Heuristic methods for solving knapsack type problems

Thekra Al-Douri

UNIVERSITE de Picardie Jules Verne

Thèse de Doctorat

Mention : Informatique
Spécialité : Recherche opérationnelle et optimisation

présentée à *l'École Doctorale en Sciences Technologie et Santé (ED 585)*

de l'Université de Picardie Jules Verne

par

Thekra AL-DOURI

pour obtenir le grade de Docteur de l'Université de Picardie Jules Verne

Méthodes heuristique pour les problèmes de type knapsack

Soutenue le 02-02-2018, après avis des rapporteurs, devant le jury d'examen :

M. Kondo ADJALLAH	**Professeur, École Nationale d'Ingénieurs de Metz**	**(Rapporteur)**
M^{me} Marie-Ange MANIER	**Maître de conférences, Université de Technologie de Befort-Montbéliard**	**(Rapporteur)**
M. Imed KACEM	**Professeur, Université de Lorraine**	**(Examinateur)**
M. Loys THIMONIER	**Professeur, Université de Picardie Jules Verne**	**(Examinateur)**
M. Mhand HIFI	**Professeur, Université de Picardie Jules Verne**	**(Directeur)**

Preface

This PhD thesis has been prepared at the EPROAD[1] laboratory at the University of Picardie Jules Verne (UPJV), during the period from November 2013 to July 2017. The work has been supervised by Professor Mhand Hifi[2].

The presentation includes a general introduction, a state of the art and three chapters, considering problems belonging to the knapsack family. Chapter 3 has been published in "International Journal of Computers and Applications" (Taylor & Francis, eds) [7]. Chapter 4 has been accepted for publication in "International Journal of Intelligent Engineering Informatics" [8]. Chapter 5 has been presented at the international conference with proceedings 47th International Conference on Computes & Industrial Engineering CIE47-2017 [6].

Acknowledgment

First of all, I would like to thank my adviser, professor Mhand Hifi for helping me through his very important suggestions and encouragements that led me to the realization of this work. I am very grateful to his supervision, steady support and inexhaustible ideas that have been extremely useful in progressing in my research.

My deep gratitude goes to professors Kondo ADJALIAH[3] and Marie-Ange MANIER[4] for accepting to review my PhD thesis and the interest that they have carried to my work. I would like to thank also Professor Imed KACEM[5] and Professor Loys THIMONIER[6] for accepting to participate in the committee of the thesis.

I'm happy to have spent my doctoral studies at the EPROAD laboratory. I would like to thank all members of this laboratory with whom I have shared lots of discussions and beautiful times.

The work on this thesis has been financially supported by research grants from ministry of higher education of Iraq. I would like to express my gratitude to them for financing my study.

Finally, I would like to express my sincere gratitude to my family, especially my mother for her unconditional support and patience over the years. Thanks for motivating me the rest of the time and her constant prayers to God.

[1] Eco-PRocédés, Optimisation et Aide à la Décision
[2] Professeur, EPROAD EA 4669, Université de Picardie Jules Verne
[3] Professeur, École Nationale d'Ingénieurs de Metz
[4] Maître de Conférences, Université de Technologie de Befort-Montbéliard
[5] Professeur, Université de Lorraine
[6] Professeur, Université de Picardie Jules Verne (UPJV)

Résumé

Titre: Méthodes heuristiques pour les problèmes de type sac à dos.

Les travaux de recherche de cette thèse s'articulent autour de la résolution du problème du sac à dos en min-max avec de multiples scénarios (en anglais, *max-min knapsack problem with multi-scenarios*). Cette thèse propose trois approches, plutôt complémentaires, en s'appuyant principalement sur l'aspect perturbation des solutions puis la reconstruction. En partant de ce principe, trois algorithmes approchés ont été étudiés, en partant d'une approche mono-solution vers des approches à base de population.

Dans une première partie, un algorithme réactif a été proposé ; il s'appuie sur deux phases imbriquées dans une recherche itérative : la phase de restauration / exploration et la phase de perturbation. La première phase part d'une solution réalisable et tente de l'améliorer en utilisant une stratégie d'exploration spécifique. Cette dernière est basée sur une série d'échanges entre les éléments appartenant ou pas à la solution courante. La deuxième phase commence par construire une solution partielle, en suppriment certains éléments de la solution courante, alors qu'une stratégie de ré-optimisation tente de sélectionner de nouveaux éléments et de les inclure dans une solution dégradée. La stratégie de destruction tente également de diversifier le processus de recherche en dégradant la qualité des solutions dans le but d'éviter des stagnations locales.

Dans une deuxième partie, une méthode à base de population a été proposée. Elle s'appuie sur trois phases. Une phase de construction de la population de départ par application d'un algorithme glouton aléatoire, une deuxième phase qui combine une série de solutions deux-à-deux, par l'utilisation de l'opérateur d'intersection et, une troisième phase qui agit sur les successeurs afin d'augmenter la qualité des solutions induites. Les deux dernières phases sont répétées jusqu'à la stabilité de la population.

Dans une troisième partie, le problème est résolu en combinant le GRASP (Greedy Randomized Adaptive Search Procedure) et le Path-relinking. Cette approche combine deux stratégies: une stratégie de construction et une autre d'amélioration. D'une part, la première stratégie produit une solution (de départ) réalisable en appliquant le GRASP. D'autre part, chaque solution courante (de départ) est améliorée en appliquant une stratégie basée sur le path-relinking : partir d'un couple de solutions « départ-arrivée », puis tenter de reconstruire le lien entre ces deux solutions en espérant rencontrer des solutions de meilleures qualités sur le chemin. Ce processus est répété sur une série de solutions.

Mots-clés: approximation, optimisation, sac à dos, scénarios, voisinage.

Abstract

Title: Heuristic methods for solving knapsack type problems.

The aim of this thesis is to propose approximate algorithms for tackling the max-min Multi-Scenarios Knapsack Problem (MSKP). Three methods have been proposed (which can be considered as complementary), where each of them is based on the perturbation aspect of the solutions and their re-buildings. The proposed methods are declined in three parts.

In the first part, we propose to solve the MSKP by using a hybrid reactive search algorithm that uses two main features: (i) the restoring/exploring phase and (ii) the perturbation phase. The first phase yields a feasible solution and tries to improve it by using an intensification search. The second phase can be viewed as a diversification search in which a series of subspaces are investigated in order to make a quick convergence to a global optimum. Finally, the proposed method is evaluated on a set of benchmark instances taken from the literature, whereby its obtained results are compared to those reached by recent methods available in the literature. The results show that the method is competitive and it is able to provide better solutions.

The second part discusses a population-based method which combines three complementary stages: (i) the building stage, (ii) the combination stage and (iii) the two-stage rebuild stage. First, the building stage serves to provide a starting feasible solution by using a greedy procedure; each item is randomly chosen for reaching a starting population of solutions. Second, the combination stage tries to provide each new solution by combining subsets of (starting) solutions. Third, the rebuild stage tries to make an intensification in order to improve the solutions at hand. The proposed method is evaluated on a set of benchmark instances taken from the literature, where its obtained results are compared to those reached by the best algorithms available in the literature. The results show that the proposed method provides better solutions than those already published.

In the third part, both greedy randomized adaptive search procedure and path-relinking are combined for tackling the MSKP. The proposed method iterates both building and improvement phases that are based upon an extended search process. The first phase yields a (starting) feasible solution for the problem by applying a greedy randomized search procedure. The second phase tries to enhance each current solution by applying the path-relinking based strategy. Finally, the proposed method is evaluated on a set of benchmark instances taken from the literature. The obtained results are compared to those reached by some best algorithms available in the literature. Encouraging results have been obtained.

Keywords: approximation; knapsack; neighborhood; optimization; scenarios.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction générale

Plusieurs situations pratiques peuvent être modélisées comme des problèmes d'optimisation combinatoire. Ce dernier consiste à rechercher un ensemble discret d'un sous-ensemble pour les meilleurs sous-ensembles, que maximisent ou minimisent un critère (ou plusieurs critères) sous certaines contraintes. Ainsi, certains problèmes pourraient être trouvés appartenant à la famille des sacs à dos. Les approches pour résoudre les problèmes d'optimisation combinatoire sont généralement divisées en deux catégories: méthodes exactes et méthodes approchées (heuristiques et méta-heuristiques). Les méthodes exactes donnent la garantie d'obtenir des solutions optimales pour les problèmes. Cependent, les méthodes approchées tentent de réaliser de "bonnes" solutions, mais pas pour obtenir les solutions optimales. En général, les méthodes utilisées pour résoudre ce type de problème sont prises en considération pour deux facteurs: la qualité des solutions et le temps de résolution. Ainsi, ces deux facteurs sont liés l'un à l'autre. Parfois, il est nécessaire de choisir entre trouver la(les) solution(s) optimale(s) ou s'appuyer sur la (des) solution(s) approchée(s). Souvent, ce choix est influencé par la nature du problème.

Les travaux de recherche de cette thèse s'articulent autour de la résolution du problème appartenant à la famille du problème du sac à dos: le problème du sac à dos max-min avec de multiples scénarios noté MSKP (en anglias max-min knapsack problem with multi-scenarios). Ce problème est une extension du problème du sac à dos dans lequel chaque élément est caractérisé par un poids et un vecteur de profits. L'objectif du problème est de maximiser le minimum de profits pour les éléments sélectionnés sur tous les scénarios sans violer la contrainte du sac à dos. Cette thèse propose trois approches, plutôt complémentaires, en s'appuyant principalement sur l'aspect perturbation des solutions puis la reconstruction. En partant de ce principe, trois algorithmes approchés ont été étudiés, en partant d'une approche mono-solution vers des approches à base de population.

Dans ce travail, nous présentons des algorithmes séquentiels basés sur les techniques de recherche par voisinage. Ces approches sont adaptées pour optimiser les instances de grande taille d'un problème d'optimisation combinatoire appelé MSKP.

ce travail est composée de deux parties. La première partie présente un état de l'art sur le problème traité avec des méthodes de résolution appropriées. Dans la deuxième partie, nous allons décrire des méthodes heuristiques pour le problème du sac à dos max-min avec de multiples scénarios.

Plus précisément, la thèse est organisée comme suit: la première partie (chapitre 2) présente l'introduction de certains problèmes appartenant à la famille du sac à dos. En fait, il commence par décrire le problème classique du sac à dos. De même, les méthodes de solution exactes et approchées décrites, que sont utilisées pour résoudre le problème. Enfin,

nous avons défini le problème MSKP avec les principales méthodes de solution utilisées dans la littérature.

La deuxième partie est composée de trois chapitres (chapitre 3 au chapitre 5). Dans le chapitre 3, un algorithme réactif a été proposé; il s'appuie sur deux phases imbriquées dans une recherche itérative: la phase de restauration / exploration et la phase de perturbation. La première phase part d'une solution réalisable et tente de l'améliorer en utilisant une stratégie d'exploration spécifique. Cette dernière est basée sur une série d'échanges entre les éléments appartenant ou pas à la solution courante. La deuxième phase peut être considérée comme une recherche locale que combine deux stratégies pour construire une solution partielle en supprimant certains éléments de la solution courante, alors que la stratégie de ré-optimisation tente de sélectionner de nouveaux éléments et de les inclure dans une solution dégradée. La stratégie de destruction tente également de diversifier le processus de recherche en dégradant la qualité de la solution dans le but d'éviter la stagnation dans un optimum local.

Dans le chapitre 4, une méthode à base de population a été proposée. La méthode proposée est basée sur trois étapes complémentaires: (i) la phase de construction, (ii) l'étape de combinaison et (iii) la phase de reconstruction en deux étapes. La phase de construction de la population de départ par application d'un algorithme glouton aléatoire; chaque élément est choisi au hasard pour atteindre une population de départ de solutions. La deuxième phase qui combine use série de solutions deux-à-deux, par l'utilisation de l'opérateur d'intersection et, la troisième phase de reconstruction tente de faire une intensification pour améliorer les solutions en main. Les deux dernières phases sont répétées jusqau's à la stabilité de la popultion.

Le chapitre 5 présente une heuristique hybride que combine le GRASP (Greedy Randomized Adaptive Search Procedure) et le Path-relinking pour aborder le MSKP. Cette approche combine deux stratégies: (i) une stratégie de construction et (ii) une autre d'amélioration. D'une part, la première stratégie produit une solution (de départ) réalisable en appliquant le GRASP. D'autre part, chaque solution courante (de départ) est améliorée en appliquant une stratégie basée sur le path-relinking: partir d'un couple de solutions « départ-arrivée », puis tenter de reconstruire le lien entre ces deux solutions en espérant recontrer des solutions de meilleures qualités sur le chemin. Ce processus est répété sur une série de solutions.

Enfin, le chapitre 6 donne une conclusion générale dans laquelle nous résumons les méthodes proposées.

1.2 Le problème du sac à dos classique et ses variantes

Dans cette partie, nous introduisons quelques problèmes de type sac à dos. Nous commençons par décrire le problème classique du sac à dos, noté KP. Nous définissons d'abord ce problème et commenterons sa formulation standard. Ensuite, nous illustrerons son rôle majeur dans le domaine de l'optimisation combinatoire, en particulier lorsqu'il se présente comme sous problème dans des applications réelles. Finalement, nous illustrons les méthodes exactes et approximatives disponibles dans la littérature c'est-à-dire ont traité le problème du sac à dos binaire.

Dans la littérature, il existe plusieurs variantes du problème du sac à dos (cf., Figure 1.1). Parmi ces variantes, dans ce qui suit, nous citons la variante du problème du sac à dos que nous avons étudié dans cette thèse, à savoir le problème du sac à dos max-min avec de multiples scénarios noté MSKP (en anglias max-min knapsack problem with multi-scenarios). Cependant, cette famille a été largement considérée et ne peut pas être entièrement couverte ici.

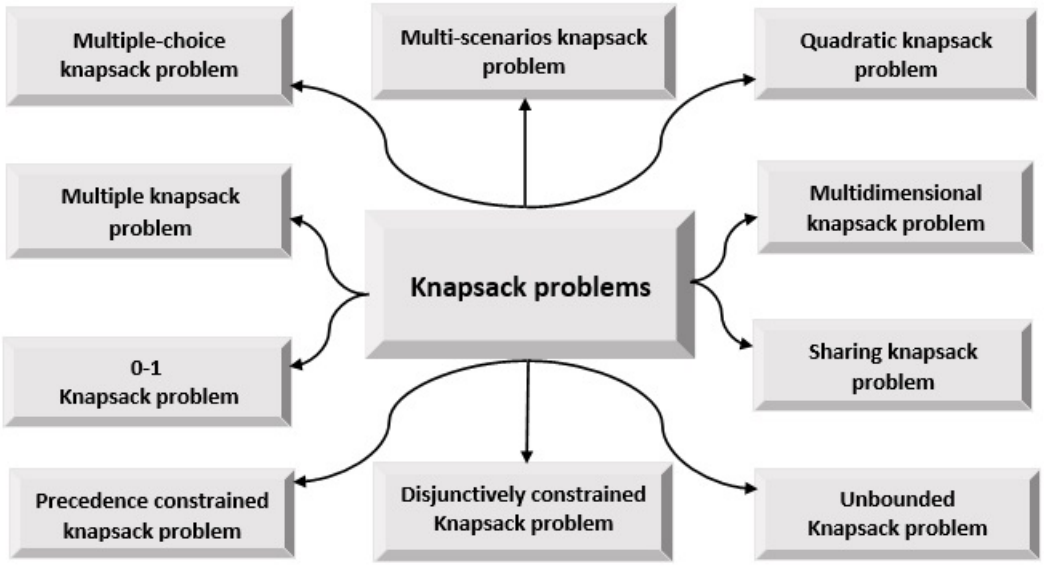

Figure 1.1: Quelques variantes du problème du sac à dos

Nous présentons aussi du problème lié à la famille KP: le problème du sac à dos max-min avec de multiples scénarios (MSKP). Pour ce problème, les méthodes correspondantes (issues de la littérature) seront décrites.

1.3 Le problème du sac à dos binaire

Le problème du sac à dos binaire (noté 0-1 KP) est considéré comme l'un des problèmes plus connus en optimisation combinatoire (cf., Martello *et al.* [54], Pisinger *et al.* [64]). Il est classé comme des problèmes NP-difficiles. La première étude est apparue en 1897 (cf., Mathews [56]), alors que Danzig a utilisé l'expression "knapsack" dans ses premiers travaux. Il existe plusieurs applications importantes pour les problèmes du sac à dos dans des domaines

pratiques et théoriques. Dans le domaine pratique, il existe de nombreuses applications, plus particulièrement dans différents domaines comme l'industrie, la logistique des transports, la gestion financière, comme pour le contrôle budgétaire (cf., Lorie et Savage [51]), sélection de projet (cf., Petersen [59]), problème du stock de coupe (cf., Gilomore *et al.* [29], [30], [31]), et problème du chargement (cf., Shih [68], Bellmam [12]). Outre les applications pratiques, les modèles théoriques sont très importants dans de nombreux problèmes appartenant à la famille du sac à dos parus souvent en tant que problèmes réduits ou sous problèmes dans les procédures de solution de problèmes plus complexes (cf., Hifi *et al.* [39], Pisinger [61]).

1.3.1 Définition et formulation mathématique

Dans la définition générale, le problème du sac à dos, étant un problème qui consiste en un ensemble I de n éléments, où chaque élément $i \in I = \{1, \ldots, n\}$ est caractérisé par le profit p_i et le poids w_i et un fixe de la capacité du sac à dos c. Le but du problème est de remplir le sac à dos avec un (sous-)ensemble d'éléments de I de manière que: le profit total de la sélection d'objets soit maximisé et le poids total ne dépasse pas c.

La représentation mathématique du 0-1 KP peut être indiquée comme suit:

$$\max \quad \sum_{i=1}^{n} p_i x_i \tag{1.1}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_i \leq c \tag{1.2}$$

$$x_i \in \{0, 1\}, \ \forall \ i \in I = \{1, ..., n\} \tag{1.3}$$

Ce modèle est représenté par une fonction objective (equation 1.1), une contrainte dite du sac à dos ou de capacité (inégalité 1.2) et la contrainte d'intégralité sur les variables de décision (inégalité 1.3). Afin d'éviter les cas triviaux, on suppose que:

- Toutes les valuers $c, p_i, w_i \ \forall i \in I$ sont des nombres entiers positifs.
- $\forall i \in I, \ w_i \leq c$ (i.e., non l'element à supérieur poids à la capacité du sac à dos).
- $\sum_{i=1}^{n} w_i > c$.

Un vecteur $(\hat{x_1}, \ldots, \hat{x_n})$ de n variables de décision binaires est considéré comme une solution réalisable si la somme des poids pour toutes les variables de décision ne dépasse pas la capacité du sac à dos c; c'est-à-dire s'il satisfait à la contrainte du sac à dos suivant:

$$\sum_{i \in I} w_i \hat{x_i} \leq c$$

Aussi, la solution réalisable est une solution optimale si elle a obtenu la meilleure (la plus grande) valeur parmi l'ensemble de toutes les solutions réalisables ; elle est notée par $\overline{x}$. Autrement dit, pour toutes les solutions réalisables $\hat{x}$, nous avons:

$$\sum_{i \in I} p_i \overline{x}_i \geq \sum_{i \in I} p_i \hat{x_i}$$

Bien qu'il s'agisse d'une formulation simple, sa résolution peut nécessiter un temps d'exécution importante. En effet, 0-1 KP est considéré l'un des premiers problèmes NP-difficile à résoudre comme le montre Karp [44]. Il existe de nombreuses méthodes dans la littérature pour la résoudre. Parmi ces méthodes, des méthodes exactes et approchées ont été proposées pour obtenir une solution optimale ou approchée. La programmation dynamique et les approches hybrides et séparation et évaluation sont considérées comme des méthodes efficaces pour résoudre 0-1 KP (cf., Horowitz and Sahni [42], Martello and Toth [54]). Les deux méthodes sont basées sur des stratégies de recherche en profondeur, d'abord pour explorer efficacement l'espace.

Afin d'améliorer certaines des procédures de solution, Balas et Zemel [10] ont proposé la *détermination*, considéré comme un outil efficace pour résoudre des problèmes de grandes dimensions. Plus tard, les méthodes les plus réussies proposées pour résoudre le KP ont été inspirées par le concept (cf., Martello et Toth [54], Fayard et Plateau [23] et Pisinger [62]). Martello *et al.* [53] ont proposé une approche hybride combinant une programmation dynamique et des bornes robustes. Certains chercheurs ont mentionné dans d'autres types de KP et ne se sont pas limités à une seule contrainte. Il s'agit du problème du sac à dos multidimensionnel (ou multicontrainte), qui a été examiné par Pisinger [62] et Chu et Beasley [16]. Le problème du sac à dos bidimensionnel est considéré comme un cas particulier de problèmes du sac à dos multiple où le nombre de contraintes est égal à deux (cf., Freville and Plateau [26]).

1.3.2 Calcul des bornes

La relaxation de la programmation linéaire du problème du sac à dos (LKP) est obtenue en permettant aux variables de prendre des valeurs réelles (généralement positives) au lieu de seulement des valeurs entières dans la solution optimale. En effet, la valeur optimale de la solution (pour un problème de maximisation) du problème relaxe est supérieure à la valeur de la solution du problème binaire (parce que l'ensemble des solutions réalisables pour 0-1 KP est un sous-ensemble des solutions réalisables pour le problème relaxe).

Le programme linéaire peut être formulé comme suit:

$$LKP \quad \begin{cases} \max & \sum_{i=1}^{n} p_i x_i \\ s.t. & \sum_{i=1}^{n} w_i x_i \leq c \\ & 0 \leq x_i \leq 1 \qquad i = 1, ..., n \end{cases}$$

Supposons que tous les objets $i, i \in I$, soient triés par ordre décroissant de leurs profits/poids, c'est-à-dire :

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

Pour le problème du sac à dos, la première limite supérieure a été proposée par Dantzig [18]. La solution de LKP peut être trouvée en utilisant la méthode gloutonne 1.4.1. Supposons que les éléments soient consécutivement mis dans le sac jusqu'à ce qu'il soit rempli, et détermine le premier élément x_s qui ne s'insère pas dans le sac à dos. De sorte que cet élément est appelle l'élément critique (élément de pause):

$$s = \min\left\{j : \sum_{i=1}^{j} w_i > c\right\}$$

Ensuite, le vecteur solution optimale $\overline{x}_i$ de LKP peut être représenté comme suit:

$$\overline{x}_i = \begin{cases} 1 & if\ i \in \{1, ..., s-1\} \\ 0 & if\ i \in \{s+1, ..., n\} \\ \frac{1}{w_s}(c - \sum_{i=1}^{s-1} w_i) & if\ i = s \end{cases}$$

La valeur solution correspondante optimale de LKP peut être représentée selon l'équation suivante:

$$Z^{LKP} = \sum_{i=1}^{s-1} p_i + (c - \sum_{i=1}^{s-1} w_i)\frac{p_s}{w_s}$$

La limite supérieure pour le problème 0-1 KP (notée UB) est une solution optimale du LKP: parce que toutes les variables de 0-1 KP sont entières, la solution optimale du 0-1 KP n'est toujours plus petite que la solution optimale du LKP: cette borne supérieure i appelée limite supérieure de Dantzig pour 0-1 KP (cf., Danzing [18]):

$$Z^{(0-1)KP} \leq UB = \lfloor Z^{LKP} \rfloor = \sum_{i=1}^{s-1} p_i + \lfloor \frac{(c - \sum_{i=1}^{s-1} w_i)}{w_l} p_l \rfloor$$

Où $\lfloor x \rfloor$ indique le plus grand nombre entier inférieur à x.

1.4 Quelques méthodes de résolution pour le sac à dos

Dans l'optimisation, les méthodes pour résoudre les problèmes NP-difficiles sont divisées en deux catégories principales: les méthodes exactes et les méthodes approchées. Dans cette partie, nous rappelons exactes et approchées les méthodes de résolution pour résoudre le 0-1 KP. Les méthodes exactes peuvent avoir la capacité de trouver une solution optimale avec un énorme effort de calcul. Ils peuvent être seulement un succès pour résoudre des problèmes de petite taille. L'obstacle est que le temps d'exécution est en croissance exponentielle avec la taille de l'instance. Les méthodes approchées donnent une solution avec une bonne qualité qui peut être suboptimale ou optimale. Cependant, ces méthodes sont capables de trouver une solution avec un temps de calcul acceptablement bas.

1.4.1 Méthodes approchées

Le problème du sac à dos est un problème NP-difficile. Cela signifie qu'il faut un long temps de fonctionnement pour résoudre. Par conséquent, cela conduit à l'utilisation de méthodes de résolution approchées, en particulier pour les grandes instances. Les méthodes de solution approximatives sont approchées et caractérisées par deux propriétés importantes: d'abord, elle trouve une solution réalisable pour tout problème dans un temps d'exécution peu élevé. Le deuxième aspect est que la qualité de l'algorithme d'approximation utilisé est le maximum entre ses solutions et les solutions optimales. Parmi ces méthodes, nous mentionnons une

méthode simple pour déterminer une solution réalisable du KP immédiatement, ce qui est une méthode gloutonne.

La méthode gloutonne est considérée comme l'une des méthodes approchées. Elle est basée sur le principe de la gloutonne (cf., Sanjoy *et al.* [66]). Elle est basée sur la construction d'une solution pièce par pièce, toujours choisir la prochaine pièce qui offre le bénéfice le plus évident et immédiat. Normalement, la stratégie gloutonne ne conduit pas toujours à une solution globalement optimale, mais elle est très puissante et fonctionne correctement pour le problème étudié.

La méthode gloutonne simple est basée sur les principes de Dantzig [18] pour résoudre le 0-1 KP. L'idée de la stratégie gloutonne trie les éléments selon le rapport du profit au poids en ordre décroissant, de sorte que:

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

Le premier élément placé dans le sac à dos a le rapport le plus élevé, puis les autres éléments sont choisis de la même manière jusqu'à ce que la capacité du sac à dos soit dépassée. L'algorithme gloutonne est décrit dans Algorithme 1.

Algorithme 1 Une solution d'initiale réalisable.

Entrée : une instance de 0-1 KP
Sortie : une solution réalisable $\overline{x}$.

1. Mettre $\overline{c}$ comme la capacité du sac à dos disponible, $\overline{c} \leftarrow c$;
2. Mettre $x_i = 0,\ \forall i \in \{1, \ldots, n\}$;
3. **Pour** $i = 1$ à n **faire**
4. **si** $w_i \leq \overline{c}$, **alors**
5. $x_i = 1$
6. $\overline{c} = \overline{c} - w_i$
7. **Finsi**
8. $i = i + 1$
9. Jusqu'à $\overline{c} = 0$ ou $i = n$
10. **FinPour.**
11. **Retourner** $\overline{x}$

1.4.2 Méthodes exactes

Plusieurs méthodes exactes pour le 0-1 KP ont été proposées dans la littérature. La plupart de ces méthodes font appel à des techniques de recherche énumératives. Dans cette partie, nous allons mentionner certaines de ces méthodes comme les plans de coupe, de branch and cut, séparation et évaluation et la programmation dynamique.

1.4.2.1 La méthode du plan de coupe

La méthode du plan de coupe a d'abord été suggérée par Gomory [34] comme une méthode pour résoudre des problèmes de programmation de nombres entiers et de nombres entiers. Il est basé sur l'approche du plan de coupe, ou l'idée principale est basée sur l'ajout de contraintes (cf., Figure 1.2) afin de réduire l'espace de recherche des solutions réalisables et de répéter la procédure jusqu'à obtenir la meilleure solution entière. Balas *et al.* [9] sont expliqué une version efficace de la méthode pour résoudre les programmes mixtes 0-1 en utilisant les coupures de lift-and-project.

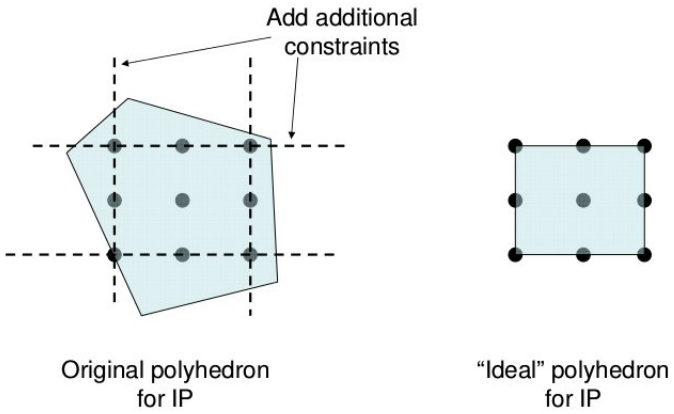

Figure 1.2: Exemple d'ajout de contraintes

Algorithme 2 un Gomory's plane de coupe
Entrée : une instance de 0-1 KP
Sortie : une solution optimale $x^\star$.

1. Mettre en place un programme de relaxation pour 0-1 KP (noté LP 0-1 KP).
2. Mettre $x^\star = 0$.
3. **Etapes itératives**
4. Résoudre le courant LP 0-1 KP, et laisser $\hat{x}$ être une solution optimale de base.
5. **Si** $\hat{x}$ est entier **alors**.
   $x^\star = \hat{x}$.
   arrêtez.
6. **sinon**
7. Générer une nouvelle coupe autour du point (comme nouvelle contrainte) et l'ajouter au LP.
8. **Finsi**
9. **Fin du** $x^\star$ est une solution entière optimale.
10. **Retourner** $x^\star$ une solution optimale.

Algorithme 2 illustre un algorithme de plan de coupe pour résoudre le 0-1 KP. Dans l'algorithme, les trois étapes principales sont répétées: d'abord, résolvez la relaxation de la programmation linéaire (en utilisant la méthode simplexe) et obtenez une solution optimale de base $\hat{x}$. D'une part, si $\hat{x}$ est une solution entière, puis la solution optimale est obtenue (noté $x^\star$). D'autre part, la dernière étape vise à compléter un algorithme en générant une nouvelle coupe (en tant que nouvelle contrainte d'inégalité) et en l'ajoutant à LP et à retourner à la première étape.

1.4.2.2 La méthode de programmation dynamique

La méthode de programmation dynamique est considérée comme un outil utile pour résoudre certains problèmes d'optimisation combinatoire. Bellman [12] a donné une première introduction inclusive à cette méthode. Elle peut s'attaquer à des problèmes qui impliquent des variables de décision qui interagissent les unes avec les autres et ont un temps dépendant. L'idée fondamentale de la méthode considère la solution optimale d'un problème résultant d'une combinaison de solutions optimales aux sous problèmes. Toth [73] a développé les algorithmes de programmation dynamique pour le problème 0-1 KP parce que les procédures originales pour le calcul de la fonction du sac à dos sont présentées et utilisées de limites pour éliminer les états ne conduisant pas à la solution optimale lors de l'analyse. Hifi [36] a proposé un algorithme qui combine les techniques de programmation dynamique et une

recherche de depth-first en utilisant Hill-Climbing stratégies pour résoudre le problème de stock du coupe bidimensionnel contraint.

1.4.2.3 La méthode par séparation et évaluation

La méthode par séparation et évaluation (en anglais Branch and Bound) est une méthode générale pour une résolution exacte optimale de différents types de problèmes d'optimisation. Land *et al.* [49] sont les concepteurs de la procédure par séparation et évaluation. Le premier algorithme de séparation et évaluation résoudre le problème du sac à dos binaire a été proposé par Kolesar [46]. L'algorithme a été développé par Horowitz *et al.* [42].

Le concept général de méthodes séparation et évaluation est basé sur une structure arbre de solutions. Chaque nœud de l'arbre divise l'espace de recherche en deux sous-espaces jusqu'à une exploration complète de l'espace de solution (cf., Dasgupta *et al.* [20]). Le nombre de solutions réalisables 2^n peut être dérivé de variables binaires n, ainsi, l'espace de recherche devient très grand (cf., Kellerer *et al.* [45]). les méthodes de séparation et évaluation sont basées sur trois principes principaux. Ils sont:

- La stratégie de séparation.
- La stratégie d'évaluation.
- La stratégie de choix du nœud.

Le but de base de la méthode BB est de diviser l'espace de solution en sous-espaces plus petits qui peuvent être résolus indépendamment (séparation). Ceci afin d'arriver à la solution optimale. Le processus est en principe répété jusqu'à ce que chaque sous-espace ne contienne qu'une seule solution réalisable. La stratégie de séparation: ce principe exécute une série à l'instance du KP. Les membres de séparation peuvent être pris dans un ordre prédéfini, ou comme un élément-défini explorant. Parfois, le choix de l'élément de séparation est important pour améliorer les performances de l'algorithme. La stratégie d'évaluation est un principe important, car elle organise si un sous-espace contient une solution optimale; dans le même temps, elle peut empêcher l'arbre de recherche de trop grande. Une autre signification, elle fonctionne en comparant et en calculant la solution du sous-espace avec la meilleure solution trouvée. Si cela signifie que le sous-espace contient la solution optimale, sinon ce sous-espace est ignoré. La stratégie du choix du nœud détermine comment choisir le prochain nœud dans l'arbre de recherche pour la séparation. De plus, il existe d'autres stratégies: les plus courantes incluent la best-first, qui détermine le nœud qui donne la meilleure valeur de la fonction de délimitation de l'arbre de recherche dans n'importe quel chemin.

Nous présentons maintenant l'algorithme suggéré par Horowitz *et al.* [42] (algorithme 3).

Algorithme 3 séparation et évaluation pour KP

Entrée : une instance de 0-1 KP
Sortie : une solution optimale $x^\star$.

1. Laisser $i = 1$; *Meilleurevaleur = 0.*
2. Calculer la borne supérieure UB pouvant être atteinte pour i.
3. **Si** $UB \geq$ *Meilleurevaleur* **alors**
4. Développer une branche de l'arbre (en profondeur d'abord) afin d'obtenir une solution réalisable x'.
5. **Si** $Z(x') \geq$ *Meilleurevaleur* **alors**
6. $x^\star = x'$.
7. *Meilleurevaleur* $= Z(x')$
8. **Finsi**
9. **Finsi**
10. $i = max\{s \leq i : x'_s = 1\}$
11. **tant que** i existe
12. $x'_i = 0$
13. **fin tant que**
14. Sortir avec $x^* = x'$.

Cet algorithme considère que tous les éléments sont triés dans le rapport décroissant de profit par poids (cf., partie 1.4.1). Ensuite, la séparation pour les éléments est la suivante: à chaque nœud, nous avons choisi l'élément i qui a un ratio maximum (profit par poids). Nous développons deux branches: la première, $x_i = 1$ correspond à l'élément i qui est placé dans le sac à dos, et l'autre $x_i = 0$ correspond à l'élément i qui n'y est pas. Nous continuons la recherche jusqu'à ce que le sac soit rempli. Cet algorithme a utilisé une stratégie de depth-first, ce qui signifie que les éléments mis en premier dans le sac à dos qui ont le meilleur profit par ratio. La fonction d'évaluation utilise les bornes de Danzig présentées dans (section 1.4.1). On évalue la solution courante en calculant la borne supérieure et en la comparant à la précédente solution.

1.4.2.4 La méthode du branch-and-cut

La méthode branch-and-cut est considérée comme très importante pour résoudre plusieurs problèmes d'optimisation combinatoire. C'est une combinaison de méthode de plan de coupe

et de séparation et évaluation. Ainsi, ce n'est pas une méthode efficace pour résoudre une programmation entière générale en utilisant uniquement une méthode de plan de coupe. Donc, il faut appliquer la méthode séparation et évaluation (cf., Crowder *et al.* [17], John [58]). L'application des plans de coupe conduit à une forte réduction importante de la taille de l'arbre de recherche pour l'approche de séparation et évaluation. Ainsi, l'approche séparation et évaluation peut être accélérée par l'emploi du plan de coupe.

Et il a expliqué que la zone des algorithmes de dérivation et de coupe évolue en permanence. John [58] a expliqué que la méthode branch-and-cut peut garantir la solution optimale pour ces problèmes. Il a expliqué que la zone des algorithmes de dérivation et de coupe évolue en permanence, et a la possibilité de devenir plus important avec le développement rapide des ordinateurs et de l'informatique parallèle. Le plus connu des utilisations des algorithmes est celle de branche-and-cut pour résoudre le travelling salesman problème (TSP). De même, pour les problèmes importants et / ou difficiles, le branch-and-cut peuvent être utilisés en conjonction avec des heuristiques ou des métaheuristiques afin d'atteindre une solution quasi optimale.

1.5 Le problème du sac à dos max-min avec de multiples scénarios (MSKP)

Le problème du sac à dos max-min avec de multiples scénarios (noté MSKP) est une variante du problème du sac à dos unique bien connu avec des multiobjctive. Le MSKP est un NP-difficile d'optimisation combinatoire problème. Précédemment, ce problème avait été examiné dans le cadre d'une prise de décision multi-critères (cf., Gass *et al.* [28], Steuer [70]). Le MSKP a été proposé pour la première fois par Yu [79] en application de l'optimisation robuste comme un problème du sac à dos max-min. Il existe plusieurs applications pratiques du MSKP dans les réseaux de distribution d'eau, les conceptions d'ingénierie et la gestion financière. Ici, mentionnons un exemple pris des conceptions d'ingénierie, plus précisément dans la conception d'une voiture exigeant les deux critères, le coût et de la fiabilité. Le client souhaite avoir une voiture avec une fiabilité maximale et un coût minimal, mais un niveau de fiabilité plus élevé entraîne généralement un coût plus élevé (cf., Wiecek *et al.* [78]).

1.5.1 Définition et modèle mathématique

Formellement, une instance du MSKP est caractérisée par un sac à dos de capacité fixe c et un ensemble I de n éléments. Chaque élément i se caractérise par un poids w_i et un ensemble du profit p_i^s, où les profits varient pour chaque scénario possible s (cf., Yu [79]). En d'autres termes, un scénario est défini par l'ensemble des profits qui s'appliquent à chaque élément (cf., Iida [43]). Le problème du sac à dos classique est un cas particulier du MSKP dans lequel s est égal à 1; c'est-à-dire qu'il n'y a qu'un scénario (cf., Kellerer *et al.* [45], Martello and Toth [54], Garey *et al.* [27]). L'objectif du MSKP est de déterminer le sous-ensemble d'éléments dont le poids total ne dépasse pas la capacité du sac à dos et dont le profit total est maximisé dans le pire scénario sur tous les scénarios possibles (cf., Pinto *et al.* [60]).

Soit x_i la variable de décision associée à l'élément i tel que x_i prend la valeur 1 si l'élément i est sélectionné, sinon x_i prend la 0. La contrainte du sac à dos est représentée par:

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

La fonction objective du problème est définie comme une formule du max-min en introduisant s des valeurs de profit au lieu d'une pour chaque élément, et l'objectif est de sélectionner le sous-ensemble d'items par rapport à la contrainte de capacité c qui a pour maximiser la valeur minimale d'un ensemble de fonctions linéaires. Ensuite, la programmation linéaire en nombres entiers du MSKP peut être formulée comme suit:

$$\max \quad \min_{s=1,\dots,S} \Big\{ \sum_{i=1}^{n} p_i^s x_i \Big\} \tag{1.4}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_i \leq c \tag{1.5}$$

$$x_i \in \{0,1\}, \qquad \forall\, i \in I \tag{1.6}$$

Dans cette programmation linéaire en nombres entiers, nous avons trois types d'équations. La première équation (1.4) est la fonction objective où l'objectif est de maximiser le profit total des items dans le pire scénario globalement les scénarios possibles. La contrainte (1.5) représente la capacité du sac à dos et impose que la somme totale des poids des éléments dans le sac à dos ne dépasse pas la capacité du sac à dos. La contrainte (1.6) impose d'inclure en totalité ou pas du tout un élément. Sans perte de généralité, nous supposons que:

(i) Toutes les données d'entrée $c, p_i, w_i, \forall i \in I$ sont des entiers strictement positifs.

(ii) $\sum_{i\in I} w_i > c$ pour éviter des solutions triviales.

1.5.2 Méthodes de résolution

Dans cette partie, nous citons les méthodes de résolution utilisées pour résoudre le problème du sac á dos max-min avec de multiples scénarios.

1.5.2.1 Méthodes approchées

Hifi *et al.* [37] ont proposé un algorithme adaptatif pour résoudre le MSKP. Leur approche a été combinée en deux phases: la première phase applique une stratégie de réduction polynomiale afin de réduire la taille du problème. La deuxième phase applique la procédure de recherche adaptative basée sur une procédure dynamique à 2-opt afin d'atteindre à une solution réalisable. Les auteurs ont utilisé l'algorithme glouton afin de construire une meilleure solution réalisable. En effet, la réduction de stratégie appliquée dans leur algorithme est un test de pegging qui est très utile pour réduire la taille du problème. Enfin, la stratégie 2-opt est appliquée afin d'améliorer une solution initiale et d'obtenir une solution de haute qualité pour le problème.

Song *et al.* [69] ont abordé dans les instances plus grandes jusqu'à $n = 10000$ et $S = 100$ en suggérant une collection d'algorithmes de m-échange incomplets comme algorithme approché afin d'obtenir une solution au MSKP. Ils ont expliqué que la solution optimale ne pouvait pas être trouvée en utilisant un algorithme de séparation et évaluation (c'est-à-dire un algorithme exact) à travers un temps raisonnable pour certaines instances à grande taille. Dans leurs algorithmes, premièrement, ils ont présenté une procédure détaillée en utilisant une procédure sous gradient pour obtenir la solution initiale réalisable et les limites

supérieures et inférieures. Ensuite, ils ont proposé d'échanger les valeurs de certaines des variables binaires d'un problème et ont obtenu des solutions de haute qualité.

1.5.2.2 Méthodes exactes

Certains articles dans les littératures abordant le MSKP avec des méthodes de solution exacte. Kouvelis et Yu [47], Yu [79] et Iida [43] ont résolu le MSKP en utilisant la relaxation de substitution basée sur l'algorithme séparation et évaluation pour les instances avec le nombre d'élément jusqu'à 90 et le nombre de scénarios jusqu'à 30. Ensuite, Taniguchi *et al.* [71] a suggéré un algorithme qui utilise d'abord une sorte la relaxation de substitution pour trouver des limites supérieures et inférieures rapidement et appliqué test de pegging pour réduire la taille du problème et, après, résoudre le problème réduit par séparation et évaluation algorithme. Leur algorithme a été effectué pour les instances quand $n \leq 1000$ et $s \leq 30$.

Taniguchi *et al.* [72] ont développé son travail dans [71] en prenant un cas particulier du MSKP avec seulement deux scénarios. Les auteurs ont proposé un algorithme approché et une exacte pour résoudre ce problème. Ils ont utilisé une relaxation de substitution pour dériver les limites supérieures et inférieures très rapidement et finalement, pour résoudre le problème réduit, ils ont utilisé cette relaxation pour obtenir la limite supérieure dans la séparation et évaluation. Ils ont expliqué que leur algorithme proposé reste très efficace pour le problème de petite taille, mais est moins efficace pour les problèmes fortement corrélés.

Hanafi *et al.* [35] ont proposé une méthode hybride comme méthode exacte pour résoudre le MSKP lorsque $s = 2$. Les auteurs ont suggéré dans leur approche combinant les heuristiques et le réglage temporaire des variables afin de réduire le gap entre les limites supérieures et inférieures. Ils ont suggéré plusieurs formulations de programme linéaire mixte équivalent MSKP et en ajoutant des pseudos-coupures dans un problème qu'ils ont essayé d'atteindre les solutions optimales. En plus de détails, leur algorithme se compose de trois étapes principales: (i) construire une ou plusieurs pseudo-coupures, et mettre à jour la meilleure limite supérieure du problème doit résoudre une ou plusieurs réflexions du problème courant; (ii) pour obtenir une ou plusieurs solutions réalisables du problème initiale et mettre à jour la meilleure limite inférieure, doivent résoudre un ou plusieurs problèmes réduits résultant des solutions optimales des relaxations précédentes; (iii) arrête l'algorithme lorsque le critère d'arrêt est satisfait et atteint les meilleurs limites supérieures et inférieures.

Finalement, Pinto *et al.* [60] ont suggéré une méthode de solution exacte basée sur la génération de la colonne et la séparation et évaluation. Les auteurs ont dit que leur méthode dépendait d'une reformulation du modèle de problème basé sur le principe de décomposition de Dantzig-Wolfe qui fournit des limites supérieures plus fortes. Alors que les sous-problèmes correspondants peuvent être utilisés en même temps pour classer les colonnes attrayantes et pour obtenir une solution réalisable pour le MSKP qui peut améliorer la solution courante.

1.5.2.3 L'algorithme par génération de colonnes

L'algorithme de génération de colonnes est une méthode efficace pour résoudre les problèmes de programmation linéaire contenant un nombre grand de variables, bloquant l'application

de l'algorithme Simplex au problème dans son ensemble. Gilmore et Gomory [29], [30] ont d'abord proposé la méthode de génération de colonnes pour le problème du stock de coupe bien connu, qui est une technique de décomposition pour résoudre un linéaire structuré Programme (LP) avec quelques lignes et beaucoup de colonnes. Vanderbeck [76] a présenté la tentative de concevoir un algorithme exact efficace basé sur la génération de colonnes pour le problème du stock de coupe.

Vance *et al.* [75] ont proposé de résoudre le problème du stock de coupe en utilisant la génération de colonnes et de séparation et évaluation pour obtenir des solutions entières optimales. Ils ont suggéré de formuler la règle de séparation qui peut être incorporée dans le sous problème pour permettre la génération de colonnes sur n'importe quel noeud dans l'arbre de séparation et évaluation.

Mehrotra et Trick [57] ont utilisé la génération de colonne pour résoudre le problème de coloration graphique et ils ont expliqué que cette méthode était très efficace pour résoudre de grandes instances problémes. Bjorklund *et al.* [13] ont proposé cette méthode pour résoudre le spécial d'accès multiple scheduling.

Hifi *et al.* [15] ont utilisé une génération de colonnes pour résoudre les problèmes du sac à dos multidimensionnels à choix multiples à grande taille en proposant une méthode dans laquelle se composent deux étapes complémentaires: une solution d'arrondissement d'une étape et une procédure de solution exacte restreinte. L'application du principe de décomposition de Dantzig-Wolfe à un problème d'entier linéaire conduit à une reformulation en un problème de base et un ou plusieurs sou-problèmes définis à partir des contraintes de la formulation originale.

Pinto *et al.* [60] ont développé une méthode exacte pour le MSKP. Leur approche repose sur la combinaison de la génération de colonnes et des procédures séparation et évaluation. L'algorithme de séparation et du Price qui en a résulté s'est révélé capable, de différentes manières, de surpasser les autres méthodes de pointe disponibles dans la littérature. La reformulation de la génération de colonnes fournissait des limites supérieures plus fortes et les sous-problèmes correspondants pouvaient, en même temps, être utilisés pour évaluer les colonnes attrayantes et générer des solutions réalisables (presque optimales). Une telle approche dépend de la reformulation du modèle de programmation d'entiers compacts standard basé sur le principe de décomposition de Danzig-Wolfe.

En général, l'idée principale de la génération de colonnes est d'utiliser des variables à taille réduite qui peuvent être actives dans la solution optimale et ignorer les autres variables qui sont fixées à zéro. Les variables réduites utilisées peuvent avoir le potentiel d'améliorer la fonction objective. Cette méthode est basée sur la décomposition de la programmation linéaire originale (LP) en un problème du maître et une sous-problème. Le problème maître contient un premier sous-ensemble des colonnes et le sous-problème, qui est un problème de séparation pour le LP dual, tel qu'il est résolu pour identifier si le problème du maître doit être étendu avec des colonnes supplémentaires ou non. La méthode de génération de colonnes alterne entre le problème du maître et le sous-problème jusqu'à ce que l'ancien contienne

toutes les colonnes nécessaires pour obtenir une solution optimale pour l'original (LP). La Figure 1.3 illustre la méthode de génération de colonnes. Une technique spéciale dans la programmation linéaire qui utilise ce type de méthode est l'algorithme de décomposition de Dantzig and Wolfe [19].

Figure 1.3: La méthode de génération de colonne

1.5.2.4 L'algorithm de branch-and-price

Branch-and-Price est une méthode d'optimisation combinatoire pour résoudre des problèmes de programmation linéaire entière (ILP) et de programmation linéaire multiple (MILP) avec de nombreuses variables. Il existe de nombreuses applications réussies dans les domaines de la Branch-and-Price dans l'industrie (cf., Lübbecke and Desrosiers [52]), aussi à des problèmes d'optimisation combinatoire générique comme bin backing et le problème du stock de coupe (cf., Vanderbeck [76]) et machine scheduling (cf., Van *et al.* [74]).

Cette méthode combine la séparation et évaluation avec des méthodes de génération de colonnes. Barnhart *et al.* [11] ont expliqué que la méthode de la branch-and-price est semblable à celle de la branche et de la coupe, sauf que la procédure se concentre sur la génération de colonnes plutôt que sur la génération de colonnes. En effet, le Price et la coupe sont des procédures complémentaires pour resserrer une relaxation LP. Dans l'algorithme de Branch-and-price, les colonnes sont exclues de la relaxation du LP car il existe de nombreuses colonnes à gérer efficacement et la plupart d'entre elles auront leur variable associée égale à zéro dans une solution optimale en toute façon. Ensuite, pour tester la solution optimale d'un LP, le sous-problème appelé problème du pricing qui est un problème de séparation pour le LP dual est résolu à tenter d'identifier les colonnes pour entrer sur la base. Si ces colonnes sont trouvées, le LP est réoptimisée la branch se produit alors qu'aucune colonne ne dépasse le price et que la solution LP ne satisfait pas les conditions d'intégralité branch-and-prcie (cf., Barnhart *et al.* [11], Lübbecke and Desrosiers [52], Desrosiers and Lübbecke [21]).

1.6 Conclusion

Ce chapitre a introduit certaines variantes appartenant à la famille du sac à dos, connues comme des problèmes d'optimisation combinatoire. Pour résoudre ces problèmes, il existe

deux directions de recherche: des méthodes exactes et approchées. Un algorithme exact tente de trouver un optimum ou un ensemble de solutions optimales pour un problème donné, alors qu'un algorithme heuristique est une procédure alternative qui vous permet de résoudre des instances à grande échelle en fournissant de bonnes solutions, pas nécessairement les optimales, mais dans un cadre acceptable temps de solution.

General introduction

Several practical situations can be modeled as a combinatorial optimization problem. The latter consists of searching a discrete set of a subset for the best subsets, which are maximizing or minimizing a criterion (or several criteria) under certain constraints. So, some problems can be found belonging to the knapsack family. Approaches for solving combinatorial optimization problems are usually divided into two categories: exact methods and approximate (heuristic and meta-heuristic) methods. The exact methods give the guarantee to obtain optimal solutions for the problems. While, the approximate try only to yield a good solution, but not obtain to get the optimal solutions. In general, the methods that are used to solve this type of problem into consideration for two factors: the quality of the solutions and the time of resolution. Thus, these two factors are related each the other. Sometimes, it is necessary to choose between finding the optimal solution(s) or relying on the approximate solution(s). Often this choice is influenced by the nature of the problem.

We are interested in studying the problem belonging to the knapsack problem family: the multi-scenarios max-min knapsack problem, denoted by MSKP. This problem is an extension of the binary knapsack problem that each item is characterized by a weight and a vector of profits. The objective of this problem is to maximize the minimum of the profits for the selected items over all the scenarios without violating the knapsack constraint.

Herein, the neighborhood search methods are considered for approximating the MSKP. Such a problem has an enormous variety of applications, especially in engineering design, health care, in energy planning (that's including renewable energy planning, energy, resource allocation, building energy management, transportation energy management, planning for energy projects and electric utility planning) and financial management. For example, suppose a real-life problem appears in the capital budgeting when the possible returns of the investment i depend on which out of m different scenarios is realized in the future, while the fixed budget c. This investment should be made when the maximizer of the lowest return under all scenarios within the given budget.

In this work, we present sequential algorithms based on neighborhood search techniques. These approaches are tailored for optimizing large-scale instances of a combinatorial optimization problem called MSKP.

This work is composed of two parts. The first part presents a state of the art of the problem treated with suitable methods of resolution. As, the second part, the approximate methods for solving the MSKP will describe.

More specifically, the thesis is organized as follows: the first part (Chapter 2) presents the introduction for some problems belonging to the knapsack family. It starts with describing the classical knapsack problem. Also, the solution methods exact and approximate ones described, that are used for solving the problem. Finally, we defined the problem MSKP with main solution methods that are used in the literature.

The second part consists of three chapters (chapter 3 to chapter 5). In chapter 3, we introduce a heuristic for MSKP. It's a goal to present a hybrid reactive search algorithm to solve the MSKP. The algorithm consists of two main phases embedded into an iterative search: the restoring/exploring phase and the perturbation phase. The first phase yields a

feasible solution and tries to improve it by using an exploring strategy that could be shown as a neighborhood search that is based on a series of exchange between items belonging or not to the current solution. The second phase can be viewed as a local search that combines two strategies for building a partial solution by removing some items of the current solution, whereas the re-optimization strategy tries to select new items and including them into a degraded solution for providing a new diversified solution. The destroying strategy also tries to diversify the search process by degrading the quality of the solution with the goal of avoiding stagnation in a local optimum.

In chapter 4, we present the two-stage hybrid method in order to solve approximately the MSKP. The proposed method is based on three complementary stages: (i) the building stage, (ii) the combination stage and (iii) the two-stage rebuild stage. First, the building stage serves to provide a starting feasible solution by using a greedy procedure; each item is randomly chosen for reaching a starting population of solutions. Second, the combination stage tries to provide each new solution by combining subsets of (starting) solutions. Third, the rebuilding stage tries to make an intensification to improve the solutions at hand.

Chapter 5 presents a hybrid heuristic that combines Greedy Randomized Adaptive Search Procedure (GRASP) and Path-relinking for tackling the MSKP. The proposed heuristic combines two principle futures: (i) the building phase and (ii) the improvement phase. The first phase provides a (starting) feasible solution by applying GRASP. On the other hand, each current (starting) solution is improved by using a strategy based on path-relinking.

Finally, Chapter 6, gives a general conclusion in which we summarize the methods proposed.

Chapter 2

Some types of knapsack problem

Contents

In this chapter, we present the state of the art of some combinatorial optimization problems belonging to the knapsack family. We start by defining and describing the standard binary knapsack problem. Then, we illustrate the exact and approximate methods available in the literature that have treated the binary knapsack problem. In addition, we present a problem related to the knapsack problems family: the max-min knapsack problem with multi-scenarios (MSKP). For this problem, the corresponding solution methods (taken from literature) are discussed.

2.1 Introduction

The knapsack problem (noted KPs) is a combinatorial optimization problem. It is classified as NP-hard problems. The first study appeared in 1897 (cf., Mathews [56]), but Dantzig [18] used the expression in his early work and thus the name of this problem could be a kind of folklore. There are several important applications of the knapsack problem in practical and theoretical domains. In the practical field, there are many applications, more especially in different domains like industry, transport logistics, financial management (such as for budget control (cf., Lorie *et al.* [51]), project selection (cf., Petersen [59]), the cutting stock problem (cf., Gilomore *et al.* [29], [30], [31]), and loading problem (cf., Shih [68], Bellmam [12])). In addition to practical applications, the theoretical models are very important since many problems belonging to the knapsack family appear often as a reduced or sub-problem in the solution procedures of more complex problems (cf., Hifi *et al.* [39], Pisinger [61]).

In the literature, there are several variants of problems of knapsack type (cf., Figure 2.1). Among of them, in what follows, we cite some problems, including the 0-1 knapsack problem

and the max-min knapsack problem with multi-scenarios (the problem studied in this thesis). However, this family has been considered widely and cannot be fully covered here.

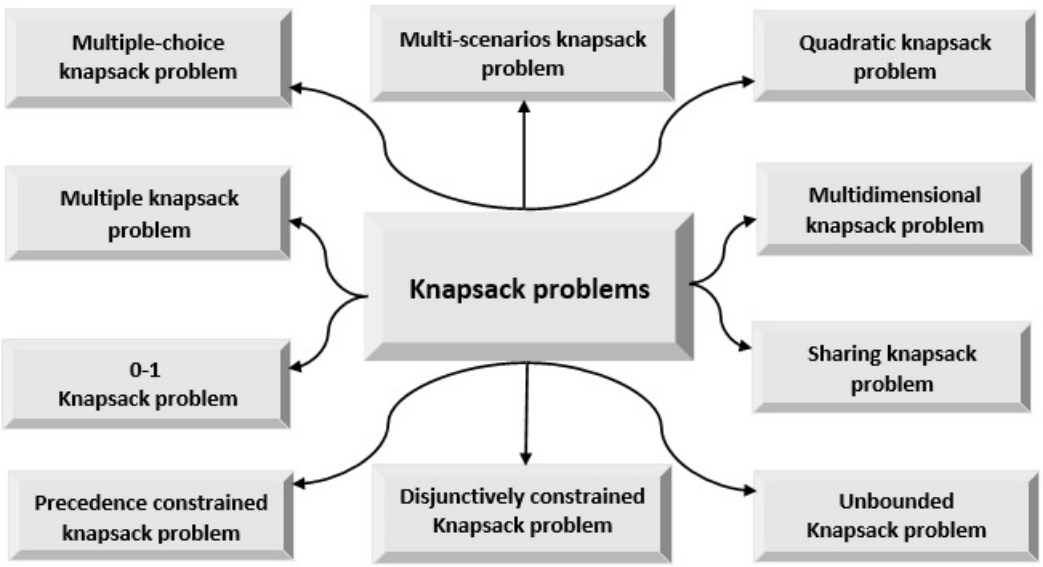

Figure 2.1: Some variants of knapsack problems

2.2 The binary knapsack problem

The classical binary knapsack problem (denoted by 0-1 KP) is one of the most linear integer programming belonging to the combinatorial optimization family (cf., Martello and Toth [54], Pisinger *et al.* [64]).

There are several applications that can be modeled as 0-1 KPs, such as cargo loading and computer science (cf., Horowitz *et al.* [42]). Let us consider a simple example: suppose a traveller has a travelling bag (knapsack) that takes a maximum of c kg of items. The traveller has n items $(1, 2, 3, \ldots, n)$, their weights are w_i and they have values to him of p_i. In this case, he should place the items so as not to exceed the maximum weight of the bag, c. Another example is the practice of selecting commercials to be played on air from a set of commercials on most TV and radio stations to generate the maximum revenue given a fixed time.

2.2.1 Definition and mathematical formulation

The general definition of the classical binary knapsack problem is that it consists of a set I of n items, where each item $i \in I = \{1, \ldots, n\}$, is characterized by a profit p_i and a weight w_i. There are also given a fixed of knapsack capacity c. The goal is to maximize the total profit of the items to be placed in the knapsack without violating the knapsack's capacity c. We consider the following example (cf., Figure 2.2):

Consider a set of four items such that $N = \{1, 2, 3, 4\}$ in a store. Each item is characterized by weight $w_i = \{w_1, w_2, w_3, w_4\} = \{$ 1 kg, 3 kg, 4 kg, 5 kg$\}$, profits $p_i = \{$ 1000 €, 2000

Figure 2.2: Example of knapsack problem

€, 4000 €, 4500 €} and a thief can carry up to 7 kg in his knapsack. What should he take to maximize the value of his haul?

The thief may take the items 1 and 2, with a value equal to 3000 €. Another possibility is to take items 1 and 4, with a value equal to 5000 €. Thus, the feasible solutions are those that are all subsets of items in which their total weight does not exceed the capacity of the thief's knapsack. One can observe that the number of possible solutions increased with increasing number of items (N). In such a case, the thief is to take the items of maximum profit among all feasible solutions (i.e., the optimal solution). Regarding this example, an optimal solution is that containing items 2 and 3 with the value of 6000 € and with a total weight of 7 kg.

The mathematical representation of the 0-1 KP can be stated as follows:

$$\max \quad \sum_{i=1}^{n} p_i x_i \tag{2.1}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_i \leq c \tag{2.2}$$

$$x_i \in \{0,1\}, \ \forall \ i \in I = \{1,...,n\} \tag{2.3}$$

where x_i is a binary decision variable such that $x_i = 1$ when item i is included in the knapsack and zero otherwise. In the mathematical representation, there are three equations. The first equation (2.1) is the objective function where the aim of the problem is to maximize the value of the profit of the items that placed in the knapsack and two inequalities (Inequalities 2.2 and 2.3). The first inequality (2.2) is known as the knapsack constraint, which imposes that

the sum of the weights of the selected items must not violate the knapsack's capacity. The second inequality (2.3) imposed on items that are to be included or not in the knapsack. In order to avoid trivial cases, it is assumed that:

- $c, \forall i \in I : p_i, w_i$ are positive integers.
- $\forall i \in I,\ w_i \leq c$ (i.e., no item has a weight greater than the knapsack's capacity).
- $\sum_{i=1}^{n} w_i > c$.

A vector $(\hat{x_1}, \ldots, \hat{x_n})$ of n binary decision variables is considered a feasible solution if the overall sum of the weights of the decision variables does not exceed the knapsack's capacity c, i.e., it satisfies the following knapsack constraint:

$$\sum_{i \in I} w_i \hat{x}_i \leq c$$

Also, a feasible solution is an optimal solution if it achieves the best (greatest) value among the set of all possible solutions; it is denoted by $\overline{x}$. In other words, for all feasible solutions $\hat{x}$, we have:

$$\sum_{i \in I} p_i \overline{x}_i \geq \sum_{i \in I} p_i \hat{x}_i$$

Although this is a simple formulation, its solution may require a significant runtime. In fact, the 0-1 KP is considered one of the first NP-hard problems to solve, as shown in Karp [44]. There are many methods in the literature to solve it. Among these methods, exact and approximate methods have been proposed for achieving either optimal or approximate solution. Dynamic programming, hybrid approaches and branch and bound are considered efficient methods to solve the 0-1 KP (cf., Horowitz and Sahni [42], Martello and Toth [54]). These methods are based on depth-first search strategies for efficiently exploring the search space.

To improve some of the solution procedures, Balas and Zemel [10] proposed the so-called core problem, that is considered an effective tool for solving large-scale problems. Later, the most successful methods proposed to solve the KP were inspired by the concept (cf., Martello and Toth [54], Fayard and Plateau [23] and Pisinger [62]). Martello *et al.* [53] suggested a hybrid approach combining dynamic programming with powerful bounds. Some researchers have mentioned to other types of KP not limited to only one constraint. Such a problem is said a multi-dimensional (or multiple) knapsack problem, which has been well-reviewed in Pisinger [62] and Chu and Beasley [16]. The bi-dimensional knapsack problem is considered the special case of the multiple knapsack problem where the number of constraints is two (cf., Freville and Plateau [26]).

2.2.2 Relaxation and bounds computing

The linear programming relaxation of the knapsack problem (LKP) is obtained by allowing the variables to take real values (usually positive) instead of only integer values in the optimal solution. Indeed, the optimal solution value (for a maximization problem) of the relaxed

problem is greater than the solution value of the binary problem (because the set of feasible solutions for a 0-1 KP is a subset of the feasible solutions for the relaxed problem).

The linear programming relaxation of the knapsack problem can be stated as follows:

$$LKP \quad \begin{cases} \max & \sum_{i=1}^{n} p_i x_i \\ s.t. & \sum_{i=1}^{n} w_i x_i \leq c \\ & 0 \leq x_i \leq 1 \qquad i = 1, ..., n \end{cases}$$

Suppose that the items $i, i \in I$, are ordered in decreasing order of their profit per weight, that is,

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

For the knapsack problem, the first upper bound was proposed by Dantzig [18]. Suppose that the items are consecutively put into the knapsack until filled, and determine the first item x_s that does not get inserted into the knapsack. This item is called the critical item (break item):

$$s = \min \left\{ j : \sum_{i=1}^{j} w_i > c \right\}$$

Then, the optimal solution vector $\overline{x}_i$ of the LKP can be represented as follows:

$$\overline{x}_i = \begin{cases} 1 & if\ i \in \{1, ..., s-1\} \\ 0 & if\ i \in \{s+1, ..., n\} \\ \frac{1}{w_s}(c - \sum_{i=1}^{s-1} w_i) & if\ i = s \end{cases}$$

The optimal corresponding solution value of LKP can be represented according to the following equation:

$$Z^{LKP} = \sum_{i=1}^{s-1} p_i + (c - \sum_{i=1}^{s-1} w_i) \frac{p_s}{w_s}$$

The upper bound for the problem 0-1 KP (denoted UB) is an optimal solution of the LKP: because all variables of 0-1 KP are integer, the optimal solution of the 0-1 KP is always smaller than the optimal solution of the LKP. This upper bound is called the Danzig upper bound (cf., Danzing [18]):

$$Z^{(0-1)KP} \leq UB = \lfloor Z^{LKP} \rfloor = \sum_{i=1}^{s-1} p_i + \lfloor \frac{(c - \sum_{i=1}^{s-1} w_i)}{w_l} p_l \rfloor$$

where $\lfloor x \rfloor$ denotes the larger integer less than x.

2.3 The binary knapsack problem and solution methods

In optimization, the methods to solve NP-hard problems are divided into two main classes: the exact and the approximate methods. In this section, we recall the exact and approximate solution methods to solve the 0-1 KP. The exact methods can have the ability to find an

optimal solution with huge computational efforts. They can be only successful for solving small sized problems. The obstacle is that the execution time grows exponentially with the size of the problem. The approximate methods give a solution with good quality that may be suboptimal or optimal. However, these methods can find a solution with an acceptably low computational time.

2.3.1 Approximate methods

The knapsack problem is an NP-hard problem. That means it needs a long runtime to solve. Therefore, this leads to the use of approximate solution methods, in particular for the large instances. Approximate solution methods are characterized by two important properties: first, it finds a feasible solution for any problem within a low runtime. The second aspect is that the quality of the approximation algorithm used is the maximum between its solutions and the optimal solutions. Among these methods, we mention a simple method to determine a feasible solution of the KP immediately, which is a greedy method.

The greedy method is considered as one of the approximate methods. It is based on the principle of being greedy (cf., Sanjoy *et al.* [66]). It is based on building a solution piece by piece, always choosing the next piece that offers the most evident and immediate benefit. Usually, the greedy strategy does not always lead to an optimal global solution, but it is very powerful and works properly for the studied problem.

The simple greedy method is based on the principles of Dantzig [18] for tackling the 0-1 KP. The idea of the greedy strategy sort the variables according to the ratio of the profit to the weight in decreasing order, such that:

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

The first item placed in the knapsack has the highest ratio, then, the other items are chosen in the same manner until the capacity of the knapsack is about to exceed. The greedy algorithm is described in Algorithm 4.

Algorithm 4 A feasible starting solution.

Input: An instance of 0-1 KP.
Output: A feasible solution $\overline{x}$.

Set $\overline{c}$ as the available knapsack capacity, $\overline{c} \leftarrow c$;
Set $x_i = 0,\ \forall i \in \{1, \ldots, n\}$;
for $i = 1$ to n **do**
  if $w_i \leq \overline{c}$ **then**
    $x_i = 1$
    $\overline{c} = \overline{c} - w_i$;
  end if
  $i = i + 1$
  Until $\overline{c} = 0$ or $i = n$;
end for
return $\overline{x}$

2.3.2 Exact methods

Several exact methods for the 0-1 KP have been proposed in the literature. In this section, we will mention some of these methods, such as the cutting plane, dynamic programming, branch and bound, and branch and cut.

2.3.2.1 The cutting plane method

The cutting plane method was first suggested by Gomory [34] as a method for solving integer programming and mixed-integer programming problems. It is based on the cutting plane approach, where the main idea is based on adding constraints (cf., Figure 2.3) in order to reduce the search space of the feasible solutions, and repeat this procedure until the best integer solution is obtained. Balas *et al.* [9] presented an effective version of this method for solving mixed 0-1 programs by using the lift-and-project cuts.

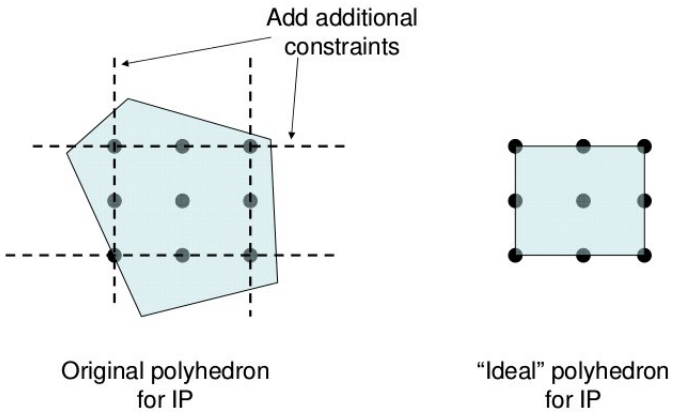

Figure 2.3: Example of adding constraints

Algorithm 5 Gomory's cutting plane algorithm.

Input: An instance for 0-1 KP.

Output: An optimal solution $x^\star$.

1: Set a relaxation program for 0-1 KP (noted LP 0-1 KP);
2: Set $x^\star = 0$
3: **repeat**
4: Solve the current LP(0-1 KP), and let $\hat{x}$ be a basic optimal solution;
5: **if** $\hat{x}$ is integer; **then**
6: $x^\star = \hat{x}$;
7: Stop;
8: **else**
9: Generate a new cut around the point $\hat{x}$ (as a new constraint) and add it to the LP;
10: **end if**
11: **until** $x^\star$ is optimum integer solution;
12: **return** $x^\star$ the optimal solution.

Algorithm 5 illustrates the cutting plane algorithm to solve the 0-1 KP. In this algorithm, three main steps are repeated: first, solve the linear programming relaxation (by using the simplex method) and obtain a basic optimal solution $\hat{x}$. On the one hand, if $\hat{x}$ is an integer solution, then the optimal solution is obtained (denoted $x^\star$). On the other hand, the last step aims to complete an algorithm by generating a new cut (as a new inequality constraint) and adding it to LP and return to the first step. The algorithm stops once the optimal integer solution $x^\star$ is obtained.

2.3.2.2 Dynamic programming method

The dynamic programming method is considered a useful tool for solving certain combinatorial optimization problems. Bellmam [12] has given a first inclusive introduction to this method. It can tackle problems that involve decision variables that interact with each other and have a time dependence. The basic idea of the method considers the optimal solution of a problem as a resulting from a combination of optimal solutions to the subproblems. Toth [73] developed the dynamic programming algorithms for 0-1 KP because the initial procedures for the computation of knapsack function are presented and used of bounds to eliminate states not leading to the optimal solution when analyzed. Hifi [36] proposed an algorithm which combines the dynamic programming techniques with a depth-first search using hill-climbing strategies for solving the constrained two-dimensional cutting stock problem.

2.3.2.3 Branch-and-bound method

Branch-and-Bound (BB) is a general method to solve different types of optimization problems in order to find an exact optimal solution. The first BB has been suggested by Land *et al.* [49]. Kolesar [46] proposed the first BB to solve the 0-1 KP. Subsequently, the BB algorithm has been developed by Horowitz *et al.* [42].

The general concept of BB methods is based on a tree structure of the solutions. Each node of the tree divides the search space into two sub-spaces until the full exploration of the solution space (cf., Dasgupta *et al.* [20]). The number of feasible solutions 2^n can be derived from n binary variables, thus, the search space becomes very large (cf., Kellerer *et al.* [45]). The BB methods are based on three main principles. They are:

- The strategy of branching.
- The strategy of bounding.
- The strategy of node choosing.

The basic aim of the BB method is to divide the solution space into smaller subspaces that can be solved independently (branching) in order to arrive at the optimal solution. The process is in principle repeated until each subspace contains only a single feasible solution. The division is called branching as new branches are created in the enumeration tree. The branching members can be taken in a predefined order, or as an element-defined exploring. Sometimes the choice of the branching element is important to improve the performance of the algorithm. The strategy of bounding is an important principle because it verifies whether a subspace contains an optimal solution; at the same time, it can prevent the search tree from

growing too much. It works by comparing and computed the solution of the subspace with the best solution found. If it achieves, this means the subspace contains the optimal solution, and if not, this subspace is ignored. The node choosing strategy determines the choice of the next node in the search tree for branching. Additionally, there are other strategies: the most common include depth-first, which determines the node that gives the best value of the bounding function of the search tree in anywhere.

We now present the algorithm that was suggested by Horowitz *et al.* [42] (Algorithm 6).

Algorithm 6 Branch and bound for KP

Input: An instance of 0-1 KP.
Output: An optimal solution x^*.

1: Let $i = 1$; $BestValue = 0$.
2: Compute the upper bound UB that can be reached for i
3: **if** $UB \geq BestValue$ **then**
4: Develop a branch of the tree (in depth-first) to obtain a feasible solution x'
5: **if** $Z(x') \geq BestValue$ **then**
6: $x^\star = x'$;
7: $BestValue = Z(x')$
8: **end if**
9: **end if**
10: $i = max\{s \leq i : x'_s = 1\}$
11: **while** i exist **do**
12: $x'_i = 0$
13: **end while**
14: $x^* = x'$.

This algorithm considers that all items are sorted in decreasing profit per weight ratio (cf., Section 2.3.1). Then, the separation of the elements is as follows: at each node, we choose the item i which has the maximum ratio (profit per weight). We develop two branches: the first branch, $x_i = 1$, corresponds to the item i that is put in the knapsack, and the other, $x_i = 0$, corresponds to the item i that is not packed in the knapsack. We continue the search until the knapsack is filled. This algorithm uses a depth-first strategy, which means the items put on the knapsack first have the best profit per weight ratio. The function of evaluation uses the Danzig bounds presented in Section 2.3.1. Computing an upper bound is very important to find the current solution and compare it with the earlier solution.

2.3.2.4 Branch-and-cut method

The branch-and-cut method is considered very important for solving several combinatorial optimization problems. It is a combination of the cutting plane method and branch-and-bound. It is not in general efficient to solve the general integer programming using only the cutting plane method. Therefore, it is necessary to apply the branch-and-bound method (cf., Crowder *et al.* [17], John [58]). The applying of the cutting planes leads to a greater reduction in the size of the search tree than the pure branch-and-bound approach. Therefore, the pure branch-and-bound approach can be accelerated by the employment of the cutting

plane scheme.

John [58] explained that the branch-and-cut method can guarantee to find an optimal solution for these problems. The researcher explained that the area of branch-and-cut algorithms is constantly evolving, and has the possibility of becoming more important with the rapid development of computers and parallel computing. The best-known use of the branch-and-cut algorithms is to solve the travelling salesman problem (TSP). For large and/or hard problems, branch-and-cut can be used in conjunction with heuristics or meta-heuristic methods in order to reach a solution near optimality.

2.4 The multi-scenarios max-min knapsack problem (MSKP)

The Multi-Scenarios max-min Knapsack Problem (abbreviated MSKP) is a variant of the well-known single knapsack problem with a multi-objective function. The MSKP is an NP-hard combinatorial optimization problem. Previously, this problem has considered under a multi-criteria decision making framework (cf., Gass *et al.* [28], Steuer [70]). The MSKP was first proposed by Yu [79] in an application of robust optimization to a max-min knapsack problem. There are several practical applications of MSKP in water distribution networks, engineering designs and in financial management. Let us mention an example, in the field of engineering designs, more precisely, in designing a car involving two criteria, cost and reliability. The customer would like to have a car with the maximum reliability and a minimal cost, but a higher level of reliability usually results in a higher cost (cf., Wiecek *et al.* [78]).

2.4.1 Definition and mathematical model

Formally, an instance of the MSKP is characterized by a knapsack of fixed capacity c and a set I of n items. Each item i is characterized by a weight w_i and a set of profits p_i^s, where the profits vary for each possible scenario s (cf., Yu [79]). In other words, a scenario is defined by the set of profits that apply to each item (cf., Iida [43]). The classical knapsack problem is a special case of the MSKP in which s is equal to 1; i.e. there is only one scenario (cf., Kellerer *et al.* [45], Martello and Toth [54], Garey *et al.* [27]). The goal of the MSKP is to determine the subset of items whose total weight does not exceed the knapsack capacity, and whose total profit is maximized in the worst scenario over all the possible scenarios (cf., Pinto *et al.* [60]).

Let x_i be the decision variable associated to the item i such that x_i takes the value 1 if the item i is selected, otherwise x_i takes the value 0. The knapsack constraint is represented by:

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

The objective function of the problem is defined as a formula of the max-min type, by introducing s profit values instead of one for every item, and the goal is to select the subset of items which respects the capacity constraint c and maximizes the minimal value of a set of linear functions. Then, the integer linear programming problem of the MSKP can be stated as follows:

$$\max \quad \min_{s=1,\dots,S} \Big\{ \sum_{i=1}^{n} p_i^s x_i \Big\} \tag{2.4}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_i \leq c \tag{2.5}$$

$$x_i \in \{0,1\}, \qquad \forall\, i \in I \tag{2.6}$$

In this integer linear programming problem, we have three equations. The equation (2.4) is the objective function, where the goal is to maximize the total profit of the items under the worst scenario over all the possible scenarios. The inequality (2.5) is the capacity constraint, which imposes the condition that the sum of the weights of the selected items does not exceed the knapsack's capacity. The inequality (2.6) is imposed so that items are either placed or not in the knapsack. In order to avoid trivial cases, it is assumed that:

(i) all input data $c, p_i, w_i, \forall i \in I$ are strictly positive integers.

(ii) $\sum_{i \in I} w_i > c$ to avoid trivial solutions.

2.4.2 Solution methods

In this section, we mention some solution methods to tackle the multi-scenarios max-min knapsack problem.

2.4.2.1 Approximate methods

Hifi *et al.* [37] proposed an adaptive algorithm for solving MSKP. Their approach combined two phases: The first phase applies a polynomial reduction strategy to reduce the size of the problem. The second phase applies an adaptive search procedure based on a dynamic 2-opt procedure to achieve a feasible solution. The authors used the greedy algorithm in order to build a better feasible solution. Indeed, the reduction strategy applied in their algorithm is a pegging test which is very useful to reduce the problem size. Lastly, the 2-opt strategy is applied to improve an initial solution and obtain a high-quality solution for the problem.

Song *et al.* [69] have been tackled larger instances up to $n = 10000$ and $s = 100$, by suggesting a collection of incomplete m-exchange algorithms as an approximate algorithm to obtain a solution to the MSKP. They explained that the optimal solution cannot be found by using a branch-and-bound algorithm (that is, an exact algorithm) through a reasonable time for some large-scale instances. In their algorithms, firstly, they present, in details, a subgradient procedure to obtain an initial feasible solution and upper and lower bounds. Then, they propose swapping the values of some of the binary variables of the problem, thus obtaining a high-quality solution.

2.4.2.2 Exact methods

Some papers in the literature tackle the MSKP with exact solution methods. Kouvelis and Yu [47], as well as Yu [79] and Iida [43] have solved the MSKP by using a surrogate relaxation based branch and bound algorithm for instances with numbers of items up to 90 and numbers of scenarios up to 30. Then, Taniguchi *et al.* [71] suggested an algorithm that first uses a

kind of surrogate relaxation to find upper and lower bounds quickly, and then applies a pegging test to reduce the size of the problem. Subsequently solving the reduced problem by a branch and bound algorithm. Their algorithm was carried out for instances when $n \leq 1000$ and $s \leq 30$.

Taniguchi *et al.* [72] developed that work in [71] by taking a special case of the MSKP with just two scenarios. The authors proposed an approximate and an exact algorithm to solve this problem. They used a surrogate relaxation in order to derive upper and lower bound very quickly, and finally, in order to solve the reduced problem, they used this relaxation to obtain the upper bound in the branch and bound. They explained that their proposed algorithm remains very efficient for problems with small size, but it is less efficient for strongly correlated ones.

Hanafi *et al.* [35] proposed a hybrid method as an exact method to solve the MSKP when the number of scenarios is two. The authors suggested combining in their approach heuristics and the temporary setting of variables in order to reduce the gap between the upper and lower bounds. They suggested several formulations of the MSKP equivalent to a mixed integer linear programming problem, and by adding pseudo-cuts into a problem, they tried to reach the optimal solutions. In more detail, their algorithm consists of three main steps: (i) to construct one or more pseudo-cuts, and update the best upper bound of the problem, they must resolve one or more relaxations of the current problem; (ii) to obtain one or more feasible solutions of the initial problem, and update the best lower bound, they must resolve one or more of the reduced problems produced by the optimal solutions of the previous relaxations; (iii) stop the algorithm when the stopping criterion is satisfied, and the best upper and lower bounds are reached.

Lastly, Pinto *et al.* [60] suggested an exact solution method that is based on column generation and branch and bound. The authors have explained that their method depends on a reformulation of the problem model based on the Dantzig-Wolfe decomposition principle, which provides stronger upper bounds. Hence, the corresponding subproblems can be used at the same time to price out attractive columns and to obtain a feasible solution for the MSKP that may improve the current solution.

2.4.2.3 The column-generation algorithm

The column generation algorithm is an efficient method to solve linear programming problems containing a huge number of variables, something which prevents the application of the Simplex method to the problem as a whole. Gilmore and Gomory [30] were the first to propose the column generation method, for the well-known cutting stock problem, which is a decomposition technique for solving a structured linear program (LP) with a few rows and lots of columns. Vanderbeck [76] presented an attempt to design an efficient exact algorithm based on column generation for the cutting stock problem.

Vance *et al.* [75] proposed solving the binary cutting stock problem by using column generation and branch-and-bound to obtain optimal integer solutions. They suggested formulating a branching rule that can be incorporated into the subproblem to allow column generation at any node in the branch-and-bound tree.

Mehrotra and Trick [57] used column generation to solve the graph coloring problem, and they explained that this method is very efficient at solving large problem instances.

Bjorklund *et al.* [13] proposed this method for solving the special multiple access scheduling problems.

Hifi [15] used column generation to solve the large-scale multiple-choice multi-dimensional knapsack problem by proposing a method which is composed of two complementary stages: a rounding solution stage and a restricted exact solution procedure. Applying the Dantzig-Wolfe decomposition principle to a linear integer problem leads to a reformulation into a master problem and one or more subproblems defined from the constraints of the original formulation.

Pinto *et al.* [60] developed an exact method for the MSKP. Their approach is based on combining both column generation and branch-and-bound procedures. The resulting branch-and-price algorithm proved able, in different ways, to outperform the other state of the art methods available in the literature. The column generation reformulation provided stronger upper bounds, and the corresponding subproblems could at the same time be used to price out attractive columns and to generate feasible (near-optimal) solutions. Such an approach depends on the reformulation of the standard compact integer programming model based on Dantzig-Wolfe's decomposition principle.

In general, The main idea of column generation is to use variables with reduced size that may be active in the optimal solution while ignoring other variables, which are fixed to zero. The reduced variables that they are used can potentially improve the objective function. This method is based upon decomposing the original linear programming (LP) into a master problem and a subproblem. The master problem contains a first subset of the columns and the subproblem, which is a separate problem for the dual LP, such that it is solved for the purpose of identifying whether the master problem should be expanded with additional columns or not. The column generation method alternates between the master problem and the subproblem until the former contains all the columns that are necessary to obtain an optimal solution for the original (LP). Figure 2.4 illustrates the column generation method. A special technique in linear programming which uses this type of method is the Dantzig–Wolfe decomposition algorithm (cf., Dantzig and Wolfe [19]).

Figure 2.4: The column generation method

2.4.2.4 The branch-and-price algorithm

Branch-and-price is a method of combinatorial optimization for solving integer linear programming (ILP) and mixed integer linear programming (MILP) problems with many variables. There are many successful applications of branch-and-price in the industry (cf., Lübbecke and Desrosiers [52]), also to general combinatorial optimization problems like bin packing and the cutting stock problems (cf., Vanderbeck [76]) and machine scheduling (cf., Van *et al.* [74]).

This method combines a branch-and-bound method with the column generation method. Barnhart *et al.* [11] explained that the method of branch-and-price is similar to that of branch-and-cut except that the procedure focuses on column generation rather than row generation. Indeed, pricing and cutting are complementary procedures for tightening an LP relaxation. In the branch-and-price algorithm, the columns are excepted from the LP relaxation because there are too many columns to handle efficiently and most of them will have their associated variable equal to zero in an optimal solution in any case. Then to test the optimal solution of an LP, the subproblem called the pricing problem, which is a separation problem for the dual LP, is solved so as to attempt to identify columns to enter the basis. If these columns are found the LP is re-optimized. Branching happens when no columns price out to enter the basis and the LP solution does not satisfy the integrality conditions of branch-and-price (cf., Barnhart *et al.* [11], Lübbecke and Desrosiers [52], Desrosiers and Lübbecke [21]).

2.5 Conclusion

This chapter introduced some variants belonging to the knapsack family, which is known as being problems of combinatorial optimization. To solve these problems, there are two directions of research: exact and approximate methods. An exact algorithm tries to find an optimal or a set of optimal solutions for a given problem, whereas a heuristic algorithm is an alternative procedure that permits you to solve large-scale instances by providing better solutions, not necessarily the optimal ones but in an acceptable solution time.

In the following chapter, we present our first sequential solution method suggested to solve the multi-scenarios max-min knapsack problem.

Chapter 3

A hybrid reactive search for solving the max-min knapsack problem with multi-scenarios

Contents

Several practical applications depend on different criteria or scenarios and belong to the combinatorial optimization family. For example, in engineering design of a car, two criteria may be needed to optimize its production: the profit that can bring a car as well as its reliability for the customer. In water distribution networks, designing contaminant warning system of water is necessary. In this case, the problem consists in determining the level of protection under the worst possible scenario on the sensors that should be placed.

In this chapter, we propose to solve MSKP by using a hybrid reactive search algorithm that uses two main features: the restoring/exploring phase and the perturbation phase. The first phase yields a feasible solution and tries to improve it by using an intensification search. The second phase can be viewed as a diversification search in which a series of subspaces are investigated in order to make a quick convergence to a global optimum. Finally, the proposed method is evaluated on a set of benchmark instances taken from the literature, whereby its obtained results are compared to those reached by recent methods available in the literature. The results show that the method is competitive and it is able to provide better solutions than those already published.

3.1 Introduction

Many practical problems depend on various criteria or scenarios, where the goal of each of them is to satisfy most of these criteria according to the recommended scenarios. In engineering designs, for which the design of a car is a central issue, the problem may depend on two main criteria: cost and reliability. The customer would like to have a car with maximum reliability with minimum cost, but a higher level of reliability usually results in a higher cost (cf., Wiecek *et al.* [78]). In network problems, such a phenomenon may be met in a different way. Indeed, in water distribution networks, designing the contamination warning system of water depends on the sensors that should be placed in order to maximize the level of protection (cf., Watson *et al.* [77]; Aissi *et al.* [4]; Carr *et al.* [14]). Likewise, in the game theory, two players try to optimize both gain and cost. Each one has different strategies, where the max-min value of the first player corresponds to maximizing the gain and the second one to minimizing the cost. It may be viewed as the largest value that the player can be sure to get without knowing the actions of the other player (cf., Maschler *et al.* [55]).

Most of these exposed problems can be formulated as a max-min model, where different constraints can be added and different scenarios can be considered. In this chapter, we investigate the use of an iterative random search-based algorithm for solving the so-called Multi-Scenario max-min Knapsack Problem (noted MSKP). Such a problem belongs to the family of NP-hard combinatorial optimization, where several variants of the problem have been tackled in the literature (cf., Aissi [3], Aissi *et al.* [4], Du *et al.* [22] and Pinto *et al.* [60]).

We recall that, the MSKP have characterized by a knapsack of fixed capacity c, a set I of n items, where each item $i \in I := \{1, \ldots, n\}$ is associated with a non-negative weight w_i and a set of profits p_i^s such that each profit varies for each possible scenario s such that $s \in S := \{1, \ldots, m\}$ (cf., Yu [79]). In other words, a scenario is defined through a set of profits that applies to each item (cf., Iida [43]). Note that the classical knapsack problem is a special case of the MSKP in which s is equal to one; i.e., only one scenario is considered (cf., Kellerer *et al.* [45]; Martello and Toth [54]). Finally, the goal of the MSKP is to determine a subset of items whose total weight does not exceed the knapsack capacity c, and whose total profit is maximized in the worst scenario over all the possible scenarios (cf., Pinto *et al.* [60]).

A formal description of MSKP follows:

$$\max \quad \min_{s=1,\ldots,S} \Big\{ \sum_{i=1}^{n} p_i^s x_i \Big\} \tag{3.1}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_i \leq c \tag{3.2}$$

$$x_i \in \{0,1\}, \qquad \forall\, i \in I \tag{3.3}$$

where x_i, $\forall i \in I$, is a binary decision variable that takes the value one if the i-th item is included in the knapsack (considered in the solution); zero otherwise. On the other hand, by setting p_i^s, we denote the value of item i under scenario s, $s \in S$. From the above formulation, inequality (3.1) represents the objective function, thus we have m objective

function, where the goal is to maximize the value of the total profit of the items placed in the knapsack in the worst scenario over all the possible scenarios. Inequality (3.2) denotes the knapsack constraint with capacity c, which imposes the condition that the sum of the weights of the selected items does not exceed the fixed knapsack capacity. Finally, Inequality (3.3) represents the integrality imposed on each decision variable x_i, $\forall i \in I$. Note that, in order to avoid trivial cases, it is assumed that:

- all input data c, p_i^s, w_i, $\forall i \in I$, are strictly positive integers.
- $\sum_{i \in I} w_i > c$; that is to avoid trivial solutions.

The chapter is organized as follows. Section 3.1 discusses the large neighborhood search method. Section 3.3 introduces a hybrid reactive search-based heuristic for approximately solving the MSKP. Section 3.4 evaluates the performance of the proposed method on a set of benchmark instances taken from the literature. Section 3.5 concludes the work presented.

3.2 Large neighborhood search

Neighborhood search algorithms (alternatively called local search algorithms) are a wide class to solve the various of the combinatorial optimization problems (cf., Lenstra [50]).

Let I be a set consists of n items, such that $I = \{1, 2, \ldots, n\}$. Let F represent the set of feasible solutions of I. The mapping $f : F \longrightarrow \Re$ is called the *objective function*. Suppose that, the combinatorial optimization problem is a maximization problem, that, we want to find a solution $x^\star$ such that:

$f(x^\star) \geq f(x)$, $\forall x \in F$

Each $x \in F$ has an associated subset $N(x) \subset F$, and the set $N(x)$ is called the *neighborhood* of the solution x. A solution $x^\star \in F$ is called to be locally optimal with respect to a neighborhood function N if $f(x^\star) \geq f(x)$ for all $x \in N(x^\star)$. The neighborhood $N(x)$ is usually called to be exponential if $N(x)$ grows exponentially in n as n increases. With these definitions, it is possible to define, generally, a neighborhood search algorithm.

In general, a basic neighborhood search algorithm consists of three steps:

1. The algorithm starts with an initial solution x.
2. The solution $x^\star$ computes as $x^\star = argmax_{x \in N(x)}\{f(x)\}$, (i.e, finds the best solution $x^\star$ in the neighborhood of x).
3. Update the solution $x = x^\star$, if $f(x^\star) > f(x)$.

These steps are repeated until reaching a local optimum and/or performing the stopping criteria. After then, the algorithm stops when the best solution found (cf., Ahuja *et al.* [1], [2]). It means that a neighborhood search algorithm gradually improves a starting solution through exploring the neighborhoods of a current solution.

Large Neighborhood Search (LNS) is a heuristic that has proved to be effective for a wide range of combinatorial optimization problems. Shaw in [67] has introduced the simplest version of the large neighborhood search to solve vehicle routing problems. LNS is based on the concepts of building and exploring a neighborhood; that is, a neighborhood defined implicitly by a destroy and a repair procedure (cf., Pisinger *et al.* [63]), Figure 3.1 illustrates the LNS.

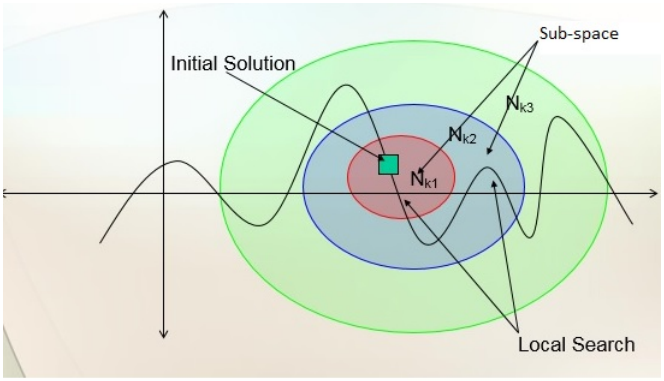

Figure 3.1: Large Neighborhood search

3.3 A hybrid search for the max-min knapsack problem with multi-scenarios

Neighborhood Search (NS) can be viewed as a framework based upon exploring solutions according to the exchange that is able to improve the solutions as a descent method. Variable Neighborhood Search (VNS) is an extended version of NS, where the descent method is coupled with a series of neighborhood changes in order to escape from local optima. Large Neighborhood Search (LNS) can be viewed as a diversification heuristic that is based upon reducing the quality of the solution at hand (in order to escape from a series of local optima) and then re-optimizing the problem by applying tailored intensification strategies (cf., Hifi *et al.* [38], Sanjoy *et al.* [63], Shaw [67]). Herein, we adapt a variety of Hifi and Michrafy's method (cf., Hifi *et al.* [38]); that can be viewed as a variant of LNS, already used for solving a series of combinatorial optimization belonging to the knapsack family (cf., Kellerer *et al.* [45]). The method consists of two complementary phases embedded in an iterative search; it combines both exploring and destroying/repairing phases.

In what follows, we develop a manner for simulating the process illustrated by the above complementary phases.

3.3.1 A (starting) solution for the max-min knapsack problem with multi-scenarios

We first describe the procedure that is able to build both a starting solution or a complementary solution to the partial one. Indeed, the proposed method can provide a starting feasible solution that is based on running m times a greedy procedure that is able to provide a quick starting feasible solution for the MSKP. Note that greedy methods are a class of heuristic solution methods that construct a solution piece by piece, focusing on an immediate improvement without considering the global consequence. Although this type of method generally does not produce an optimal solution, to our knowledge, it is one of the best known methods for providing feasible solutions in negligible runtime.

For the rest of the chapter, we assume that all items are sorted in decreasing order of their profits per weights, i.e.,

$$\frac{p_1^s}{w_1} \geq \frac{p_2^s}{w_2} \geq \cdots \geq \frac{p_n^s}{w_n},\ s = 1, ..., m$$

where s corresponds to the current scenario.

Algorithm 7 A Greedy Procedure for MSKP: (GP_s)

Input: an instance of P_{MSKP} and the current scenario $s, s \in \{1, ..., m\}$.
Output: A feasible solution S^s_{MSKP} associated with the s-th scenario.

1: Sort items corresponding to the s-th scenario in non-increasing order of their profit per weight;
2: Set $\overline{c} = c$, where $\overline{c}$ denotes the residual knapsack capacity;
3: Set $i = 1$
4: **repeat**
5: set $x_i = min\{1, \lfloor \overline{c}/w_i \rfloor\}$;
6: $\overline{c} = \overline{c} - w_i * x_i$;
7: Increment i
8: **until** ($\overline{c} = 0$ or $i = n$);
9: **return** S^s_{MSKP} the best solution corresponding to the s-th scenario.

Algorithm 7 describes the main steps of the greedy procedure (noted GP_s) when it is applied to a given scenario s, $1 \leq s \leq m$. In fact, GP_s uses an adaptation of Dantzig's procedure applied for, providing both lower and upper bounds for the single KP (cf., Dantzig [18]). Indeed, initially all items of each given scenario s are sorted in non-increasing order of the ratio *profit/weight* (as described above). The main loop (cf., lines from 4 to 8) is repeated until no item can be placed in the knapsack. More specifically, at each step, an item is selected according to the previously defined order. The selected item is placed in the knapsack if its weight does not exceed the residual knapsack capacity $\overline{c}$ after the selection of the other items. This process is repeated until no other items can be added in the knapsack. The algorithm stops and exits with a feasible solution S^s_{MSKP} associated with the s-th scenario for an instance of MSKP (P_{MSKP}). Of course, in order to determine the worst solution corresponding to all scenarios, one can apply GP_s, for $s = 1, \ldots, m$, and get the worst solution; that is considered as the starting feasible solution for P_{MSKP}. In this case, each s-th scenario's solution is noted by S^s_{MSKP} and the global solution containing all configurations (a global vector containing the structure of all feasible solutions corresponding to all scenarios s, $s \in \{1, \ldots, m\}$) is noted by S_{MSKP}.

3.3.2 The exploring strategy

The exploring phase can be viewed as a neighborhood search that is based on a series of exchanges between items belonging or not to the current solution. The neighborhood solution is used for improving the quality of the current solution or a series of built solutions. It also consists in finding a good strategy for generating a final solution x_p, $p \geq 1$, localized in the neighborhood solution. The process applies a series of moves for constructing a series of feasible solutions $x_1, x_2, \ldots, x_p$.

For an instance of MSKP, a standard binary representation scheme is an obvious choice since it represents the underlying 0-1 non-negative variables (Figure 3.2 shows a simple

1	1	1	1	...	1	0	0	...	0	0
1	2	3	4	...	$i^s_{c-}1$	i^s_c	i^s_c+1	...	$n-1$	n

Figure 3.2: Binary representation of MSKP's solution for the s-th scenario.

representation of MKSP's solution). Herein, a feasible solution $\bar{x}$, for each scenario, is such that $\sum w_j \bar{x}_j \leq c$. It is obtained by running one time GP_s (cf., Algorithm 7), which provides a solution with a *critical element* (noted i^s_c for each scenarios, $s \in S$, as described in Figure 3.2). All variables x_i such that $i < i^s_c$ are fixed to one and all other variables such that $i \geq i^s_c$ are fixed to zero.

Therefore, for each scenario $s \in S$, by moving i^s_c either to the left hand of i^s_c or its right hand, one can build a set of feasible solutions characterizing its associated neighbors. On the other hand, Figure 3.3 illustrates an eventual MSKP's partial feasible solution whenever some variables remain free in the current solution according to the current scenario s, $s \in S$.

1	1	1	1	...	0	0	$\star$	...	$\star$	$\star$
1	2	3	4	...	$i^s_{c-}1$	i^s_c	i^s_c+1	...	$n-1$	n

Figure 3.3: Configuration of MSKP's partial feasible solution associated to the s-th scenario. The symbol $\star$ denotes the free item

Let Δ be a nonnegative integer representing a *depth parameter*. The depth parameter allows us to jump backward from i^s_c to $i^s_c - 1$ and so on until the total depth Δ has been explored. It also allows the search to explore the entire left neighborhood and to attend to explore the entire right neighborhood. Such a strategy tries to make combinations of different solutions never visited before, and it tries to modify choice rules to encourage move combinations. Of course, one can also consider a series of critical elements that can be associated with the current solution; it means that an extended definition of the critical element can be used for more diversifying the search process (our limited computational results showed that the simple exploring strategy was sufficient in providing good behavior for the proposed algorithm, especially on benchmark instances available in the literature for the MSKP).

Hence, for each scenario s, $s \in S$, and with reference to a feasible solution, several manner can be explored for making a move regarding the critical item. In our study, we have followed the standard strategy already used in Hifi *et al.* [40] which works as follows:

1. Let $x_{i^s_c-1}$ be the $(i^s_c - 1)$-*th* item of the current solution such that $s \in \{1, \ldots, m\}$;
2. Set $x_{i^s_c-1}$ equal to zero, maintain $x_{i^s_c-2}, \ldots, x_1$ to one, and
3. Complete the partial configuration by applying GP_s (cf., Algorithm 7).

On the one hand, by jumping from $i^s_c - 1$ to $i^s_c - \Delta$, a set of new solutions is provided; that is called the *left neighborhood* of the solution at hand. A new search is then applied in order to improve the quality of the solution, i.e., the solution with the best value in the

neighborhood is selected. On the other hand, by applying the same process with a look-ahead, from $i_c^s + 1$ to $i_c^s + \Delta$ and by fixing the current item to one, we obtain the *right neighborhood*; that is provided by applying the following equivalent process:

1. Let $x_{i_c^s+1}$ be the $(i_c^s + 1)$-*th* item of the current solution such that $s \in \{1, \ldots, m\}$;
2. Set $x_{i_c^s+1}$ to one, make free some variable $x_i,\ i \leq i_c^s - 1$, and set $x_i,\ \forall i \geq i_c^s + 2$, to free;
3. Complete the partial configuration by applying GP_s (cf., Algorithm 7).

Hence, both *left* and *right* neighborhoods realize the current neighborhood of the solution at hand, namely S_{MSKP}. In fact, its mimics a truncated branch-and-bound where only a first level is considered; in this case, each successor of the current critical element has the following form: the current critical element is fixed either to zero or one and some of its successor, limited to the Δ-th element, is fixed either to 1 or 0, respectively. Of course, such a manner can be viewed as an extension of the "core problem" used for the single knapsack problem applied herein for the MSKP.

Algorithm 8 An Exploring Procedure for the MSKP: EP

Input. A feasible solution S_{MSKP} and a depth parameter Δ.
Output. An improved local optimum S'_{MSKP}.

1: **for** $s \in \{1, \ldots, m\}$ **do**
2: Set $\rho = 0$;
3: **while** $(\rho < \Delta)$ **do**
4: Increment ρ, let S^s_{MSKP} be the current solution, $I_1 = \{1, \ldots, i_c^s - \rho - 1\},\ I_2 = I \backslash \{I_1 \cup \{i_c^s - \rho\}\}$;
5: Set $x_{i_c^s-\rho} = 0,\ x_i = 1,\ i \in I_1$ and $x_i,\ \forall i \in I_2$, are free;
6: Call GP_s for completing S^s_{MSKP} and let S'_{MSKP} be the best solution found so far.
7: **end while**
8: Set $\rho = 0$;
9: **while** $(\rho < \Delta)$ **do**
10: Increment ρ, set $x_{i_c^s+\rho} = 1$ and $\forall i \in I' := I \setminus \{i_c^s\}$, set x_i to free;
11: Call GP_s for solving MSKP with I' and let S'_{MSKP} be the best solution found so far.
12: **end while**
13: **end for**
14: **return** S'_{MSKP};

According to the neighborhood defined above, Algorithm 8 describes the principle of the exploring search (also considered as an improved procedure) applied either to a *partial* or a *complete* feasible solution. As shown from Algorithm 8, it starts with a feasible solution (noted S^s_{MSKP}) reached by applying Algorithm 7 and by using a prefixed depth parameter Δ. The main loop (cf., lines from 1 to 13) describes the main steps of the exploring method: for each scenario $s \in S$, both internal loops (the first one from the line 3 to line 7 and the second one from (line 9 to line 12) the original MSKP's instance is reconsidered. Indeed, on the one hand, the first part (the internal loop from line 3 to line 7), the *left exploring phase* is applied in order to build the first part of the neighborhood. The current neighborhood contains the solutions with $x_{i_c^s-\rho}$ fixed to zero, where $\rho \in \{1, \ldots, \Delta\},\ \Delta \geq 1$. For each fixed ρ in the discrete interval $\{1, \ldots, \Delta\}$, a new solution is provided by calling the standard GP

on the remaining sub-instance with its residual capacity. On the other hand, the second loop, from (line 9 to line 12), represents the *right exploring phase* that is considered for completing the global neighborhood. In this case, $x_{i_c^s-\rho}$ is fixed to one, where $\rho \in \{1, \ldots, \Delta\}$, $\Delta \geq 1$; for each ρ belonging to $\{1, \ldots, \Delta\}$, a new solution is reached by applying GP to the reduced instance. Finally, Algorithm 8 (cf., line 14) stops with the best solution S'_{MSKP} found so far.

3.3.3 A reactive phase

The solution obtained from the exploring phase (cf., Algorithm 8) can be improved by applying several neighborhood procedures-based strategies. As used in (cf., Hifi *et al.* [38]), a reactive search can be viewed as a local search that combines two complementary strategies:(i) degrading strategy and (ii) re-optimization strategy. The degrading strategy tries to build a partial solution by removing some items of the current solution, whereas the re-optimization strategy tries to select new items and including them into the degraded solution for providing a new diversified solution. The destroyed strategy tries also to diversify the search process by degrading the quality of the solution with the aim of avoiding stagnating in a local optimum.

In what follows, we assume that a *degrading operator* is used, which can be viewed as a procedure that removes $\alpha\%$, $\alpha > 0$, of items belonging to the current solution. Let suppose that S^p_{MSKP} be the *partial solution* reached by applying the *degrading operator*. Then, in order to consider a stochastic search, one can randomly remove $\alpha\%$ of items from the solution at hand. Such a random destroying strategy may enlarge the chance of reaching a series of improved solutions or to escape from a series of local optima. After the removing stage, a reduced problem, namely P_{MSKP^r} is obtained and it is subjected to optimization. Finally, the second partial solution completes the first one and so, the provided solution is also submitted to an improvement stage.

More precisely, the reactive phase can be described as follows:

1. Let S'_{MSKP} be the current feasible solution and α be a predefined value.
2. Remove $\alpha\%$ of items from the current solution S'_{MSKP} and let S^p_{MSKP} be the partial feasible solution reached and P_{MSKP^r} be the new reduced subproblem containing all items removed from S'_{MSKP} and the items fixed to zero in S'_{MSKP}.
3. Solve P_{MSKP^r} by using a black-box solver and let S_{MSKP^r} be the (sub)optimal solution reached.
4. Combine both S^p_{MSKP} and S_{MSKP^r} to provide a starting solution and call Algorithm 8 for enhancing the quality of the solution.

3.3.4 An overview of the hybrid reactive search

Algorithm 9 summarizes the main steps of the proposed Hybrid Reactive Search (noted HRS). The input of HRS is a starting feasible solution S'_{MSKP} reached by applying Algorithm 8. Lines from 2 to 7 represent the main loop of HRS. Indeed, the destroying operator is first called (cf., line 3) in order to remove $\alpha\%$ of items from the current solution, namely S'_{MSKP}

(considered initially −line 1− as the best solution found so far, $S^{\star}_{MSKP}$). It then builds a reduced problem S_{MSKP^r}; that is, a subproblem composed of items already removed and those fixed to zero in the same current solution. In this case, a partial solution S^{p}_{MSKP} is reached; that is, a solution containing $(100-\alpha)\%$ of items fixed to one already fixed to one in the current solution S'_{MSKP}. In line 4, an optimization procedure, using the Cplex as a black-box solver, is applied in order to optimally solve the reduced problem P_{MKSP^r}. Of course, one can observe that a better diversification for HRS is to apply another strategy instead of re-applying the exploring phase which serves in intensifying the solutions.

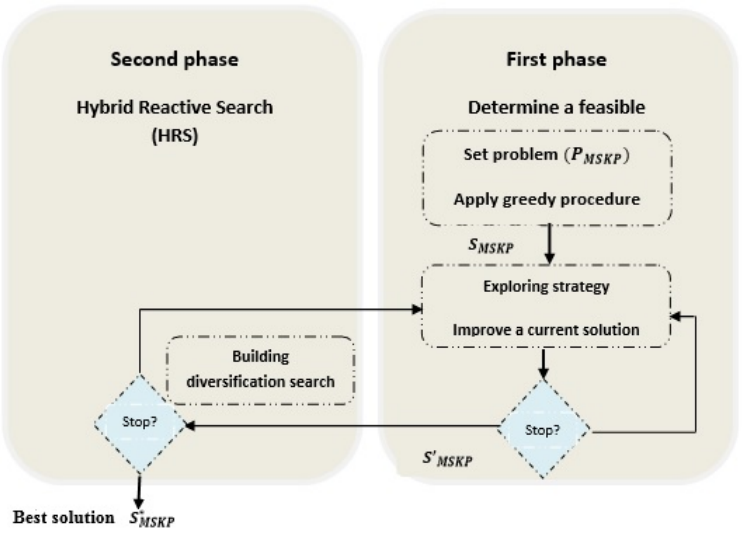

Figure 3.4: Hybrid Reactive Search HRS for MSKP

Algorithm 9 : A Hybrid Reactive Search: HRS

Input: S'_{MSKP}, a starting solution of P_{MSKP}.
Output: $S^{\star}_{MSKP}$, the best solution of P_{MSKP}

1: Affect S'_{MSKP} to $S^{\star}_{MSKP}$ //`the best solution found so far`
2: **while** (the time limit is not performed) **do**
3: Apply the destroying stage by removing $\alpha\%$ of items from the current solution:
- Let P_{MSKP^r} be the resulting reduced subproblem
- Set S^{p}_{MSKP} as the reached partial solution after the destroying stage

4: Optimize P_{MSKP^r} by using a black-box solver and complete S^{p}_{MSKP} with the obtained complementary solution S^{r}_{MSKP} for realizing a new solution S_{MSKP}
5: Apply the exploring phase (Algorithm 8) to S_{MSKP} and let S'_{MSKP} be the new solution reached.
6: Update $S^{\star}_{MSKP}$ with S'_{MSKP}, if necessary.
7: **end while**
8: **return** $S^{\star}_{MSKP}$.

Note that, on the one hand, the reduced problem P_{MKSP^r} is very difficult to solve, but

for small sized instances, the Cplex solver is generally able to solve them within the very short runtime. On the other hand, the resolution of P_{MKSP^r} depends on the value of α, as discussed in the experimental part (cf., Section 3.4). Then, a new solution S_{MSKP} is built by combining both S^p_{MKSP} and S^r_{MKSP}. In line 6, the best solution $S^\star_{MSKP}$ is updated with S'_{MSKP} whenever the objective value of the new solution dominates $S^\star_{MSKP}$'s objective value. In line 5, HRS tries to improve the current solution S_{MSKP} by getting a neighbor solution S'_{MSKP}; that is obtained by applying the exploring stage, i.e., by calling Algorithm 8. Finally, HRS returns $S^\star_{MSKP}$, the best solution found so far (cf. line 8).

Figure 3.4 illustrates the different phases used by algorithm HRS for approximately solving an instance of MSKP. This representation is used as a simplified representation of HRS that is already detailed in Algorithm 9.

3.4 Computational results

This section evaluates the effectiveness of the proposed Hybrid Reactive Search (abbreviated to HRS) on benchmark instances taken from the literature. These instances were extracted from (c., Pinto *et al.* [60]) and generated following the standard scheme used by (cf., Taniguchi *et al.* [72]).

This section is divided into two parts. The first part is focused on the performance of the proposed HRS on instances related to the max-min knapsack problem with two scenarios. The second part evaluates the effectiveness of HRS on more hardness instances containing at least three scenarios. Note also that HRS was coded in C++ and run on an Intel Pentium Core i7 with 2.1 GHz (which is generally equivalent to the computers' runtimes used in the last papers published in the literature).

3.4.1 Performance of HRS on the first group of instances

In order to evaluate the effectiveness of HRS, we considered a first set of instances (taken from (cf., Pinto *et al.* [60])), where each instance contains two scenarios. For this set, two types of instances are considered: weakly and strongly correlated ones. For each type of instances (weakly and strongly correlated), there are three groups represented according to the size of the capacity of the knapsack. The knapsack capacity c is setting equal to $\delta \times \sum_{i=1}^{n} w_i$, where δ is equal either to $0.25, 0.5$ or 0.75. For each pair of (I, δ), there are fifteen instances, where for the weakly (resp. strongly) correlated type, the cardinality of the set of items I varies in the discrete interval $\{5000, 7000, 10000, 13000, 15000, 18000, 20000\}$ (resp. $\{500, 1000, 4000, 5000, 7000, 10000\}$).

The results provided by HRS are compared to the best results available in the literature (cf., Pinto *et al.* [60]). In what follows, we first (Section 3.4.1.1) evaluate the behavior of the exploring phase used by the algorithm. Second and last (Section 3.4.1.2), we discuss the performance of HRS on the same set of instances when compared to the best available methods in the literature.

3.4.1.1 Effect of the exploring phase

In the preliminary results, the exploring phase (noted EP) is evaluated in order to show its effectiveness when it is used by HRS. We mainly study its effectiveness when used as an improved stage for the greedy procedure (noted GP). We recall that GP is repeated for each scenario and the EP is called for improving GPs' solutions and all internal (sub)solutions considered by HRS.

We then compare the quality of the solutions (representing the objective value) reached by both GP and EP. Indeed, Table 3.1 reports the results realized by GP and those reached by applying EP on both groups of instances: weakly correlated instances (the first four columns of Table 3.1) and strongly correlated instances (the last four columns of Table 3.1). In fact, columns 1 and 2 (respectively, columns 5 and 6) represent the instance information: n, the number of items, and δ, the value of the parameter used for calculating the capacity c of the instance's knapsack. Column 3 (respectively, column 7) tallies the average solutions obtained by GP when it is applied for all scenarios on the weakly (respectively, strongly) correlated instances whereas column 4 (respectively, column 8) displays the average solutions provided by the exploring procedure EP. Finally, the last line of the table summarizes the average solutions of all solutions reached by both GP and EP. Note also that both GP and EP provide the solutions within negligible runtime.

Weakly correlated instances				Strongly correlated instances			
n	δ	GP	EP	n	δ	GP	EP
5000	0.25	885508.4	892674.27	500	0.25	87411.20	87413.47
7000		1223063	1241955.6	1000		174528.13	174536.13
10000		1727676.8	1764285.33	4000		700045	700054
13000		2223105.07	2277617.53	5000		873853.33	874093.20
15000		2558235.8	2620578.27	7000		1223526.73	1224226.65
18000		3056033.13	3128239.93	10000		1630600	1650600
20000		3381609.33	3465992.47				
5000	0.5	1628112.47	1631387	500	0.5	154687.2	156587.2
7000		2274377.87	2281981.87	1000		318575	319036
10000		3232815.20	3252579.47	4000		1281366.33	1281562.53
13000		4186896.8	4219339.07	5000		1602499	1602799
15000		4820733.60	4860508.6	7000		2239220	2240480
18000		5769747.33	5823474.07	10000		3204793.33	3204952.33
20000		6411656	6470482.27				
5000	0.75	2333101.67	2334723.53	500	0.75	226566.33	227266.33
7000		3264633.20	3267177.87	1000		445162.47	449363.6
10000		4661729	4666322.33	4000		1835026.93	1837028.73
13000		6058825.6	6067003.67	5000		2165173.33	2254173.33
15000		6982283.93	6995199.73	7000		3125061.73	3226062.13
18000		8369450.93	8389560.53	10000		4605606.67	4607606.67
20000		9292438.4	9317258.4				
Average		4016287.31	4046111.51	Average		1439094.60	1451518.96

Table 3.1: Effect of the exploring phase EP

From Table 3.1, one can observe that EP is able to improve the quality of the solutions provided by GP. Indeed,

- for the weakly correlated instances, EP improves the quality of the solutions for the three subgroups, i.e., with $\delta \in \{0.25, 0.50, 0.75\}$. Indeed, the improvement varies from 1621.86 units (the first line of the third subgroup with $\delta = 0.75$) to 84383.14 units (the

seventh line of the first subgroup with $\delta = 0.25$);.

- for the strongly correlated instances, EPs' percentage improvements varies from 2.24 units (the first line of the first subgroup with $\delta = 0.25$) to 101000.40 units (the fifth line of the third subgroup with $\delta = 0.75$). Globally, EP realizes an average improvement of 29824.20 units (resp. 14485.45 units) overall treated weakly (respectively, strongly) correlated instances.

It means that searching around a critical item, for a given solution, is a good strategy and so, iterating such a strategy on a series of diversified solutions can also provide several solutions with high quality.

3.4.1.2 HRS versus other available methods

Generally, when using approximate methods to solve optimization problems, it is well-known that different parameter settings for the method lead to results of varying quality. Of course, a different adjustment of the method's parameters would lead to a high percentage of good solutions. But this better adjustment would sometimes lead to heavier runtime requirements. Because the runtime limit was fixed to 5 seconds by some recent available methods in the literature, we then decided to choose the set of values representing a satisfactory trade-off between solution quality and this running time.

Instance I		Variation of α		
n	δ	20%	25%	30%
5000	0.25	896064.47	896064.47	896064.47
7000		1253426.6	1253426.6	1253426.6
10000		1792891.67	1792891.67	1789007.67
13000		2331661.47	2331661.47	2324112.6
15000		2664303	2688439.73	2668301.33
18000		3216050.47	3225073.6	3217603.87
20000		3577328.6	3585801.47	3578827.6
Average		2247389.47	2253337.00	2246763.45
5000	0.5	1632860	1632860	1632860
7000		2284891.6	2284891.6	2284891.6
10000		3261098.67	3263826.73	3261144.67
13000		4235491.47	4243494.73	4236758.47
15000		4882089.27	4896677.93	4884944.6
18000		5854021.33	5875557.8	5857668.47
20000		6515294.33	6528232.47	6516560.13
Average		4095106.667	4103648.751	4096403.991
5000	0.75	2335620.53	2335620.53	2335620.53
7000		3268652.8	3268652.8	3268652.8
10000		4669022.27	4669022.27	4667506.6
13000		6069680.93	6072676.73	6070000.4
15000		6998535.6	7003990.67	6999055.47
18000		8395435.4	8406036.87	8396968.4
20000		9326280.4	9338815.47	9327525.2
Average		5866175.419	5870687.906	5866475.629
Global Average		4069557.185	**4075891.22**	4069881.023

Table 3.2: Behavior of HRS when varying the parameter α for the weakly correlated instances (the value in the bold space indicates the best average tuning).

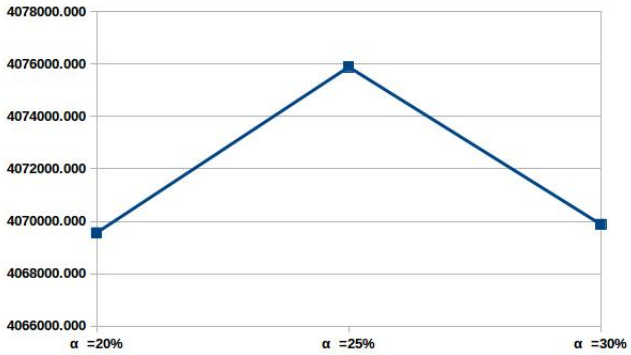

Figure 3.5: Performance of varaint α for weakly correlated instances

The proposed HRS involves three decision variables: (i) the runtime limit, used as the stopping criterion, (ii) the depth parameter Δ representing the size of the solution's neighborhood and (iii) the value of α (that is the percentage of items iteratively removed from the solution at hand). The proposed HRS evaluates as follows:

- First, the runtime limit was fixed to 5 seconds (as mentioned above).

- Second, because neighboring search can be viewed as simple exploring procedure, which tries a series of perturbations of the critical items from the current position to either the left position or the right position, we then fixed Δ to the greatest value such that $0 \leq \Delta \leq n$ regarding the position of the current critical item.

- Third and last, in order to evaluate the behavior of HRS according to the parameter α, we introduced a variation on the number of removing items in the discrete interval $\{20, 25, 30\}$.

Of course, the aim of the hybrid method is to diversify the solutions using another strategy instead of the EP. Herein, the Cplex is used as a black-box optimizer for solving a series of reduced new subproblems (as mentioned in Section 3.3.4). Furthermore, because of the stochastic aspect of HRS, we also considered ten trials for each α variation in the aforementioned interval.

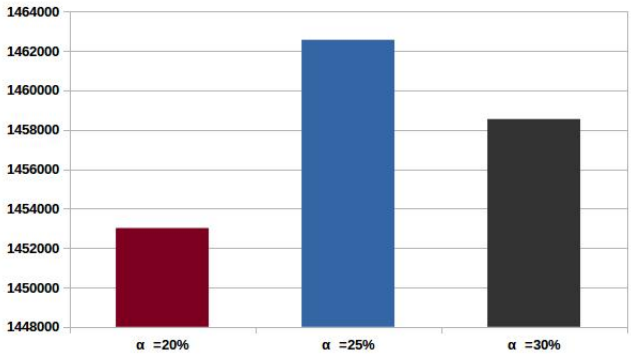

Figure 3.6: Performance of varaint α for strongly correlated instances

Instance I		Variation of α		
n	δ	20%	25%	30%
500	0.25	87415.73	87420.93	87417.53
1000		174545.40	174608.87	174608.53
4000		700061	700080	700075
5000		874250.12	875093.33	874952.27
7000		1224526.53	1225526.67	1225126.27
10000		1670600	1750600	1720600
500	0.5	157287.2	160187.2	158787.2
1000		319152	320780	319640
4000		1281756.47	1282466.67	1282146.33
5000		1603199.07	1603799.07	1603299.07
7000		2241410	2244420	2243415
10000		3205213.20	3206393.33	3205913.13
500	0.75	227566.33	230566.33	229566.33
1000		451363.6	461363.8	455363.6
4000		1839028.87	1847028.87	1842028.87
5000		2261173.33	2309173.33	2290173.33
7000		3227062.13	3231062.13	3228062.13
10000		4608606.67	4615606.67	4612606.67
Average		1453012.09	**1462565.40**	1458543.40

Table 3.3: Behavior of HRS when varying the parameter α for the strongly correlated instances.

Table 3.2 (resp. Table 3.3) reports the average solutions obtained by HRS when varying the parameter α in the discrete interval $\{20, 25, 30\}$ for the weakly (resp. strongly) correlated instances (cf., Figure 3.5 and 3.6). Columns 1 and 2 of both tables report the instance information: the number n of items and the value δ reaching c, the knapsack capacity. Column 3 tallies the average solutions of 15 instances of each line (of the table) when setting α to 20%, column 4 displays the same average solutions for $\alpha = 25\%$ whereas column 5 reports the average solutions when fixing α to the third value of 30%. And the detailed results are exposed in Tables 3.7 - 3.9 (respectively, in Tables 3.10 - 3.12), where, we mentioned to one instance among fifteen in all Tables for two types. We also recall that HRS's runtime limit was fixed to five seconds as considered by more recent methods available in the literature. Each bold value reported in Tables 3.2 and 3.3 represents the best global average value, which characterizes the best value assigned to the destroying parameter.

From Tables 3.2 and 3.3, one can observe what follows:

- For the weakly correlated instances (Table 3.2),

  1. HRS produces better average results for $\alpha = 25\%$ (last line, column 4).
  2. If the percentage value becomes large (i.e., $\alpha = 30\%$, last line, column 5), then the used diversification is less important in the sense that the search process is unable to reach better solutions. In this case, we think that for more largest values HRS may explore a larger space and so, the reactive search is not able to locate a good direction for improving the quality of some visited solutions.
  3. On the other hand, with $\alpha = 20\%$ (last line, column 3), HRS degrade the average quality of the solutions when compared to that realized with $\alpha = 20\%$, but this average value remains not better than that reached by HRS with $\alpha = 25\%$. However, the smallest values for α remain insufficient, because the runtime limit used

remains insufficient for intensifying the search.

- For the strongly correlated instances (Table 3.3), the same phenomenon is observed. Indeed, the best average solution (1462565.40: reported at the last line and the fourth column of Table 3.3) is provided for $\alpha = 25\%$ whereas such a quality is reduced for both $\alpha = 20\%$ and $\alpha = 30\%$. We also believe that the parameter α is very sensitive to the type of instances treated and so, taking other values for α induces the increasing of the runtime.

From Tables 3.2 and 3.3, we can conclude that an intermediate value for α maintains the highest quality of the solutions. Therefore, for the rest of the chapter, we maintain the value of α to 25%.

Weakly correlated instances				Strongly correlated instances			
		HRS				HRS	
n	δ	Best Lit.	av. ten trials	n	δ	Best Lit.	av. ten trials
5000	0.25	896064.13	**896064.47**	500	0.25	87420.93	87420.93
7000		1253426.4	**1253426.6**	1000		174608.87	174608.87
10000		1792890.8	**1792891.67**	4000		700080	700080
13000		2331660.73	**2331661.47**	5000		875093.33	875093.33
15000		2688438.6	**2688439.73**	7000		1225526.67	1225526.67
18000		3225071.27	**3225073.6**	10000		1750600	1750600
20000		3585799.93	**3585801.47**				
5000	0.5	1632859.73	**1632860**	500	0.5	160187.2	160187.2
7000		2284890.07	**2284891.6**	1000		320780	320780
10000		3263825.27	**3263826.73**	4000		1282466.67	1282466.67
13000		4243493	**4243494.73**	5000		1603799.07	1603799.07
15000		4896676.53	**4896677.93**	7000		2244420	2244420
18000		5875554.4	**5875557.8**	10000		3206393.33	3206393.33
20000		6528230.33	**6528232.47**				
5000	0.75	2335620.2	**2335620.53**	500	0.75	230566.33	230566.33
7000		3268651.73	**3268652.8**	1000		461363.8	461363.8
10000		4669021.13	**4669022.27**	4000		1847028.87	1847028.87
13000		6072675.87	**6072676.73**	5000		2309173.33	2309173.33
15000		7003990.13	**7003990.67**	7000		3231062.13	3231062.13
18000		8406034.47	**8406036.87**	10000		4615606.67	4615606.67
20000		9338812.87	**9338815.47**				
Total Av.		4075889.89	**4075891.22**			1462565.4	1462565.4

Table 3.4: Performance of HRS for instances with two scenarios

With $\alpha = 25\%$, Table 3.4 displays the average objective values reached by HRS, compared to the best average solutions available in the literature (extracted from Pinto *et al.* [60]). Column 1 (respectively, column 4) reports the instance information for the first (respectively, second) type of instances. Column 2 (respectively, column 5) shows the best average solution available in the literature. Column 3 (respectively, column 6) displays the average solutions obtained by HRS for the weakly (respectively, strongly) correlated instances. Moreover, because HRS uses a random removing of $\alpha\%$ of items, we also run ten trials of HRS for each considered instance; the average solutions provided for these ten trials are reported in column 4 (respectively, column 8).

Finally, the last line of the table tallies the average solutions of ten trials of HRS for each instance of both types weakly and strongly correlated instances.

From Table 3.4, we observe what follows:

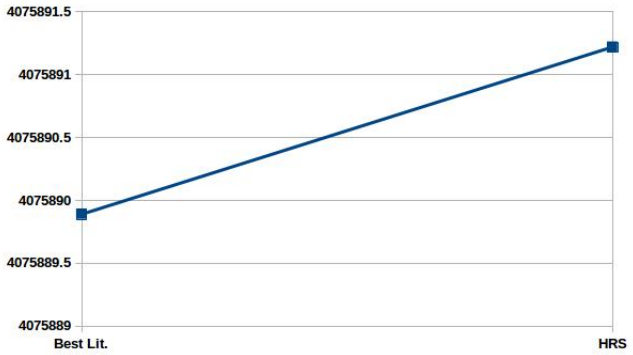

Figure 3.7: Performance of HRS vs Best Lit. for weakly correlated instances

- For the weakly correlated instances, HRS dominates all tested methods available in the literature (cf., Figure 3.7). Indeed, first, for the first group of instances with $\delta = 0.25$, the variation of the average improvement varies from 0.20 to 2.33. Globally, for this value of δ, the average improvement, when compared to the average solutions taken from the literature (Pinto *et al.* [60]), is equal to 1.54. Second, for the group with $\delta = 0.5$, its global average variation becomes equal to 1.70 whereas it is equal to 1.28 for the third group of instances (i.e., with $\delta = 0.75$).
- For the strongly correlated instances, our approach matches all the results published in the literature (Pinto *et al.* [60]). In this case, the strongly correlated instances available in the literature seam more easy to optimally solved by all methods.

Because HRS has a good behavior for two scenarios, we then decided to extend the study to more than two scenarios on the instances available in the literature (Pinto *et al.* [60]).

3.4.2 Performance of HRS on the second group of instances

In this section, we perform a comparative study between HRS and the best available methods of the literature. We do it on the second set of instances containing instances with at least three scenarios. This set is divided in three groups, where each group corresponds to the number of scenarios m that is generated in the discrete interval $\{100, 500, 1000\}$ (these instances are also taken from (cf., Pinto *et al.* [60])). Each group is composed of four subgroups of instances, where each subgroup depends on the number of items n which varies in the interval $\{1000, 5000, 10000, 20000\}$ and for each number of items there are ten instances.

3.4.2.1 Sensitivity of the diversification parameter α on HRS

In order to evaluate the performance of HRS, three versions of that method are considered: HRS_a (by fixing the runtime limit to 60 seconds), HRS_b (by fixing the runtime limit to 300 seconds) and HRS_c (by fixing the runtime limit to 600 seconds). The different runtime limits are those considered in the literature using equivalent computers. This comparison is performed by varying the diversification parameter α in the interval $\{20\%, 25\%, 30\%\}$, as used in the first part (Section 3.4.1).

Table 3.5 evaluates the performance of the three versions of HRS on the four sets of instances. The table contains three parts, where each part is associated with each version of HRS, i.e., HRS_a for the first part, HRS_b for the second part and HRS_c for the last part. For each part, columns 1 and 2 denote the considered instance, columns from 3 to 5 of each part contain the best average solutions reached by each version of HRS when varying the diversification parameter α from 20% to 30%, respectively. And the detailed results are exposed in Tables 3.13 - 3.15, where, we cited to one instance among ten.

	Instance		Variation of α		
CPU's variation	n	m	20%	25%	30%
	1000	100	297196.7	299795.5	298794.4
		500	297290.7	299991.1	298988.8
		1000	300247	300554.9	300352.5
	5000	100	1474614.8	1494614.4	1489614.5
		500	1485610	1502629.6	1489815
		1000	1434577.2	1497486.5	1465848.2
HRS_a	10000	100	2954637.3	2992541	2965545.3
		500	2925009.2	2995130.6	2955326.7
		1000	2945610	2998754.8	2975600
	20000	100	5992137	5992838	5992537.9
		500	5987525.8	5988633.1	5987511.3
		1000	5992121.8	5994321.8	5993421.8
		Average	2673881.46	**2696440.94**	2684446.37
	1000	100	297595	299795.9	298697.1
		500	297298.3	299989.6	298990.7
		1000	300250.1	300561.2	300355.9
	5000	100	1475115	1494615.9	1489715.7
		500	1478845.8	1502839.9	1489847.1
		1000	1485800.3	1497678.6	1490813.1
HRS_b	10000	100	2955545.6	2992545.8	2959546.4
		500	2926706.5	2995727.3	2955737.3
		1000	2947772.1	2999705.2	2975935.9
	20000	100	5992236.9	5992838.8	5992540.1
		500	5988015.3	5989983.5	5987994.5
		1000	5992321.8	5994321.8	5993521.8
		Average	2678125.23	**2696716.96**	2686141.30
	1000	100	297695.2	299794.9	298796.3
		500	297391.1	299991.4	299092.1
		1000	300262.5	300562.4	300361.8
	5000	100	1476615.9	1494616.2	1490615.7
		500	1480848.1	1502843.4	1490848.5
		1000	1486814.8	1497810	1491819.6
HRS_c	10000	100	2956546.5	2992546.9	2960546.2
		500	2927754.9	2995757.7	2956754.5
		1000	2948990	3000035.7	2976936.9
	20000	100	5992338.5	5992840.1	5992639.9
		500	5988059.2	5990058.9	5988022.1
		1000	5992452	5994739.2	5993621.8
		Average	2678814.06	**2696799.73**	2686671.28

Table 3.5: Behavior of HRS_a, HRS_b and HRS_c when varying the diversification parameter α.

From Table 3.5, one can observe that the three versions of HRS (HRS_a, HRS_b and HRS_c) with $\alpha = 25\%$ are competitive and are able to produce, in general, better average solutions than those reached with α equals either to 20% or 30%. On the other hand, HRS_c (with a CPU fixed to 600 seconds) performs better than all other versions by reaching more interesting solutions. Then, for the rest of the experimental part, we use the value of 25%, since it represents the best tuning for the treated instances (the bold values in Table 3.5 represent the best average values achieved by the three versions of HRS for the value 25%)

3.4.2.2 HRS versus other methods on more complex instances

With $\alpha = 25\%$, Table 3.6 displays the average objective values reached by the three versions of HRS when compared to the best average solutions available in the literature (extracted from (cf., Pinto *et al.* [60])). Column 1 shows the runtime limit used by each version of HRS, columns 2 and 3 contains the instance information, column 4 shows the best average solution available in the literature and column 5 tallies the average solution values given by HRS (the average solutions of ten trials) following the three versions: the first (respectively, second and third) part of the table according to the HRS_a (respectively, HRS_b and HRS_c). Finally, the last line of each part summarizes the average solutions of all treated instances.

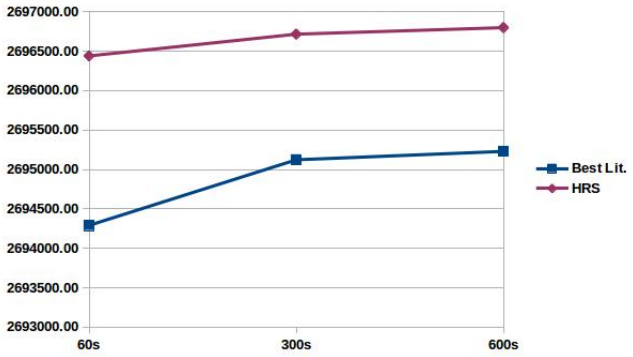

Figure 3.8: Performance of HRS vs Best Lit.

In what follows, we comment on the results reported in Table 3.6 (cf., Figure 3.8):

- On the one hand, one can observe that HRS_a (the first part of Table 3.6) produces better average solutions when compared to those extracted from (cf., Pinto *et al.* [60]) (displayed in column 4). On the other hand, both HRS_b (the second part of Table 3.6) and HRS_c (the third part of Table 3.6) realizes better average solutions.
- The analysis of the average solution value of 2696440.94 (column 5, line corresponding to "Average" of the first part of the table) realized by HRS_a is better than those realized by the best published solution values when the corresponding methods are executed in 300 seconds (an average value of 2695123.42) and 600 seconds (an average value of 2695230.41), respectively. Then, HRS_a is faster than all methods of the literature.
- More increasing the runtime, more the quality of HRS_b and HRS_c are improved. Indeed, the global average value increases from 2696440.94 to 2696716.96 for HRS_b and from 2696716.96 to 2696799.73 for HRS_c.

	n	m	$\text{Best}_{\text{Lit.}}$	Sol_{HRS}
	1000	100	299588.3	**299795.5**
		500	299751.5	**299991.1**
		1000	300314.9	**300554.9**
	5000	100	1493284.8	**1494614.4**
		500	1501777.6	**1502629.6**
		1000	1488904.1	**1497486.5**
HRS_a	10000	100	2989772.1	**2992541**
		500	2993870.3	**2995130.6**
		1000	2998419.4	**2998754.8**
	20000	100	5987260.3	**5992838**
		500	5986620.1	**5988633.1**
		1000	5991924.4	**5994321.8**
		Average	2694290.65	**2696440.94**
	1000	100	299670	**299795.9**
		500	299768.9	**299989.6**
		1000	300341.4	**300561.2**
	5000	100	1493600.4	**1494615.9**
		500	1501828	**1502839.9**
		1000	1496929.2	**1497678.6**
HRS_b	10000	100	2990278	**2992545.8**
		500	2993932.6	**2995727.3**
		1000	2998500.8	**2999705.2**
	20000	100	5987879.3	**5992838.8**
		500	5986719.4	**5989983.5**
		1000	5992033	**5994321.8**
		Average	2695123.42	**2696716.96**
	1000	100	299703.1	**299794.9**
		500	299779.3	**299991.4**
		1000	300347.1	**300562.4**
	5000	100	1493812.8	**1494616.2**
		500	1501866.3	**1502843.4**
		1000	1496944.7	**1497810**
HRS_c	10000	100	2990667.6	**2992546.9**
		500	2993940.7	**2995757.7**
		1000	2998505.2	**3000035.7**
	20000	100	5988340.2	**5992840.1**
		500	5986765.8	**5990058.9**
		1000	5992092.1	**5994739.2**
		Average	2695230.41	**2696799.73**

Table 3.6: Performance of HRS on the second set of instances (the value in bold face indicates the best solution reached by the corresponding algorithm).

We can conclude that the used strategies seem, among the different strategies explored, to be a good compromise in terms of solution quality and runtime. Of course, applying HRS by increasing its maximum number of iterations would require largest average runtime and perhaps better solution qualities.

3.5 Conclusion

In this chapter, we proposed a hybrid reactive search-based algorithm for approximately solving the max-min knapsack problem with multi-scenarios. The method combines a series of neighborhood search techniques in order to provide solutions of high quality. It can be viewed as a two phase search: (i) the first phase is applied for intensifying the search space and, (ii) the second phase applies both destroying and repairing strategies for diversifying the search process. Such a diversification uses a black-box solver in order to escape from a series of local optima. Both phases are also embedded in an iterative search until satisfied the stopping condition. The computational results show that the proposed hybrid method is competitive and it is able to improve several solutions available in the literature. This work has been accepted in an International Journal of Computers and Applications (Taylor & Francis, Eds) with paper: T. Al-Douri, M. Hifi. "A Hybrid Reactive Search for solving the max-min knapsack problem with multi-scenarios" [7].

Instances	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	Best	Average
1_5000_0.25	896276	896277	896275	896277	896276	896277	896277	896276	896277	896277	896277	896276.5
1_7000_0.25	1254479	1254481	1254481	1254481	1254481	1254481	1254481	1254480	1254481	1254481	1254481	1254480.7
1_10000_0.25	1790610	1790615	1790614	1790612	1790611	1790615	1790615	1790615	1790615	1790615	1790615	1790613.7
1_13000_0.25	2337840	2337841	2337842	2337844	2337841	2337843	2337845	2337845	2337845	2337845	2337845	2337843.1
1_15000_0.25	2655195	2655199	2662175	2665182	2665184	2665179	2665181	2665184	2665184	2665184	2665184	2662884.7
1_18000_0.25	3200225	3205211	3210213	3210214	3210213	3210215	3210216	3210216	3210216	3210216	3210216	3208715.5
1_20000_0.25	3554219	3554220	3554222	3583250	3584221	3584222	3584223	3584223	3584223	3584223	3584223	3575124.6
1_5000_0.5	1635223	1635225	1635226	1635227	1635224	1635227	1635227	1635227	1635228	1635227	1635227	1635226.1
1_7000_0.5	2286991	2286990	2286993	2286994	2286995	2286995	2286995	2286995	2286995	2286995	2286995	2286993.8
1_10000_0.5	3254859	3252863	3254864	3264865	3264868	3264868	3264868	3264868	3264868	3264868	3264868	3261665.9
1_13000_0.5	4140216	4240215	4240216	4240218	4240218	4240218	4240218	4240218	4240218	4240218	4240218	4230217.3
1_15000_0.5	4780929	4860930	4890928	4900932	4900933	4900934	4900934	4900934	4900934	4900934	4900934	4883932.2
1_18000_0.5	5577318	5877320	5877319	5877320	5877319	5877320	5877321	5877321	5877321	5877321	5877321	5847320
1_20000_0.5	6421170	6421171	6521172	6521175	6521177	6521178	6521178	6521176	6521178	6521178	6521178	6501175.3
1_5000_0.75	2333590	2333589	2333590	2333594	2333593	2333594	2333592	2333594	2333594	2333594	2333594	2333592.4
1_7000_0.75	3267201	3267202	3267201	3267205	3267204	3267205	3267205	3267205	3267205	3267205	3267205	3267203.8
1_10000_0.75	4672640	4672641	4672647	4672647	4672642	4672646	4672646	4672647	4672647	4672647	4672647	4672645
1_13000_0.75	6011410	6081411	6081412	6081414	6081416	6081415	6081416	6081416	6081416	6081416	6081416	6074414.2
1_15000_0.75	6975050	6975050	6995051	7005052	7005054	7005055	7005055	7005055	7005055	7005055	7005055	6998053.2
1_18000_0.75	8206295	8406297	8406298	8406299	8406299	8406300	8406299	8406300	8406300	8406300	8406300	8386298.7
1_20000_0.75	9238240	9318241	9328261	9338241	9338245	9338245	9338245	9338245	9338245	9338245	9338245	9325245.3

Table 3.7: The quality of the solutions reached by ten trails of HRS for the two scenarios (weakly correlated instances) and $\alpha = 20$

Instances	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	Best	Average
1_5000_0.25	896276	896277	896275	896277	896276	896277	896277	896276	896277	896277	896277	896276.5
1_7000_0.25	1254479	1254481	1254481	1254481	1254481	1254481	1254481	1254480	1254481	1254481	1254481	1254480.7
1_10000_0.25	1790610	1790615	1790614	1790612	1790611	1790615	1790615	1790615	1790615	1790615	1790615	1790613.7
1_13000_0.25	2337840	2337841	2337842	2337844	2337841	2337843	2337845	2337845	2337845	2337845	2337845	2337843.1
1_15000_0.25	2695181	2695182	2695183	2695182	2695184	2695179	2695181	2695184	2695184	2695184	2695184	2695182.4
1_18000_0.25	3220212	3220211	3220213	3220214	3220213	3220215	3220216	3220216	3220216	3220216	3220216	3220214.2
1_20000_0.25	3584219	3584220	3584222	3584220	3584221	3584222	3584223	3584223	3584223	3584223	3584223	3584221.6
1_5000_0.5	1635223	1635225	1635226	1635227	1635224	1635227	1635227	1635227	1635228	1635227	1635227	1635226.1
1_7000_0.5	2286991	2286990	2286993	2286994	2286995	2286995	2286995	2286995	2286995	2286995	2286995	2286993.8
1_10000_0.5	3264861	3264863	3264864	3264865	3264868	3264868	3264868	3264868	3264868	3264868	3264868	3264866.1
1_13000_0.5	4240216	4240215	4240216	4240218	4240218	4240218	4240218	4240218	4240218	4240218	4240218	4240217.3
1_15000_0.5	4900929	4900930	4900928	4900932	4900933	4900934	4900934	4900934	4900934	4900934	4900934	4900932.2
1_18000_0.5	5877318	5877320	5877319	5877320	5877319	5877320	5877321	5877321	5877321	5877321	5877321	5877320
1_20000_0.5	6521170	6521171	6521172	6521175	6521177	6521178	6521178	6521176	6521178	6521178	6521178	6521175.3
1_5000_0.75	2333590	2333589	2333590	2333594	2333593	2333594	2333592	2333594	2333594	2333594	2333594	2333592.4
1_7000_0.75	3267201	3267202	3267201	3267205	3267204	3267205	3267205	3267205	3267205	3267205	3267205	3267203.8
1_10000_0.75	4672640	4672641	4672647	4672647	4672642	4672646	4672646	4672647	4672647	4672647	4672647	4672645
1_13000_0.75	6081410	6081411	6081412	6081414	6081416	6081415	6081416	6081416	6081416	6081416	6081416	6081414.2
1_15000_0.75	7005050	7005050	7005051	7005052	7005054	7005055	7005055	7005055	7005055	7005055	7005055	7005053.2
1_18000_0.75	8406295	8406297	8406298	8406299	8406299	8406300	8406299	8406300	8406300	8406300	8406300	8406298.7
1_20000_0.75	9338240	9338241	9338242	9338241	9338245	9338245	9338245	9338245	9338245	9338245	9338245	9338243.4

Table 3.8: The quality of the solutions reached by ten trails of HRS for the two scenarios (weakly correlated instances) and $\alpha = 25$

Instances	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	Best	Average
1_5000_0.25	896276	896277	896275	896277	896276	896277	896277	896276	896277	896277	896277	896276.5
1_7000_0.25	1254479	1254481	1254481	1254481	1254481	1254481	1254481	1254480	1254481	1254481	1254481	1254480.7
1_10000_0.25	1779610	1780615	1780614	1780612	1790611	1790615	1790615	1790615	1790615	1790615	1790615	1786513.7
1_13000_0.25	2307840	2307841	2307842	2327844	2337841	2337843	2337845	2337845	2337845	2337845	2337845	2327843.1
1_15000_0.25	2655181	2655182	2662183	2665182	2665184	2665179	2665181	2685181	2665184	2695181	2695181	2667881.8
1_18000_0.25	3210225	3215211	3210213	3210214	3220216	3220216	3220216	3220216	3220216	3220216	3220216	3216715.9
1_20000_0.25	3564219	3564220	3564222	3583250	3584221	3584222	3584223	3584223	3584223	3584223	3584223	3578124.6
1_5000_0.5	1635223	1635225	1635226	1635227	1635224	1635227	1635227	1635227	1635228	1635227	1635227	1635226.1
1_7000_0.5	2286991	2286990	2286993	2286994	2286995	2286995	2286995	2286995	2286995	2286995	2286995	2286993.8
1_10000_0.5	3254859	3254863	3254864	3264865	3264868	3264868	3264868	3264868	3264868	3264868	3264868	3261865.9
1_13000_0.5	4220216	4220215	4220216	4220218	4240218	4240218	4240218	4240218	4240218	4240218	4240218	4232217.3
1_15000_0.5	4810929	4860930	4890928	4900932	4900933	4900934	4900934	4900934	4900934	4900934	4900934	4886932.2
1_18000_0.5	5677318	5877320	5877319	5877320	5877319	5877320	5877321	5877321	5877321	5877321	5877321	5857320
1_20000_0.5	6441170	6421171	6521172	6521175	6521177	6521178	6521178	6521176	6521178	6521178	6521178	6503175.3
1_5000_0.75	2333590	2333589	2333590	2333594	2333593	2333594	2333592	2333594	2333594	2333594	2333594	2333592.4
1_7000_0.75	3267201	3267202	3267201	3267205	3267204	3267205	3267205	3267205	3267205	3267205	3267205	3267203.8
1_10000_0.75	4650640	4671641	4672647	4672647	4672642	4672646	4672646	4672647	4672647	4672647	4672647	4670345
1_13000_0.75	6021400	6081411	6081412	6081414	6081416	6081415	6081416	6081416	6081416	6081416	6081416	6075413.2
1_15000_0.75	6975050	6975050	7005051	7005052	7005054	7005055	7005055	7005055	7005055	7005055	7005055	6999053.2
1_18000_0.75	8276295	8406297	8406298	8406299	8406299	8406300	8406299	8406300	8406300	8406300	8406300	8393298.7
1_20000_0.75	9218240	9338241	9338242	9338241	9338245	9338245	9338245	9338245	9338245	9338245	9338245	9326243.4

Table 3.9: The quality of the solutions reached by ten trails of HRS for the two scenarios (weakly correlated instances) and $\alpha = 30$

Instances	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	Best	Average
1_500_0.25	87900	87850	88000	88000	88000	88000	88000	88000	88000	88000	88000	87975
1_1000_0.25	173310	173320	173300	173350	173400	173400	173400	173400	173400	173400	173400	173368
1_4000_0.25	701000	701100	701100	701100	701100	701100	701100	701100	701100	701100	701100	701090
1_5000_0.25	872100	872150	872000	871300	871900	870900	871900	871700	872300	872300	872300	871855
1_7000_0.25	1212300	1212300	1222300	1222300	1222300	1222300	1222300	1222300	1222300	1222300	1222300	1220300
1_10000_0.25	1662600	1671600	1670600	1671600	1671600	1671600	1671600	1671600	1671600	1671600	1671600	1670600
1_500_0.5	156500	156500	156500	156500	156500	156500	156500	156500	156500	156500	156500	156500
1_1000_0.5	310700	310700	319700	320700	320700	320700	320700	320700	320700	320700	320700	318600
1_4000_0.5	1273900	1283500	1283900	1283900	1283900	1283900	1283900	1283900	1283900	1283900	1283900	1282860
1_5000_0.5	1591800	1603800	1603800	1603800	1603800	1603800	1603800	1603800	1603800	1603800	1603800	1602600
1_7000_0.5	2206800	2246700	2246800	2246800	2246800	2246800	2246800	2246800	2246800	2246800	2246800	2242790
1_10000_0.5	3206800	3207600	3207800	3207800	3207800	3207800	3207800	3207800	3207800	3207800	3207800	3207680
1_500_0.75	227400	227500	227600	227600	227600	227600	227600	227600	227600	227600	227600	227570
1_1000_0.75	450400	450400	450400	450400	450400	450400	450400	450400	450400	450400	450400	450400
1_4000_0.75	1833400	1833400	1833400	1833400	1833400	1833400	1833400	1833400	1833400	1833400	1833400	1833400
1_5000_0.75	2251000	2251000	2252000	2257000	2259000	2259000	2260000	2261000	2262000	2266000	2266000	2257800
1_7000_0.75	3214900	3214900	3214900	3214900	3214900	3214900	3214900	3214900	3214900	3214900	3214900	3214900
1_10000_0.75	4603200	4603200	4603200	4603200	4603200	4603200	4603200	4603200	4603200	4603200	4603200	4603200

Table 3.10: The quality of the solutions reached by ten trails of HRS for the two scenarios (strongly correlated instances) and $\alpha = 20$

Instances	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	Best	Average
1_500_0.25	88000	88000	88000	88000	88000	88000	88000	88000	88000	88000	88000	88000
1_1000_0.25	173400	173400	173400	173400	173400	173400	173400	173400	173400	173410	173410	173401
1_4000_0.25	701100	701100	701100	701100	701100	701100	701100	701100	701100	701100	701100	701100
1_5000_0.25	872300	872300	872300	872300	872300	872300	872300	872300	872300	872300	872300	872300
1_7000_0.25	1222300	1222300	1222300	1222300	1222300	1222300	1222300	1222300	1222300	1222300	1222300	1222300
1_10000_0.25	1745600	1745600	1745600	1745600	1745600	1745600	1745600	1745600	1745600	1745600	1745600	1745600
1_500_0.5	158900	158900	158900	158900	158900	158900	158900	158900	158900	158900	158900	158900
1_1000_0.5	320700	320700	320700	320700	320700	320700	320700	320700	320700	320700	320700	320700
1_4000_0.5	1283900	1283900	1283900	1283900	1283900	1283900	1283900	1283900	1283900	1283900	1283900	1283900
1_5000_0.5	1603800	1603800	1603800	1603800	1603800	1603800	1603800	1603800	1603800	1603800	1603800	1603800
1_7000_0.5	2246800	2246800	2246800	2246800	2246800	2246800	2246800	2246800	2246800	2246800	2246800	2246800
1_10000_0.5	3207800	3207800	3207800	3207800	3207800	3207800	3207800	3207800	3207800	3207800	3207800	3207800
1_500_0.75	230600	230600	230600	230600	230600	230600	230600	230600	230600	230600	230600	230600
1_1000_0.75	460400	460400	460400	460400	460400	460400	460400	460400	460400	460400	460400	460400
1_4000_0.75	1843400	1843400	1843400	1843400	1843400	1843400	1843400	1843400	1843400	1843400	1843400	1843400
1_5000_0.75	2308000	2308000	2308000	2308000	2308000	2308000	2308000	2308000	2308000	2308000	2308000	2308000
1_7000_0.75	3224900	3224900	3224900	3224900	3224900	3224900	3224900	3224900	3224900	3224900	3224900	3224900
1_10000_0.75	4613200	4613200	4613200	4613200	4613200	4613200	4613200	4613200	4613200	4613200	4613200	4613200

Table 3.11: The quality of the solutions reached by ten trails of HRS for the two scenarios (strongly correlated instances) and $\alpha = 25$

Instances	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	Best	Average
1_500_0.25	87500	88000	88000	88000	88000	88000	88000	88000	88000	88000	88000	87950
1_1000_0.25	173370	173400	173400	173400	173400	173400	173400	173400	173400	173400	173400	173397
1_4000_0.25	701000	701100	701100	701100	701100	701100	701100	701100	701100	701100	701100	701090
1_5000_0.25	871000	871300	872300	872300	872300	872300	872300	872300	872300	872300	872300	872070
1_7000_0.25	1221300	1221300	1221300	1222300	1222300	1222300	1222300	1222300	1222300	1222300	1222300	1222000
1_10000_0.25	1715600	1715600	1715600	1715600	1715600	1715600	1715600	1715600	1715600	1715600	1715600	1715600
1_500_0.5	157100	157100	157100	157100	157100	157100	157100	157100	157100	157100	157100	157100
1_1000_0.5	318700	318700	318700	318700	318700	318700	318700	318700	318700	318700	318700	318700
1_4000_0.5	1282700	1281900	1283900	1283900	1283900	1283900	1283900	1283900	1283900	1283900	1283900	1283580
1_5000_0.5	1603500	1601800	1602800	1603800	1603800	1603800	1603800	1603800	1603800	1603800	1603800	1603470
1_7000_0.5	2236800	2246700	2246800	2246800	2246800	2246800	2246800	2246800	2246800	2246800	2246800	2245790
1_10000_0.5	3206790	3206800	3206800	3207800	3207800	3207800	3207800	3207800	3207800	3207800	3207800	3207499
1_500_0.75	228600	228600	229600	229600	229600	229600	229600	229600	229600	229600	229600	229400
1_1000_0.75	456400	456400	456400	456400	456400	456400	456400	456400	456400	456400	456400	456400
1_4000_0.75	1831400	1833400	1833400	1833400	1833400	1843400	1833400	1843350	1843400	1843400	1843400	1837195
1_5000_0.75	2288000	2298000	2298000	2298000	2298000	2288000	2288000	2288000	2288000	2288000	2288000	2292000
1_7000_0.75	3214900	3214900	3214900	3214900	3214900	3224900	3224900	3224900	3224900	3224900	3224900	3219900
1_10000_0.75	4603200	4603200	4603200	4603200	4603200	4603200	4603200	4603200	4603200	4603200	4603200	4603200

Table 3.12: The quality of the solutions reached by ten trails of HRS for the two scenarios (strongly correlated instances) and $\alpha = 30$

						HRS$_a$						
Instances	**t1**	**t2**	**t3**	**t4**	**t5**	**t6**	**t7**	**t8**	**t9**	**t10**	**Best**	**Average**
1000_100_1	297233	297235	297234	297236	297232	297235	297233	297231	297236	297235	297236	297234
1000_500_1	297436	297438	297436	297437	297435	297436	297437	297436	297438	297438	297438	297436.7
1000_1000_1	301997	301999	302001	301998	302000	302000	302001	301999	301992	302002	302002	301998.9
5000_100_1	1506648	1506646	1506650	1506651	1506649	1506650	1506649	1506649	1506651	1506649	1506651	1506649.2
5000_500_1	1508108	1508114	1508105	1508116	1508110	1508110	1508115	1508112	1508117	1508118	150818	1508112.5
5000_1000_1	1499490	1499500	1499492	1499495	1499500	1499492	1499489	1499494	1499500	1499597	1499500	1499504.9
10000_100_1	2984203	2984413	2984550	2984587	2984564	2984458	2984550	2984587	2984550	2984592	2984592	2984505.4
10000_500_1	3003150	3003194	3003220	3003764	3003038	3003308	3003010	3003338	3003338	3003764	3003764	3003312.4
10000_1000_1	2992002	2992001	2992003	2992000	2992003	2992002	2992001	2992003	2992002	3000004	3000004	2992802.1
20000_100_1	5994581	5994582	5994584	5994581	5994582	5994584	5994589	5994590	5994584	5994587	5994590	5994584.4
20000_500_1	5986610	5986610	5986610	5986610	5986610	5986610	5986610	5986610	5986610	5986610	5986610	5986610
20000_1000_1	5988505	5988505	5988505	5988505	5988505	5988505	5988505	5988505	5988505	5988505	5988505	5988505
						HRS$_b$						
Instances	**t1**	**t2**	**t3**	**t4**	**t5**	**t6**	**t7**	**t8**	**t9**	**t10**	**Best**	**Average**
1000_100_1	297234	297235	297234	297236	297237	297235	297233	297237	297236	297235	297237	297235.2
1000_500_1	297321	297349	297400	297325	297349	297400	297325	297341	297320	297400	297400	297353
1000_1000_1	301990	301989	301980	301992	301990	301985	301991	301987	301990	301992	301992	301988.6
5000_100_1	1506635	1506638	1506633	1506641	1506644	1506633	1506623	1506643	1506641	1506644	1506644	1506637.5
5000_500_1	1508028	1508050	1508055	1508055	1508051	1508041	1508005	1508014	1508015	1508055	1508055	1508036.9
5000_1000_1	1499905	1500000	1499925	1500001	1499914	1500000	1499938	1499948	1500002	1500002	1500019	1499963.5
10000_100_1	2984403	2984453	2984458	2984404	2984404	2984480	2984401	2984413	2984404	2984480	2984480	2984430
10000_500_1	3004546	3004594	3004592	3004564	3004595	3004595	3004578	3004595	3004591	3004585	3004595	3004583.5
10000_1000_1	3002001	3002001	3002001	3002001	3002001	3002001	3002001	3002001	3002001	3002001	3002001	3002001
20000_100_1	5994293	5994293	5994291	5994308	5994293	5994291	5994308	5994293	5994594	5994594	5994594	5994355.8
20000_500_1	5988177	5988177	5988177	5988177	5988177	5988177	5988177	5988177	5988177	5988177	5988177	5988177
20000_1000_1	5988313	5988313	5988313	5988313	5988313	5988313	5988313	5988313	5988313	5988313	5988313	5988313
						HRS$_c$						
Instances	**t1**	**t2**	**t3**	**t4**	**t5**	**t6**	**t7**	**t8**	**t9**	**t10**	**Best**	**Average**
1000_100_1	297234	297235	297234	297236	297237	297235	297233	297237	297236	297238	297238	297235.5
1000_500_1	297350	297402	297390	297375	297350	297350	297365	297385	297402	297405	297405	297377.4
1000_1000_1	301982	301985	301990	301987	301983	301972	301982	301991	301993	301982	301993	301984.7
5000_100_1	1506645	1506643	1506644	1506645	1506645	1506642	1506637	1506642	1506645	1506645	1506645	1506643.3
5000_500_1	1508048	1508050	1508055	1508055	1508051	1508047	1508015	1508051	1508050	1508055	1508055	1508047.7
5000_1000_1	1500001	1500005	1500003	1500001	1500005	1500003	1500006	1500007	1500000	1500001	1500007	1500003.2
10000_100_1	2984570	2984573	2984571	2984574	2984570	2984573	2984571	2984574	2984565	2984573	2984573	2984571.4
10000_500_1	3004705	3004705	3004705	3004705	3004705	3004705	3004705	3004705	3004705	3004705	3004705	3004705
10000_1000_1	3001993	3001993	3001993	3001993	3001993	3001993	3001993	3001993	3001993	3001993	3001993	3001993
20000_100_1	5994397	5994397	5994397	5994397	5994397	5994397	5994397	5994397	5994397	5994397	5994397	5994397
20000_500_1	5988475	5988475	5988475	5988475	5988475	5988475	5988475	5988475	5988475	5988475	5988475	5988475
20000_1000_1	5989021	5989021	5989021	5989021	5989021	5989021	5989021	5989021	5989021	5989021	5989021	5989021

Table 3.13: The quality of the solutions reached by ten trails of HRS for the multi-scenarios and $\alpha = 20$

						HRS$_a$						
Instances	**t1**	**t2**	**t3**	**t4**	**t5**	**t6**	**t7**	**t8**	**t9**	**t10**	**Best**	**Average**
1000_100_1	297235	297236	297237	297239	297237	297239	297236	297237	297237	297239	297239	297237.2
1000_500_1	297441	297441	297440	297437	297440	297440	297439	297439	297439	297441	297441	297439.7
1000_1000_1	301998	302001	302001	301999	302003	302001	302004	301999	301992	302012	302012	302001
5000_100_1	1506650	1506651	1506650	1506651	1506649	1506650	1506649	1506649	1506653	1506649	1506654	1506650.1
5000_500_1	1508108	1508124	1508105	1508128	1508121	1508110	1508115	1508122	1508126	1508118	1508126	1508117.7
5000_1000_1	1499500	1499500	1499500	1499500	1499500	1499500	1499500	1499500	1499500	1499500	1499500	1499500
10000_100_1	2984203	2984603	2984600	2984602	2984604	2984600	2984601	2984604	2984602	2984601	2984604	2984562
10000_500_1	3003446	3003194	3003920	3003964	3003338	3003338	3003338	3003338	3003338	3004380	3004380	3003559.4
10000_1000_1	2992013	2992023	2992023	2992023	2992013	2992012	2992013	2992033	3000023	3000033	3000033	2993620.9
20000_100_1	5994593	5994593	5994591	5994591	5994594	5994591	5994594	5994593	5994594	5994594	5994594	5994592.8
20000_500_1	5986623	5986623	5986623	5986623	5986623	5986623	5986623	5986623	5986623	5986623	5986623	5986623
20000_1000_1	5988513	5988513	5988513	5988513	5988513	5988513	5988513	5988513	5988513	5988513	5988513	5988513
						HRS$_b$						
Instances	**t1**	**t2**	**t3**	**t4**	**t5**	**t6**	**t7**	**t8**	**t9**	**t10**	**Best**	**Average**
1000_100_1	297237	297238	297237	297239	297237	297239	297236	297237	297237	297239	297239	297237.6
1000_500_1	297438	297439	297440	297437	297440	297440	297439	297439	297439	297441	297441	297439.2
1000_1000_1	302000	302002	302002	302001	302002	302000	302002	302002	302001	302002	302002	302001.4
5000_100_1	1506650	1506651	1506653	1506651	1506654	1506653	1506653	1506653	1506653	1506654	1506654	1506652.5
5000_500_1	1508148	1508150	1508155	1508155	1508151	1508147	1508115	1508151	1508150	1508155	1508155	1508147.7
5000_1000_1	1500011	1500010	1500015	1500009	1500010	1500017	1500009	1500008	1500014	1500019	1500019	1500012.2
10000_100_1	2984603	2984603	2984600	2984604	2984604	2984600	2984601	2984604	2984602	2984601	2984604	2984602.2
10000_500_1	3004646	3004694	3004692	3004664	3004695	3004695	3004678	3004695	3004691	3004695	3004695	3004684.5
10000_1000_1	3002073	3002073	3002073	3002073	3002073	3002073	3002073	3002073	3002073	3002073	3002073	3002073
20000_100_1	5994593	5994593	5994591	5994591	5994594	5994591	5994594	5994593	5994594	5994594	5994594	5994592.8
20000_500_1	5988477	5988477	5988477	5988477	5988477	5988477	5988477	5988477	5988477	5988477	5988477	5988477
20000_1000_1	5988513	5988513	5988513	5988513	5988513	5988513	5988513	5988513	5988513	5988513	5988513	5988513
						HRS$_c$						
Instances	**t1**	**t2**	**t3**	**t4**	**t5**	**t6**	**t7**	**t8**	**t9**	**t10**	**Best**	**Average**
1000_100_1	297237	297238	297237	297239	297237	297235	297236	297237	297237	297239	297239	297237.2
1000_500_1	297441	297441	297440	297437	297440	297440	297439	297439	297439	297441	297441	297439.7
1000_1000_1	302000	302002	302002	302001	302005	302003	302004	302005	302001	302005	302005	302002.8
5000_100_1	1506652	1506651	1506653	1506654	1506654	1506653	1506653	1506653	1506653	1506654	1506654	1506653
5000_500_1	1508148	1508150	1508155	1508155	1508151	1508147	1508115	1508151	1508150	1508155	1508155	1508147.7
5000_1000_1	1500016	1500018	1500020	1500020	1500019	1500017	1500018	1500019	1500020	1500020	1500020	1500018.7
10000_100_1	2984603	2984603	2984600	2984604	2984604	2984600	2984601	2984604	2984602	2984601	2984604	2984602.2
10000_500_1	3004905	3004905	3004905	3004905	3004905	3004905	3004905	3004905	3004905	3004905	3004905	3004905
10000_1000_1	3002073	3002073	3002073	3002073	3002073	3002073	3002073	3002073	3002073	3002073	3002073	3002073
20000_100_1	5994597	5994597	5994597	5994597	5994597	5994597	5994597	5994597	5994597	5994597	5994597	5994597
20000_500_1	5988595	5988595	5988595	5988595	5988595	5988595	5988595	5988595	5988595	5988595	5988595	5988595
20000_1000_1	5989335	5989335	5989335	5989335	5989335	5989335	5989335	5989335	5989335	5989335	5989335	5989335

Table 3.14: The quality of the solutions reached by ten trails of HRS for the multi-scenarios and $\alpha = 25$

						HRS$_a$						
Instances	**t1**	**t2**	**t3**	**t4**	**t5**	**t6**	**t7**	**t8**	**t9**	**t10**	**Best**	**Average**
1000_100_1	297235	297236	297231	297236	297232	297237	297236	297232	297237	297237	297237	297234.9
1000_500_1	297431	297435	297430	297437	297438	297438	297439	297439	297439	297440	297440	297436.6
1000_1000_1	301958	301987	301975	301999	302000	301997	301982	301989	301972	302001	302001	301986
5000_100_1	1506550	1506551	1506550	1506551	1506549	1506550	1506549	1506549	1506553	1506549	1506554	1506550.1
5000_500_1	1507908	1507915	1508001	1508002	1507931	1507950	1507915	1508004	1508007	1508008	1508008	1507964.1
5000_1000_1	1499250	1499314	1499320	1499314	1499320	1499314	1499318	1499314	1499320	1499314	1499320	1499309.8
10000_100_1	2983903	2984003	2984000	2984005	2984004	2984002	2984001	2984004	2984002	2984010	2984010	2983993.4
10000_500_1	3003216	3003204	3003520	3003564	3003138	3003138	3003138	3003138	3003138	3003580	3003580	3003277.4
10000_1000_1	2991813	2991823	2991823	2991823	2991813	2991812	2991813	2991833	2991895	2991898	2991898	2991834.6
20000_100_1	5994193	5994193	5994191	5994191	5994194	5994191	5994194	5994193	5994194	5994194	5994194	5994192.8
20000_500_1	5986223	5986223	5986223	5986223	5986223	5986223	5986223	5986223	5986223	5986223	5986223	5986223
20000_1000_1	5988213	5988213	5988213	5988213	5988213	5988213	5988213	5988213	5988213	5988213	5988213	5988213
						HRS$_b$						
Instances	**t1**	**t2**	**t3**	**t4**	**t5**	**t6**	**t7**	**t8**	**t9**	**t10**	**Best**	**Average**
1000_100_1	297235	297236	297231	297236	297232	297233	297231	297232	297234	297237	297237	297233.7
1000_500_1	297438	297439	297440	297437	297440	297440	297439	297439	297439	297441	297441	297439.2
1000_1000_1	301974	301978	301965	301948	301958	301980	301951	301957	301981	301978	301981	301967
5000_100_1	1506600	1506601	1506603	1506602	1506604	1506603	1506613	1506603	1506613	1506614	1506614	1506605.6
5000_500_1	1508148	1508150	1508155	1508155	1508151	1508147	1508115	1508151	1508150	1508155	1508155	1508147.7
5000_1000_1	1499150	1499114	1499120	1499114	1499120	1499114	1499118	1499114	1499120	1499114	1499120	1499119.8
10000_100_1	2984005	2984003	2984100	2984004	2984004	2984000	2984001	2984004	2984002	2984001	2984004	2984012.4
10000_500_1	3003546	3003594	3003592	3003564	3003595	3003595	3003578	3003595	3003591	3003595	3003595	3003584.5
10000_1000_1	3001873	3001873	3001873	3001873	3001873	3001873	3001873	3001873	3001873	3001873	3001873	3001873
20000_100_1	5994293	5994193	5994291	5994291	5994294	5994291	5994294	5994293	5994294	5994294	5994294	5994282.8
20000_500_1	5986777	5986777	5986777	5986777	5986777	5986777	5986777	5986777	5986777	5986777	5986777	5986777
20000_1000_1	5988289	5988289	5988289	5988289	5988289	5988289	5988289	5988289	5988289	5988289	5988289	5988289
						HRS$_c$						
Instances	**t1**	**t2**	**t3**	**t4**	**t5**	**t6**	**t7**	**t8**	**t9**	**t10**	**Best**	**Average**
1000_100_1	297235	297236	297231	297236	297232	297237	297236	297232	297237	297237	297237	297234.9
1000_500_1	297441	297441	297440	297437	297440	297440	297439	297439	297439	297441	297441	297439.7
1000_1000_1	301974	301978	301965	301957	301964	301975	301965	301957	301988	301988	301988	301971.1
5000_100_1	1506612	1506611	1506613	1506614	1506614	1506613	1506613	1506613	1506613	1506614	1506614	1506613
5000_500_1	1508148	1508150	1508155	1508155	1508151	1508147	1508115	1508151	1508150	1508155	1508155	1508147.7
5000_1000_1	1499170	1499224	1499180	1499175	1499125	1499135	1499125	1499137	1499175	1499221	1499224	1499166.7
10000_100_1	2984103	2984103	2984100	2984104	2984104	2984100	2984101	2984104	2984102	2984101	2984104	2984102.2
10000_500_1	3004005	3004005	3004005	3004005	3004005	3004005	3004005	3004005	3004005	3004025	3004025	3004007
10000_1000_1	3001915	3001915	3001915	3001915	3001915	3001915	3001915	3001915	3001915	3001915	3001915	3001915
20000_100_1	5994323	5994323	5994323	5994323	5994323	5994323	5994323	5994323	5994323	5994323	5994323	5994323
20000_500_1	5987010	5987010	5987010	5987010	5987010	5987010	5987010	5987010	5987010	5987010	5987010	5987010
20000_1000_1	5988335	5988335	5988335	5988335	5988335	5988335	5988335	5988335	5988335	5988335	5988335	5988335

Table 3.15: The quality of the solutions reached by ten trails of HRS for the multi-scenarios and $\alpha = 30$

Chapter 4

A two-stage hybrid method for the multi-scenarios max-min knapsack problem

Contents

In this chapter, we propose a two-stage hybrid method in order to solve approximately the MSKP. The proposed method is based upon three complementary stages: (i) the building stage, (ii) the combination stage and (iii) the two-stage rebuild stage. First, the building stage serves to provide a starting feasible solution by using a greedy procedure; each item is randomly chosen for reaching a starting population of solutions. Second, the combination stage tries to provide a new solution by combining subsets of (starting) solutions. Third, the rebuild stage tries to make an intensification in order to improve the solutions at hand. The proposed method is evaluated on a set of benchmark instances taken from the literature. The obtained results are compared to those reached by the best algorithms available in the literature. These results show that the solution of the proposed method outperforms existing solutions.

4.1 Introduction

In this chapter, we investigate the use of a two-stage hybrid method (abbreviated TSHM) for solving approximately the Multi-Scenario max-min Knapsack Problem (denoted MSKP). The MSKP is an NP-hard combinatorial optimization problem (cf., Hifi *et al.* [37]). Previously, this problem has considered under a multi criteria decision-making framework (cf., Gass *et al.* [28], Steuer [70]) and it was first discussed by Yu [79] for application of robust optimization as a max-min knapsack problem. For such a problem, several practical applications can be

found in Yu [79], where the well-known problem is the capital budgeting problem. In this case, the possible returns of an investment decision i depend on which out of s different scenarios is realized in the future, while the available budget c is known in advance. Such an investment should be made that maximizes the lowest return under all scenarios within the given budget.

An instance of the MSKP is characterized by a knapsack of fixed capacity c and a set I of n items. Each item $i \in I$ is represented by a non-negative weight w_i and set of profits p_i^s, where the profits vary for each possible scenario s (cf., Yu [79]). In other words, a scenario is defined through the set of profits that applies to each item (cf., Iida [43]). The classical knapsack problem is a special case of the MSKP in which s is equal to one; i.e. there is only one scenario (cf., Kellerer *et al.* [45]; Martello and Toth [54]). The aim of the MSKP is to select a subset of items whose total weight fills the knapsack, and whose total profit is maximized in the worst scenario of all the possible scenarios (cf., Pinto *et al.* [60]). The problem can be stated as:

$$\max \quad \min_{s=1,\ldots,S} \Big\{ \sum_{i=1}^{n} p_i^s x_i \Big\} \tag{4.1}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_i \leq c \tag{4.2}$$

$$x_i \in \{0,1\}, \quad \forall\, i \in I, \tag{4.3}$$

where the decision variable $x_i, \forall i \in I$, is equal to one if item i-th is included in the knapsack (considered in the solution) and zero otherwise. From the above formulation, Inequality (4.2) represents the knapsack constraint of capacity c, which imposes the condition that the sum of the weights of the selected items does not exceed the fixed knapsack capacity c. Finally, the last inequality (4.3) represents the integrality constraint imposed on each decision variable x_i, $\forall i \in I$. Without loss of generality, we assume that: (i) all input data $c, p_i^s, w_i, \forall i \in I$, are strictly positive integers. (ii) $\sum_{i \in I} w_i > c$, which avoids trivial solutions.

This chapter is organized as follows. Section 4.2 introduces a two-stage hybrid approach for solving the MSKP. Section 5.3 evaluates the performance of the proposed method on a set of benchmark instances taken from the literature. The presented work is concluded in section 5.4.

4.2 A two-stage hybrid method for the multi-scenarios max-min knapsack problem

In this section, we discuss the suggested two-stage hybrid approach (TSHM) for solving the MSKP. In fact, the proposed algorithm is composed of three main steps. The first step yields a feasible solution by following a greedy algorithm. The second step tries to combine an initial population that enhances the quality of the solution at hand. The third and last step tries to improve the produced solutions by two-stage: (i) stage one applies a series of moves for constructing a series of feasible solutions by critical elements, and (ii) stage two allows the search to explore the left and the right neighborhood of critical elements by using a *depth parameter*.

4.2.1 A starting population

The two-stage hybrid method needs a set of solutions that represent the initial population. In our study, we propose a greedy random algorithm where the order of each selected item is randomly generated. Indeed, we assume that all items are sorted in a random order. Then, the used procedure can be summarized as follows:

Algorithm 10 A greedy random building procedure for MSKP.
Input: An instance of MSKP represented by the set I of items, the weight vector w and the capacity c of the knapsack.
Output: A feasible solution represented by a set S.

1. Let $\bar{c} = c$ be the available (residual) knapsack capacity;
   - Set $S = \emptyset$ (the indexes of the feasible solution; that is, a solution to build);
   - $\bar{S} = \emptyset$ (the indexes of items out of the knapsack, i.e., their capacities are greater than the residual capacity);
   - $nb = 0$ (the number of items considered by the loop repeat);
2. **Repeat**
   a) Randomly select i from $I \setminus \{S \cup \bar{S}\}$.
   b) Set $nb = nb + 1$;
   If $(w_i > \bar{c})$ then set $\bar{S} = \bar{S} \cup \{i\}$;
   Else set $S = S \cup \{i\}$ and reduce $\bar{c}$ by w_i.
   c) Increment i;
   Until $\Big(nb > n$ or $\bar{c} \leq 0\Big)$.
3. Exit with S representing a feasible solution.

Note that the above procedure (Algorithm 10) is applied in order to build a starting solution. So, in order to generate an initial population, we just run the procedure many times, according to the size of the population. Algorithm 11 summarizes how the starting population may generate according to the procedure described by Algorithm 10.

Algorithm 11 is a standard description used for generating a set of diversified solutions. Each solution S is built by applying Algorithm 10 and introduced in the set P representing the population. One can observe that introducing S into P implies that the solution does not belong to the current population. Otherwise, a random removing / adding of some items from the solution S makes it possible to construct a feasible and different solution (as described at step 2).

Algorithm 11 A starting population.
Input: A size N of the population.
Output: A population P of feasible solutions.

1. Set $P = \emptyset$ (the population);
   Set $nb = 0$ (the number of configurations belonging to the population P);
2. **Repeat**
   a) Set $nb = nb + 1$;
   b) Call Algorithm 10 and let S be the solution built.
   c) If $S \notin P$, then set $P = P \cup S$;
   Else perturbe S till obtaining a different configuration before introducing it in P.
   Until $\Big(nb = N\Big)$.
3. Exit with the population P.

4.2.2 A fusion stage

The fusion stage tries to build a set of diversified solutions which depend on the solutions belonging to the current population. In this case, it tries to provide a new trial solution by building a series of new solutions according to a series of pairs of solutions belonging to the population at hand. Then, we propose a new greedy random procedure, called a Fusion Procedure (FP). More precisely, let $S^{(1)}$ and $S^{(2)}$ be the solutions (belonging to the population) being combined and S_{new} be the new resulting solution. Such a solution S_{new} is provided by applying Algorithm 12.

Algorithm 12 A fusion strategy: building a diversified solution.

Input: Two configurations $S^{(1)}$ and $S^{(2)}$ of the current population P.
Output: A new configuration S_{new} (to be introduced in P replacing its worst configuration).

1. Set $S_{new} = S^{(1)} \cap S^{(2)}$, where only items fixed to one in both solutions are saved.
2. For each item fixed to one either in $S^{(1)}$ and not in $S^{(2)}$ or in $S^{(2)}$ and not in $S^{(1)}$, let $x = \text{random}\{0; 1\}$ and for each x, add to S_{new} the index representing x whenever $x = 1$.
3. If S_{new} is unfeasible, then for each scenario set temporary order all items as follows:

   $$p_1^s/w_1 \geq p_2^s/w_2 \geq ... \geq p_n^s/w_n, s = 1, ..., m,$$

   Remove all items from the first one ($i = 1$) of S_{new} until satisfied the capacity constraint c.
4. Call the Cplex as a black-box solver in order to optimize the complementary problem C obtained as follows: for each scenario of S_{new}, let C be the indexes out of S_{new}, i.e., $C = I \setminus S_{new}$.
5. Let S_{MSKP_r} be the solution provided by the Cplex solver (when combined with the partial solution S_{new}).

4.2.3 A two-stage procedure: an intensification procedure

The two-stage procedure can be viewed as a local search that is based on a series of exchanges between items belonging or not to the current solution. The local solution is used to improve the quality of the current solution or a series of building solutions. It also includes to finding a good strategy for generating a final solution x_p, $p \geq 1$, localized in the neighborhood of the current local solution.

4.2.3.1 The first stage

It's a process applied to a series of moves for constructing a series of feasible solutions $x_1, x_2, \ldots, x_p$ by using a *critical element*.

For instance, in MSKP, a standard binary representation scheme is an obvious choice since it represents the underlying $0-1$ non-negative value (Figure 5.1 shows a simple representation of MSKP's solution).

1	1	1	1	...	1	0	0	...	0	0
1	2	3	4	...	$i_{c-}^s 1$	i_c^s	$i_c^s + 1$	...	$n-1$	n

Figure 4.1: Binary representation of MSKP's solution for the s-th scenario.

Herein, a feasible solution $\overline{x}$, for each scenario, is such that

$$\sum w_j \overline{x}_j \leq c.$$

It is obtained by applying a greedy procedure, which provides a solution with a *critical element* (noted i_c^s for each scenario, $s \in S$, as described in Figure 5.1). All variables x_i such that $i < i_c^s$ are fixed to one and all other variables such that $i \geq i_c^s$ are fixed to zero.

Therefore, for each scenario $s \in S$ by moving i_c^s either to the left hand of i_c^s or its right hand, one can build a set of feasible solutions characterizing its associated neighbors. On the other hand, Figure 5.2 illustrates an eventual MSKP's partial feasible solution whenever some variables remain free in the current solution according to the current scenario s, $s \in S$.

1	1	1	1	...	0	0	$\star$	...	$\star$	$\star$
1	2	3	4	...	$i_{c-}^s 1$	i_c^s	$i_c^s + 1$	...	$n-1$	n

Figure 4.2: Configuration of MSKP's partial feasible solution associated with the s-th scenario. The symbol $\star$ denotes the free item

4.2.3.2 The second stage

This stage depends on nonnegative integer parameter, namely Δ, representing a *depth parameter*. The depth parameter allows us to jump backward from i_c^s to $i_c^s - 1$ and so on until the total depth Δ has been explored. It also allows the search to explore the entire left neighborhood and attend to explore the entire right neighborhood. Such a method tries to make solutions combinations of solutions never visited before, and it tries to modify choice rules to encourage move combinations.

Certainly, one can also consider a series of critical elements that can be associated with the current solution; it means that an extended definition of the *critical element* can be used for more diversifying the search process (our limited computational results showed that the simple exploring method was sufficient in providing good behavior for the proposed algorithm, especially on benchmark instances available in the literature for the MSKP).

Hence, for each scenario s, $s \in S$, and with reference to a feasible solution, several manner can be explored for making a move regarding the critical item. In our study, we have followed the standard strategy already used in Hifi *et al.* [41] which works as follows:

1. Let $x_{i_c^s - 1}$ be the $(i_c^s - 1)$-th item of the current solution such that $s \in \{1, ..., m\}$.
2. Set $x_{i_c^s - 1}$ equal to zero, maintain $x_{i_c^s - 2}, ..., x_1$ to one,
3. Complete the partial configuration by applying the greedy procedure, where we suppose that all items are ordered in decreasing order of their profits per weights, i.e.,

$$p_1^s / w_1 \geq p_2^s / w_2 \geq ... \geq p_n^s / w_n, \ \ s = 1, ..., m,$$

On the one hand, by jumping from $i_c^s - 1$ to $i_c^s - \Delta$, a set of new solutions is provided; that is called the *left neighborhood* of the solution at hand. A new search is then applied

in order to improve the quality of the solution. On the other hand, by applying the same process with a look-ahead, from $i_c^s + 1$ to $i_c^s + \Delta$ and by fixing the current item to one, we obtain the *right neighborhood*; that is provided by applying the following equivalent process:

1. Let $x_{i_c^s+1}$ be the $(i_c^s + 1)$-th item of the current solution such that $s \in \{1, ..., m\}$.
2. Set $x_{i_c^s+1}$ to one, make free some variable x_i, $i \leq i_c^s - 1$, and set x_i, $\forall i \geq i_c^s + 2$, to free;
3. Complete the partial configuration by applying the greedy procedure.

Hence, both *left* and *right* neighborhood realize the current neighborhood of the solution at hand, namely S_{MSKP}. In fact, its mimics a truncated branch-and-bound where only a first level is considered; in this case, each successor of the current critical element has the following form: the current critical element is fixed either to zero or one and some of its successor, limited to the Δ-th element, is fixed to one or zero, respectively. Of course, such a manner can be viewed as an extension of the "core problem" used for the single knapsack problem applied herein for the MSKP.

Algorithm 13 An exploring procedure for MSKP.

Input: A feasible solution S_{MSKP} and a depth parameter Δ.
Output: An improved local optimum S'_{MSKP}.

1: **for** $s \in \{1, ..., m\}$ **do**
2: Set $\rho = 0$;
3: **while** $(\rho < \Delta)$ **do**
4: Increment ρ;
let S^s_{MSKP} be the current solution;
Set $I_1 = \{1, ..., i_c^s - \rho - 1\}, I_2 = I \setminus \{I_1 \bigcup \{i_c^s - \rho\}\}$;
5: Set $x_{i_c^s-\rho} = 0, x_i = 1, i \in I_1$ and $x_i, \forall i \in I_2$, are free;
6: Call the greedy procedure for completing S^s_{MSKP} and let S'_{MSKP} be the best solution found so far;
7: **end while**
8: Set $\rho = 0$;
9: **while** $(\rho < \Delta)$ **do**
10: Increment ρ, set $x_{i_c^s+\rho} = 1$ and $\forall i \in I' := I \setminus \{i_c^s\}$, set x_i to free;
11: Call the greedy procedure for solving MSKP with I' and let S'_{MSKP} be the best solution found so far;
12: **end while**
13: **end for**
14: **return** S'_{MSKP};

Algorithm 13 describes the principle of the exploring search applied either to a *partial* or a *complete* feasible solution. As shown from Algorithm 13, it starts with a feasible solution (noted S^s_{MSKP}) reached by applying a greedy procedure and by using a prefixed depth parameter Δ. The main loop (cf., lines from 1 to 13) describes the main steps of the exploring method: for each scenario $s \in S$, both internal loops (the first one from line 3 to

line 7 and the second one from line 9 to line 12) the original MSKP's instance is reconsidered. In fact, on the one hand, the first part (the internal loop from line 3 to line 7), the *left exploring phase* is applied in order to build the first part of the neighborhood. The current neighborhood contains the solution with $x_{i_c^s} - \rho$ fixed to zero, where $\rho \in \{1, ..., \Delta\}$, $\Delta \geq 1$. For each fixed ρ in the discrete interval $\{1, ..., \Delta\}$, a new solution is provided by calling the standard greedy procedure on the remaining sub-instance with its residual capacity. On the other hand, the second loop, from line 9 to line 12, represents the *right exploring phase* that is considered for completing the global neighborhood. In this case $x_{i_c^s} - \rho$ is fixed to one, where $\rho \in \{1, ..., \Delta\}$, $\Delta \geq 1$; for each ρ belonging to $\{1, ..., \Delta\}$, a new solution is reached by applying the greedy procedure to the reduced instance. Finally, Algorithm 13 (cf., line 14) stops with the best solution S'_{MSKP} found so far.

4.3 Computational results

This section evaluates the effectiveness of the proposed two-stage hybrid method on benchmark instances taken from the literature. These instances were extracted from Pinto *et al.* [60] and generated following the standard scheme used by Taniguchi *et al.* [72]. This section is divided into two parts. The first part is focused on the performance of the proposed TSHM on instances related to the max-min knapsack problem with two scenarios and runtime limit fixed to five seconds (as used in the literature (cf., Pinto *et al.* [60])). The second part evaluates the effectiveness of TSHM on more hardness instances containing at least three scenarios, and in this case, the runtime limit is varied. Note also that TSHM was coded in C++ and run on an Intel Pentium Core i7 with 2.1 GHz.

We recall that TSHM involves several decisions. The decisions associated directly to TSHM correspond to (i) the size of the initial population, (ii) the maximum number of local generations plus the times that restarting the populations set (generated). The presented results have been conducted using the following tunings. First, the size of the initial populations is generally recommended between 25 and 45; herein, we used the value of 35 which find as a good compromise between the computing time and the quality of the obtained solutions (this value was also used in Al-Douri and Hifi [5] for studying the behavior of the diversified method). As in Al-Douri and Hifi [5], we also set the number of the local generations to 30 and, the number of the initial populations refreshment to 15.

4.3.1 Behavior of TSHM on instances with two scenarios

In order to evaluate the effectiveness of TSHM, we considered a first set of instances (taken from Pinto *et al.* [60]), where each instance contains two scenarios. For this set, two types of instances are considered: weakly and strongly correlated ones. For each type of instances (weakly and strongly correlated), there are three groups represented according to the size of the capacity of the knapsack. The knapsack capacity c is setting equal to $\delta \times \sum_{i=1}^{n} w_i$, where δ is equal to $0.25, 0.5$, or 0.75. For each pair of (I, δ), there are fifteen instances, where for the weakly (respectively, strongly) correlated type, the cardinality of the set of items I varies in the discrete interval $\in \{5000, 7000, 10000, 13000, 15000, 18000, 20000\}$ (resp. $\in \{500, 1000, 4000, 5000, 7000, 10000\}$).

The results provided by TSHM are compared to the best results available in the literature

(cf., Al-Douri and Hifi [7] and Pinto *et al.* [60]). Indeed, Table 4.1 reports the results reached by applying TSHM on both groups of instances: weakly correlated instances (the first three columns of Table 4.1) and strongly correlated instances (the last three columns of Table 4.1). In fact, columns 1 and 2 (resp. columns 4 and 5) represent the instance information: I, the number of items, and δ, the value of the parameter used for calculating the capacity c of the instance's knapsack. Column 3 (resp. column 6) tallies the average solutions obtained by TSHM when it is applied for all scenarios on the weakly (resp. strongly) correlated instances and, we considered ten trials for each instance. According to these trails, column 3 (respectively column 6) shows the average results of the solution values reached by these trials and those displayed in Table 4.4 (respectively Table 4.5), where, we cited to one instance among fifteen in these Tables for two types of instances. Finally, the last line of the table summarizes the average values of all solutions reached by TSHM. Note also that TSHM provides the solutions within the negligible runtime.

Weakly correlated instance			Strongly correlated instance		
I	δ	TSHM	I	δ	TSHM
5000	0.25	896065	500	0.25	87420.93
7000		1253427.10	1000		174608.87
10000		1792893.05	4000		700080
13000		2331663.00	5000		875093.33
15000		2688441.80	7000		1225526.67
18000		3225075.69	10000		1750600
20000		3585803.05			
5000	0.5	1632861.00	500	0.5	160187.2
7000		2284892.73	1000		320780
10000		3263828.29	4000		1282466.67
13000		4243496.30	5000		1603799.07
15000		4896679.28	7000		2244420
18000		5875560.85	10000		3206393.33
20000		6528235.15			
5000	0.75	2335621.85	500	0.75	230566.33
7000		3268653.87	1000		461363.8
10000		4669023.49	4000		1847028.87
13000		6072678.10	5000		2309173.33
15000		7003992.36	7000		3231062.13
18000		8406039.09	10000		4615606.67
20000		9338817.10			
Total Average		4075892.77			1462565.4

Table 4.1: Performance of TSHM on instances with two scenarios

Generally, when using approximate methods to solve optimization problems, it is well-known that different parameter settings for the method lead to results of varying quality. Certainly, a different adjustment of the parameters' method would lead to a high percentage of good solutions. But this better adjustment would sometimes lead to heavier runtime requirements. Because the runtime limit was fixed to 5 seconds by some recent available

methods in the literature, we then decided to choose the set of values representing a satisfactory trade-off between solution quality and this running time.

The proposed TSHM involves two decision variables: (i) the runtime limit, used as the stopping criterion, and (ii) the depth parameter Δ representing the size of the solution's neighborhood. First, the runtime limit was fixed to 5 seconds for the first set of instances, as discussed above. Second, because neighboring search can be viewed as a simple exploring procedure, which tries a series of perturbations of the critical items from the current position to either the left position or the right position, we then fixed Δ to the greatest value such that $0 \leq \Delta \leq n$ regarding the position of the current critical item.

Weakly correlated instance				Strongly correlated instance			
I	δ	HRS	TSHM	I	δ	HRS	TSHM
5000	0.25	896064.47	896065	500	0.25	87420.93	87420.93
7000		1253426.6	1253427.10	1000		174608.87	174608.87
10000		1792891.67	1792893.05	4000		700080	700080
13000		2331661.47	2331663.00	5000		875093.33	875093.33
15000		2688439.73	2688441.80	7000		1225526.67	1225526.67
18000		3225073.6	3225075.69	10000		1750600	1750600
20000		3585801.47	3585803.05				
5000	0.5	1632860	1632861.00	500	0.5	160187.2	160187.2
7000		2284891.6	2284892.73	1000		320780	320780
10000		3263826.73	3263828.29	4000		1282466.67	1282466.67
13000		4243494.73	4243496.30	5000		1603799.07	1603799.07
15000		4896677.93	4896679.28	7000		2244420	2244420
18000		5875557.8	5875560.85	10000		3206393.33	3206393.33
20000		6528232.47	6528235.15				
5000	0.75	2335620.53	2335621.85	500	0.75	230566.33	230566.33
7000		3268652.8	3268653.87	1000		461363.8	461363.8
10000		4669022.27	4669023.49	4000		1847028.87	1847028.87
13000		6072676.73	6072678.10	5000		2309173.33	2309173.33
15000		7003990.67	7003992.36	7000		3231062.13	3231062.13
18000		8406036.87	8406039.09	10000		4615606.67	4615606.67
20000		9338815.47	9338817.10				
Total Av.		4075891.22	4075892.77			1462565.4	1462565.4

Table 4.2: Behavior of TSHM vs HRS on instances with two scenarios

Table 4.2 illustrates the objective value obtained with TSHM on both sets of instances (the weakly and strongly correlated instances). The solutions obtained are compared to those published in Al-Douri and Hifi [7] (noted HRS-Hybrid Reactive Search-). Columns 1 and 2 (respectively, columns 5 and 6) show the instance information: I is the number of items, and δ the value of the parameter used for calculating the capacity c of that instances knapsack. Column 3 (respectively, column 7) represents the best average solution available from the literature. Finally, the average solutions obtained by TSHM for all scenarios for the weakly (strongly) correlated instances are shown in column 4 (respectively, column 8).

From Table 4.2, we observe what follows:

1. For the weakly correlated instances, TSHM outperforms HRSs solutions. Indeed, on the one hand, for the first group of instances when $\delta = 0.25$, the variation of the improvement varies from 0.5 to 2.09. Globally, for the aforementioned of δ, the average improvement, when compared to the average solutions HRS, is equal to 2.09. Second, for the group with $\delta = 0.5$, the variation of the improvement varies from 1 to 3.05, and, its global average equal to 3.05. Third and last, when $\delta = 0.75$, the variation of the improvement varies from 1.07 to 2.22, and, its global average equal to 2.22.

2. For the strongly correlated instances, our approach matches all the results HRS. In this case, the strongly correlated instances available in the literature seem more easier to optimally solved by all methods.

4.3.2 Behavior of TSHM on instances with more than two scenarios

This section discusses a comparative study between TSHMs results and those published in the literature. The study is made on three groups of instances divided according to the number of scenarios m, which is generated in the discrete interval $\{100, 500, 1000\}$ (these sets are taken from Pinto *et al.* [60]). Each group is composed of four subgroups of instances, where each subgroup depends on the number of items n in the interval $\{1000, 5000, 10000, 20000\}$ and, for each number of items, there are ten instances.

In order to evaluate the performance of the proposed TSHM, there are three different runtime limits (60, 300 and 600) seconds as considered in the literature using equivalent computers. Table 4.3 illustrates the objective value obtained by TSHM on sets containing more than two scenarios.

These results are compared to the best solutions available in the literature. Column 1 shows the variation of the runtime used, columns 2 and 3 contain the instance information, column 4 displays the best average solution values taken from the literature and column 5 reports the average solution values given by TSHM and, we considered ten trials for each instance. According to these trails, column 5 shows the average results of the solution values reached by these trials and those displayed in Table 4.6, where, we mentioned to one instance among ten in this Table.

From Table 4.3 one can observe what follows:

1. For the first fixed runtime (60 seconds), TSHM provides better average solutions than those reached by HRS. In this case, increasing the fixed runtime to 300 (respectively, to 600) seconds increases TSHM's average solution quality.

2. Solutions realized by TSHM outperforms those reached by the best methods available in the literature. It happens when fixing the runtime to 60 seconds (respectively, 300 or 600 seconds). Finally, the average solution values obtained by TSHM is equal to 2696593.59 (respectively, 2696731.20 or 2696802.76) depending on the three fixed runtimes.

It can be concluded that the utilized strategies seem, among the different strategies explored, to be a good compromise in terms of solution quality and runtime.

CPU's variation	n	m	HRS	TSHM
	1000	100	299795.5	299796.2
		500	299991.1	299992.1
		1000	300554.9	300558.5
	5000	100	1494614.4	1494615.2
		500	1502629.6	1502726.8
		1000	1497486.5	1497515.8
60 sec	10000	100	2992541	2992543.2
		500	2995130.6	2995590
		1000	2998754.8	2998982
	20000	100	5992838	5992839
		500	5988633.1	5989642.5
		1000	5994321.8	5994321.8
		Average	2696440.94	2696593.59
	1000	100	299795.9	299796.6
		500	299989.6	299990.1
		1000	300561.2	300562.1
	5000	100	1494615.9	1494616
		500	1502839.9	1502844
		1000	1497678.6	1497683.8
300 sec	10000	100	2992545.8	2992546.1
		500	2995727.3	2995730.4
		1000	2999705.2	2999781
	20000	100	5992838.8	5992839.8
		500	5989983.5	5990002
		1000	5994321.8	5994382.5
		Average	2696716.96	2696731.20
	1000	100	299794.9	299796.8
		500	299991.4	299991.5
		1000	300562.4	300563.7
	5000	100	1494616.2	1494616.5
		500	1502843.4	1502847.5
		1000	1497810	1497814.2
600 sec	10000	100	2992546.9	2992546.7
		500	2995757.7	2995758.4
		1000	3000035.7	3000061.2
	20000	100	5992840.1	5992841
		500	5990058.9	5990024.6
		1000	5994739.2	5994771
		Average	2696799.73	2696802.76

Table 4.3: Performance of TSHM vs HRS on instances with multi-scenarios

4.4 Conclusion

In this chapter, a two-stage hybrid approach for solving the multi-scenarios max-min knapsack problem is proposed. Suggested approach uses a greedy algorithm by choosing the order of items randomly in order to generate a starting population solution. The proposed method combines initial population solutions in order to provide solutions of high qualities. It tries to improve the current solutions by exploring and combining a series of solutions. Compu-

tational results showed that the two-stage hybrid approach yields a high-quality solutions and was able to improve most solutions available in the literature. This work is extended to the diversified method for the multi-scenarios max-min knapsack problem which is published in an international conference with proceeding *IEEE on Control Decision and Information Technologies (CoDit'16)* [5]. It's accepted in an International journal "Intelligent Engineering Informatics" with the paper: T., Al-douri and M., Hifi "A Two-Stage Hybrid Method for the Multi-Scenarios Max-Min Knapsack Problem" [8].

Instances	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	Best	Average
1_5000_0.25	896276	896277	896275	896277	896276	896277	896277	896278	896277	896277	896278	896276.7
1_7000_0.25	1254479	1254481	1254481	1254481	1254481	1254481	1254481	1254482	1254482	1254481	1254482	1254481
1_10000_0.25	1790610	1790615	1790614	1790612	1790611	1790617	1790615	1790615	1790615	1790617	1790617	1790614.1
1_13000_0.25	2337840	2337841	2337842	2337844	2337841	2337846	2337845	2337846	2337845	2337846	2337846	2337843.6
1_15000_0.25	2695181	2695182	2695183	2695182	2695187	2695179	2695181	2695187	2695184	2695184	2695187	2695183
1_18000_0.25	3220212	3220211	3220213	3220214	3220213	3220215	3220216	3220219	3220216	3220219	3220219	3220214.8
1_20000_0.25	3584219	3584220	3584222	3584220	3584221	3584228	3584228	3584223	3584225	3584227	3584228	3584223.3
1_5000_0.5	1635223	1635225	1635226	1635227	1635224	1635227	1635230	1635227	1635228	1635230	1635230	1635226.7
1_7000_0.5	2286991	2286990	2286993	2286994	2286995	2286999	2286995	2286999	2286995	2286998	2286999	2286994.9
1_10000_0.5	3264861	3264863	3264864	3264865	3264868	3264868	3264868	3264868	3264868	3264868	3264868	3264866.1
1_13000_0.5	4240216	4240215	4240216	4240218	4240218	4240218	4240220	4240218	4240221	4240221	4240221	4240218.1
1_15000_0.5	4900929	4900930	4900928	4900932	4900933	4900934	4900934	4900934	4900934	4900934	4900934	4900932.2
1_18000_0.5	5877318	5877320	5877319	5877320	5877319	5877320	5877325	5877324	5877325	5877324	5877325	5877321.4
1_20000_0.5	6521170	6521171	6521172	6521175	6521177	6521178	6521182	6521180	6521183	6521181	6521183	6521176.9
1_5000_0.75	2333590	2333589	2333590	2333594	2333596	2333597	2333597	2333594	2333595	2333597	2333597	2333593.9
1_7000_0.75	3267201	3267202	3267201	3267205	3267204	3267205	3267205	3267208	3267208	3267208	3267208	3267204.7
1_10000_0.75	4672640	4672641	4672647	4672647	4672642	4672646	4672649	4672650	4672651	4672651	4672651	4672646.4
1_13000_0.75	6081410	6081411	6081412	6081414	6081416	6081415	6081416	6081419	6081419	6081419	6081419	6081415.1
1_15000_0.75	7005050	7005050	7005051	7005052	7005060	7005055	7005055	7005060	7005061	7005061	7005060	7005055.5
1_18000_0.75	8406295	8406297	8406298	8406299	8406299	8406300	8406299	8406302	8406302	8406302	8406302	8406299.3
1_20000_0.75	9338240	9338241	9338242	9338241	9338245	9338245	9338245	9338249	9338248	9338249	9338249	9338244.5

Table 4.4: The quality of the solutions reached by ten trails of TSHM for the two scenarios (weakly correlated instances)

Instances	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	Best	Average
1_500_0.25	88000	88000	88000	88000	88000	88000	88000	88000	88000	88000	88000	88000
1_1000_0.25	173400	173400	173400	173400	173400	173400	173400	173400	173400	173410	173410	173401
1_4000_0.25	701100	701100	701100	701100	701100	701100	701100	701100	701100	701100	701100	701100
1_5000_0.25	872300	872300	872300	872300	872300	872300	872300	872300	872300	872300	872300	872300
1_7000_0.25	1222300	1222300	1222300	1222300	1222300	1222300	1222300	1222300	1222300	1222300	1222300	1222300
1_10000_0.25	1745600	1745600	1745600	1745600	1745600	1745600	1745600	1745600	1745600	1745600	1745600	1745600
1_500_0.5	158900	158900	158900	158900	158900	158900	158900	158900	158900	158900	158900	158900
1_1000_0.5	320700	320700	320700	320700	320700	320700	320700	320700	320700	320700	320700	320700
1_4000_0.5	1283900	1283900	1283900	1283900	1283900	1283900	1283900	1283900	1283900	1283900	1283900	1283900
1_5000_0.5	1603800	1603800	1603800	1603800	1603800	1603800	1603800	1603800	1603800	1603800	1603800	1603800
1_7000_0.5	2246800	2246800	2246800	2246800	2246800	2246800	2246800	2246800	2246800	2246800	2246800	2246800
1_10000_0.5	3207800	3207800	3207800	3207800	3207800	3207800	3207800	3207800	3207800	3207800	3207800	3207800
1_500_0.75	230600	230600	230600	230600	230600	230600	230600	230600	230600	230600	230600	230600
1_1000_0.75	460400	460400	460400	460400	460400	460400	460400	460400	460400	460400	460400	460400
1_4000_0.75	1843400	1843400	1843400	1843400	1843400	1843400	1843400	1843400	1843400	1843400	1843400	1843400
1_5000_0.75	2308000	2308000	2308000	2308000	2308000	2308000	2308000	2308000	2308000	2308000	2308000	2308000
1_7000_0.75	3224900	3224900	3224900	3224900	3224900	3224900	3224900	3224900	3224900	3224900	3224900	3224900
1_10000_0.75	4613200	4613200	4613200	4613200	4613200	4613200	4613200	4613200	4613200	4613200	4613200	4613200

Table 4.5: The quality of the solutions reached by ten trails of TSHM for the two scenarios (strongly correlated instances)

						60s						
Instances	**t1**	**t2**	**t3**	**t4**	**t5**	**t6**	**t7**	**t8**	**t9**	**t10**	**Best**	**Average**
1000_100_1	297235	297236	297237	297239	297237	297239	297236	297237	297237	297239	297239	297237.2
1000_500_1	297441	297441	297440	297437	297440	297440	297439	297439	297439	297441	297441	297439.7
1000_1000_1	301998	302001	302001	301999	302003	302001	302004	301999	301992	302012	302012	302001
5000_100_1	1506650	1506651	1506650	1506651	1506649	1506650	1506649	1506649	1506653	1506649	1506654	1506650.1
5000_500_1	1508108	1508124	1508105	1508128	1508121	1508110	1508115	1508122	1508126	1508118	1508126	1508117.7
5000_1000_1	1499500	1499500	1499500	1499500	1499500	1499500	1499500	1499500	1499500	1499500	1499500	1499500
10000_100_1	2984203	2984603	2984600	2984602	2984604	2984600	2984601	2984604	2984605	2984605	2984605	2984562.7
10000_500_1	3004546	3004594	3004620	3004664	3004638	3004648	3004648	3004648	3004648	3004647	3004648	3004630.1
10000_1000_1	3001001	3001000	3001000	3001007	3001006	3001007	3001007	3001000	3001005	3001007	3001007	3001004
20000_100_1	5994593	5994593	5994591	5994591	5994594	5994595	5994594	5994593	5994594	5994597	5994597	5994593.5
20000_500_1	5986623	5986623	5986630	5986634	5986633	5987638	5987638	5987638	5986635	5986636	5987638	5986932.8
20000_1000_1	5988513	5988513	5988513	5988513	5988513	5988513	5988513	5988513	5988513	5988513	5988513	5988513
						300s						
Instances	**t1**	**t2**	**t3**	**t4**	**t5**	**t6**	**t7**	**t8**	**t9**	**t10**	**Best**	**Average**
1000_100_1	297237	297238	297237	297239	297237	297239	297236	297237	297237	297239	297239	297237.6
1000_500_1	297438	297439	297440	297437	297440	297440	297439	297439	297439	297441	297441	297439.2
1000_1000_1	302000	302002	302008	302007	302009	302009	302002	302005	302008	302009	302009	302005.9
5000_100_1	1506650	1506651	1506653	1506651	1506654	1506653	1506653	1506653	1506653	1506654	1506654	1506652.5
5000_500_1	1508148	1508150	1508155	1508155	1508151	1508147	1508115	1508151	1508150	1508155	1508155	1508147.7
5000_1000_1	1500011	1500010	1500015	1500009	1500010	1500017	1500009	1500008	1500014	1500019	1500019	1500012.2
10000_100_1	2984603	2984603	2984600	2984604	2984604	2984603	2984601	2984604	2984604	2984604	2984604	2984603
10000_500_1	3004846	3004844	3004462	3004867	3004795	3004865	3004875	3004875	3004875	3004875	3004875	3004817.9
10000_1000_1	3002093	3002093	3002093	3002093	3002093	3002093	3002093	3002093	3002093	3002093	3002093	3002093
20000_100_1	5994593	5994597	5994597	5994597	5994597	5994597	5994597	5994597	5994597	5994597	5994597	5994596.6
20000_500_1	5988385	5988385	5988385	5988385	5988385	5988385	5988385	5988385	5988385	5988385	5988385	5988385
20000_1000_1	5988733	5988733	5988733	5988733	5988733	5988733	5988733	5988733	5988733	5988733	5988733	5988733
						600s						
Instances	**t1**	**t2**	**t3**	**t4**	**t5**	**t6**	**t7**	**t8**	**t9**	**t10**	**Best**	**Average**
1000_100_1	297237	297238	297237	297239	297237	297235	297236	297237	297237	297239	297239	297237.2
1000_500_1	297441	297441	297440	297442	297440	297440	297341	297340	297343	297343	297343	297401.1
1000_1000_1	302000	302002	302004	302001	302005	302006	302005	302009	302009	302009	302009	302005
5000_100_1	1506652	1506651	1506653	1506654	1506654	1506653	1506653	1506653	1506653	1506654	1506654	1506653
5000_500_1	1508148	1508150	1508155	1508155	1508155	1508147	1508115	1508158	1508158	1508157	1508158	1508149.8
5000_1000_1	1500016	1500018	1500035	1500035	1500019	1500028	1500027	1500029	1500032	1500035	1500035	1500027.4
10000_100_1	2984603	2984603	2984600	2984604	2984604	2984600	2984601	2984604	2984602	2984601	2984604	2984602.2
10000_500_1	3004907	3004907	3004907	3004907	3004907	3004907	3004907	3004907	3004907	3004907	3004907	3004907
10000_1000_1	3002169	3002169	3002169	3002169	3002169	3002169	3002169	3002169	3002169	3002169	3002169	3002169
20000_100_1	5994597	5994597	5994597	5994597	5994597	5994597	5994597	5994597	5994597	5994597	5994597	5994597
20000_500_1	5988595	5988595	5988595	5988595	5988595	5988595	5988595	5988595	5988595	5988595	5988595	5988595
20000_1000_1	5989459	5989459	5989459	5989459	5989459	5989459	5989459	5989459	5989459	5989459	5989459	5989459

Table 4.6: The quality of the solutions reached by ten trails of TSHM for the multi-scenarios

Chapter 5

Combining GRASP and Path-Relinking for the multi-scenario max-min knapsack problem

Contents

In this chapter, both greedy randomized adaptive search procedure and path-relinking are combined for tackling the max-min knapsack problem with multi-scenarios. The proposed method iterates a building phase and an improvement phase that is based upon an extended search process. The first phase yields a (starting) feasible solution for the problem by applying a greedy randomized adaptive search procedure. The second phase tries to enhance each current solution by applying the path-relinking based strategy. Finally, the proposed method is evaluated on a set of benchmark instances taken from the literature. The obtained results are compared to those reached by some best algorithms available in the literature. Encouraging results have been obtained.

5.1 Introduction

In this chapter, we propose to combine the greedy randomized adaptive search procedure (namely GRASP) with the path-relinking strategy (namely PR) for solving the Multi-Scenario max-min Knapsack Problem (abbreviated to MSKP). We recall that, an instance of MSKP is characterized by a knapsack of fixed capacity c and a set I of n items. Each item $i,\ i \in I$, is represented by a non-negative weight w_i and set of profits p_i^s, where profits vary for each possible scenario $s,\ s = 1, \ldots, m$ (cf., Yu [79]). In other words, a scenario is defined

through a set of profits associated with each item (cf., Iida [43]). It is noticed that the classical knapsack problem is a special case of MSKP in which only one scenario is considered, i.e., s is setting equal to one (cf., Kellerer *et al.* [45]; Martello and Toth [54]). Finally, the goal of MSKP is to find (select) a subset of items of I whose total weight does not exceed the knapsack's capacity c, and whose total profit is maximized for the worst scenario among all considered scenarios (cf., Pinto *et al.* [60]). Formally, MSKP can be written as follows:

MSKP

$$\max \quad \min_{s=1,\ldots,S} \Big\{ \sum_{i=1}^{n} p_i^s x_i \Big\} \tag{5.1}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_i \leq c \tag{5.2}$$

$$x_i \in \{0,1\}, \quad \forall\, i \in I \tag{5.3}$$

where $x_i, \forall i \in I$, denotes a binary decision variable that takes the value one if the i-th item is included in the knapsack (considered in the solution), zero otherwise. Equation (5.1) denotes the objective of the problem where it represents m scenarios. In this case, one can observe that the aim is to optimize (maximize) the value of the total profit on the items placed in the knapsack regarding the worst scenario among the m available scenarios. Inequality (5.2) is the knapsack constraint of capacity c, which imposes the condition that the sum of the weights of the selected items does not exceed the fixed capacity c. Finally, inequality (5.3) are the integrality constraints imposed on each decision variable x_i, $\forall i \in I$. In order to avoid trivial cases, it is assumed that (i) all input data $c, p_i^s, w_i, \forall i \in I$, are strictly positive integers, and (ii) $\sum_{i \in I} w_i > c$, i.e., all items can't be selected.

The remainder of the chapter is organized as follows. Section 5.2 discusses the combined method proposed for approximately solving the MSKP. Section 5.3 evaluates the performance of the proposed method throughout a set of benchmark instances taken from the literature and its results are compared to those achieved by some better method available in the literature. The summary of this chapter is drawn in Section 5.4.

5.2 A combined method for the MSKP

This section is composed of three parts. In the first part, we discuss the Greedy Randomized Adaptive Search Procedure (GRASP). The second part shows how a starting solution can be reached by using a standard greedy randomized adaptive search procedure: that is, the first phase of the method. The third part shows how an intensification search can be introduced in order to highlight the quality of the solutions; that is, the second phase of the method where both path-relinking and exploring procedure are combined.

5.2.1 Greedy Randomized Adaptive Search Procedure (GRASP)

A greedy randomized adaptive search procedure (GRASP) is a metaheuristic for combinatorial optimization. Such as an approach was first proposed by Feo and Resende [24], [25]. A GRASP is a repeated process, each iteration of GRASP composed two steps: The construction step and the local search step. In the construction step, the initial solution builds in an

adaptive and iterative technique. At each iteration one element is selected randomly from a *restricted candidate list (RCL)* and added to the current solution. The *RCL* is a set of high quality ordered elements with respect to a greedy function, which estimates the benefit of element selection based on the current solution. The relevant information is then updated by adding a new element. The iteration is repeated until a feasible solution is found. In this case, at each iteration of a GRASP, there are different solutions, not necessarily improving the previous solution.

The local search step tries to improve the solution obtain from the first step by exploring neighbourhoods of the current solution and replacing it with a better one. The local search step is terminated when no improvement can be detected. If there is no better solution in the neighbourhood, the current solution is called the locally optimal.

The Algorithm 14 is showing the main steps of GRASP.

Algorithm 14 A generic GRASP pseudo-code.

Input: Instance().
Output: Best solution found.
1: **while** GRASP stopped criterion not satisfied **do**
2: Construction step.
3: Local search step.
4: Update best solution found;
5: **end while**
6: end GRASP;

5.2.2 Greedy Randomized Adaptive Search Procedure (GRASP) for MSKP

A starting solution can be built by applying GRASP. In such approach, an iteration of the method involves building a current solution by applying some local search strategies. The building stage may be iterative (step by step), random or adaptive regarding the structure of the problem. Herein, an adaptive approach is proposed, where an (or a set of) solution(s) is generated by using an exploring procedure.

5.2.2.1 The first solution

From literature survey, we can point on the *constructive greedy procedure* proposed in Aldouri and Hifi [5], where an initial solution is constructed "step by step" in a greedy way. Indeed, for a given scenario s, $s \in \{1, \ldots, m\}$, the constructive procedure is based on Dantzig's procedure applied for, providing both lower and upper bounds for the single KP (cf., Dantzig [18]). Initially, all items of each given scenario s are sorted in non-increasing order of their ratios $\frac{\text{profit}}{\text{weight}}$, i.e.,

$$\frac{p_1^s}{w_1} \geq \frac{p_2^s}{w_2} \geq ... \geq \frac{p_n^s}{w_n}, \ s = 1, \ldots, m. \tag{5.4}$$

Then, for each scenario s, $s \in \{1, \ldots, m\}$, the current item is selected whenever its weight is less than or equal to the residual weight and the process is repeated until all items have been examined. Finally, the initial solution is that realizing the worst value among all scenarios.

Algorithm 15 GCP for MSKP.

Input. An instance of P_{MSKP}.

Output. A feasible solution S^{worst}_{MSKP} for P_{MSKP}.

1: set $s = 1$ and $S^{worst}_{MSKP} = +\infty$;
2: **repeat**
3: sort items of the s-th scenario according to Eq. (5.4);
4: set $\overline{c} = c$ ($\overline{c}$ is the residual knapsack capacity);
5: set $i = \text{random}\{1, \ldots, n\}$ and $I' = \{i\}$;
6: Set $S^s_{MSKP} = 0$;
7: **repeat**
8: **if** $(w_i \leq \overline{c})$ **then**
9: set $\overline{c} = \overline{c} - w_i$;
10: set $S^s_{MSKP} = S^s_{MSKP} + p^s_i$;
11: **end if**
12: set $i = \text{random}\{1, \ldots, n\} \setminus \{I'\}$ and $I' = I' \cup \{i\}$;
13: **until** $(\overline{c} = 0$ or $|I'| = n)$;
14: **if** $(S^{worst}_{MSKP} > S^s_{MSKP})$ **then**
15: set $worst = s$ and $S^{worst}_{MSKP} = S^s_{MSKP}$;
16: **end if**
increment s;
17: **until** $(s > m)$;
18: **return** S^{worst}_{MSKP}.

Algorithm 15 illustrates the main steps of the greedy constructive procedure (GCP) when, it is applied for a given instance P_{MSKP}. Its main loop (cf., lines from 2 to 17) is repeated until all scenarios are treated. For each given scenario s, $s = 1, \ldots, m$, the internal loop (cf., lines from 7 to 13) is repeated until no item can be placed in the knapsack regarding the current scenario. More specifically, at each scenario s, an item is selected according to the order established by Equation (5.4). In the current scenario, the selected item is placed in the knapsack if, its weight does not exceed the residual knapsack capacity $\overline{c}$ after the selection of the other items. The process is repeated until no other items can be added in the knapsack. The algorithm stops and exits with a feasible solution (noted S^{worst}_{MSKP}) associated with the scenario (for the instance P_{MSKP}) achieving the worst objective value.

5.2.2.2 Improving a solution at hand

Aldouri and Hifi [5] proposed a Diversified Method (DM) for the MSKP. DM is based on the perturbation phase and restoring/exploring phase. The first phase mimics a diversification search and the second phase considers a current feasible solution and tries to improve it by using an intensification.

Herein, we consider the intensification phase that will be explained in what follows and the diversification phase is replaced with the path relinking as explained in Section 5.2.3.

The exploring phase can be viewed as a neighborhood search that is based on a series of exchanges between items belonging or not to the current solution. The neighborhood solution is used for improving the quality of the current solution. It also consists in finding a good strategy for generating a final solution x_p, $p \geq 1$, localized in the neighborhood solution. The process applies a series of moves for constructing a series of feasible solutions

$x_1, x_2, \ldots, x_p$.

1	1	1	1	...	1	0	0	...	0	0
1	2	3	4	...	i_c^s-1	i_c^s	i_c^s+1	...	$n-1$	n

Figure 5.1: Binary representation of MSKP's solution for the s-th scenario.

For an instance of MSKP, a standard binary representation scheme is an obvious choice since it represents the underlying 0-1 non-negative variables (Figure 5.1 shows a simple representation of MKSP's solution). Herein, a feasible solution $\bar{x}$, for each scenario, is such that $\sum w_j \bar{x}_j \leq c$. It is obtained by running one time GCP, which provides a solution with a *critical element* (noted i_c^s for each scenario s, $s \in S$, as described in Figure 5.1). All variables x_i such that $i < i_c^s$ are fixed to one and all other variables such that $i \geq i_c^s$ are fixed to zero.

Therefore, for each scenario $s \in S$, by moving i_c^s either to the left hand of i_c^s or its right hand, one can build a set of feasible solutions characterizing its associated neighbors. On the other hand, Figure 5.2 illustrates an eventual MSKP's partial feasible solution whenever some variables remain free in the current solution according to the current scenario s, $s \in S$.

1	1	1	1	...	0	0	$\star$	...	$\star$	$\star$
1	2	3	4	...	i_c^s-1	i_c^s	i_c^s+1	...	$n-1$	n

Figure 5.2: Configuration of MSKP's partial feasible solution associated to the s-th scenario. The symbol $\star$ denotes the free item

Let Δ be a nonnegative integer representing a *depth parameter*. The depth parameter allows us to jump backward from i_c^s to $i_c^s - 1$ and so on until the total depth Δ has been explored. In addition, it allows the search to explore the left and right neighborhood. Such a strategy tries to make combinations of different solutions never visited before, and it tries to modify choice rules to encourage move combinations. Certainly, one can also consider a series of critical elements that can be associated with the current solution; it means that an extended definition of the critical element can be used for more diversifying the search process. Our limited computational results showed that the simple exploring strategy was sufficient in providing good behavior of the proposed algorithm, especially on benchmark instances available in the literature for the MSKP.

Hence, for each scenario s, $s \in S$, and with reference to a feasible solution, several manner can be explored for making a move regarding the critical item. In our study, we followed the standard strategy which works as follows:

1. Let $x_{i_c^s-1}$ be the $(i_c^s - 1)$-*th* item of the current solution such that $s \in \{1, \ldots, m\}$;
2. Set $x_{i_c^s-1}$ equal to zero, maintain $x_{i_c^s-2}, \ldots, x_1$ to one, and
3. Complete the partial configuration by applying GCP.

On the one hand, by jumping from i_c^s-1 to $i_c^s-\Delta$, a set of new solutions is provided; that is called the *left neighborhood* of the solution at hand. A new search is then applied to improving the quality of the solution, i.e., the solution with the best value in the neighborhood is

selected. On the other hand, by applying the same process with a look-ahead, from $i_c^s + 1$ to $i_c^s + \Delta$ and by fixing the current item to one, we obtain the *right neighborhood*; that is provided by applying the following equivalent process:

1. Let $x_{i_c^s+1}$ be the $(i_c^s + 1)$-*th* item of the current solution such that $s \in \{1, \ldots, m\}$;
2. Set $x_{i_c^s+1}$ to one, make free some variable $x_i,\ i \leq i_c^s - 1$, and set $x_i,\ \forall i \geq i_c^s + 2$, to free;
3. Complete the partial configuration by applying GCP.

Hence, both *left* and *right* neighborhoods realize the current neighborhood of the solution at hand, namely S_{MSKP}. In fact, it mimics a truncated branch-and-bound where only a first level is considered; in this case, each successor of the current critical element has the following form: the current critical element is fixed either to zero or one and some of its successor, limited to the Δ-th element, is fixed either to 1 or 0, respectively. Of course, such a manner can be viewed as an extension of the "core problem" used for the single knapsack problem applied herein for the MSKP.

Algorithm 16 An Exploring Procedure for the MSKP: EP

Input. A feasible solution S_{MSKP} and a depth parameter Δ.

Output. An improved local optimum S'_{MSKP}.

1: **for** $s \in \{1, \ldots, m\}$ **do**
2: Set $\rho = 0$;
3: **while** $\big(\rho < \Delta\big)$ **do**
4: Increment ρ, let S_{MSKP}^{worst} be the current solution, $I_1 = \{1, \ldots, i_c^{worst} - \rho - 1\},\ I_2 = I \setminus \{I_1 \cup \{i_c^s - \rho\}\}$;
5: Set $x_{i_c^{worst}-\rho} = 0$, $x_i = 1,\ i \in I_1$ and $x_i,\ \forall i \in I_2$, are free;
6: Call GCP for completing S_{MSKP}^{worst} and let S'_{MSKP} be the best solution found so far.
7: **end while**
8: Set $\rho = 0$;
9: **while** $\big(\rho < \Delta\big)$ **do**
10: Increment ρ, set $x_{i_c^{worst}+\rho} = 1$ and $\forall i \in I' := I \setminus \{i_c^{worst}\}$, set x_i to free;
11: Call GCP for solving MSKP with I' and let S'_{MSKP} be the best solution found so far.
12: **end while**
13: **end for**
14: **return** S'_{MSKP}.

Algorithm 16 describes the principle of the exploring search (also considered as an improved procedure) applied either to a *partial* or a *complete* feasible solution. Algorithm 16 starts with a feasible solution (noted S^s_{MSKP}) and by using a prefixed depth parameter Δ. The main loop (cf., lines from 1 to 13) describes the main steps of the exploring method: for each scenario $s \in S$, both internal loops (the first one from the line 3 to line 7 and the second one from the line 9 to line 12) the original MSKP's instance is reconsidered. In the first part (the internal loop from line 3 to line 7), the *left exploring phase* is applied for building the first part of the neighborhood. The current neighborhood contains the solutions with $x_{i_c^s-\rho}$ fixed to zero, where $\rho \in \{1, \ldots, \Delta\},\ \Delta \geq 1$. For each fixed ρ in the interval $\{1, \ldots, \Delta\}$, a new solution is provided by using GCP on the remaining sub-instance with its residual capacity. The second loop, line 9 to line 12, represents the *right exploring phase* that completes the

global neighborhood. In this case, $x_{i_c^s-\rho}$ is fixed to one, where $\rho \in \{1,\ldots,\Delta\}$, $\Delta \geq 1$; for each $\rho \in \{1,\ldots,\Delta\}$, a new solution is reached by applying CGP to the reduced instance. Finally, it stops (cf., line 14) with the best solution S'_{MSKP} found so far.

5.2.3 Path-Relinking

The use of the Path-Relinking (PR) within a GRASP procedure was originally proposed by Laguna and Marti [48]. Such a strategy was used for intensifying the search space of each given local solution at hand. Note that PR was first considered by Glover [32], where the aforementioned intensification strategy was used for exploring trajectories connecting some solutions, namely elite solutions, which have been obtained when applying either tabu or scatter search (cf., Glover *et al.* [33]). The aim of the combination is to build new solutions with high-quality by starting from one of these elite solutions (starting solution) and generating a path in the neighborhood space that leads towards another solution (guiding solution).

As described in Resende *et al.* [65], several strategies-based methods can be used in order to mimic the creation of the path that should be created between an initial solution and the target one. Among these strategies, one can run a PR by using a greedy swapping between elements of both solutions. Creating randomness on the generated solutions through the path can also be obtained by applying a swap between an element of the starting solution and another element randomly taken from the target solution. Truncated search and evolutionary strategies can also be used either for accelerating the search process or enhancing the quality of the solutions (for more details, the reader can refer to Resende *et al.* [65]). Figure 5.3 explains two hypothetical paths (i.e., a sequence of moves) that connect the initial solution A to the target solution B.

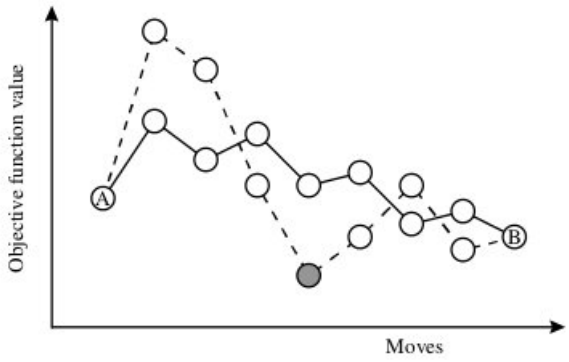

Figure 5.3: Path relinking

In this thesis, a new way of simulating the path between the starting solution and the target one is proposed. Let x and y be two solutions of the MSKP, where each of them is represented by the sets of selected items S_x and S_y, respectively. The greedy path relinking starts with x and iteratively tries to modify it to reach the guiding solution y. The transformation is achieved by applying some swaps between un-selected items of x with items belonging to y. The items selected from both solutions represent a *partial solution* of MSKP; that is a part of an intermediate solution in the path between both solutions.

Let $S_{y\setminus x}$ be the subset of items belonging to y and not in x. Observe that $S_{y\setminus x}$ contains some elements belonging to y and any local search applied to the elements out of $S_{y\setminus x}$ induces a solution belonging either to a path between x and y or a several paths where the current solution contains all items of $S_{y\setminus x}$. In this case, one way toward y is similar to building a series of solutions that can take different ways. Indeed, finding neighbor solutions leading y can be summarized as follows:

1. Set $k = 0$ and let $x = p_k(x, y)$ be an initialing solution $y = p_r(x, y)$ be the guiding solution, where $r \geq 1$.
2. Determine intermediate solution $p_{k+1}(x, y)$, set $k = k + 1$ and $x = p_k(x, y)$;
3. Repeat step (2) until performing r neighbor-solutions close to y.

Observe that the above process tends to introduce some diversifications according to some selected items belonging to y. Such a principle can obviously focalise on the convergence toward y, but at the same time attempting to provide solutions of good qualities. Indeed, almost all of our use of the greedy search that restricts the neighborhood, we propose to determine intermediate solutions by completing the partial solution associated with $S_{y\setminus x}$ by applying the exploring procedure (cf. Algorithm 16). We have already mentioned that the constructive greedy procedure GCP is able either to generate a starting solution or a complementary one according to a partial solution. Hence, the exploring phase is applied whenever the complementary solution is built by using GCP.

Algorithm 17 Path-Relinking (PR)

Input. Initial solution x_P and guiding solution y_P

Output. Best solution $S^{\star}_{MSKP}$.

Compute $S_{y_P\setminus x_P}$;
$f^{\star} \leftarrow \max\{f(x_P), f(y_P)\}$;
$S^{\star}_{MSKP} \leftarrow \text{argmin}\{f(x_P), f(y_P)\}$;
$iter \leftarrow 1$ and $z_P \leftarrow x_P$;
while ($S_{y_P\setminus x_P} \neq \emptyset$ and $iter \leq r$) **do**
  Let z_P^{par} be the *partial solution* associated to $S_{y_P\setminus x_P}$.
  Call GCP (Algo. 15) and let z_P^{comp} be the *complementary solution* associated to $S_{y_P\setminus x_P}$.
  Set $z_P \leftarrow z_P^{par} \cup z_P^{comp}$.
  Improve z_P by applying EP (Algo. 16).
  if ($f(z_P) > f(S^{\star}_{MSKP})$) **then**
    $f^{\star} \leftarrow f(z_P)$;
    $S^{\star}_{MSKP} \leftarrow z_P$;
  end if
  $x_P \leftarrow z_P$;
end while
return $S^{\star}_{MSKP}$;

Algorithm 17 illustrates the path-relinking procedure between the pair of solutions x_P (starting solution) and y_P (guiding solution). The procedure starts by computing the symmetric difference between both solutions by favoring the items belonging to the target solution y_P. The paths corresponding to of solutions are generated by linking x_P and y_P. The best

solution $S^{\star}_{MSKP}$ in the generated paths is always returned by the algorithm. The main loop (cf., lines from 5 to 15) is repeated until the best solution $S^{\star}_{MSKP}$ is reached. At each step, the algorithm builds a partial solution associated to $S_{y_P \setminus x_P}$ and calls the greedy constructive procedure (GCP) for completing the current partial solution. The provided solution $z_P^{par} \cup z_P^{comp}$ is improved by applying the *exploring phase* (EP). Finally, the best solution achieving the smallest cost, the one minimizing $f(z_P)$, is returned. The main loop stops whenever the final solution y_P is obtained and the number of explored solutions is reached. Note that, our preliminary results (cf. Section 5.4) were conducted by setting r to $n - |S_{y \setminus x}|$, which means that we give a change for each item out of the set $S_{y \setminus x}$.

5.2.4 An overview of the combined method

Algorithm 18 summarizes the main steps of the combined method (noted CM). CM's input is an instance of MSKP and its output is the best solution $x^{\star}$ found. Line 1 uses GRASP that generates a population of solutions, where P is ordered in decreasing order of their profits. Because, CM uses a runtime limit as the stopping criteria, we then fixed the size of the population to $\max\{m, 10\}$. Next, a random selection of a solution is made in line 4, which represents the initial solution improved by the exploring procedure (EP) at line 5.

Algorithm 18 Combined Method (CM) for the MSKP

Input. An instance of MSKP.

Output. Best solution $x^{\star}$.

1: Apply GRASP as a local search and let $P = \{x^1, \ldots, x^s, \ldots, x^{\alpha}\}$ be a population of n different solutions (in decreasing order).
2: $iter \leftarrow 1$;
3: **while** ($iter \leq \alpha$ and the runtime limit is not performed) **do**
4: Let $y \leftarrow random\{x^1, \ldots, x^s, \ldots, x^{\alpha}\}$.
5: $y \leftarrow EP(x)$.
6: Randomly select x from P.
7: Apply $PR(x, y)$ and let z be the best solution found.
8: **if** ($f(z) > f(x^{\alpha})$) **then**
9: $x^k \leftarrow$ closest solution to y in P with $f(z) > f(x^k)$.
10: Replace x^k with y and resort P.
11: **end if**
12: **if** (the runtime limit is not performed and $iter =\mid P \mid$) **then**
13: $iter \leftarrow 1$.
14: **else** Increment $iter$.
15: **end if**
16: **end while**
17: $x^{\star} \leftarrow x^1$;
18: return $x^{\star}$;

Line 6 makes a random selection from the population P and line 7 applies the path-relinking for building the intermediate solution z. From line 8 to line 11, the population replacement is made. The population of solutions is updated whenever the new obtained solution is better than the best solution of the population (x^1) or best of the worst solution (x^{α}). The algorithm iterates until performing the stopping criterion.

5.3 Experimental part

Herein, we evaluate the effectiveness of the proposed method on benchmark instances taken from the literature (Pinto *et al.* [60]). These instances were generated following the scheme proposed in Taniguchi *et al.* [71]. It is composed of two groups of instances: a group containing instances with two scenarios and a second one containing instances with more than two scenarios. For the first group, the running time is limited to five seconds (as used in the literature) and for the second group, the running time varies. The results reached by the proposed method are compared to those reached by the best methods available in the literature (cf., Aldouri and Hifi [8]). The method was coded in C++ and run on the Intel Pentium Core i7 with 2.1 GHz.

5.3.1 The first group: instances with two scenarios

There are two types of instances: weakly and strongly correlated instances. For each type, three sets can be distinguished, that depend on the capacity of the knapsack c. This capacity is set to be equal to $\delta \times \sum_{i=1}^{n} w_i$, where δ is equal to 0.25, 0.5, or 0.75. For each pair of (I, δ), there are fifteen instances, where for the weakly (respectively strongly) correlated instances, the set of items I varies in the discrete interval $\in \{5000, 7000, 10000, 13000, 15000, 18000, 20000\}$ (respectively, $\in \{500, 1000, 4000, 5000, 7000, 10000\}$).

Weakly correlated instance				Strongly correlated instance			
I	δ	TSHM	CM	I	δ	TSHM	CM
5000	0.25	896065	**896065**	500	0.25	87420.93	87420.93
7000		1253427.10	**1253427.13**	1000		174608.87	174608.87
10000		1792893.05	**1792893.33**	4000		700080	700080
13000		2331663.00	**2331662.80**	5000		875093.33	875093.33
15000		2688441.80	**2688441.47**	7000		1225526.67	1225526.67
18000		3225075.69	**3225075.80**	10000		1750600	1750600
20000		3585803.05	**3585803.13**				
5000	0.5	1632861.00	**1632861.07**	500	0.5	160187.2	160187.2
7000		2284892.73	**2284892.80**	1000		320780	320780
10000		3263828.29	**3263828.33**	4000		1282466.67	1282466.67
13000		4243496.30	**4243496.47**	5000		1603799.07	1603799.07
15000		4896679.28	**4896679.33**	7000		2244420	2244420
18000		5875560.85	**5875561.20**	10000		3206393.33	3206393.33
20000		6528235.15	**6528235.40**				
5000	0.75	2335621.85	**2335621.93**	500	0.75	230566.33	230566.33
7000		3268653.87	**3268653.93**	1000		461363.8	461363.8
10000		4669023.49	**4669023.53**	4000		1847028.87	1847028.87
13000		6072678.10	**6072678.13**	5000		2309173.33	2309173.33
15000		7003992.36	**7003992.53**	7000		3231062.13	3231062.13
18000		8406039.09	**8406039.13**	10000		4615606.67	4615606.67
20000		9338817.10	**9338817.93**				
Average		4075892.77	**4075892.87**			1462565.4	1462565.4

Table 5.1: Performance of CM vs TSHM on the benchmark instances with two scenarios

Table 5.1 illustrates the solution value realized by the combined method on the weakly (respectively strongly) correlated instances. The solutions obtained are compared to the best solutions taken from the Two-Stage Hybrid Method (abbreviated TSHM) of Aldouri and Hifi [8]. Columns 1 and 2 (respectively, columns 5 and 6) show the instance information: I is the number of items, and δ the value of the parameter used for calculating the capacity c of that instances knapsack. Column 3 (respectively, column 7) displays the best average

solution achieved by TSHM. Finally, the average CM's solutions are shown in column 4 (respectively column 8), and we considered ten trials for each instance. According to these trails, column 4 (respectively column 8) shows the average results of the solution values reached by these trials and those displayed in Table 5.3 (respectively, Table 5.4), where, we cited to one instance among fifteen in these Tables for two types of instances.

From Table 5.1, we observe that

1. For the weakly correlated instances, CM outperforms TSHM. Indeed, one can observe that the total average value achieved by CM exceeds the TSHM value by about 0.1.
2. For the strongly correlated instances, CM matches all optimal solutions.

5.3.2 The second group: instances with more than two scenarios

These instances are composed of three groups which are divided according to the number of scenarios m (belonging to the discrete interval $\{100, 500, 1000\}$. Each group contains four subgroups of instances, where each subgroup depends on the number of items n in the interval $\{1000, 5000, 10000, 20000\}$ and, for each interval $\{n, m\}$, there exist ten instances.

compared to the best solutions taken from the Two-Stage Hybrid Method (abbreviated TSHM) of Aldouri and Hifi *et al.* [8]

Table 5.2 reports the solution values reached by CM, compared to the Two-Stage Hybrid Method (abbreviated TSHM) of Aldouri and Hifi *et al.* [8]. Column 1 tallies the variation of the runtime limit used, columns 2 and 3 show the instance information, column 4 displays the best average solution values reached by TSHM. Finally, the average solution values achieved by CM are give in column 5 and, we considered ten trials for each instance. According to these trails, column 5 shows the average results of the solution values reached by these trials and those displayed in Table 5.5, where, we cited to one instance among ten in this Table.

From Table 5.2, we observe what follows:

- For the first runtime (60 seconds) the CM obtains better average solutions than those reached by TSHM. On the other hand, increasing the runtime (either to 300 or 600 seconds) increases the average solution quality.
- CMs' solution values are better than those published by the best methods available in the literature. Indeed, with the first running time (60 seconds), the average solution obtained is equal to 2696634.98 and its becomes more interesting (2696736.42) for the second running time and the third running time (2696806.22).

5.4 Conclusion

In this chapter, we proposed a combined approach based on greedy randomized adaptive search procedure (GRASP) and path-relinking for the max-min knapsack problem with multi-scenarios. The method uses a GRASP procedure in order to generate a starting solution and improves it by using an exploring phase. In order to mimic the path-relinking strategy, we proposed a new searching process in order to diversify the search space. The

CPU	n	m	TSHM	CM
	1000	100	299796.2	299796.5
		500	299992.1	299992.5
		1000	300558.5	300559.9
	5000	100	1494615.2	1494615.5
		500	1502726.8	1502826.5
		1000	1497515.8	1497616
60 sec	10000	100	2992543.2	2992543.7
		500	2995590	2995726.9
		1000	2998982	2999132
	20000	100	5992839	5992839.4
		500	5989642.5	5989649.1
		1000	5994321.8	5994321.8
		Average	2696593.59	2696634.98
	1000	100	299796.6	299797
		500	299990.1	299990.4
		1000	300562.1	300562.5
	5000	100	1494616	1494616.3
		500	1502844	1502845.3
		1000	1497683.8	1497685.3
300 sec	10000	100	2992546.1	2992546.3
		500	2995730.4	2995731
		1000	2999781	2999806.8
	20000	100	5992839.8	5992840.1
		500	5990002	5990010.8
		1000	5994382.5	5994405.2
		Average	2696731.20	2696736.42
	1000	100	299796.8	299797.3
		500	299991.5	299991.5
		1000	300563.7	300565.8
	5000	100	1494616.5	1494616.9
		500	1502847.5	1502850.5
		1000	1497814.2	1497816.8
600 sec	10000	100	2992546.7	2992546.7
		500	2995758.4	2995758.8
		1000	3000061.2	3000085.4
	20000	100	5992841	5992841.2
		500	5990024.6	5990024.6
		1000	5994771	5994779.1
		Average	2696802.76	2696806.22

Table 5.2: Performance of CM vs TSHM on the benchmark instances with multi-scenarios

experimental part shows that the proposed method yields good-quality solutions and adds and improvement of the several existing solutions available in the literature. This work has been presented at the International Conference on Computers & Industrial Engineering CIE47-2017 [6].

Instances	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	Best	Average
1_5000_0.25	896276	896277	896275	896277	896276	896277	896277	896278	896277	896277	896278	896276.7
1_7000_0.25	1254479	1254481	1254481	1254481	1254481	1254481	1254481	1254483	1254482	1254481	1254483	1254481.1
1_10000_0.25	1790610	1790615	1790614	1790612	1790611	1790617	1790615	1790615	1790615	1790618	1790618	1790614.2
1_13000_0.25	2337840	2337841	2337842	2337844	2337841	2337846	2337846	2337846	2337847	2337847	2337847	2337844
1_15000_0.25	2695181	2695182	2695183	2695182	2695187	2695179	2695181	2695187	2695184	2695184	2695187	2695183
1_18000_0.25	3220212	3220211	3220213	3220214	3220213	3220215	3220216	3220220	3220220	3220219	3220220	3220215.3
1_20000_0.25	3584219	3584220	3584222	3584220	3584221	3584228	3584228	3584229	3584229	3584227	3584229	3584224.3
1_5000_0.5	1635223	1635225	1635226	1635227	1635224	1635227	1635230	1635227	1635228	1635231	1635231	1635226.8
1_7000_0.5	2286991	2286990	2286993	2286994	2286995	2286999	2286997	2286999	2286995	2286998	2286999	2286995.1
1_10000_0.5	3264861	3264863	3264864	3264867	3264868	3264868	3264868	3264868	3264868	3264868	3264868	3264866.3
1_13000_0.5	4240216	4240215	4240216	4240218	4240218	4240218	4240220	4240222	4240221	4240222	4240222	4240218.6
1_15000_0.5	4900929	4900930	4900928	4900932	4900933	4900934	4900934	4900934	4900935	4900934	4900935	4900932.3
1_18000_0.5	5877318	5877320	5877319	5877320	5877322	5877323	5877325	5877324	5877325	5877324	5877325	5877322
1_20000_0.5	6521170	6521171	6521172	6521175	6521177	6521178	6521182	6521180	6521183	6521181	6521183	6521176.9
1_5000_0.75	2333590	2333589	2333596	2333594	2333596	2333597	2333597	2333594	2333595	2333597	2333597	2333594.5
1_7000_0.75	3267201	3267202	3267201	3267205	3267204	3267205	3267205	3267208	3267208	3267208	3267208	3267204.7
1_10000_0.75	4672640	4672641	4672647	4672647	4672642	4672646	4672649	4672650	4672651	4672652	4672652	4672646.5
1_13000_0.75	6081410	6081411	6081412	6081414	6081417	6081418	6081416	6081419	6081419	6081419	6081419	6081415.5
1_15000_0.75	7005050	7005050	7005051	7005052	7005060	7005061	7005055	7005060	7005061	7005061	7005061	7005056.1
1_18000_0.75	8406295	8406297	8406298	8406299	8406299	8406300	8406301	8406302	8406302	8406302	8406302	8406299.5
1_20000_0.75	9338240	9338241	9338242	9338241	9338245	9338245	9338247	9338249	9338248	9338249	9338249	9338244.7

Table 5.3: The quality of the solutions reached by ten trails of CM for the two scenarios (weakly correlated instances)

Instances	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	Best	Average
1_500_0.25	88000	88000	88000	88000	88000	88000	88000	88000	88000	88000	88000	88000
1_1000_0.25	173400	173400	173400	173400	173400	173400	173400	173400	173400	173410	173410	173401
1_4000_0.25	701100	701100	701100	701100	701100	701100	701100	701100	701100	701100	701100	701100
1_5000_0.25	872300	872300	872300	872300	872300	872300	872300	872300	872300	872300	872300	872300
1_7000_0.25	1222300	1222300	1222300	1222300	1222300	1222300	1222300	1222300	1222300	1222300	1222300	1222300
1_10000_0.25	1745600	1745600	1745600	1745600	1745600	1745600	1745600	1745600	1745600	1745600	1745600	1745600
1_500_0.5	158900	158900	158900	158900	158900	158900	158900	158900	158900	158900	158900	158900
1_1000_0.5	320700	320700	320700	320700	320700	320700	320700	320700	320700	320700	320700	320700
1_4000_0.5	1283900	1283900	1283900	1283900	1283900	1283900	1283900	1283900	1283900	1283900	1283900	1283900
1_5000_0.5	1603800	1603800	1603800	1603800	1603800	1603800	1603800	1603800	1603800	1603800	1603800	1603800
1_7000_0.5	2246800	2246800	2246800	2246800	2246800	2246800	2246800	2246800	2246800	2246800	2246800	2246800
1_10000_0.5	3207800	3207800	3207800	3207800	3207800	3207800	3207800	3207800	3207800	3207800	3207800	3207800
1_500_0.75	230600	230600	230600	230600	230600	230600	230600	230600	230600	230600	230600	230600
1_1000_0.75	460400	460400	460400	460400	460400	460400	460400	460400	460400	460400	460400	460400
1_4000_0.75	1843400	1843400	1843400	1843400	1843400	1843400	1843400	1843400	1843400	1843400	1843400	1843400
1_5000_0.75	2308000	2308000	2308000	2308000	2308000	2308000	2308000	2308000	2308000	2308000	2308000	2308000
1_7000_0.75	3224900	3224900	3224900	3224900	3224900	3224900	3224900	3224900	3224900	3224900	3224900	3224900
1_10000_0.75	4613200	4613200	4613200	4613200	4613200	4613200	4613200	4613200	4613200	4613200	4613200	4613200

Table 5.4: The quality of the solutions reached by ten trails of CM for the two scenarios (strongly correlated instances)

						60s						
Instances	**t1**	**t2**	**t3**	**t4**	**t5**	**t6**	**t7**	**t8**	**t9**	**t10**	**Best**	**Average**
1000_100_1	297235	297236	297237	297239	297237	297239	297236	297237	297237	297239	297239	297237.2
1000_500_1	297441	297441	297440	297437	297440	297440	297439	297439	297439	297441	297441	297439.7
1000_1000_1	301998	302001	302001	301999	302003	302001	302004	301999	301992	302012	302012	302001
5000_100_1	1506650	1506651	1506650	1506651	1506649	1506650	1506649	1506649	1506653	1506649	1506654	1506650.1
5000_500_1	1508143	1508144	1508145	1508148	1508141	1508140	1508145	1508142	1508151	1508151	1508151	1508145
5000_1000_1	1499986	1499986	1499986	1499986	1499986	1499986	1499986	1499986	1499986	1499986	1499986	1499986
10000_100_1	2984203	2984603	2984600	2984602	2984604	2984600	2984601	2984604	2984605	2984605	2984605	2984562.7
10000_500_1	3004846	3004844	3004820	3004844	3004838	3004848	3004848	3004848	3004848	3004847	3004848	3004843.1
10000_1000_1	3002002	3002000	3002007	3002007	3002006	3002007	3002007	3002000	3002005	3002007	3002007	3002004.8
20000_100_1	5994593	5994593	5994597	5994596	5994594	5994595	5994594	5994593	5994594	5994597	5994597	5994594.6
20000_500_1	5986623	5986623	5986630	5986634	5986633	5987638	5987638	5987638	5986635	5986636	5987638	5986932.8
20000_1000_1	5988513	5988513	5988513	5988513	5988513	5988513	5988513	5988513	5988513	5988513	5988513	5988513
						300s						
Instances	**t1**	**t2**	**t3**	**t4**	**t5**	**t6**	**t7**	**t8**	**t9**	**t10**	**Best**	**Average**
1000_100_1	297237	297238	297237	297239	297237	297239	297236	297237	297237	297239	297239	297237.6
1000_500_1	297438	297439	297440	297437	297440	297440	297439	297439	297439	297441	297441	297439.2
1000_1000_1	302007	302008	302008	302007	302009	302009	302002	302005	302008	302009	302009	302007.2
5000_100_1	1506650	1506651	1506653	1506651	1506654	1506653	1506653	1506653	1506653	1506654	1506654	1506652.5
5000_500_1	1508148	1508150	1508155	1508155	1508151	1508147	1508115	1508151	1508150	1508155	1508155	1508147.7
5000_1000_1	1500011	1500010	1500015	1500009	1500010	1500017	1500009	1500008	1500014	1500019	1500019	1500012.2
10000_100_1	2984603	2984603	2984600	2984604	2984604	2984603	2984601	2984604	2984604	2984604	2984604	2984603
10000_500_1	3004846	3004844	3004862	3004867	3004795	3004865	3004875	3004875	3004875	3004875	3004875	3004857.9
10000_1000_1	3002140	3002135	3002145	3002145	3002150	3002150	3002150	3002148	3002150	3002150	3002150	3002146.3
20000_100_1	5994597	5994597	5994597	5994597	5994597	5994597	5994597	5994597	5994597	5994597	5994597	5994597
20000_500_1	5988467	5988467	5988467	5988467	5988467	5988467	5988467	5988467	5988467	5988467	5988467	5988467
20000_1000_1	5988963	5988963	5988963	5988963	5988963	5988963	5988963	5988963	5988963	5988963	5988963	5988963
						600s						
Instances	**t1**	**t2**	**t3**	**t4**	**t5**	**t6**	**t7**	**t8**	**t9**	**t10**	**Best**	**Average**
1000_100_1	297237	297238	297237	297240	297240	297235	297240	297238	297237	297240	297240	297238.2
1000_500_1	297441	297441	297440	297442	297440	297440	297341	297340	297343	297343	297343	297401.1
1000_1000_1	302008	302009	302010	302011	302010	302011	302010	302013	302013	302013	302013	302010.8
5000_100_1	1506652	1506651	1506653	1506654	1506654	1506653	1506653	1506653	1506653	1506654	1506654	1506653
5000_500_1	1508148	1508150	1508155	1508160	1508155	1508147	1508160	1508158	1508158	1508160	1508160	1508155.1
5000_1000_1	1500016	1500038	1500035	1500045	1500046	1500046	1500046	1500039	1500042	1500046	1500046	1500039.9
10000_100_1	2984603	2984603	2984600	2984604	2984604	2984600	2984601	2984604	2984602	2984601	2984604	2984602.2
10000_500_1	3004908	3004908	3004908	3004908	3004908	3004908	3004908	3004908	3004908	3004908	3004908	3004908
10000_1000_1	3002169	3002169	3002169	3002169	3002169	3002169	3002169	3002169	3002169	3002169	3002169	3002169
20000_100_1	5994597	5994597	5994597	5994597	5994597	5994597	5994597	5994597	5994597	5994597	5994597	5994597
20000_500_1	5988595	5988595	5988595	5988595	5988595	5988595	5988595	5988595	5988595	5988595	5988477	5988595
20000_1000_1	5989459	5989459	5989459	5989459	5989459	5989459	5989459	5989459	5989459	5989459	5989459	5989459

Table 5.5: The quality of the solutions reached by ten trails of CM for the multi-scenarios

Chapter 6

General Conclusions

Conclusions

In this thesis, we studied the multi-scenarios max-min knapsack problem (MSKP), a combinatorial optimization problem. The MSKP received increasing research attention in recent years, both due to their capabilities to accommodate a wide range of practical applications (include engineering design, health care and financial management). In this work, the MSKP was tackled by using three approximate methods (heuristic and metaheuristic), where the goal is to efficiently solve large-scale problem instances: good solution's quality within moderate average runtime.

In chapter 3, the MSKP was tackled by using a hybrid reactive search, where a series of neighborhood search techniques were combined in order to achieve solutions with "high quality". The used method can be viewed as a two phase approach, where the first phase intensifies the search space and the second one diversifies the search space (by using both destroying and repairing strategies). In the first phase, the starting solution yielded by applying a greedy procedure and improved by an exploring step; that is based on exchanges between items belonging or not to the current solution. Such a diversification uses a black-box solver in order to escape from a series of local optima. Finally, both phases were embedded in an iterative search until satisfied the stopping condition. The proposed method was evaluated on a set of benchmark instances taken from the literature, where its obtained results are compared to those reached by the best methods available in the literature. The results showed that the method remained competitive and it was able to provide better solutions than those already published ones.

In chapter 4, a two-stage hybrid approach has been proposed. The proposed algorithm is composed of three main steps. The first step yields a feasible solution by using a greedy procedure. The second step tries to combine an initial population of solutions in order enhances the quality of the solutions (at hand). The third and last step tries to improve the solutions achieved by applying a series of moves (for constructing a series of feasible solutions around a series of critical elements), and exploring both left and right neighborhoods of critical elements. Computational results showed that the proposed approach yielded high-quality solutions and was able to improve most solutions available in the literature.

In chapter 5, a combined approach based upon greedy randomized adaptive search procedure (GRASP) and path-relinking has been presented. The method consists of two phases: the first phase tries to generate a starting solution by using a GRASP and improves the current solution by applying an intensification strategy. The second phase attempts to diversify the search space by applying the path-relinking strategy. The method uses some strategies already developed in chapters 3 and 4 for enhancing the accuracy of the method. Finally, the proposed method was evaluated on a set of instances of the literature and its achieved

results were compared to those obtained by the best methods available in the literature. The results showed that the method was able to reproduce several solutions of the literature while improving some of these solutions by consuming a lowest average runtime.

Perspective and future work

The work presented in this thesis could be adapted to deal with several various combinatorial optimization problems, especially when tackling knapsack problem types, like quadratic knapsack and quadratic multiple knapsack problems. We strongly believe that the proposed methods can be beneficial for developing efficient heuristics for such problems. On the other hand, we think that is also interesting to consider other methods in order to develop guided neighborhood search algorithms, such as particle swarm optimization, tabu search, and genetic algorithms.

Bibliography

[1] R. K. Ahuja, Ö. Ergun, J. B. Orlin, and A. P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1):75–102, 2002. (Cited in page 37.)

[2] R. K. Ahuja, Ö. Ergun, J. B. Orlin, and A. P. Punnen. Very large-scale neighborhood search: Theory, algorithms, and applications. *Handbook of Approximation Algorithms and Metaheuristics*, 10, 2007. (Cited in page 37.)

[3] H. Aissi. *Approximation et résolution des versions min-max et min-max regret de problémes d'optimisation combinatoire*. PhD thesis, Université Paris-Dauphine, 2005. (Cited in page 36.)

[4] H. Aissi, C. Bazgan, and D. Vanderpooten. Min–max and min–max regret versions of combinatorial optimization problems: A survey. *European journal of operational research*, 197(2):427–438, 2009. (Cited in page 36.)

[5] T. Aldouri and M. Hifi. A diversified method for the multi-scenarios max-min knapsack problem. In *Control, Decision and Information Technologies (CoDIT), 2016 International Conference on*, pages 355–359. IEEE, 2016. (Cited in pages 72, 77, 83 and 84.)

[6] T. Aldouri and M. Hifi. A combined grasp and path-relinking for the multi-scenario max-min knapsack problem. In *Computers & Industrial Enginering (CIE47), 2017 International Conference on*. IEEE, 2017. (Cited in pages iii and 92.)

[7] T. Aldouri and M. Hifi. A hybrid reactive search for solving the max-minknapsack problem with multi-scenarios. *International Journal of Computers and Applications*, 40(1):1–13, 2018. (Cited in pages iii, 54, 73 and 74.)

[8] T. Aldouri and M. Hifi. A two-stage hybrid approach for solving the max-minknapsack problem with multi-scenarios. *Intelligent Engineering Informatics*, under revision 2017. (Cited in pages iii, 77, 90 and 91.)

[9] E. Balas, S. Ceria, and G. Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0–1 programs. *Mathematical programming*, 58(1-3):295–324, 1993. (Cited in pages 8 and 27.)

[10] E. Balas and E. Zemel. An algorithm for large zero-one knapsack problems. *operations Research*, 28(5):1130–1154, 1980. (Cited in pages 5 and 24.)

[11] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329, 1998. (Cited in pages 16 and 34.)

[12] R. Bellmam. Dynamic programming princeton university press. *Princeton, NJ*, 1957. (Cited in pages 4, 9, 21 and 28.)

[13] P. Björklund, P. Värbrand, and D. Yuan. A column generation method for spatial tdma scheduling in ad hoc networks. *Ad hoc networks*, 2(4):405–418, 2004. (Cited in pages 15 and 33.)

[14] R. D. Carr, H. J. Greenberg, W. E. Hart, G. Konjevod, E. Lauer, H. Lin, and C. A. Phillips. Robust optimization of contaminant sensor placement for community water systems. *Mathematical Programming*, 107(1-2):337–356, 2006. (Cited in page 36.)

[15] N. Cherfi and M. Hifi. A column generation method for the multiple-choice multi-dimensional knapsack problem. *Computational Optimization and Applications*, 46(1):51–73, 2010. (Cited in pages 15 and 33.)

[16] P. C. Chu and J. E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of heuristics*, 4(1):63–86, 1998. (Cited in pages 5 and 24.)

[17] H. Crowder, E. L. Johnson, and M. Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31(5):803–834, 1983. (Cited in pages 12 and 29.)

[18] G. B. Dantzig. Discrete-variable extremum problems. *Operations research*, 5(2):266–288, 1957. (Cited in pages 5, 6, 7, 21, 25, 26, 39 and 83.)

[19] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations research*, 8(1):101–111, 1960. (Cited in pages 16 and 33.)

[20] S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani. *Algorithms*. McGraw-Hill, Inc., 2006. (Cited in pages 10 and 28.)

[21] J. Desrosiers and M. E. Lübbecke. Branch-price-and-cut algorithms. *Wiley encyclopedia of operations research and management science*, 2011. (Cited in pages 16 and 34.)

[22] D. Z. Du and P. M. (Eds.) Pardalos. *Minimax and applications*, volume 4. Springer Science & Business Media, 2013. (Cited in page 36.)

[23] D. Fayard and G. Plateau. An algorithm for the solution of the 0–1 knapsack problem. *Computing*, 28(3):269–287, 1982. (Cited in pages 5 and 24.)

[24] T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, 1989. (Cited in page 82.)

[25] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133, 1995. (Cited in page 82.)

[26] A. Fréville and G. Plateau. The 0-1 bidimensional knapsack problem: toward an efficient high-level primitive tool. *Journal of Heuristics*, 2(2):147–167, 1996. (Cited in pages 5 and 24.)

[27] M. R. Garey and D. S. Johson. Computer and inractability: a guide to the theory of np-completeness. *W.H. Freeman*, 1979. (Cited in pages 12 and 30.)

[28] S. I. Gass and C. M. Harris. *Encyclopedia of Operations Research and Management Science: Centennial Edition*. Springer Science & Business Media, 2001. (Cited in pages 12, 30 and 65.)

[29] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations research*, 9(6):849–859, 1961. (Cited in pages 4, 15 and 21.)

[30] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting stock problem-part ii. *Operations research*, 11(6):863–888, 1963. (Cited in pages 4, 15, 21 and 32.)

[31] P. C. Gilmore and R. E. Gomory. The theory and computation of knapsack functions. *Operations Research*, 14(6):1045–1074, 1966. (Cited in pages 4 and 21.)

[32] F. Glover. Tabu search and adaptive memory programing—advances, applications and challenges. pages 1–75. Springer, 1997. (Cited in page 87.)

[33] F. Glover, M. Laguna, and R. Marti. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 29(3):653–684, 2000. (Cited in page 87.)

[34] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278, 1958. (Cited in pages 8 and 27.)

[35] S. Hanafi, R. Mansi, C. Wilbaut, and A. Fréville. Hybrid approaches for the two-scenario max–min knapsack problem. *International Transactions in Operational Research*, 19(3):353–378, 2012. (Cited in pages 14 and 32.)

[36] M. Hifi. Dynamic programming and hill-climbing techniques for constrained two-dimensional cutting stock problems. *Journal of combinatorial optimization*, 8(1):65–84, 2004. (Cited in pages 9 and 28.)

[37] M. Hifi, H. Mhalla, and M. Michaphy. The multi-scenario knapsack problem: An adaptive search algorithm. *World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 4(11):1772–1775, 2010. (Cited in pages 13, 31 and 65.)

[38] M. Hifi and M. Michrafy. A reactive local search-based algorithm for the disjunctively constrained knapsack problem. *Journal of the Operational Research Society*, 57(6):718–726, 2006. (Cited in pages 38 and 42.)

[39] M. Hifi and C. Roucairol. Approximate and exact algorithms for constrained (un) weighted two-dimensional two-staged cutting stock problems. *Journal of combinatorial optimization*, 5(4):465–494, 2001. (Cited in pages 4 and 21.)

[40] M. Hifi, S. Sadfi, and A. Sbihi. An efficient algorithm for the knapsack sharing problem. *Computational Optimization and Applications*, 23(1):27–45, 2002. (Cited in page 40.)

[41] M. Hifi, S. Sadfi, and A. Sbihi. An efficient algorithm for the knapsack sharing problem. *Computational Optimization and Applications*, 23(1):27–45, 2002. (Cited in page 70.)

[42] E. Horowitz and S. Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM (JACM)*, 21(2):277–292, 1974. (Cited in pages 5, 10, 22, 24, 28 and 29.)

[43] H. Iida. A note on the max-min 0-1 knapsack problem. *Journal of Combinatorial Optimization*, 3(1):89–94, 1999. (Cited in pages 12, 14, 30, 31, 36, 66 and 82.)

[44] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972. (Cited in pages 5 and 24.)

[45] H. Kellerer, U. Pferschy, and D. Pisinger. *Introduction to NP-Completeness of knapsack problems*. Springer, 2004. (Cited in pages 10, 12, 28, 30, 36, 38, 66 and 82.)

[46] P. J. Kolesar. A branch and bound algorithm for the knapsack problem. *Management Science*, 13(9):723–735, 1967. (Cited in pages 10 and 28.)

[47] P. Kouvelis and G. Yu. *Robust discrete optimization and its applications*. Kluwer Academic Publishers, Dordrecht, 1997. (Cited in pages 14 and 31.)

[48] M. Laguna and R. Marti. Grasp and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11(1):44–52, 1999. (Cited in page 87.)

[49] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pages 497–520, 1960. (Cited in pages 10 and 28.)

[50] J. K. Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 2003. (Cited in page 37.)

[51] J. H. Lorie and L. J. Savage. Three problems in rationing capital. *The journal of business*, 28(4):229–239, 1955. (Cited in pages 4 and 21.)

[52] M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005. (Cited in pages 16 and 34.)

[53] S. Martello, D. Pisinger, and P. Toth. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management Science*, 45(3):414–424, 1999. (Cited in pages 5 and 24.)

[54] S. Martello and P. Toth. Knapsack problems: algorithms and computer implementations. 1990. (Cited in pages 3, 5, 12, 22, 24, 30, 36, 66 and 82.)

[55] M. S. Maschler and E. Sh. Zamir. Sh.(2013): Game theory. (Cited in page 36.)

[56] G. B. Mathews. On the partition of numbers. *Proceedings of the London Mathematical Society*, 1(1):486–490, 1896. (Cited in pages 3 and 21.)

[57] A. Mehrotra and M. A. Trick. A column generation approach for graph coloring. *informs Journal on Computing*, 8(4):344–354, 1996. (Cited in pages 15 and 32.)

[58] J. E. Mitchell. Branch-and-cut algorithms for combinatorial optimization problems. *Handbook of applied optimization*, pages 65–77, 2002. (Cited in pages 12, 29 and 30.)

[59] C. C. Petersen. Computational experience with variants of the balas algorithm applied to the selection of r&d projects. *Management Science*, 13(9):736–750, 1967. (Cited in pages 4 and 21.)

[60] T. Pinto, C. Alves, R. Mansi, and J. Valério de Carvalho. Solving the multiscenario max-min knapsack problem exactly with column generation and branch-and-bound. *Mathematical Problems in Engineering*, 2015. (Cited in pages 12, 14, 15, 30, 32, 33, 36, 44, 49, 50, 52, 66, 72, 73, 75, 82 and 90.)

[61] D. Pisinger. *Algorithms for Knapsack Problem*. PhD thesis, Ph. D thesis, Dept. of Computer Science, University of Copenhagen, 1995. (Cited in pages 4 and 21.)

[62] D. Pisinger. An exact algorithm for large multiple knapsack problems. *European Journal of Operational Research*, 114(3):528–541, 1999. (Cited in pages 5 and 24.)

[63] D. Pisinger and S. Ropke. Large neighborhood search. In *Handbook of metaheuristics*, pages 399–419. Springer, 2010. (Cited in pages 37 and 38.)

[64] D. Pisinger and P. Toth. Knapsack problems. In *Handbook of combinatorial optimization*, pages 299–428. Springer, 1998. (Cited in pages 3 and 22.)

[65] Mauricio GC Resende, Rafael Martí, Micael Gallego, and Abraham Duarte. Grasp and path relinking for the max–min diversity problem. *Computers & Operations Research*, 37(3):498–508, 2010. (Cited in page 87.)

[66] D. Sanjoy, P. Christos, and V. Umesh. *Algorithms*. McGraw-Hill, Inc., 2006. (Cited in pages 7 and 26.)

[67] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming—CP98*, pages 417–431. Springer, 1998. (Cited in pages 37 and 38.)

[68] W. Shih. A branch and bound method for the multiconstraint zero-one knapsack problem. *Journal of the Operational Research Society*, 30(4):369–378, 1979. (Cited in pages 4 and 21.)

[69] X. Song, R. Lewis, J. Thompson, and Y. Wu. An incomplete m-exchange algorithm for solving the large-scale multi-scenario knapsack problem. *Computers & Operations Research*, 39(9):1988–2000, 2012. (Cited in pages 13 and 31.)

[70] R. Steuer. *Multiple Criteria Optimization: Theory, Computations, and Application*. John Wiley & Sons, Inc., New York, 1986. (Cited in pages 12, 30 and 65.)

[71] F. Taniguchi, T. Yamada, and S. Kataoka. Heuristic and exact algorithms for the max–min optimization of the multi-scenario knapsack problem. *Computers & Operations Research*, 35(6):2034–2048, 2008. (Cited in pages 14, 31, 32 and 90.)

[72] F. Taniguchi, T. Yamada, and S. Kataoka. A virtual pegging approach to the max–min optimization of the bi-criteria knapsack problem. *International Journal of Computer Mathematics*, 86(5):779–793, 2009. (Cited in pages 14, 32, 44 and 72.)

[73] P. Toth. Dynamic programming algorithms for the zero-one knapsack problem. *Computing*, 25(1):29–45, 1980. (Cited in pages 9 and 28.)

[74] J. M. van Den Akker, J. A. Hoogeveen, and S. L. van de Velde. Parallel machine scheduling by column generation. *Operations Research*, 47(6):862–872, 1999. (Cited in pages 16 and 34.)

[75] P. H. Vance, C. Barnhart, E. L. Johnson, and G. L. Nemhauser. Solving binary cutting stock problems by column generation and branch-and-bound. *Computational optimization and applications*, 3(2):111–130, 1994. (Cited in pages 15 and 32.)

[76] F. Vanderbeck. Computational study of a column generation algorithm for bin packing and cutting stock problems. *Mathematical Programming*, 86(3):565–594, 1999. (Cited in pages 15, 16, 32 and 34.)

[77] J. P. Watson, W. E. Hart, and R. Murray. Formulation and optimization of robust sensor placement problems for contaminant warning systems. In *Eighth Annual Water Distribution Systems Analysis Symposium (WDSA), August 27-30, 2006*, pages 1–13, 2008. (Cited in page 36.)

[78] M. M. Wiecek, V. Y. Blouin, G. M. Fadel, A. Engau, B. J. Hunt, and V. Singh. Multi-scenario multi-criteria optimization with applications in engineering design. In *Multi-objective Programming and Goal Programming*, pages 283–298. Springer, 2009. (Cited in pages 12, 30 and 36.)

[79] G. Yu. On the max-min 0-1 knapsack problem with robust optimization applications. *Operations Research*, 44(2):407–415, 1996. (Cited in pages 12, 14, 30, 31, 36, 65, 66 and 81.)