THE CARDINALITY CONSTRAINED MULTIPLE KNAPSACK PROBLEM

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MURAT ASLAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
INDUSTRIAL ENGINEERING

NOVEMBER 2008

Approval of the thesis:

THE CARDINALITY CONSTRAINED MULTIPLE KNAPSACK PROBLEM

submitted by **Murat ASLAN** in partial fulfillment of the requirements for the degree of **Master of Science in Industrial Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Nur Evin Özdemirel _____
Head of Department, **Industrial Engineering**

Prof. Dr. Meral Azizoğlu _____
Supervisor, **Industrial Engineering Dept., METU**

Examining Committee Members:

Prof. Dr. Ömer Kırca _____
Industrial Engineering Dept., METU

Prof. Dr. Meral Azizoğlu _____
Industrial Engineering Dept., METU

Asst. Prof. Dr. F. Can Çetinkaya _____
Industrial Engineering Dept., Çankaya University

Asst. Prof. Dr. Sedef Meral _____
Industrial Engineering Dept., METU

Asst. Prof. Dr. Seçil Savaşaneril _____
Industrial Engineering Dept., METU

Date: 21.11.2008

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Murat ASLAN

Signature :

ABSTRACT

THE CARDINALITY CONSTRAINED MULTIPLE KNAPSACK PROBLEM

ASLAN, Murat
M.S., Department of Industrial Engineering
Supervisor : Prof. Dr. Meral AZİZOĞLU

November 2008, 61 pages

The classical multiple knapsack problem selects a set of items and assigns each to one of the knapsacks so as to maximize the total profit. The knapsacks have limited capacities. The cardinality constrained multiple knapsack problem assumes limits on the number of items that are to be put in each knapsack, as well. Despite many efforts on the classical multiple knapsack problem, the research on the cardinality constrained multiple knapsack problem is scarce.

In this study we consider the cardinality constrained multiple knapsack problem. We propose heuristic and optimization procedures that rely on the optimal solutions of the linear programming relaxation problem. Our computational results on the large-sized problem instances have shown the satisfactory performances of our algorithms.

Keywords: Cardinality Constrained Multiple Knapsack Problem, Linear Programming Relaxation, Optimization

ÖZ

SAYI KISITLI ÇOKLU SIRT ÇANTASI PROBLEMİ

ASLAN, Murat
Yüksek Lisans, Endüstri Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Meral AZİZOĞLU

Kasım 2008, 61 sayfa

Klasik çoklu sırt çantası problemi toplam kazancı en çoklayan parça kümesini seçer ve seçilen her parçayı sırt çantalarından birine atar. Sırt çantalarının sınırlı kapasiteleri vardır. Sayı kısıtlı çoklu sırt çantası problemi her bir sırt çantasına konan parça sayısında da kısıt olduğunu varsayar. Klasik çoklu sırt çantası problemi için pek çok çaba sarf edilmiş olsa da, sayı kısıtlı çoklu sırt çantası problemi üzerindeki araştırmalar sınırlıdır.

Bu çalışmada sayı kısıtlı çoklu sırt çantası problemini ele aldık. Doğrusal programlama gevşetmesi probleminin en iyi çözümlerine dayanan sezgisel ve eniyileme yöntemleri önerdik. Büyük boyutlu problemler üzerinde aldığımız deneysel sonuçlar yöntemlerimizin tatmin edici performansını göstermektedir.

Anahtar Kelimeler: Sayı Kısıtlı Çoklu Sırt Çantası Problemi, Doğrusal Programlama Gevşetimi, En İyileme

To my parents, Çağlayan and Yasemin

ACKNOWLEDGEMENTS

I am deeply grateful to my thesis supervisor Prof. Dr. Meral Azizoğlu for her efforts, guidance and support throughout the study. She not only guided me perfectly, but also encouraged me to perform better in the study.

I would like to thank jury members for their valuable contributions on the thesis.

I would like to thank to my family members: Hasan Aslan, Nuray Aslan, Çağlayan Aslan and Yasemin Saldanlı for their moral support.

I would like to thank my company, ASELSAN, for the support about thesis study permissions.

I would like to thank my manager Zafer Dokuzoğlu for allowing me to go for thesis study whenever I needed and covering my vacancy.

I would like to thank my dear friends Alper Taş and Güvenç Değirmenci for their invaluable help; Altay Emre Poyraz for the articles he supplied from the Library of Bilkent University and cheerful company; Şafak Baykal for her support and guidance; Zafer Yahşi and Tunç Taner Gürlek for their insight and wisdom; Banu Lokman, Eda Göksoy and Neslihan Özlü for their moral support, and my dear friends from the group of "Endustri Tayfasi" for their useful feedbacks and invaluable friendship.

And finally, thanks to TÜBİTAK for the scholarship, which is provided throughout my master study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Consider a capable sportsman who is good at playing several branches of sports including football, basketball, volleyball and tennis. The sportsman gets different utility from each sport branch. To play a game he has to take the corresponding ball having a specified volume. His sport bag has a limited capacity so that he cannot take all balls one at a time. His problem is to find the set of balls to take, hence the set of sports to play, so that the capacity of bag is not exceeded and the total utility is maximized. This optimization problem is known to be the Single Knapsack Problem (KP) in OR literature. If the sportsman has more than one sport bag then the associated problem is referred to as Multiple Knapsack Problem (MKP). Additionally, if there are limits on the number of balls that each sports bag can take, then the associated problem is the Cardinality Constrained Multiple Knapsack Problem (kMKP).

The part selection problem in automated manufacturing systems is analogous to the kMKP. Assume there are n part types that are to be selected for processing by m Computerized Numerically Controlled (CNC) machines where machine i has a limited capacity of C_i time units. Part type j has a processing requirement of w_j time units and there is a profit p_j money units if selected for processing. The problem is to select a subset of the part types so as to maximize the total profit. The cardinality constraints of the KMKP may correspond to the tool magazine capacities of the CNC machines in the part selection problem. The number of setups that can be made on each machine may also define the cardinality constraints.

The knapsack problems have several practical application areas in the manufacturing and service industries. One application area for the KP is the hitch-hiker problem that is stated in Martello and Toth (1990). The hitch-hiker has to fill up his knapsack among various possible objects so as to maximize his comfort. In the problem p_i is the measure of his comfort taken from object i, w_j is its size, and C is the size (volume) of the knapsack.

The cutting stock problem is another area where the knapsack problems find their applications. The cutting stock problem can be stated as follows: Assume you work in a paper mill and you have a number of rolls of paper of fixed width waiting to be cut, yet different customers want different numbers of rolls of various-sized widths. The problem is to find a way to cut the rolls so that the scrap is minimized. Solving this problem to optimality can be economically significant: a difference of 1% for a modern paper machine can be worth more than 1 million US$ per annum (Wikipedia, free encyclopedia). The KP is used as a subproblem in solving the cutting stock problems with column generation technique.

In the business environments, the KP is used for investment planning. Consider an investor who has a certain amount of money (C) and a list of possible investment alternatives. Each investment alternative has a capital required (w_j) and an expected return (p_j) over a planning period. The problem is to select the set of investment alternatives so that the budget is not exceeded and the total return is maximized. Clearly, such an investment problem associates to the KP.

Despite its simple structure, the solution of the KP is not that easy. Each item selection is defined by a binary decision variable that takes value 1 if the item is selected and 0 otherwise. A simple approach would be to examine all possible arrangements of the binary variables. For n items problem there are 2^n such binary variables. As Martello and Toth (1990) state, on a computer that examines one billion variables per second, it takes 30 years to enumerate the possible solutions when there are 60 items. If you increase the number of items to 65 then it will take ten centuries.

The knapsack problems have been studied for several decades as they are the simplest maximization problems. Mathews (1897) shows how several constraints may be compiled into one single knapsack constraint. What Mathews (1897) had done is now called as “reduction of an integer program to KP”.

Garey and Johnson (1979) focus comprehensively on the theory of intractability and NP-completeness of the KP. They show that the KP is NP-hard in the ordinary sense. The problem can be solved by pseudopolynomial time dynamic programming algorithm. On the other hand, the Multiple Knapsack Problem is strongly NP-hard, hence there cannot exist polynomial, even pseudopolynomial algorithm, to find optimal solutions. The cardinality constrained multiple knapsack problem is strongly NP-hard, as all generalizations of the MKP.

The KP and MKP are widely studied in the literature. However, there are limited reported studies on the kMKP. Recognizing this fact, we introduce solution algorithms for the kMKP. We observe that the linear programming relaxation of the problem produces very satisfactory results and can be used as a basis in developing solution algorithms.

The rest of the thesis is organized as follows. In Chapter 2 we give the mathematical representation of the knapsack problems and review the related literature. Chapter 3 is the main body of our work where the heuristic algorithm and branch and bound algorithm are presented. In Chapter 4 we present the results of our computational experiments. In Chapter 5 we conclude by pointing our main results and possible future research directions.

CHAPTER 2

PROBLEM DEFINITION AND LITERATURE REVIEW

In this chapter, we first discuss several versions of the knapsack problem. We then review the literature on cardinality constrained single knapsack problem and multiple knapsack problem. Those two problems appear as special cases of the cardinality constrained multiple knapsack problem.

2.1 Problem Definition

2.1.1 Classical Knapsack Problem

The classical knapsack problem can be formally defined as follows: Consider a set of items $N=\{1,\ldots,n\}$ and a knapsack with capacity of C time units. Item j in set N has a profit p_j and capacity usage of w_j time units. The profit can be interpreted as the relative importance of the item or simply the benefit (like money) brought due to its selection. The capacity usage w_j for item j can as well be interpreted as the amount of space occupied by the item. In a production environment, w_j can represent the processing time, i.e., requirement, by item j. C is the knapsack capacity which may represent the amount of space available. In production environments, C may represent the time capacity, i.e, available machine time. One more important point is that w_j and C are in the same units. Usually p_j, w_j and C are assumed to be positive integer numbers. The classical knapsack problem, denoted as KP, is the simplest non-trivial integer programming model with binary variables, with a single constraint and n binary variables. The model is presented below.

(KP) maximize $\sum_{j=1}^{n} p_j x_j$

subject to $\sum_{j=1}^{n} w_j x_j \leq C$

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

The single knapsack problem is shown to be NP-hard in the ordinary sense (Garey and Johnson, 1979).

The studies in the literature assume that

$$w_j \leq C,\ j=1,\ldots,n \text{ and } \sum_{j=1}^{n} w_j > C$$

$w_j \leq C$ is required as otherwise item j would be trivially removed. $\sum_{j=1}^{n} w_j > C$ is assumed as otherwise a trivial solution that assigns all items would be found.

2.1.2 Subset Sum Problem (SSP)

If the profits and weights are identical for all items, i.e., $p_j=w_j$ for all j, we get the well known Subset Sum Problem (SSP). The SSP finds a subset of N items such that the corresponding total profit is maximized without exceeding the available capacity C. In production environments, the objective function can be interpreted as the used capacity which is to be maximized. The SSP is NP-hard in ordinary sense. (Martello and Toth, 1990)

2.1.3 Multiple Knapsack Problem (MKP)

The Multiple Knapsack Problem is a generalization of the single classical knapsack problem. In the MKP we are given a set of items $N = \{1,\ldots,n\}$ with profits p_j and weights w_j, $j=1,\ldots,n$ and set of knapsacks $M = \{1,\ldots,m\}$ with positive capacities C_i, $i=1,\ldots,m$. The MKP is NP-hard in the strong sense (Martello and Toth, 1990).

The MKP Model can be stated as

$$\text{maximize} \quad \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_{ij} \le C_i \qquad i=1,\ldots..,m$$

$$\sum_{i=1}^{m} x_{ij} \le 1 \qquad j=1,\ldots..,n$$

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

The following assumptions are made for the MKP in the literature.

1. Every item must fit to at least one of the knapsacks, i.e.,

$$w_{\max} \le C_{\max}$$

where $C_{max}=max\{C_1,C_2,\ldots C_m\}$ and $w_{max}=max\{w_1,w_2,\ldots.,w_n\}$ otherwise the associated items are trivially eliminated.

2. All knapsacks should take at least one item, i.e.,

$$w_{\min} \le C_{\min}$$

where $C_{min}=min\{C_1,C_2,\ldots C_m\}$ and $w_{min}=min\{w_1,w_2,\ldots.,w_n\}$ otherwise the associated knapsacks are eliminated.

3. Moreover, the trivial solutions that select all items should be avoided, i.e.,

$$\sum_{j=1}^{n} w_j > C_{\max}$$

otherwise all items would be put to the maximum capacity knapsack.

2.1.4 Cardinality Constrained Single Knapsack Problem

If there is a constraint on the number of items included in each knapsack, the associated problem is referred to as cardinality constrained single knapsack problem. When the selection of the item causes explicit handling, the solutions with a small number of larger items will be preferred to the one with large number of smaller items.

The number of items $\sum_{j=1}^{n} x_j$ can be included in the objective function so as to obtain a bi-objective knapsack problem, or $\sum_{j=1}^{n} x_j$ can be a constraint, i.e., $\sum_{j=1}^{n} x_j \leq K$. The latter problem is a cardinality constrained single knapsack problem, which is denoted as kKP in the OR literature. The kKP model is given below.

(KP) maximize $\sum_{j=1}^{n} p_j x_j$

subject to $\sum_{j=1}^{n} w_j x_j \leq C$

$$\sum_{j=1}^{n} x_j \leq K$$

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

Recall that the KP is NP-hard but pseudopolynomially solvable by a dynamic program. Caprara et al. (2000) show that the kKP is NP-hard in the ordinary sense, as well.

2.1.5 Cardinality Constrained Multiple Knapsack Problem

If the cardinality constrained single knapsack problem has multiple knapsacks then it is referred to as the cardinality constrained multiple knapsack problem (kMKP). That means each bag has a limit on the number of balls you put in the bag. In production environments, there can be limits on the number of jobs if each job requires a set-up, hence they can have upper bounds on the number of the items.

MKP is a special case of the kMKP with infinite knapsack cardinalities. MKP is strongly NP-hard, so is the kMKP.

The kMKP model is expressed below:

Let K_i be the maximum number of items that can be assigned to knapsack i. Then the kMKP model is expressed below.

$$\text{maximize} \quad \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_{ij} \le C_i \qquad i=1,\ldots..,m$$

$$\sum_{i=1}^{m} x_{ij} \le 1 \qquad j=1,\ldots..,n$$

$$\sum_{j=1}^{n} x_{ij} \le K_i \qquad i=1,\ldots\ldots,m$$

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

The model has mxn binary decision variables x_{ij} and $2m+n$ constraints

2.2 Literature Survey for Cardinality Constrained Multiple Knapsack Problem

The classical knapsack problem and its variations are studied enormously in the OR literature due to its simple structure and ability to model many industrial situations. The practical implications include but not limited to, capital budgeting, cargo loading, production planning or cutting stock cases.

We study the literature on the knapsack problems that are most closely related to our problem: namely single knapsack problem with cardinality constraints and multiple knapsack problem. For both problems, we make the survey in the chronological order.

For the classical single and multiple knapsack problems and their variations, we refer the reader to the book by Kelleler, Pferschy and Psinger (2004). In the book, the models and the associated solution algorithms are studied thoroughly.

2.2.1 Single Knapsack Problems with the Cardinality Constraint

Campello and Makulan (1987) are one of the first researchers who focused on the cardinality constrained single knapsack problem (kKP). They study the cardinality constrained linear programming relaxation of the knapsack problem (LPK-k). They introduce the following model:

$$Q(k) = \max \sum_{j=1}^{n} p_j x_j$$

subject to

$$\sum_{j=1}^{n} w_j x_j \le C$$

$$\sum_{j=1}^{n} x_j = K$$

$$0 \le x_j \le 1 \qquad j=1,\ldots,n$$

An $O(n^3)$ algorithm is developed by Campello and Makulan (1987) for solving the problem (LPK-k).

Dudzinski (1989) also deals with the cardinality constrained linear programming knapsack problem (LPK-k). To find an upper bound he presents a more relaxed model than the one introduced by Campello and Makulan (1987). Dudzinski (1989) revises the notation used in Campello and Maculan (1987) as follows,

$$Q(k) = \max \ (Kp_i + \sum_{j=1}^{n} (p_j - p_i)x_j)$$

subject to

$$\sum_{j=1}^{n} (w_j - w_i)x_j \le T - kw_i$$

$$\sum_{j=1}^{n} x_j = K$$

$$0 \le x_j \le 1 \qquad j=1,\ldots,n$$

Dudzinski (1989) improves the $O(n^3)$ algorithm, developed by Campello and Makulan (1987) and proposes $O(n^2)$ algorithm for the cardinality constrained linear programming knapsack problem (LPK-k) and obtains the following model LPK-i-k.

$$Q_i(k) = kp_i + \max \sum_{j=1}^{n} (p_j - p_i)x_j$$

subject to

$$\sum_{j=1}^{n} (w_j - w_i)x_j \le T - Kw_i$$

$$0 \le x_j \le 1 \qquad j=1,\ldots,n$$

$$Q(k) \le Q_i(k) \qquad \forall i \in N$$

Caprara et al. (2000) study the k-item Knapsack Problem (kKP) and exact k-item Knapsack Problem (E-kKP). The kKP can be formulated as KP with an additional constraint. The E-kKP is a variant of the kKP where the number of items must be exactly K, i.e., $\sum_{j=1}^{n} x_j = K$. Caprara et al. (2000) show that the kKP and E-kKP can be transformed to each other, hence any kKP instance can be solved using the methods developed for the E-kKP.

Caprara et al. (2000) develop a ½ approximation algorithm that runs in $O(n)$ by using the LP relaxation of the kKP. This algorithm is used by, Caprara et al. (2000) in developing a Polynomial Time Approximation Scheme. The scheme runs in $(O^{1/\varepsilon - 1})$ time and requires a linear space.

Finally in 2006, Mastrolilli and Hutter study the same problem and present a linear-storage polynomial time approximation scheme (PTAS) and a fully polynomial time approximation scheme (FPTAS). They use input rounding (arithmetic or geometric rounding) techniques and show that PTAS requires linear space and has a running time of $O(n+k(\log 1/\varepsilon)^{O(1/\varepsilon)})$. Hence it is superior to PTAS proposed by Caprara et al. (2000).

2.2.2 Literature Related With Multiple Knapsack Problems

Ingargiola and Korsh (1975) propose an algorithm for 0-1 Loading Problem in which ten random instances are solved with 15 items and 6 knapsacks. This paper was one of the first papers focusing on the Multiple Knapsack Problem. Hung and Fisc (1978) also focus on the Multiple Knapsack Problem and present Lagrangean and Surrogate relaxation techniques. A branch-and-bound algorithm that avoids the redundancy of the partial solutions is presented. They compare their results by those of Ingargiola and Korsh (1975) and show the superiority of their approach.

Martello and Toth (1981) propose a heuristic algorithm for the Multiple Knapsack Problem (MKP). They solve practical instances with up to 1000 items and 100 knapsacks and show that their solutions are satisfactory.

A Polynomial Time Approximation Scheme (PTAS) is provided by Murgolo (1987). Hochbaum and Shymoys (1988) give PTAS using the *dual* based approach where they convert the scheduling problem into a bin packing problem. Lawler et al. (1993) also propose a PTAS that uses the ideas from uniform multi-processor scheduling. The objective is to assign a set of jobs with given processing times to the machines of different speeds so as to minimize the makespan. The most recent PTAS is proposed by Chekuri and Khanna (2000).

CHAPTER 3

THE kMKP

The mathematical model of the kMKP is restated below for the ease of reference.

$$\text{maximize} \quad \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij}$$

$$\text{subject to} \quad \sum_{i=1}^{m} x_{ij} \le 1 \qquad j=1,\ldots..,n \qquad (1)$$

$$\sum_{j=1}^{n} w_j x_{ij} \le C_i \qquad i=1,\ldots..,m \qquad (2)$$

$$\sum_{j=1}^{n} x_{ij} \le K_i \qquad i=1,\ldots\ldots.,m \qquad (3)$$

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

where,

Objective : Maximizing the total profit

Parameters

p_j : profit of item j

w_j : weight (capacity usage) of item j

C_i : capacity of knapsack i

K_i : cardinality of knapsack i

Decision Variables

x_{ij} : the binary decision variable about whether item j is assigned to knapsack i or not.

$$x_{ij} = \begin{cases} 1 \text{ if item j is assigned to knapsack i} \\ 0 \text{ otherwise} \end{cases}$$

Constraint sets 1, 2 and 3 explain assignment, capacity and cardinality constraints respectively.

In this chapter, we present our approach to study the kMKP. We first discuss our bounding mechanism: upper bound and lower bound (heuristic), and then present our branch and bound algorithm.

3.1 Upper Bounds and Heuristic

In this section, we first present several upper bounds that are obtained through various relaxations of the problem. We then present our heuristic procedure that aims to find a satisfactory approximate solution. The heuristic solution is used as an initial feasible solution in our branch and bound algorithm, discussed in Section 3.2.

3.1.1 Upper Bounds

Recall that our problem has a maximization type objective function. This follows that any relaxation of the problem provides an upper bound on the optimal objective function value. In this study we use two types of relaxations: constraint relaxation and continuous relaxation.

Each of these relaxations is discussed below;

3.1.1.1 Constraint Relaxation

Our problem has three constraint sets: assignment, capacity and cardinality constraints. When any one of the capacity and cardinality constraint sets is removed, an optimal solution to the resulting problem provides an upper bound on the maximum total profit value. If the resulting solution, by chance satisfies the removed constraints, then the optimal solution is also optimal for the original problem. We now discuss each of these constraint relaxations.

Relaxation of the Capacity Constraints:

When the capacity constraints are removed, the resulting model can be stated as follows:

kMKP Model (Capacity Constraint Relaxed) :

maximize $\sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij}$

subject to $\sum_{i=1}^{m} x_{ij} \le 1 \qquad j=1,\ldots..,n$

$\sum_{j=1}^{n} x_{ij} \le K_i \qquad i=1,\ldots\ldots,m$

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

An optimal solution to this relaxed problem assigns a total of $Min\left(\sum_{i=1}^{m} K_i, n\right)$ items.

If $n \le \sum_{i=1}^{m} K_i$ then all n items will be assigned with a total profit of $\sum_{j=1}^{n} p_j$.

If $n > \sum_{i=1}^{m} K_i$ then the items having the maximum $\sum_{i=1}^{m} K_i$ profits will be assigned to m knapsacks. In such a case, the maximum total profit is $\sum_{j=1}^{R} p_{[j]}$ where $p_{[j]}$ is the j^{th} largest profit and $R=\sum_{i=1}^{m} K_i$. The overall upper bound is $\sum_{j=1}^{Min\{n,R\}} p_{[j]}$, and is denoted as UB_1.

Relaxation of the Cardinality Constraints:

When the cardinality constraints are removed, the resulting model is a classical multiple knapsack model that is stated below.

kMKP Model (Cardinality Constraint Relaxed) :

maximize $\sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij}$

subject to $\sum_{j=1}^{n} w_j x_{ij} \le C_i$ $i=1,\ldots..,m$

$\sum_{i=1}^{m} x_{ij} \le 1$ $j=1,\ldots..,n$

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

Recall that the multiple knapsack problem is strongly NP-hard. In place of using optimal solutions that are obtained in exponential time, a polynomial time upper bound can be used. An upper bound on the multiple knapsack problem is a valid upper bound for our problem. This is due to the fact that an upper bound for any relaxation of a maximization problem is a valid upper bound on the original problem.

In the literature, several upper bounds are proposed for the multiple knapsack problem. An optimal solution to the continuous relaxation of the problem (that replaces $x_{ij} \in \{0,1\}$ with $0 \le x_{ij} \le 1$) is an upper bound. This optimal solution is stated below:

Consider a single surrogate knapsack with capacity $\sum C_i$. Order the items in their nonincreasing p_j/w_j values, and assign them to the knapsack according to the order until the capacity is fully used or no item remains, whichever is observed first (Martello and Toth, 1990). Note that such an assignment ends up with at most one fractional item.

We refer the bound found by the continuous relaxation of the multiple knapsack problem as UB_2. Formally the upper bound, UB_2, is stated below:
Assume R satisfies,

$\sum_{j=1}^{R} w_{[j]} \le \sum C_i$ and $\sum_{j=1}^{R+1} w_{[j]} > \sum C_i$

$\sum C_i - \sum_{j=1}^{R} w_{[j]}$ is the total capacity remaining to the $(R+1)^{th}$ item.

The contribution of the $(R+1)^{th}$ item to the total profit is,

$$p_{[R+1]} \times \frac{\sum C_i - \sum_{j=1}^{R} w_{[j]}}{w_{[R+1]}}$$

The overall upper bound, UB_2, becomes

$$UB_2 = \sum_{j=1}^{R} p_{[j]} + p_{[R+1]} \times \frac{\sum C_i - \sum_{j=1}^{R} w_{[j]}}{w_{[R+1]}}$$

3.1.1.2 Linear Programming Relaxation (LPR)

When the constraints on the integrality of the assignment variables are removed the problem becomes

$$\text{maximize} \quad \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_{ij} \le C_i \qquad i=1,\ldots..,m$$

$$\sum_{i=1}^{m} x_{ij} \le 1 \qquad j=1,\ldots..,n$$

$$\sum_{j=1}^{n} x_{ij} \le K_i \qquad i=1,\ldots\ldots.,m$$

$$0 \le x_{ij} \le 1 \qquad i=1,\ldots\ldots,m \quad j=1,\ldots\ldots,n.$$

The above model is the Linear Programming Relaxation (LPR) of the original problem, and its optimal solution provides an upper bound on the maximum total profit value. To the best of our knowledge, no simple algorithm is available to solve the LPR of the problem. Hence an optimal solution, UB_3, can be found by using any commercial LP software.

We use the upper bounds in the following sequel: UB_1-UB_2-UB_3, i.e., from easiest to hardest. That is, we first evaluate the partial solutions by UB_1, if we cannot make any elimination then we compute UB_2. If UB_2 is not of any help then we compute the most powerful, however most costly upper bound, UB_3.

We benefit from the optimal LPR solution in finding a feasible solution to our problem as well. The associated heuristic is discussed in the next section.

3.1.2 Heuristic Procedure

In the heuristic procedure, we basically follow three steps that are stated below:

Step 1. Solve the LP relaxed problem, LPR.

Let x_{ij}^{LP} be the optimal LP assignment.

Assign item *j* to knapsack *i* only if $x_{ij}^{LP} = 1$ for all *i* and *j*.

Such an assignment results in a feasible solution as it satisfies the capacity and cardinality constraints and the integrality requirements. This value is a lower bound too but it is naive. We call it as naive lower bound (LB_1).

Step 2. Let *T* be the set of items that are not assigned in Step 1, and *S* be the set of assigned in Step 1, i.e., $S=\{1,2,\ldots..n\}/T$. Then, solve the following reduced problem.

$$\text{maximize} \quad \sum_{j\in T}\sum_{i} p_j x_{ij}$$

$$\text{subject to} \quad \sum_{j} w_j x_{ij} \le C_i - \sum_{j\in S} w_j x_{ij} \qquad i=1,\ldots..,m$$

$$\sum_{i=1}^{m} x_{ij} \le 1 \qquad j=1,\ldots..,n$$

$$\sum_{j} x_{ij} \le K_i - \sum_{j\in S} x_{ij} \qquad i=1,\ldots..,m$$

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

The reduced problem is strongly NP-hard, as well. However due to the exponential nature of the problem finding an optimal solution to the problem is much easier than finding an optimal solution to the original problem. Alternately the reduced problem can be solved heuristically through the following procedure.

Step 2.1 Assign the unassigned items according to a greedy procedure.

Step 2.2 Improve the solution obtained in Step 2.1 via interchanges. Step 2.1 is referred to as construction phase whereas Step 2.2 is an improvement phase.

3.1.2.1 Construction Phase

The items are sorted according to a priority rule and they are assigned to a knapsack according to an assignment rule. We use the following three priority rules for item ordering:

1. **Maximum p_j Rule:** The items are sorted in their nonincreasing order of p_j values. According to this rule, priority is given to the items having higher profit values.

2. **Minimum w_j Rule:** The items are sorted in their nondecreasing order of w_j values. According to this rule, priority is given to the items having lower capacity usages.

3. **Maximum p_j/w_j Rule:** The items are sorted in their nonincreasing order of p_j/w_j values. According to this rule, priority is given to the items having higher unit profit values.

We use the following four knapsack assignment rules:

1. **Maximum Remaining Capacity Rule:** The first item of the order is assigned to the knapsack having the maximum unused capacity.

2. **Minimum Capacity Used Rule:** The first item of the order is assigned to the knapsack having the minimum used capacity. As the capacities of the knapsacks are not necessarily identical, the knapsack with maximum remaining capacity is not necessarily the one having the minimum used capacity.

3. **Maximum Remaining Cardinality Rule:** The first item of the order is assigned to the knapsack having the maximum number of items that can be assigned.

4. **Minimum Number of Items Used Rule:** The first item of the order is assigned to the knapsack having the minimum number of items already assigned. As the cardinalities of the knapsacks are not necessarily identical, the knapsack with the maximum remaining cardinality is not necessarily the one having the minimum used cardinality.

Note that we have 3 priority rules and 4 assignment rules. This results in 12 solutions, some of which may be identical. We obtain all these solutions and select the one having the maximum total profit value. The selected solution is improved by the improvement phase discussed next.

3.1.2.2 Improvement Phase

The improvement phase looks for the possibility of increasing the maximum total profit by putting an unassigned item to a knapsack in place of an already assigned item. We let r be an unassigned item and s be an assigned item. We check whether putting item r in place of item s is feasible and $p_r > p_s$, i.e., the exchange increases the total profit value. Among all feasible pairs that increase the total profit, we select the one that leads to a maximum improvement. We terminate either all pairs lead to infeasible or nonimproving solutions.

Example:

We illustrate the heuristic via an example problem.

Table 3.1 gives the profit and weight values in a 12-item 2-knapsack problem

Table 3.1: The Profit and Weight Values for the 12-Item and 2-Knapsack Example Problem

Item	1	2	3	4	5	6	7	8	9	10	11	12
p_j	50	50	64	46	50	5	50	40	70	62	16	28
w_j	56	59	80	64	75	17	25	20	35	31	12	10

Capacities of knapsacks are C_1 = 190 and C_2= 170.

Cardinalities for knapsacks are K_1= 4 and K_2=4.

Figure 3.1 summarizes the problem environment. The representation is $\overset{profit}{\underset{weight}{ItemNumber}}$. For example, $\overset{50}{\underset{56}{1}}$ states that item 1 has a profit of 50 units and a weight of 56 units.

$\overset{50}{\underset{56}{1}}$ $\overset{50}{\underset{59}{2}}$ $\overset{64}{\underset{80}{3}}$ $\overset{46}{\underset{64}{4}}$ $\overset{50}{\underset{75}{5}}$ $\overset{5}{\underset{17}{6}}$ $\overset{50}{\underset{25}{7}}$ $\overset{40}{\underset{20}{8}}$ $\overset{70}{\underset{35}{9}}$ $\overset{62}{\underset{31}{10}}$ $\overset{16}{\underset{12}{11}}$ $\overset{28}{\underset{10}{12}}$

Figure 3.1: Summary of Problem in Item Representation.

We first construct the knapsack model for the example problem as

k-MKP Model

$$\text{maximize} \quad \sum_{i=1}^{2} \sum_{j=1}^{12} p_j x_{ij}$$

$$\text{subject to} \quad \sum_{j=1}^{12} w_j x_{1j} \le 190$$

$$\sum_{j=1}^{12} w_j x_{2j} \le 170$$

$$\sum_{i=1}^{2} x_{ij} \le 1 \qquad j=1,\ldots..,12$$

$$\sum_{j=1}^{12} x_{1j} \le 4$$

$$\sum_{j=1}^{12} x_{2j} \le 4$$

$$x_{ij} \in \{0,1\} \qquad i=1,2 \quad j=1,\ldots\ldots,12.$$

The optimal values of the decision variables are summarized in the below table.

Table 3.2: The Optimal Integer Programming Solution of the k-MKP

Variable	Value	Variable	Value
x_{11}	0	x_{17}	0
x_{21}	0	x_{27}	1
x_{12}	1	x_{18}	1
x_{22}	0	x_{28}	0
x_{13}	0	x_{19}	1
x_{23}	1	x_{29}	0
x_{14}	0	x_{110}	0
x_{24}	0	x_{210}	1
x_{15}	1	x_{111}	0
x_{25}	0	x_{211}	0
x_{16}	0	x_{112}	0
x_{26}	0	x_{212}	0

Next we find an approximate solution with the heuristic procedure.

Heuristic Method Solution:

Step 1. Solve the following LP relaxed problem, LPR.

maximize $\sum_{i=1}^{2}\sum_{j=1}^{12} p_j x_{ij}$

subject to $\sum_{j=1}^{12} w_j x_{1j} \le 190$

$\sum_{j=1}^{12} w_j x_{2j} \le 170$

$\sum_{i=1}^{2} x_{ij} \le 1 \qquad j=1,\ldots..,12$

$\sum_{j=1}^{12} x_{1j} \le 4$

$\sum_{j=1}^{12} x_{2j} \le 4$

$0 \le x_{ij} \le 1 \qquad i=1,2 \quad j=1,\ldots\ldots,12.$

Solution:

The optimal LP Relaxed solution has an objective function value of 428.89, and the solution values of the decision variables are tabulated below.

Table 3.3: LP-Relaxed Solution of the k-MKP

Variable	Value	Variable	Value
x_{11}	0	x_{17}	1
x_{21}	1	x_{27}	0
x_{12}	0	x_{18}	0.45
x_{22}	1	x_{28}	0.55
x_{13}	1	x_{19}	1
x_{23}	0	x_{29}	0
x_{14}	0	x_{110}	0
x_{24}	0	x_{210}	1
x_{15}	0.55	x_{111}	0
x_{25}	0.13	x_{211}	0
x_{16}	0	x_{112}	0
x_{26}	0	x_{212}	0,32

We assign item j to knapsack i if $x_{ij}^{LP}=1$

Accordingly, $x_{21}=1$, $x_{22}=1$, $x_{13}=1$, $x_{17}=1$, $x_{19=}1$ and $x_{210}=1$.

The remaining items 4, 5, 6, 8, 11 and 12 are either partially assigned or unassigned.

Step 2. Solve the following problem with the remaining items and reduced capacities and cardinalities

$$\text{maximize} \quad \sum_{j\in T}\sum_{i} p_j x_{ij}$$

$$\text{subject to} \quad \sum_{j} w_j x_{1j} \le 190 - \sum_{j\in S} w_j x_{1j}$$

$$\sum_{j} w_j x_{2j} \le 170 - \sum_{j\in S} w_j x_{2j}$$

$$\sum_{i=1}^{2} x_{ij} \le 1 \qquad j=1,\ldots..,12$$

$$\sum_{j} x_{1j} \le 4 - \sum_{j\in S} x_{1j}$$

$$\sum_{j} x_{ij} \le 4 - \sum_{j\in S} x_{2j}$$

$$x_{ij} \in \{0,1\}$$

where assigned items set $S=\{1,2,3,7,9,10\}$ and unassigned items set $T=\{4,5,6,8,11,12\}$. Next we solve the reduced problem with our heuristic assignment procedures.

Step 2.1 Assign the unassigned items according to a greedy procedure (construction phase)

This step is summarized in the tables below;

In each column the items are listed according to their p_j, $1/w_j$ *or* p_j/w_j values.

In case of a tie in p_j, $1/w_j$ and p_j/w_j orders we select according to $1/w_j$ and p_j and p_j orders, respectively. For further ties we use p_j/w_j and p_j/w_j and $1/w_j$, respectively.

The remaining items are assigned to the remaining capacities and cardinalities according to the assignment rules: maximum capacity left, minimum capacity used maximum cardinality left and minimum cardinality used.

Table 3.4: Greedy Assignment of Items According to Maximum p_j Rule

p_j	Max Capacity Left	Min Capacity Used	Max Cardinality Left	Min Cardinality Used
$\overset{50}{\underset{75}{5}}$ $\overset{46}{\underset{64}{4}}$ $\overset{40}{\underset{20}{8}}$	Rem C_1=50 Fixed: $\overset{64}{\underset{80}{3}}, \overset{50}{\underset{25}{7}}, \overset{70}{\underset{35}{9}}$ Assigned: $\overset{40}{\underset{20}{8}}$	Used C_1=140 Fixed: $\overset{64}{\underset{80}{3}}, \overset{50}{\underset{25}{7}}, \overset{70}{\underset{35}{9}}$ Assigned: $\overset{40}{\underset{20}{8}}$	Rem K_1=1 Fixed: $\overset{64}{\underset{80}{3}}, \overset{50}{\underset{25}{7}}, \overset{70}{\underset{35}{9}}$ Assigned: $\overset{40}{\underset{20}{8}}$	Used K_1=3 Fixed: $\overset{64}{\underset{80}{3}}, \overset{50}{\underset{25}{7}}, \overset{70}{\underset{35}{9}}$ Assigned: $\overset{40}{\underset{20}{8}}$
$\overset{28}{\underset{10}{12}}$ $\overset{16}{\underset{12}{11}}$ $\overset{5}{\underset{17}{6}}$	Rem C_2=24 Fixed: $\overset{50}{\underset{56}{1}}, \overset{50}{\underset{59}{2}}, \overset{62}{\underset{31}{10}}$ Assigned: $\overset{28}{\underset{10}{12}}$	Used C_2=146 Fixed: $\overset{50}{\underset{56}{1}}, \overset{50}{\underset{59}{2}}, \overset{62}{\underset{31}{10}}$ Assigned: $\overset{28}{\underset{10}{12}}$	Rem K_2=1 Fixed: $\overset{50}{\underset{56}{1}}, \overset{50}{\underset{59}{2}}, \overset{62}{\underset{31}{10}}$ Assigned: $\overset{28}{\underset{10}{12}}$	Used K_2=3 Fixed: $\overset{50}{\underset{56}{1}}, \overset{50}{\underset{59}{2}}, \overset{62}{\underset{31}{10}}$ Assigned: $\overset{28}{\underset{10}{12}}$
Obj Value	414 346(fixed) + 68 (greedy as)	414 346(fixed) + 68 (greedy as)	414 346(fixed) + 68 (greedy as)	414 346(fixed) + 68 (greedy as)

Table 3.5: Greedy Assignment of Items According to Maximum $1/w_j$ Rule

$1/w_j$	Max Capacity Left	Min Capacity Used	Max Cardinality Left	Min Cardinality Used
$\overset{28}{\underset{10}{12}}$ $\overset{16}{\underset{12}{11}}$ $\overset{5}{\underset{17}{6}}$ $\overset{40}{\underset{20}{8}}$ $\overset{46}{\underset{64}{4}}$ $\overset{50}{\underset{75}{5}}$	Rem C_1=50 Fixed: $\overset{64}{\underset{80}{3}}, \overset{50}{\underset{25}{7}}, \overset{70}{\underset{35}{9}}$ Assigned: $\overset{28}{\underset{10}{12}}$	Used C_1=140 Fixed: $\overset{64}{\underset{80}{3}}, \overset{50}{\underset{25}{7}}, \overset{70}{\underset{35}{9}}$ Assigned: $\overset{28}{\underset{10}{12}}$	Rem K_1=1 Fixed: $\overset{64}{\underset{80}{3}}, \overset{50}{\underset{25}{7}}, \overset{70}{\underset{35}{9}}$ Assigned: $\overset{28}{\underset{10}{12}}$	Used K_1=3 Fixed: $\overset{64}{\underset{80}{3}}, \overset{50}{\underset{25}{7}}, \overset{70}{\underset{35}{9}}$ Assigned: $\overset{28}{\underset{10}{12}}$
	Rem C_2=24 Fixed: $\overset{50}{\underset{56}{1}}, \overset{50}{\underset{59}{2}}, \overset{62}{\underset{31}{10}}$ Assigned: $\overset{16}{\underset{12}{11}}$	Used C_2=146 Fixed: $\overset{50}{\underset{56}{1}}, \overset{50}{\underset{59}{2}}, \overset{62}{\underset{31}{10}}$ Assigned: $\overset{16}{\underset{12}{11}}$	Rem K_2=1 Fixed: $\overset{50}{\underset{56}{1}}, \overset{50}{\underset{59}{2}}, \overset{62}{\underset{31}{10}}$ Assigned: $\overset{16}{\underset{12}{11}}$	Used K_2=3 Fixed: $\overset{50}{\underset{56}{1}}, \overset{50}{\underset{59}{2}}, \overset{62}{\underset{31}{10}}$ Assigned: $\overset{16}{\underset{12}{11}}$
Obj Value	390 346(fixed) + 44 (greedy as)	390 346(fixed) + 44 (greedy as)	390 346(fixed) + 44 (greedy as)	390 346(fixed) + 44 (greedy as)

Table 3.6: Greedy Assignment of Items According to Maximum p_j/w_j Rule

p_j/w_j	Max Capacity Left	Min Capacity Used	Max Cardinality Left	Min Cardinality Used
$\overset{28}{\underset{10}{12}}$ $\overset{40}{\underset{20}{8}}$ $\overset{16}{\underset{12}{11}}$ $\overset{46}{\underset{64}{4}}$ $\overset{50}{\underset{75}{5}}$ $\overset{5}{\underset{17}{6}}$	Rem C_1=50 Fixed: $\overset{64}{\underset{80}{3}}, \overset{50}{\underset{25}{7}}, \overset{70}{\underset{35}{9}}$ Assigned: $\overset{28}{\underset{10}{12}}$	Used C_1=140 Fixed: $\overset{64}{\underset{80}{3}}, \overset{50}{\underset{25}{7}}, \overset{70}{\underset{35}{9}}$ Assigned: $\overset{28}{\underset{10}{12}}$	Rem K_1=1 Fixed: $\overset{64}{\underset{80}{3}}, \overset{50}{\underset{25}{7}}, \overset{70}{\underset{35}{9}}$ Assigned: $\overset{28}{\underset{10}{12}}$	Used K_1=3 Fixed: $\overset{64}{\underset{80}{3}}, \overset{50}{\underset{25}{7}}, \overset{70}{\underset{35}{9}}$ Assigned: $\overset{28}{\underset{10}{12}}$
	Rem C_2=24 Fixed: $\overset{50}{\underset{56}{1}}, \overset{50}{\underset{59}{2}}, \overset{62}{\underset{31}{10}}$ Assigned: $\overset{40}{\underset{20}{8}}$	Used C_2=146 Fixed: $\overset{50}{\underset{56}{1}}, \overset{50}{\underset{59}{2}}, \overset{62}{\underset{31}{10}}$ Assigned: $\overset{40}{\underset{20}{8}}$	Rem K_2=1 Fixed: $\overset{50}{\underset{56}{1}}, \overset{50}{\underset{59}{2}}, \overset{62}{\underset{31}{10}}$ Assigned: $\overset{40}{\underset{20}{8}}$	Used K_2=3 Fixed: $\overset{50}{\underset{56}{1}}, \overset{50}{\underset{59}{2}}, \overset{62}{\underset{31}{10}}$ Assigned: $\overset{40}{\underset{20}{8}}$
Obj Value	414 346(fixed) + 68 (greedy as)	414 346(fixed) + 68 (greedy as)	414 346(fixed) + 68 (greedy as)	414 346(fixed) + 68 (greedy as)

We obtain all these solutions and select the one having the maximum total profit value. Accordingly, *Max*{ 414,414, 414,414,390,390,390,390, 414,414, 414,414} = 414 is the total profit value of the selected solution.

Step 2.2 Improve the solution obtained in Step 2.1 via interchanges.

Unassigned Set (*T*): The set consists of items which are not assigned to any knapsack in previous steps.

$$T = \{ \underset{75}{\overset{50}{5}}, \underset{64}{\overset{46}{4}}, \underset{17}{\overset{5}{6}}, \underset{12}{\overset{16}{11}} \}$$

Assigned Set (*S*): The set consists of items which are assigned to any knapsack in the previous steps.

$$S = \{ \underset{80}{\overset{64}{3}}, \underset{25}{\overset{50}{7}}, \underset{35}{\overset{70}{9}}, \underset{56}{\overset{50}{1}}, \underset{59}{\overset{50}{2}}, \underset{31}{\overset{62}{10}}, \underset{20}{\overset{40}{8}}, \underset{10}{\overset{28}{12}} \}$$

This phase looks for the opportunity of increasing the maximum total profit by putting an unassigned item to a knapsack in place of an already assigned item. We terminate when all pairs lead to infeasible or nonimproving solutions.

Pick the pair that causes the maximum improvement

$\underset{75}{\overset{50}{5}} \rightarrow \underset{10}{\overset{28}{12}}$ Violates the capacity constraint.

$\underset{64}{\overset{46}{4}} \rightarrow \underset{10}{\overset{28}{12}}$ Violates the capacity constraint.

$\underset{75}{\overset{50}{5}} \rightarrow \underset{20}{\overset{40}{8}}$ Violates the capacity constraint.

$\underset{64}{\overset{46}{4}} \rightarrow \underset{20}{\overset{40}{8}}$ Violates the capacity constraint.

Note that any pair does not lead to an improvement, hence the solution is not changed.

The optimal total profit value is 414. Note that construction phase, by chance, has ended up with the optimal solution. Hence, no improvement is possible in the succeeding phase.

3.2 Branch and Bound

Recall that, the cardinality constrained multiple knapsack problem is strongly NP-hard. This justifies the use of an implicit enumeration technique to find an exact solution. In this study, we design a branch and bound algorithm that uses the bounding mechanisms discussed so far.

Our branching scheme is based on the optimal solution of the LPR problem. We observe that the LPR produces very few continuous variables, hence base our branching scheme on these variables.

There are different branching methods that are used for the knapsack problem in the literature. According to the first scheme, the levels are represented by items. At each level, for a particular item, there are $m+1$ nodes where the first node represents the decision of not assigning the item to any knapsack, and each of the remaining nodes represents the assignment of an item to a particular knapsack. This branching scheme is proposed by Valerio (1996) and is figured below.

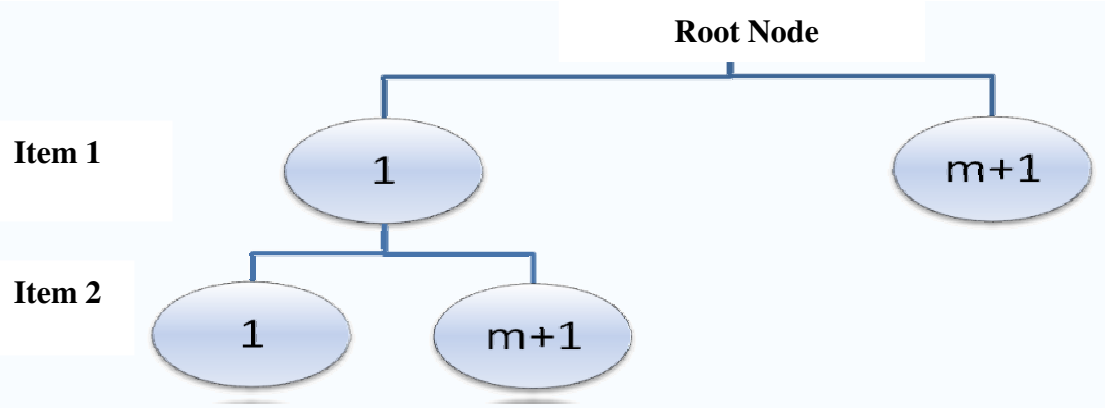

Figure 3.2: Knapsacks on Nodes Representation

At each level, $m+1$ decisions are considered. There are n levels. At the last level, all possible n^{m+1} decisions become available.

The second branching scheme first considers knapsack 1 and tries to fill the knapsack as much as possible, and then proceeds to the second knapsack, when the first knapsack is full, i.e., cannot take an additional item. The branching terminates whenever all knapsacks are considered.

The below figure gives the associated branching scheme. The scheme is proposed by Kelleler et al. (2004).

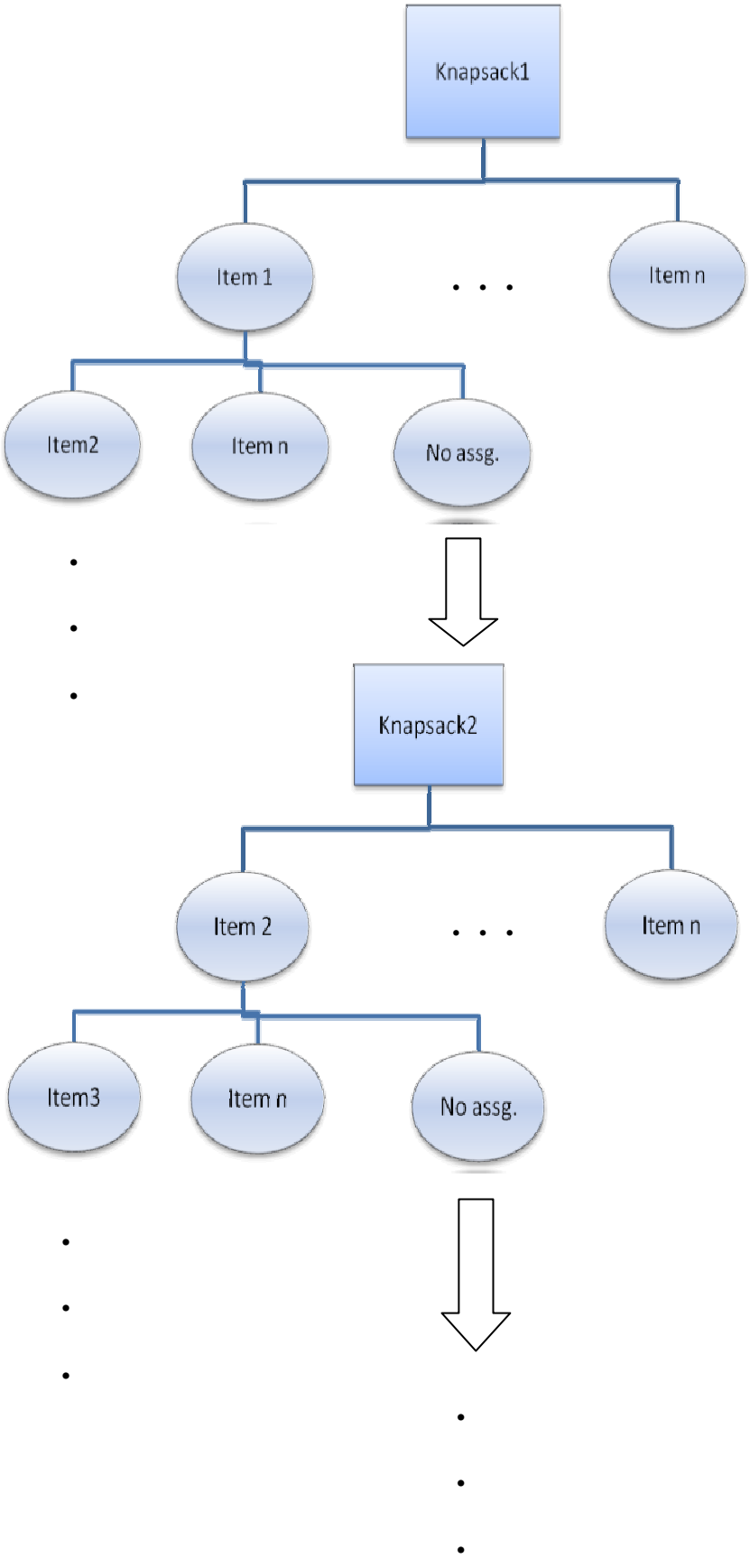

Figure 3.3: Items on Nodes B&B Representation

Instead of using the above alternatives we construct our branching method based on the fractional x_{ij} solutions. As mentioned before the optimal LPR solution produces very few continuous variables and this motivates us using this type of branching.

We use the result of the optimal LPR solution to define our branching structure. At every branch, we solve the LP problem and branch on a fractional variable of the LP solution. For the chosen fractional variable x_{ij} such that $0 < x_{ij} < 1$, we generate the following two subproblems.

Subproblem 1. $x_{ij} = 0$

Subproblem 2. $x_{ij} = 1$

The associated tree is given in the figure below.

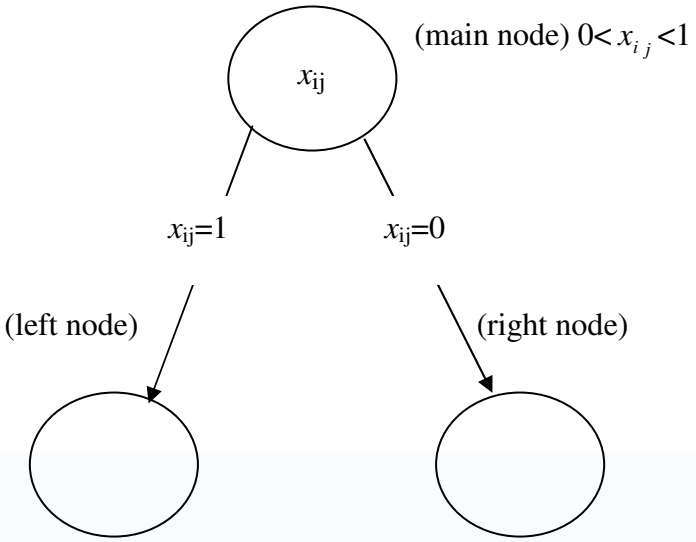

Figure 3.4: Branching Tree

We employ the following three strategies to select the fractional variable from which two subproblems are generated.

- Strategy 1 : Select the highest x_{ij} value
- Strategy 2 : Select the lowest x_{ij} value
- Strategy 3 : Select the x_{ij} value randomly

Strategies 1 and 2 expect that the optimal integer solution is close to the optimal LP relaxed solution. This forces big x_{ij} values to one (Strategy 1) and small x_{ij} values to zero (Strategy 2). Strategy 3, on the other hand, selects a fractional variable randomly, thereby looking for the effect of a solution found without any intuitive reasoning.

Example :

We illustrate the branching strategies via an example problem, having 8 items and 2 knapsacks. The data are tabulated in Table 3.7.

Table 3.7 : The Profit and Weight Values for the 8-Item 2-Knapsack Example Problem

Items	1	2	3	4	5	6	7	8
p_j	35	65	64	46	60	5	50	40
w_j	56	59	80	64	75	17	25	20

Capacity of knapsacks: C_1 = 65 and C_2= 120
Cardinalities for knapsacks: K_1= 2 and K_2=2

Above mentioned three strategies are based on the LP Relaxed solution of the kMKP. The LP Relaxed solution of the kMKP is given in below table.

Table 3.8: The Optimal LPR Solution of 8-Item 2-Knapsack Example Problem

Variable *	Value
x_{12}	0.59
x_{22}	0.41
x_{23}	1
x_{25}	0.02
x_{17}	0.43
x_{27}	0.57
x_{18}	0.98

* The variables that do not appear in the table receive value zero.

Strategy 1 forces the fractional variable having the biggest fractional variable, i.e., x_{18}. If strategy 2 is selected, then branching starts with the smallest fractional variable, i.e., x_{25}. On the other hand Strategy 3 depends on picking randomly between variables x_{12}, x_{22}, x_{25}, x_{17}, x_{27} and x_{18}.

Note that the highest fraction 0.98 associates to the variable x_{18}. In the optimal solution, it is very likely to have value one for variable x_{18}, hence Strategy 1 makes a conscious choice.

We find an initial feasible solution using our heuristic procedure discussed in Section 3.1.2. We update the best known, i.e., incumbent, solution wherever we find a feasible solution with higher total profit value.

We fathom the node if any one of the following cases occurs:

i. The LP solution is infeasible. This occurs for the partial solution in which $x_{ij}=1$, but not for $x_{ij}=0$.

ii. The solution has all integer decision variables. In such a case, an optimal solution from that node is already found. The incumbent solution is updated, if the resulting solution value is better.

iii. Whenever the upper bound is no greater than the incumbent solution. In such a case, the resulting solution cannot lead to a unique optimal solution, i.e., it is not promising.

We backtrack whenever both nodes at a level are fathomed. We stop whenever we reach the root node; hence search all partial solutions implicitly.

For a particular node, we calculate the total realized profit by collecting the profits of the items that are already assigned to any knapsack. We let $TC(S) = \sum p_i x_{ij}(S)$ where $x_{ij}(S)$ is the value of x_{ij} for node S and $TC(S)$ is the total realized profit.

The upper bound is found by adding the associated constraint, '$x_{ij}=0$' or '$x_{ij}=1$' to the problem solved in the parent node. We calculate the upper bounds in sequel from the easiest to the hardest to compute. Accordingly we first find UB_1 then UB_2 and finally compute the LP based upper bound, UB_3. We benefit from the LP solution of the node in deriving lower bounds. We compute a naïve lower bound by taking the integer part of the solution and update the incumbent if the lower bound at the node is higher.

In place of solving the LP at each node, we implement the addition and deletion of a constraint idea. In doing so, we solve the LP only at the root node and use the addition of a constraint option while branching and the deletion of a constraint while backtracking. The added or deleted constraints are $x_{ij}=0$ or $x_{ij}=1$.

Below is the algorithmic description of our branch and bound algorithm, we state the algorithm according to Branching Strategy 1.

Branch and Bound Algorithm:

Step 0. Find an initial feasible solution using the procedure discussed in Section 3.1.2.

Let *INC* be the total profit of schedule, and set the incumbent solution to *INC*.

Level=1

Solve the LP relaxation of the problem, and let the solution be LPR

If LPR produces all integer decision variables then it is optimal, STOP.

Step 1. Let $x_{kr} = Max\{ x_{ij} / 0 < x_{ij} < 1\}$ where x_{ij} is the optimal LP relaxation solution. Generate the following two subproblems:

Subproblem 1. $x_{kr} = 1$

Subproblem 2. $x_{kr} = 0$

Fathom subproblem *i* if any of the following conditions holds:

i. The solution has all integer variables.

If *LPR* > *INC*

INC = *LPR*

ii. The solution is infeasible.

This happens only for subproblem 1.

iii. $UB_i \leq INC$, (first try i=1, then i=2 and finally i=3) i.e., the subproblem cannot lead to a unique optimal solution.

Step 2. If both subproblems in Step 1 are eliminated then go Step 3.

If both subproblems remain, branch from the one having the largest upper bound value.

If only a single subproblem remains, continue from this subproblem.

Level=*Level*+1

Go to Step 1

Step3. *Level*=*Level*-1

If *Level*=1 then Stop else Go to Step 1.

We implement our Branch and Bound algorithm on an example problem, with 6 items and 2 knapsacks. The profit and weight values are tabulated in the below table.

Table 3.9: The Data for the 6-Item 2-Knapsack Example Problem

Items	1	2	3	4	5	6
p_j	35	65	64	46	60	5
w_j	56	59	80	64	75	17

Capacity of knapsacks are $C_1 = 65$ and $C_2 = 95$
Cardinalities for knapsacks are $K_1 = 1$ and $K_2 = 1$

At the root node:
$UB_1 = 189$
$UB_2 = 145.8$
$UB_3 = 145.8$
LB Naïve = 65
LB Heuristic = 116
INC = 116

The branch and bound solution is summarized in the below figure. The numbers on the nodes show the solution path. The maximum fractional variables are shown in nodes.

There are three fathoming conditions as stated in algorithm: ALL: we hit a full integer solution, INF: the subproblem is infeasible, UB_i (i=1,2,3) : the branch is

fathomed as $UB_i \leq INC$. The type of upper bound that fathoms the branch is shown on the nodes. UB_1 means that UB_2 and UB_3 are not computed as $UB_1 \leq INC$.

Note that Initial Incumbent Solution (*INC*) is updated only once at the 14th node where it becomes 130.

The branches following nodes 2, 4, 7, 9, 11, 13, 17, 23 and 28 are fathomed due to infeasibility. The other branches are fathomed by upper bounds, i.e., they are not promising.

The solution is found at the 8th level. A total of 32 nodes are searched and the optimal solution is found at the 14th node.

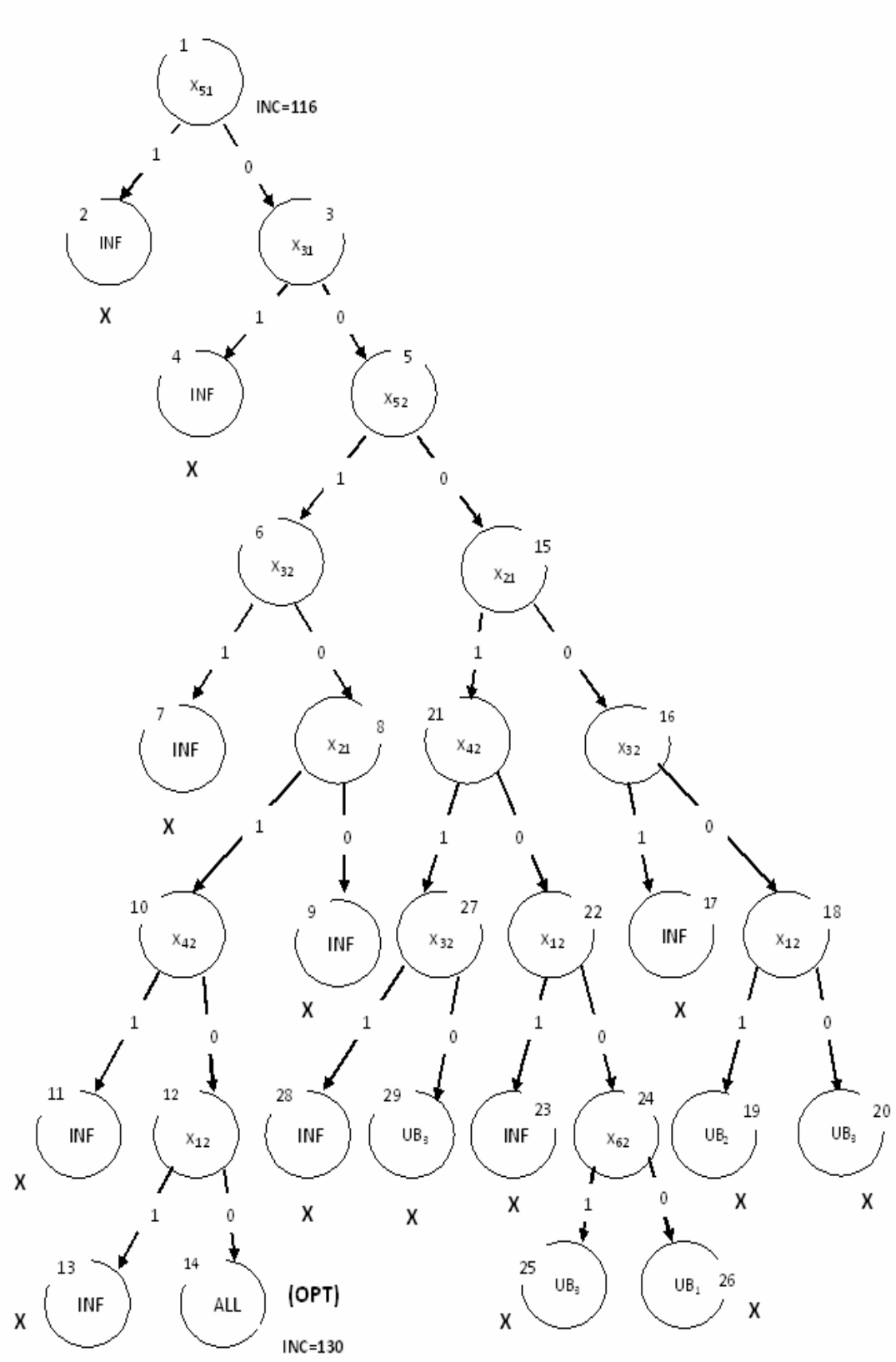

Figure 3.5: 6-Item 2-Knapsack Branch and Bound Tree

CHAPTER 4

COMPUTATIONAL RESULTS

4.1 Input Generation

In this chapter, we first present the data generation scheme and then discuss the results of our computational experiment. Our aim here is to test the efficiency of our algorithms and detect the effects of certain parameters on the difficulty of the solutions.

To generate p_j, w_j and C_i values, we use the scheme proposed by Martello and Toth (1990) for the multiple knapsack problem. According to this scheme, the p_j and w_j values are generated from discrete uniform distribution [10,100] and C_i values are discrete uniform between 0 and $(0.5\sum_{j=1}^{n} w_j - \sum_{k=1}^{i-1} C_k)$. We set the lower limit of the discrete distribution to w_{min} for C_i values, otherwise the knapsack having a capacity between 0 and w_{min}-1 would never be used.

We generate the cardinality of knapsack i, i.e., K_i from discrete uniform distribution U[1,($\lfloor n/m \rfloor$-1)]. Our upper limit somewhat guarantees that the cardinality constraint is forcing, i.e., nonredundant.

We use two discrete uniform distributions U[10,100] and U[10,250] for low and high profit variability. We hereafter call these profit sets as Set I and Set II.

The number of items and number of knapsacks are tabulated below.

Table 4.1: Generated Problems

Number of Items (*n*)	Number of Knapsacks (*m*)
100	5
100	8
150	5
150	8
200	10

For each combination in the table, we generate and solve 10 problems and we perform the experiments for both sets I and II.

We also generate large-sized problem instances with up to 1900 items and 20 knapsacks to set the solution capability of our branch and bound algorithm.

Input generation part is coded with C programming language with Microsoft Visual 6.0. (2003). The experiments are conducted with C# programming language with Microsoft Visual 8.0.(2005).

4.2 Performance Measures

In this section we set our performance measures that are used to evaluate the performance of the heuristic algorithm, branch and bound algorithm and upper bounds. The performances measures used are as listed below:

For Upper Bounds

- The average and maximum percentage deviation from the optimal solution.

The percentage deviation is defined as $\frac{UB - OPT}{OPT} \times 100$ where *UB* is the upper bound value and *OPT* is the optimal total profit.

For Heuristic Algorithm

- The average and maximum CPU times in seconds
- The average and maximum percentage deviation from the optimal solution

The percentage deviation is defined as $\frac{OPT - LB}{OPT} \times 100$ where *LB* is the lower bound value.

For Branch and Bound Algorithm

- The average and maximum CPU times
- The average and maximum number of nodes generated
- The average and maximum number of nodes generated until *OPT* is found.
- The average and maximum level of the tree reached (the depth of the branch)

The algorithms are coded with C# programming language with Microsoft Visual 8.0.(2005) and run on Microsoft Windows XP. For optimization problems Cplex version 10.1 is used. The instance runs are performed on the Intel ® 4 CPU 3.20 GHz and 1 MB of Ram computer.

4.3 Strategy Selection

In Section 3.2 we mentioned that the branching strategy of selecting highest fractional variable is likely to be the best strategy. Now we will perform an experiment to verify this issue. In below table results of three cases are observed. Remember that strategy 1 refers to selecting maximum fractional x_{ij} value, strategy

2 refers to selecting minimum fractional x_{ij} value and strategy 3 refers to selecting random fractional x_{ij} value while performing branch and bound algorithm.

Table 4.2: Branching Strategy Evaluation for B&B

		Strategy 1				Strategy 2				Strategy 3			
		# of nodes		CPU (seconds)		# of nodes		CPU (seconds)		# of nodes		CPU (seconds)	
n	***m***	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**	**max**
100	**5**	96,3	212	1,5	4,1	112,4	233	1,7	4,8	136,2	233	2,4	5,9
100	**8**	116,4	245	1,6	3,0	136,1	301	2,5	5,1	159,3	325	3,6	7,2
150	**5**	86,5	253	1,5	5,1	121,8	319	2,6	4,7	139,6	321	3,4	6,1
150	**8**	97,1	264	1,6	4,1	112,4	321	2,8	5,2	128,3	340	3,2	5,6
200	**10**	126,6	281	1,8	5,4	141,2	345	3,1	5,8	158,6	369	4,2	7,1

Note that selecting the maximum fractional variable, i.e., Strategy 1, produces smaller number of nodes and CPU times, when compared with the other strategies. Hence, we use Strategy 1 in our experiments.

We next analyze the effects of parameters and mechanisms on the problem, and base our main runs on the results from these experiments.

4.4 Effects of Parameters

The kMKP has certain parameters; number of items n, number of knapsacks m, profit value p_j, weight usage w_j, capacity usage C_i and cardinality amount K_i. In this section we analyze the effects of these parameters on the difficulty of the solutions.

Our main runs include the n, m and p_j effects. In this section, we analyze the effects of C_i, K_i and w_j values.

Effect of Capacity Value C_i

To see the effect of the capacity value on the difficulty of the problem we use two different capacity settings. We first use $U[w_{min}, (0.5\sum_{j=1}^{n} w_j - \sum_{k=1}^{i-1} C_k)]$ to generate C_i s and then generate another class by halving the capacities of the first two knapsacks.

We refer to first set as $C1$ and second set as $C2$. Note that $C2$ has restricted capacity.

Table 4.3: Capacity Effect on the Performance of B&B

		SET $C1$				SET $C2$			
		CPU(seconds)		# of nodes		CPU(seconds)		# of nodes	
n	m	avg	max	avg	max	avg	max	avg	max
100	5	1,5	4,1	96,3	212	1,4	3,6	81,1	138
100	8	1,6	3,0	116,4	245	1,3	2,7	104,6	212
150	5	1,5	5,1	86,5	253	1,4	4,9	83,8	221
150	8	1,6	4,1	97,1	264	1,5	3,9	78,1	169
200	10	1,8	5,4	126,6	281	1,6	5,9	97,2	191

As can be observed from the above table, the average number of nodes and CPU times decrease when capacities become tighter. This is due to the fact that more solutions become feasible when the capacities are larger and this leaves more nodes for further investigation. Note that when n=150 the maximum number of nodes

searched is 253 when the capacities are larger, this number reduces to 221 when the capacities are decreased.

We continue our runs with harder problem combination, i.e., with set *C*1.

Effect of Cardinality Value K_i

The effect of the cardinality value on the difficulty of the solutions is same as the capacity case. We use two cardinality values. Initially we use U[1,($\lfloor n/m \rfloor$-1)] to generate K_i values. We then generate another set by halving the cardinalities of the first two knapsacks.

We refer to the first set as *K*1 and second set as *K*2. Note that *K*2 has restricted cardinality.

Table 4.4: Cardinality Effect on the Performance of B&B

		SET *K*1				SET *K*2			
		CPU (seconds)		# of nodes		CPU (seconds)		# of nodes	
n	***m***	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**	**max**
100	**5**	1,5	4,1	96,3	212	1,4	3,8	72,5	179
100	**8**	1,6	3,0	116,4	245	1,4	2,7	107,1	237
150	**5**	1,5	5,1	86,5	253	1,4	4,7	75,6	221
150	**8**	1,6	4,1	97,1	264	1,3	4,1	71,2	248
200	**10**	1,8	5,4	126,6	281	1,5	4,0	112,6	265

We observe that as we decrease the cardinality, the number of nodes and CPU times decrease because more solutions become feasible when the remaining cardinalities are larger and this leaves more nodes for further investigation. For example when there are 100 items and 8 knapsacks, the average numbers of nodes searched are 116 and 107 for large and small cardinality cases respectively.

We continue our main runs with larger combination, i.e., with set $K1$. Now we analyze the effect of weight (usage).

Effect of Weight Value w_i

A weight, i.e., capacity usage, increase is similar to the knapsack capacity decrease. Hence, we expect the problem becomes easier when we increase the weights of the items.

We will analyze the effects of the weights by changing the distribution range of weight values from U[10, 100] to U[10, 250] and call these sets as $W1$ and $W2$ respectively.

The construction and lower bound deviations are also included for this effect. The below table indicates the heuristic deviation before and after improvement.

Table 4.5: Weight Effect-Lower Bound Comparison

		SET $W1$				SET $W2$			
		%Dev of Construction		%Dev of Heuristic-LB_2		%Dev of Construction		%Dev of Heuristic-LB_2	
n	***m***	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**	**max**
100	**5**	3,1%	19,6%	1,0%	3,0%	3,1%	18,7%	1,4%	4,4%
100	**8**	2,6%	5,0%	2,6%	5,0%	3,1%	7,1%	2,6%	4,4%
150	**5**	1,7%	4,0%	1,4%	4,0%	2,4%	4,2%	1,8%	2,9%
150	**8**	3,4%	7,1%	1,6%	2,8%	4,1%	7,2%	1,9%	3,6%
200	**10**	2,8%	5,1%	1,1%	2,4%	3,6%	6,1%	1,8%	4,2%

As we observe, deviations increased on average for both construction and heuristic parts. If $W2$ is used, the branch and bound algorithm starts with a weaker lower bound.

Next, we focus on the branch and bound performance.

Table 4.6: Weight Effect-Branch and Bound Comparison

		SET *W*1				SET *W*2			
		CPU (seconds)		# of nodes		CPU (seconds)		# of nodes	
n	***m***	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**	**max**
100	**5**	1,5	4,1	96,3	212	1,6	3,2	97,6	158
100	**8**	1,6	3,0	116,4	245	1,2	2,7	104,1	214
150	**5**	1,5	5,1	86,5	253	1,3	4,8	81,3	214
150	**8**	1,6	4,1	97,1	264	1,2	2,9	101,2	202
200	**10**	1,8	5,4	126,6	281	1,4	3,4	108,6	191

In general the instances of *W*2 set are solved easier but there exist counter cases like 100 items and 5 knapsacks case. The number of the generated nodes is higher for set *W*2, which can be attributed to the random effect.

We continue our main experiment with smaller weight values, i.e, with set *W*1.

So far we have focused on the parameter effects; now we analyze the effects of the mechanisms on the difficulty of solution.

4.5 Effects of Mechanisms

In this section, we investigate the effects of the mechanisms we developed on the performance of our branch and bound algorithm. These mechanisms decide branching strategies and bounding schemes.

Effects of Upper Bounds-UB_1:

Note that removing UB_1 does not affect the number of nodes generated because the stronger bound UB_3 already covers it. However, when UB_1 fathoms a branch UB_3, i.e., the most complex upper bound, is not necessarily computed.

The effects of UB_1 on the CPU times of branch and bound algorithm are tabulated below.

Table 4.7: Effects of UB_1

		With UB_1, UB_2 and UB_3		With UB_2 and UB_3	
		BB-CPU (seconds)		BB-CPU (seconds)	
n	***m***	**avg**	**max**	**avg**	**max**
100	**5**	1,5	4,1	1,4	4,0
100	**8**	1,6	3,0	1,6	2,8
150	**5**	1,5	5,1	1,5	4,5
150	**8**	1,6	4,1	1,4	3,6
200	**10**	1,8	5,4	1,7	4,1

Note that, when UB_1 is not used, the average and maximum seconds are smaller. Therefore, we conclude that reduction due to UB_1 is outweighed by the effort spent to compute it. Hence, we do not use UB_1 in our main runs.

Effects of Upper Bounds-UB_2:

Now we analyze the effects of UB_2 on the performance of the algorithm. As in UB_1 case, the number of nodes generated is not expected to change because UB_3 is stronger than UB_2. On the other hand; when UB_2 fathoms a branch there is no need to compute UB_3.

The table below reports on the performance of branch and bound algorithm that uses and does not use UB_2.

Table 4.8: Effects of UB_2

		With UB_1, UB_2 and UB_3		With UB_1 and UB_3	
		BB-CPU (seconds)		BB-CPU (seconds)	
n	***m***	**avg**	**max**	**avg**	**max**
100	**5**	1,5	4,1	1,7	4,2
100	**8**	1,6	3,0	1,7	3,2
150	**5**	1,5	5,1	1,6	5,2
150	**8**	1,6	4,1	1,8	4,9
200	**10**	1,8	5,4	2,1	4,1

As can be observed from the above table UB_2 when used together UB_3, reduces the solution times. Hence, we use UB_2 in our main runs.

Effects of Upper Bounds-UB_3:

UB_3 does not only help for fathoming but also it decides the branching path. We experiment on the performance of branch and bound algorithm with and without UB_3 in evaluating the nodes. We always continue for x_{ij}=1 branch and use UB_2 in evaluation.

The effects of UB_3 on the CPU times of branch and bound algorithm are tabulated below.

Table 4.9: Effects of UB_3

		With UB_1, UB_2 and UB_3				With UB_1 and UB_2			
		BB-CPU (seconds)		# of nodes		BB-CPU (seconds)		# of nodes	
n	***m***	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**	**max**
100	**5**	1,5	4,1	96,3	212	5,9	11,1	292,5	512
100	**8**	1,6	3,0	116,4	245	7,3	12,1	408,6	596
150	**5**	1,5	5,1	86,5	253	9,7	14,4	389,6	485
150	**8**	1,6	4,1	97,1	264	8,2	13,8	326,4	445
200	**10**	1,8	5,4	126,6	281	10,6	15,8	421,1	635

As can be observed from the above table, using UB_3 greatly reduces the number of nodes and CPU times. The reduction is about 4 times for both. We can conclude that UB_3 is quite powerful and should be used to evaluate the nodes. Considering all upper bound experiments, we decide to use UB_2 and UB_3 in sequel.

Now we will focus on heuristic lower bound.

Effects of Lower Bound-Heuristic Solution:

To investigate the effect of the lower bounds on the performance of the branch and bound algorithm, we compare two cases: the algorithm that uses no lower bounds as initial feasible solution (i.e., starts with value zero) and the algorithm that uses our heuristic procedure to produce an initial feasible solution. The results are reported below.

Table 4.10: Effects of *LB* on Branch and Bound CPU Values.

		With LB_2				Without LB_2			
		CPU(seconds)		# of nodes		CPU(seconds)		# of nodes	
n	***m***	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**	**max**
100	**5**	1,5	4,1	96,3	212	3,8	7,0	165,4	312
100	**8**	1,6	3,0	116,0	245	7,3	11,5	236,6	446
150	**5**	1,5	5,1	86,0	253	6,3	11,4	186,3	341
150	**8**	1,6	4,1	97,1	264	7,1	10,8	225,2	325
200	**10**	1,8	5,4	126,6	281	6,7	9,8	201,4	295

As can be observed from the above table, incorporating of initial lower bound highly improves the performance of the branch and bound algorithm. This means finding an LP solution with all integer variables takes significant time. The most significant reduction is due to 100 items and 8 knapsacks case.

We next discuss our main runs.

4.6 Main Runs

Recall that in Section 3.1.1 three different upper bounds namely capacity relaxed upper bound, cardinality relaxed upper bound and integrality relaxed upper bound are developed and referred to as UB_1, UB_2 and UB_3 respectively. In Section 4.4 we found that removing UB_1, but not UB_2 and UB_3, results in better solutions. We now report on the performances of UB_2 and UB_3. The performance of an upper bound for instance i is measured by its deviation from optimal solution as a percentage of the optimal solution and calculated as;

$$DevU_i = \frac{UB_i - OPT}{OPT} \times 100$$

Hence $AvgDev = \sum_{i=1}^{10} DevU_i / 10$ and $MaxDev = max_i\{DevU_i\}$

The deviations for 10 combinations are reported in Table 4.11

Table 4.11: Upper Bound Performances at Root Node

		SET I				SET II			
		%Dev. UB_2		%Dev. UB_3		%Dev. UB_2		%Dev. UB_3	
n	***m***	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**	**max**
100	**5**	55,3%	99,0%	0,5%	2,1%	47,5%	102,7%	0,3%	0,6%
100	**8**	33,2%	77,2%	0,5%	3,0%	37,6%	93,1%	0,6%	3,2%
150	**5**	180,7%	418,7%	0,5%	2,8%	231,9%	456,8%	0,5%	3,1%
150	**8**	48,3%	108,1%	0,5%	1,4%	51,6%	118,3%	0,8%	2,6%
200	**10**	70,5%	207,6%	0,1%	0,4%	76,1%	232,8%	0,6%	1,3%

Note from the above table that UB_3 is the most powerful upper bound, deviates from the optimal by less than 0.5% on the average over all combinations and the deviations do not deteriorate when the problem sizes become larger. This is due to the satisfactory behavior of the LPR solution. From Table 4.14, it can be observed that the number of fractional variables by LP is much smaller than the total number of the integer variables.

The average deviation of UB_2 is not satisfactory, however it is very quick. Due to its high speed we first compute UB_2 if it cannot eliminate we compute UB_3. Note that, as the ranges for the profit values increase, the upper bound deviations increase. This is due to the fact that the feasible solutions are apart from each other when distributions have wider ranges.

Now we evaluate the performance of the lower bounds that is found by the heuristic method, stated in Section 3.1.2.

Note that the higher the lower bound deviation, the stronger it is.

For instance *i* we compute,

$$DevLB_i = \frac{OPT - LB_i}{OPT} \times 100,$$

The lower bound is constructed in three steps; first, we find a naive lower bound then construction step is performed followed by the improvement step.

Table 4.12 reports on the performance of the naïve lower bounds.

Table 4.12: Performance of Naïve Lower Bound

		SET I		SET II	
n	***m***	**%Dev LB_1 avg**	**%Dev LB_1 max**	**%Dev LB_1 avg**	**%Dev LB_1 max**
100	**5**	12.1%	36,8%	10.6%	22.5%
100	**8**	15.5%	23,7%	15.4%	26.6%
150	**5**	14.4%	47.1%	12.5%	31.1%
150	**8**	10.4%	12.7%	13.5%	16.5%
200	**10**	10.1%	17.4%	13,0%	19.6%

The average deviations are between 10 and 15 percent, and maximum deviations are close to 50%. We observe that the deviations are smaller when the profit ranges are wider. We did not report on the CPU times, as they are negligibly small.

The below table summarizes the percentage deviation of the heuristic lower bound from the optimal solution.

Table 4.13: Performance of Heuristic Algorithm (LB_2)

		SET I				SET II			
		% Dev. of Construction		% Dev. of Heuristic (LB_2)		% Dev. of Construction		% Dev. of Heuristic (LB_2)	
n	***m***	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**	**max**
100	**5**	3,1%	19,5%	1,0%	3,0%	2,5%	9,1%	1,1%	3,6%
100	**8**	2,6%	5,0%	2,6%	5,0%	4,5%	8,1%	2,4%	5,1%
150	**5**	1,7%	4,0%	1,4%	4,0%	3,2%	6,1%	2,7%	4,2%
150	**8**	1,5%	2,4%	1,5%	2,4%	2,1%	4,0%	1,8%	2,8%
200	**10**	1,2%	2,9%	1,0%	2,2%	1,3%	2,6%	2,1%	4,2%

As can be observed from Table 4.13, the heuristic performs quite satisfactory. The worst maximum deviation is 5.1%. The average deviations are mostly below 1.5%. The deviations do not deteriorate with an increase in problem size. Note that the minimum average deviation is observed for the maximum problem size, i.e., 200 items and 10 knapsacks. We also observe that if the construction phase results in high deviations, the improvement phase recovers. Note that for the first combination, the construction heuristic deviates about 3% on average, whereas this deviation is reduced to 1% by the improvement phase.

We observe that the lower bound deviations are higher when the variability of the profits is higher.

Due to its satisfactory performance we use the improvement heuristic as an initial feasible solution in our branch and bound algorithm.

The number of fractional variables by LP and total number of fraction variables are reported in below table.

Table 4.14: Number of Fractional x_{ij} Variables

		SET I		SET II		
n	***m***	**# of fractional x_{ij} variables avg**	**# of fractional x_{ij} variables - max**	**# of fractional x_{ij} variables avg**	**# of fractional x_{ij} variables - max**	**# of total x_{ij} variables**
100	**5**	7,1	11	7,2	12	500
100	**8**	9,6	20	10,8	17	800
150	**5**	7,1	14	8,4	13	750
150	**8**	13,2	20	16,7	21	1200
200	**10**	15,5	23	16,2	19	2000

The above table shows that the number of fractional x_{ij} variables is quite small when compared to the total number of the decision variables. Note that the LPR gives at most 23 fractional variables out of 2000 decision variables. These computational results motivate us to base our branching rule on the fractional variables of the LPR solution. Due to the satisfactory behavior of UB_3, its closeness to the optimal total profit values, and few continuous variables, we expect a satisfactory performance from our branch and bound algorithm.

We next discuss the performance of the branch and bound algorithm that is measured by the CPU times and the number of nodes searched. Clearly the lower and upper bounds highly affect the performance.

In general all bounds, in particular UB_3 and LB_2, perform quite satisfactory even at the root node (see Tables 4.11, 4.12 and 4.13). Hence one can expect satisfactory behavior from a branch and bound algorithm that employs these bounds. The table below reports on the performance results of our branch and bound algorithm.

Table 4.15: Performance Evaluation for B&B Part

		SET I						SET II					
		# of nodes		optimality node		depth of search		# of nodes		optimality node		depth of search	
n	*m*	avg	max	avg	max	avg	max	avg	max	avg	max	avg	max
100	**5**	96,3	212	14,3	35	27,4	62	82,6	220	15,2	43	27,8	62
100	**8**	116,4	245	17,1	32	32,7	49	118,6	262	18,3	41	34,9	54
150	**5**	86,5	253	12,6	48	24,1	63	93,7	271	14,9	61	26,0	52
150	**8**	135,2	288	13,3	26	35,6	69	149,6	296	18,3	43	41,3	76
200	**10**	116,8	285	13,7	26	37,2	72	115,2	287	14,0	32	35,6	51

Note there are nxm decision variables, hence 2^{nxm} complete solutions. The number of the partial solutions used to generate these complete solutions is thus much higher. Due to the power of the LPR solution, both in leading our branch strategy and fathoming the partial solutions, we only generate a small portion of those solutions. Note from the Table 4.15 that we generate at most 296 nodes.

We also report on the CPU times of the CPLEX algorithm and compare them with our branch and bound algorithm.

Table 4.16: Performances of B&B and CPLEX algorithms

		SET I					SET II				
		BB-CPU (seconds)		IP-CPU (seconds)			BB-CPU (seconds)		IP-CPU (seconds)		
n	***m***	**avg**	**max**	**min**	**avg**	**max**	**avg**	**max**	**min**	**avg**	**max**
100	**5**	1,5	4,1	0,2	8139,5	27150	1,8	4,4	0,1	3280,4	28389
100	**8**	1,6	3,0	1,5	4563,5	30000	1,8	4,2	0,3	6089,3	30000
150	**5**	1,5	5,1	0,1	6984,4	30000	1,8	5,3	0,1	2515,1	30000
150	**8**	2,4	6,4	0,6	2290,2	14315	3,4	5,8	0,2	7859,6	29659
200	**10**	2,0	7,1	0,2	1900,5	18950	3,8	6,4	0,1	5216,3	18354

For CPLEX runs, we set a termination limit of 30.000 seconds (approximately 8 hours). The instances that are not solved in 30.000 seconds contribute to the total CPU time by 30.000 seconds.

As can be observed from the above table, the CPU times by CPLEX algorithm are too high. Moreover, unlike our branch and bound algorithm the CPLEX performs too inconsistent that there is a huge gap between minimum and maximum CPU time values. Hence the need for an implicit enumeration algorithm is well justified. Note that our algorithm returns optimal solutions in very small CPU times, consistently.

We solve the problem instances with up to 200 items 10 knapsacks easily. As the number of the knapsacks and items increase, the performance of our algorithm deteriorates. This is due to an increase in the search size and the effort spent by LP at each node.

4.7 Limit of Our Solution Method

We aim to find the limit on the problem size that our algorithm can handle, using the profit set II (U[10,250]).

Table 4.17 gives the performance of our branch and bound algorithm on large-sized problem instances, for problem set includes item sizes from 500 to 1900 and knapsack sizes 10, 15 and 20.

Table 4.17: Limit Run Experiments

n	***m***	**Total CPU- avg minutes**	**Number of solved instances (10)**
500	10	5,56	10
800	15	10,20	10
1200	15	16,65	10
1400	20	28,28	10
1700	20	49,38	8
1800	20	54,39	6
1900	20	62,59	2

As can be observed from the above table, all problems can be solved in reasonable times when n=1400 and m=20. When n is 1700, 1800 and 1900, the number of problems (out of 10) that can be solved in one hour reduces to 8, 6 and 2 respectively.

CHAPTER 5

CONCLUSIONS

In this study, we consider the Cardinality Constrained Multiple Knapsack problem (kKMP). The knapsack problems in general and kKMP in particular find their application both in service and manufacturing industries. Despite this fact, the associated reported research on the kKMP is quite limited.

We observe that the kKMP cannot be solved to optimality, even by the most powerful Integer Programming Solver, CPLEX. So, efficient implicit enumeration techniques are required to arrive at optimal solutions. Recognizing this fact, we propose optimization and approximation procedures with the hope of generating satisfactory solutions in reasonable times.

We first study the Linear Programming Relaxation (LPR) of the problem and verify its quality in producing very few continuous variables. Hence we base our approximation (heuristic) and optimization algorithms on the optimal LPR solutions. Our heuristic procedure first finds an initial solution by taking the integer part of the optimal LPR solution and then improves this solution by pair wise interchanges. We introduce the total profit value of the heuristic procedure as an initial feasible solution for the branch and bound algorithm. We use the optimal LPR solutions not only to evaluate the partial solutions but also to guide our search by setting the branching strategy.

The results of our extensive computational experiment show that our heuristic procedure generates solutions that deviate from the optimal solutions by no more than three percentages on average. Moreover, at the root node, our LPR based upper bound deviates from the optimal solution by at most one percent. Our branch and

bound algorithm finds optimal solutions to the problem instances with up to 1400 items and 20 knapsacks in less than 1800 seconds, on average.

The lower and upper bounds used and the branching strategy employed are quite significant in improving the efficiency of the branch and bound algorithm. The most efficient results are obtained when the LPR based upper bounds together with maximum fractional variable strategy are employed. Moreover using our heuristic procedure as a lower bound at the root node reduced the solution times.

We observe that as the number of items or the number of knapsacks increases, the solution times increase, however not in exponential rate. We also observe that the capacities, cardinalities, weights and profits are significant parameters that affect the problem complexity.

We hope our work opens new research avenues some note-worthy of which are listed below:

- Investigating the special cases of the kMKP like identical knapsack cardinalities or capacities.
- Studying the kMKP with dependent profit and weight values, i.e., the profit and weight values are dependent on the knapsack assigned.
- Studying the assignment restricted the kMKP, i.e., there is an assignment restriction such that some items cannot be put in all knapsacks.
- Investigating the properties of the LPR solution.
- Developing Polynomial Time Approximation Schemes for the kMKP.

REFERENCES

A. Caprara, H. Kelleler, U. Pferschy, D. Pisinger, Approximation Algorithms for Knapsack Problems with Cardinality Constraints, *European Journal Operations Research*, 123: 333-345, 2000.

C.Chekuri and S. Khanna, A PTAS for the Multiple Knapsack Problem, *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, 213-222, 2000.

D.S. Hochbaum and D.B. Shymos, A Polynomial Time Approximation Scheme for Scheduling on Uniform Processors: Using Dual Approximation Approach, *SIAM Journal on Computing,* 17(3): 539-551, 1988.

E.L. Lawler, J.K. Lenstra, A.H. G. Rinooy Kan, and D.B. Shymoys, Sequencing and Scheduling: Algorithms and Complexity, *Handbooks in OR&MS*, 4: 445-522, 1993.

F.D. Murgolo, An Efficient Approximation Acheme for Variable-Sized Bin Packing, *SIAM Journal on Computing*, 16(1):149-161,1987.

G.B. Mathews, On the Partition of the Numbers, *Proceedings of the London Mathematical Society*, 28:486-490,1897.

G. Ingargiola and J. Korsh, An Algorithm for the Solution of 0-1 Loading Problems, *Operations Research,* 23:1110-1119,1975.

H. Kelleler, U. Pferschy, D. Pisinger, Knapsack Problems, *Springer*, 2004.

J.M. Valerio de Carvalho, Exact Solution of Bin-Packing Problems Using Column Generation and Branch-and-Bound, *Annals of Operations Research*, 86: 629–659, 1996.

K. Dudzinski, On a Cardinality Constrained Linear Programming Knapsack Problem, *Operations Research Letters,* 8: 215-218, 1989.

M. Mastrolilli, M. Hutter, Hybrid Rounding Techniques for Knapsack Problems, *Discrete Applied Mathematics*, 154: 640-649, 2006.

M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to The Theory of NP-Completeness, *W. H. Freeman*, 1979.

M.S. Hung and J.C. Fisk, An Algorithm for 0-1 Multiple Knapsack Problems, *Naval Research Logistical Quarterly*, 24:571-579, 1978.

R.E. Campello and N.F. Maculan, An O(n^3) Worst Case Bounded Special LP Knapsack (0-1) with two Constraints, *Recherce Operationelle / Operational Research*, 22: 27-32, 1988.

S. Martello and P. Toth, Heuristics Algorithms for the Multiple Knapsack Problem, *Computing*, 27:93-112, 1981.

S. Martello and P. Toth, Knapsack Problems Algorithms and Computer Implementations*, Chichester ; J. Wiley & Sons*, 1990.