COMPUTATIONAL ANALYSIS OF THE SEARCH BASED CUTS ON THE MULTIDIMENSIONAL 0-1 KNAPSACK PROBLEM

A THESIS
SUBMITTED TO THE DEPARTMENT OF INDUSTRIAL ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCES
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

by

Duygu Pekbey

September 2003

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Osman Oğuz (Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Mustafa Ç. Pınar

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Bahar Y. Kara

Approved for the Institute of Engineering and Sciences:

Prof. Mehmet Baray
Director of Institute of Engineering and Sciences

ABSTRACT

COMPUTATIONAL ANALYSIS OF THE SEARCH BASED CUTS ON THE MULTIDIMENSIONAL 0-1 KNAPSACK PROBLEM

Duygu Pekbey

M. S. in Industrial Engineering

Supervisor: Assoc. Prof. Osman Oğuz

September 2003

In this thesis, the potential use of a recently proposed cut (the search based cut) for 0-1 programming problems by Oguz (2002) is analyzed. For this purpose, the search based cuts and a new algorithm based on the search based cuts are applied to multidimensional 0-1 knapsack problems from the literature as well as randomly generated multidimensional 0-1 knapsack problems. The results are compared with the implementation of CPLEX v8.1 in MIP mode and the results reported.

Key Words: 0-1 Integer Programming, Multidimensional 0-1 Knapsack Problem

ÖZET

ARAŞTIRMA TABANLI KESMELERİN ÇOK BOYUTLU 0-1 SIRT ÇANTASI PROBLEMLERİ ÜZERİNDE HESAPSAL ANALİZİ

Duygu Pekbey
Endüstri Mühendisliği Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Osman Oğuz
Eylül 2003

Bu çalışmada, yakın zamanda 0-1 programlama problemleri için Oguz (2002) tarafından önerilen bir kesmenin (araştırma tabanlı kesme) potansiyel faydası analiz edilmektedir. Bu amaçla, araştırma tabanlı kesmeler ve bunlar üzerine kurulan yeni bir algoritma literatürdeki çok boyutlu 0-1 sırt çantası problemlerine ve rastlantısal olarak oluşturulan çok boyutlu 0-1 sırt çantası problemlerine uygulanmaktadır. Sonuçlar CPLEX v8.1' in MIP biçimindeki uygulaması ve literatürdeki sonuçlarla karşılaştırılmaktadır.

Anahtar Kelimeler: 0-1 tamsayılı programlama, çok boyutlu 0-1 sırt çantası problemleri

To my parents and my sister . . .

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Assoc. Prof. Osman Oğuz for his guidance, attention and patience throughout our study.

I wish to express my thanks to the readers Assoc. Prof. Mustafa Ç. Pınar and Asst. Prof. Bahar Y. Kara for their effort, kindness and time.

I would like to thank Aykut Özsoy, Güneş Erdoğan, Ünal Akmeşe, Çağatay Kepek, Hakan Ümit and Pınar Tan for their assistance, support and friendship during the graduate study. I am also very thankful to all Industrial Engineering Department staff.

I am grateful to Çağdaş Yavuz and Karim Saadaoui for their support and friendship.

Finally, I would like to thank my family, and especially my sister for her invaluable support and love.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

C h a p t e r 1

Introduction

In this thesis, a computational study on the potential use of a recently proposed cut for 0-1 programming problems by Oguz (2002) is presented. The cut is of a general type, and can be used for almost any 0-1 integer programming problem. We have chosen to focus on and limit the scope of the study to the multidimensional 0-1 knapsack problems for a couple of reasons. Firstly, extending the scope to cover several types of combinatorial optimization problems would require development of much more sophisticated and expert programming skills and effort, and longer time. Secondly, a wide array of test problems and computational results on this specific problem already exist in the literature and it is easier to carry out comparative analysis with this problem.

In this study, in order to analyze the potential use of the search based cuts (see Oguz (2002)), we have applied them to 60 multidimensional 0-1 knapsack problems (0-1 MDKP) from the literature as well as 25 randomly generated set covering problems. In addition, in order to check the efficiency of the partitioning algorithm-a new algorithm based on the search based cuts-, we have made computational experiments with 30 small-sized 0-1 MDKP from the literature (Shih (1979)) as well as randomly generated 0-1 MDKP with different numbers of variables and different numbers of constraints.

In order to reduce the computational time and effort required by the partitioning algorithm, we presented a modification of the partitioning

algorithm which we named "one step partitioning algorithm" and made computational experiments with 60 randomly generated large sized 0-1 MDKP as well as 30 large-sized 0-1 MDKP from the literature (Chu and Beasley (1998)). We compared our results with the implementation of CPLEX v8.1 in MIP mode and the results reported by Chu and Beasley (1998).

The remaining part of this thesis is organized as follows: After introducing the multidimensional 0-1 knapsack problem briefly, we present a comprehensive survey of the work done for it in the literature in the next chapter. In chapter 3, the "search based cuts" and the "partitioning algorithm" are discussed. In chapter 4, computational analysis of the search based cuts is presented and in chapter 5, computational results of the application of one step partitioning algorithm to the large-sized 0-1 MDKP are reported. Finally, some conclusions and remarks for future works are given in chapter 5.

The Multidimensional 0-1 Knapsack Problem

The multidimensional 0-1 knapsack problem (0-1 MDKP) is an important combinatorial optimization problem which is widely studied in the literature and can be employed to formulate many practical problems such as capital budgeting or resource allocation. As an example, think that there are n projects with known profits c_j and project j consumes a_{ij} units from resource i given b_i as the capacity of resource i. The goal is to select a group of projects to allocate resources in such a way that the profit is maximized and the capacity of any resource is not exceeded. Other applications of the multidimensional 0-1 knapsack problem include cutting stock problems

(Gilmore and Gomory (1966)), cargo loading (Shih (1979)) and allocating processors and databases in distributed systems (Gavish and Pirkul (1982)).

The 0-1 MDKP can be considered as a general 0-1 integer programming problem with non-negative coefficients and can be formulated as follows:

$$\text{maximize} \sum_{j=1}^{n} c_j x_j ,$$

$$\text{subject to} \sum_{j=1}^{n} a_{ij} x_j \leq b_i , \; i = 1, \ldots , m,$$

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

where $a_{ij} \geq 0$ for $i = 1, \ldots , m$; $j = 1, \ldots , n$; $b_i > 0$ for $i = 1, \ldots , m$ and $c_j > 0$ for $j = 1, \ldots ,n$.

In addition, for the problem to be meaningful, the following must be true:

$b_i < \sum_{j=1}^{n} a_{ij}$ for $i = 1, \ldots , m$ (otherwise i^{th} constraint will be redundant),

$a_{ij} < b_i$ for $i = 1, \ldots , m$; $j = 1, \ldots , n$ (otherwise x_j will be fixed to zero).

If $m = 1$, the problem is the standard knapsack problem which is proven to be NP-complete(see Garey and Johnson (1979)). Since the standard knapsack problem is NP-complete, the multidimensional 0-1 knapsack problem is also NP-complete.

Chapter 2

Literature Review

Most of the research on knapsack problems deals with the standard knapsack problem ($m = 1$) and a good review of exact and heuristic algorithms for the standard knapsack problem is given by Martello and Toth (1990). Below we review exact and heuristic algorithms designed to solve the multidimensional 0-1 knapsack problem (0-1 MDKP).

Gilmore and Gomory (1966), and Nemhauser and Ullmann(1969) developed dynamic programming based methods to solve the 0-1 MDKP. However, they were not able to solve large instances. Nemhauser and Ullmann(1969) reported that for the problems that they had solved the number of variables was at most 50. Weingartner (1967), and Weingartner and Ness (1967) also developed dynamic programming based methods to solve the 0-1 MDKP and Cabot (1970) developed an enumeration algorithm based on Fourier-Motzkin elimination.

Soyster, Lev, and Slivka (1978) developed an algorithm for solving zero-one integer programs with many variables and few constraints. In their algorithm, sub-problems were generated from the linear programming relaxation and solved through implicit enumeration. The variables in these sub-problems corresponded to the fractional variables obtained in the linear program. The number of variables in the sub-problems is much less than the number of variables in the original zero-one integer program because the number of fractional variables in the linear program is bounded by the number of constraints in the linear program.

Senju and Toyoda (1968) proposed a dual gradient method that starts with a possibly infeasible initial solution (all decision variables set to 1) and achieves feasibility by dropping the non-rewarding variables one by one, while following an effective gradient path.

Zanakis (1977) compared three heuristic algorithms for the 0-1 MDKP from Hillier (1969); Kochenberger, McCarl, and Wymann (1974); Senju and Toyoda(1968) and reported that none was found to dominate the others computationally.

In order to apply the greedy method for standard knapsack problem where items are picked from the top of a list sorted in decreasing order on c_j/a_j (Martello and Toth (1987)) to the 0-1 MDKP, Toyoda (1975) proposed a new measurement called aggregate resource consumption. He developed a primal gradient method that improves the initial feasible solution (all decision variables set to zero) by incrementing the value of the decision variable with the steepest effective gradient. Using the basic idea behind Toyoda's primal gradient method, Loulou and Michaelides (1979) developed a greedy-like algorithm that expands the initial feasible solution by including the decision variable with the maximum pseudo-utility. The pseudo-utility is defined as $u_j = c_j/v_j$, where v_j is the penalty factor of variable j , which depends on the resource coefficients a_{ij} and can be defined in several ways. They tested this method on small-sized randomly generated problems as well as some larger real-world problems and showed that the average deviation from optimum ranged from 0.26% to 1.08% for smaller problems and up to 14% for larger problems.

Balas and Martin (1980) presented a heuristic algorithm for the 0-1 MDKP which utilizes linear programming by relaxing the integrality constraints x_j Є

$\{0, 1\}$ to $0 \leq x_j \leq 1$. The fractional x_j are then set to 0 or 1 according to a heuristic which maintains feasibility.

Shih (1979) proposed a branch and bound algorithm for the 0-1 MDKP. In this algorithm, estimation of an upper bound for a node was made by solving m single constraint knapsack problems with the same objective function. An optimal fractional solution (Dantzig (1957)) was found for each of the m single constraint knapsack problems separately. To find optimal fractional solution one must include as much as possible of each item in the order of decreasing c_j/a_{ij} to the knapsack i until the constraint i is satisfied exactly as an equation. Minimum of the objective function values associated with each optimal fractional solution was chosen as the upper bound for that node.

The node selected for next branching would be the end node whose upper bound is maximum of all end nodes and where the solution associated with such an upper bound is infeasible(if solution is feasible it is also an optimal solution). The branching variable would be the one whose c_j/a_{ij} ratio is minimum of all non-zero free variables in this infeasible solution.

Shih solved thirty 5-constraint knapsack problems with 30-90 variables (we also used this data to test our first algorithm) and reported that his algorithm is faster than original Balas and improved Balas additive algorithms (Balas (1965)) with respect to the total as well as the individual solution times.

Using Senju and Toyoda 's dual gradient algorithm and Everett (1963) 's generalized lagrange multipliers approach, Magazine and Oguz (1984) proposed another heuristic that moves from the initial infeasible solution (all variables set to 1) towards a feasible solution by following a direction which reduces the aggregate weighted infeasibility among all resource constraints. Their algorithm was tested on randomly generated problems with sizes from

m = 20 to 1,000 and n = 20 to 1,000, and its computational efficiency is compared with two other well-known heuristics: the primal heuristic of Kochenberger, McCarl, and Wymann (1974) (1) and the dual approach of Senju and Toyoda (1968)(2). They reported that, in terms of solution quality, (1) produced slightly better results than their heuristic and their heuristic was superior to (2). Their heuristic and (2) were much better than (1) in terms of computation time.

Fox and Scudder (1985) presented a heuristic based on starting from setting all variables to zero(one) and successively choosing variables to set to one(zero). They reported results for randomly generated test problems with sizes up to m = 100 and n = 100, and with c_j = 1 and a_{ij} = 0 or 1.

Gavish and Pirkul (1985) proposed another branch and bound algorithm in which they used tighter upper bounds obtained with relaxation techniques such as lagrangean, surrogate and composite relaxations. They tried to evaluate the quality of the bounds generated by these different relaxations and showed that the composite relaxation (which used a subgradient optimization procedure to determine the multipliers) yielded the best bounds overall, but needed extra computational effort. They developed new algorithms for obtaining surrogate bounds and suggested rules for reducing the problem size. They tested their algorithm on a set of randomly generated problems with sizes up to m = 5 and n = 200 and reported that it is faster than the branch and bound algorithm of Shih (1979). They showed that if their algorithm is used as a heuristic by terminating it before the tree search is completed, then it is superior to the heuristic developed by Loulou and Michaelides (1979).

In addition, Pirkul (1987) presented an efficient algorithm in which m knapsack constraints were transformed into a single knapsack constraint

using the dual variables (known as the surrogate multipliers) obtained from the linear programming relaxation of the 0-1 MDKP. He then obtained a feasible solution to this problem using a greedy algorithm based on the ordering of the profit to weight ratios. This ratio was defined as,

$$c_j / \sum_{i=1}^{m} w_i \, a_{ij}$$

where w_i is the surrogate multiplier for constraint i. Surrogate multipliers were determined using a descent procedure. He reported that the algorithm was considerably better than the heuristic of Loulou and Michaelides (1979) and similar to the pivot and complement heuristic of Balas and Martin (1980) in terms of solution quality.

Lee and Guignard (1988) presented a heuristic that combined Toyoda's primal heuristic (1975) with variable fixing, linear programming and a complementing procedure from Balas and Martin (1980). Computational experiments were done with standard test problems and randomly generated problems with sizes up to m = 20 and n = 200. They reported that their heuristic produced better results than Toyoda (1975) and Magazine and Oguz (1984), but is out-performed by Balas and Martin (1980).

Drexl (1988) presented a heuristic based upon simulated annealing. They made experiments with 57 standard 0-1 MDKP test problems from the literature and found optimal solutions for 25 of these problems.

Volgenant and Zoon (1990) extended Magazine and Oguz's heuristic in two ways: (1) in each step, not one, but more multiplier values are computed simultaneously, and (2) at the end the upper bound is sharpened by changing some multiplier values. They showed that these extensions yielded an

improvement, on average, at the cost of only a modest amount of extra computing time.

Crama and Mazzola (1994) showed that although the bounds obtained with relaxation techniques such as lagrangean, surrogate, or composite relaxations, are stronger than the bounds obtained from the linear programming relaxation, the improvement in the bound that can be achieved using these relaxations is limited. In fact, the improvement in the quality of the bounds using any of these relaxations cannot exceed the magnitude of the largest coefficient in the objective function.

There are a few number of papers considering a statistical-asymptotic analysis of the 0-1 MDKP. Schilling (1990) presented an asymptotic analysis and computed the asymptotic objective function value of a particular m constraint, n variable 0-1 random integer programming problem as n increases and m remaining fixed. In this analysis, the a_{ij} 's and c_j 's were uniformly and independently distributed over the unit interval and $b_i = 1$. Szkatula (1994) generalized that analysis to the case where the b_i were not restricted to be one (see also Szkatula (1997)). A statistical analysis was presented by Fontanari (1995) in which he investigated the dependence of the multidimensional knapsack objective function on the knapsack capacities and on the number of capacity constraints, in the case when all n objects were assigned the same profit value and the a_{ij} 's were uniformly distributed over the unit interval. A rigorous upper bound to the optimal profit was obtained employing the annealed approximation and then compared with the exact value obtained through a lagrangean relaxation method.

Freville and Plateau (1994) presented an efficient preprocessing procedure for the 0-1 MDKP based on their previous work ((Freville and Plateau

(1986)) and they also proposed a heuristic for the bidimensional knapsack problem (Freville and Plateau (1997)).

Dammeyer and Voss (1993) proposed a tabu search heuristic for the 0-1 MDKP based on reverse elimination method. They made computational experiments with 57 standard test problems from the literature and reported that they found optimal solutions for 41 of these problems.

Aboudi and Jörnsten (1994) combined tabu search with the pivot and complement heuristic of Balas and Martin (1980) in a heuristic for general zero-one integer programming. They made computational experiments with 57 standard test problems and found optimal solutions for 49 of these problems.

Løkketangen, Jörnsten, and Storøy (1994) presented a tabu search heuristic within a pivot and complement framework and gave computational results for the same set of test problems. They found optimal solutions for 39 of these problems.

Glover and Kochenberger (1996) presented a heuristic based on tabu search. They employed a flexible memory structure that integrates recency and frequency information keyed to "critical events" in the search process. Their method was enhanced by a strategic oscillation scheme that alternates between constructive (current solution feasible) and destructive (current solution infeasible) phases. They define a "critical event" as the last feasible solution found after a transition between phases. They found optimal solutions for each of 57 standard test problems from the literature.

Løkketangen and Glover (1996) presented a heuristic based on probabilistic tabu search for solving general zero-one mixed-integer programming

problems. They made computational experiments with 18 standard test problems and found optimal solutions for 13 of these problems.

Løkketangen and Glover (1997) presented a tabu search heuristic for solving general zero-one mixed-integer programming problems. They made experiments with 57 standard test problems and found optimal solutions for 54 of these problems.

Hanafi and Freville (1997) presented an efficient tabu search approach for the 0-1 MDKP, a heuristic algorithm strongly related to the work of Glover and Kochenberger (1996). They described a new approach to tabu search based on strategic oscillation and surrogate constraint information that provides a balance between intensification and diversification strategies. New rules needed to control the oscillation process were given for the 0-1 MDKP. They tested their approach on 54 instances from Freville and Plateau (1986) and 24 instances from Glover and Kochenberger(1996). Optimal solutions were obtained for the first set of problems and better results than Glover and Kochenberger (1996) were reported for the second set.

Khuri, Bäck, and Heitkötter (1994) presented a genetic algorithm to solve the 0-1 MDKP. In their algorithm, infeasible solutions were allowed to participate in the search and a simple fitness function that uses a graded penalty term was used. They applied their algorithm on 9 test problems taken from the literature and reported moderate results. The problem sizes ranged from 15 objects to 105 and from 2 to 30 knapsacks.

Thiel and Voss (1994) suggested an algorithm for the 0-1 MDKP by combining a genetic algorithm implementation with tabu search. They tested their heuristic on a set of standard test problems, but the results were not

computationally competitive with the results obtained using other heuristic methods.

Hoff, Løkketangen, and Mittet (1996) presented a genetic algorithm for the 0-1 MDKP in which only feasible solutions were allowed. They found optimal solutions for 56 of the 57 test instances from the literature. They also applied their algorithm on 9 test instances used by Khuri, Bäck and Heitkötter (1994) and obtained better results for all of them. They also compared their results to those obtained by Thiel and Voss (1993). When compared to their pure genetic algorithm approach, they got better results for 44 of the 57 instances. With the genetic algorithm-tabu search approach of Thiel and Voss (1993), they got slightly better average results than Hoff, Løkketangen, and Mittet. One reason for this is that genetic algorithms have a problem on focusing on some types of local optima, but these are for these cases easily found by the tabu search component.

Chu and Beasley (1998) presented a heuristic based upon genetic algorithms for solving the 0-1 MDKP. It appears to be the most successful genetic algorithm to date for the 0-1 MDKP. In their heuristic, a heuristic operator which utilises problem-specific knowledge is incorporated into the standard genetic algorithm approach.

They initially tested the heuristic on 55 standard test problems and showed that it finds the optimal solution for all of them. However, these problems were solved in very short computing times using CPLEX, and Chu and Beasley generated a set of large 0-1 MDKP instances using the procedure suggested by Freville and Plateau (1994). These data contained randomly generated 0-1 MDKP 's with different numbers of constraints (m = 5, 10, 30), variables (n = 100, 250, 500), and different tightness ratios (α = 0.25, 0.5, 0.75). The coefficients c_j were correlated to a_{ij} making the problems in

general more difficult to solve than uncorrelated problems, see (Gavish and Pirkul (1985), Pirkul (1987)). There were 10 problem instances for each combination of m, n, and α, and 270 test problems in total.

Chu and Beasley solved 270 problems that they generated using both CPLEX and their genetic algorithm heuristic. They solved 30 of these problems to optimality using CPLEX and for the remaining 240 problems, they terminated CPLEX whenever tree memory exceeds 42 Mb or after 1800 CPU seconds. The quality of the solutions generated were measured by the percentage gap between the best solution value found and the optimal value of the LP relaxation(100 * (optimal LP value-best solution value)/(optimal LP value)). The average percentage gap (over all 270 test problems) was much lower for their heuristic (0.54%) than for CPLEX (3.14%).

They also compared the performance of their heuristic with the heuristic of Magazine and Oguz (1984), the heuristic of Volgenant and Zoon (1990) and the heuristic of Pirkul (1987) on the newly generated problems and reported that their heuristic was superior over these heuristic methods in terms of the solution quality. However, in terms of computation time, their heuristic required much more computation time than that required by the other heuristics.

Günther R. Raidl (1998) improved a genetic algorithm for solving the 0–1 MDKP by introducing a pre-optimized initial population, a repair and a local improvement operator. These new techniques were based on the solution of the linear programming relaxation of the 0-1 MDKP. The pre-optimization of the initial population and the repair and local improvement operators all contained random elements for retaining population diversity. The algorithm was tested on standard large-sized test data proposed by Chu and Beasley (1998) and compared to the genetic algorithm from Chu and Beasley (1998).

They showed that most of the time the new genetic algorithm converged much faster to better solutions, especially for large problems.

Barake, Chardaire and McKeown (2001) presented the application of a new technique that they have proposed, known as PROBE(Population Reinforced Optimization Based Exploration), to the 0-1 MDKP. PROBE is a population based metaheuristic that directs optimization algorithms towards good regions of the search space using some ideas from genetic algorithms. They tested their algorithm on the 270 test problems generated by Chu and Beasley (1998). They showed that for problems with a small number of constraints and variables the genetic algorithm of Chu and Beasley was slightly better than PROBE in terms of solution quality. For problems with a large number of constraints, PROBE gave slightly better solutions than the genetic algorithm but the PROBE computing times were slightly larger than the genetic algorithm computing times on average.

Osorio, Hammer and Glover (2000) used surrogate analysis and constraint pairing for solving the 0-1 MDKP to fix some variables to zero and to separate the rest into two groups, those that tend to be zero and those that tend to be one, in an optimal integer solution. They generated logic cuts based on their analysis using an initial feasible integer solution, before solving the problem with branch and bound.

In order to test the efficiency of the logic cuts generated, they presented two experiments and solved the problems using CPLEX v6.5.2 both with and without the addition of their procedure. For the first experiment, they used 270 large-sized test problems from the OR-library, and for the second one, they generated a new set of test problems which are harder.

They reported that in the first experiment, the average objective value for the genetic algorithm(Chu and Beasley(1998)) was 120153.1, for CPLEX, 120162.5 and for their procedure, 120167.2. Their procedure was able to solve 100 problems to optimality, while CPLEX alone could solve only 95 and in the rest of the problems, CPLEX usually terminated because the tree size memory (250 Mb) was exceeded. Their procedure kept the tree size memory within the limits for a larger number of instances and finished because the time limit (10800 sec) was reached.

For the second experiment, they generated problems with 5 constraints and 100, 250 and 500 variables, and examined tightness values of 0.25, 0.5 and 0.75. They showed that CPLEX performed much better on average with the addition of their procedure and problems were solved by constructing smaller search trees.

Vasquez and Hao (2001) presented a hybrid approach for the 0–1 MDKP. The proposed approach combines linear programming and tabu search. They tested their approach on the 56 test problems from OR-library for which n varies from 6 to 105 and m from 2 to 30. They showed that their approach finds the optimal value in an average time of 1 second. They also tested their approach on the 90 largest test problems (n = 500) of OR-library . They compared the best results they obtained for these problems to those obtained by Chu and Beasley (1998), by Osorio, Hammer and Glover (2000) and by the MIP solver CPLEX v6.5.2 alone. They reported that their approach outperforms all the other algorithms except for the instances m = 5 and α = 0.75 (10 problems).

Gabrel and Minoux (2002) described a constraint-generation procedure for systematically building strengthened formulations for the 0-1 MDKP, which

is based on a new scheme for exact separation of extended cover inequalities for knapsack constraints.

In order to check the relevance of the separation scheme, they made experiments on a series of 80 randomly generated instances of sizes n = 120 and m = 30, 60, 90; n = 150 and m = 30, 75, 100; n = 180 and m = 40, 60. They also used instances(with 100 variables, 5 and 10 constraints) from the OR-library defined by Chu and Beasley (1998) .

In a majority of the test problems solved, the computing times obtained by the standard branch and bound procedure of CPLEX applied to the strengthened formulation (without automatic cover inequality generation) were improved over the time taken by CPLEX in MIP mode with automatic cover inequality generation. In addition, they showed that the fraction of total computing time taken by the constraint generation procedure was on average, less than 5% of total computation time. They also tried to strengthen the extended cover inequalities generated by constraint-generation procedure by sequential lifting and reported that 97% of the inequalities could not be further strengthened.

C h a p t e r 3

The Partitioning Algorithm

In this chapter, firstly the search based cuts are described and the integrality gap is defined. Then, the search and cut algorithm is introduced and how to partition a problem for generating a sub-problem is explained. The partitioning algorithm and an application of this algorithm to a small example are presented. Finally, the computational results of the application of partitioning algorithm on multidimensional 0-1 knapsack problems are given and the one step partitioning algorithm is explained.

3.1 Description of the Search Based Cuts

Consider the following 0-1 integer programming problem:

$$\text{maximize} \sum_{j=1}^{n} c_j x_j , \tag{1}$$

$$\text{subject to} \sum_{j=1}^{n} a_{ij} x_j \leq b_i , \; i = 1, \ldots , m, \tag{2}$$

$$x_j \in \{0, 1\}, \; j = 1, \ldots , n. \tag{3}$$

Let $X^* = (x_1^*, x_2^*, \ldots ,x_n^*)$ denote a solution to the linear programming relaxation of this problem. We generate $X_{int} = (x_1', x_2', \ldots ,x_n')$ as our candidate solution by the following: $x_j' = 1$ if $x_j^* \geq 0.5$, $x_j' = 0$ if $x_j^* < 0.5$. [*]

[*] The cuts and algorithms described in this chapter are from *Oguz, 2002, "Search and Cut: New Class of Cutting Planes for 0-1 Programming".*

The following equality $\sum_{j \in S_1} x_j' + \sum_{j \in S_2} (1 - x_j') = n$ holds for $S_1 = \{ j \mid x_j' = 1\}$ and $S_2 = \{ j \mid x_j' = 0\}$.

The candidate solution (X_{int}) to our problem can be feasible or infeasible. Suppose that it is infeasible and there exists a feasible solution which is represented by $X = (x_1, x_2, \ldots, x_n)$ to the same problem, then

$$\sum_{j \in S_1} x_j + \sum_{j \in S_2} (1 - x_j) \leq n - 1 \qquad (4)$$

must hold, because at least one x_j must be different than x_j' for $j \in S_1 \cup S_2$.

Now, consider carrying out a one dimensional search on the vector X_{int}. That is, if x_i' is equal to 1, we will change its value to zero while keeping all the other components of the vector X_{int} at their current values and then check the resulting vector for feasibility. If x_i' is equal to 0, we will change its value to one and check the resulting vector for feasibility. If we repeat this process for $i = 1, 2, \ldots, n$; either we can find one or several feasible solutions or we can't find any. If we find a feasible solution we can compare the objective value associated with this solution with the maximum objective value found so far and keep a record of the best solution encountered. After completing one dimensional search we can reduce the right hand side constant of the inequality (4) to $n - 2$.

If t dimensional searches have been done for $t = 1, \ldots, k$, in the same way described above, then the inequality given in (4) is a valid inequality for the problem given in (1)-(3) with the right hand side value of $n - k - 1$.

The difference $\delta = n - \sum_{j=1}^{n} \max [x_j^*, (1 - x_j^*)]$ is called the *integrality gap*

associated with the vector X*. If X* is integer, then the integrality gap is zero, taking its smallest possible value. If all $x_j^* = 1/2$, j = 1, . . . , n, then the integrality gap is at its largest possible value, *n*/2. For a valid inequality resulting from a search with depth *k* to be a cut, we must have $k \geq \lfloor \delta \rfloor$.

Now, we can give an algorithm based on the search based cuts described above.

The Search and Cut Algorithm

1. Set the value of the incumbent solution z_{inc} to $-\infty$.

2. Solve the LP relaxation of the problem given in (1)-(3) plus any cuts generated so far. Stop if the problem is infeasible, or the objective function value is less than z_{inc} +1, concluding that the solution vector associated with z_{inc} is optimal. Otherwise go to *3*.

3. Compute the value of the integrality gap (δ) and set $k \geq \lfloor \delta \rfloor$.

4. Carry out *t* dimensional searches on the vector X_{int} for t = 1, ... k. If a feasible solution is found, compare the objective value associated with this solution with z_{inc}. If it is larger than z_{inc}, then set z_{inc} equal to this new value.

5. Append a new cut as $\sum_{j \in S_1} x_j + \sum_{j \in S_2} (1 - x_j) \leq n - k - 1$, and go to *2*.

3.2 Partitioning the Problem

Since it depends on enumerative search to generate a cut, the efficiency of the search and cut algorithm will decrease as the value of integrality gap (δ) gets larger. For this reason, if integrality gap is large, we suggest to partition the problem into smaller sub-problems instead of doing enumerative search to generate a cut.

Suppose that we have a subset of T of the components of the vector X* such that:

$$T \subset N = \{1, 2, \ldots, n\}, \text{ and } |T| - \sum_{j \in T} \max [x_j^*, (1 - x_j^*)] < 1$$

Then if,

$$\text{maximize} \sum_{j \in N \setminus T} c_j x_j , \tag{5}$$

$$\text{subject to} \sum_{j \in N \setminus T} a_{ij} x_j \leq b_i - \sum_{j \in T} a_{ij} x_j' \quad , i = 1, \ldots, m, \tag{6}$$

$$x_j \in \{0, 1\}, j \in N \setminus T \tag{7}$$

has no solution , the following inequality is valid for the solution set of our problem:

$$\sum_{j \in T \cap S_1} x_j + \sum_{j \in T \cap S_2} (1 - x_j) \leq |T| - 1$$

Again, it is possible to improve the quality of this cut by carrying out one dimensional search on the components of X_{int} in the set T.

Algorithm to obtain T:

1. Order x_j^* with increasing max $[x_j^*, (1 - x_j^*)]$ values so that $|x_{j(i)}^* - 1/2| \leq |x_{j(i+1)}^* - 1/2|$. Set Q = { }. (The subscript j(i) means that the j^{th} component is in the i^{th} position in the ordering.)

2. Set k = 1.

3. If $n - \sum_{i=k+1}^{n} \max [x_{j(i)}^*, (1 - x_{j(i)}^*)] \geq 1$ is true,

set $Q = Q \cup \{j(k)\}$, k = k + 1, n = n - 1 and repeat *3*. Otherwise go to *4*.

4. Set T = N \ Q and stop.

The Partitioning Algorithm

Let's call the problem defined by the equations (5)-(7) $P(T_0)$ and the set T on which $P(T_0)$ is based T_0. $P(T_0)$ is our initial sub-problem. This sub-problem may still have a large number of variables. Therefore, instead of solving it with the search and cut algorithm, we suggest to solve this problem with the aid of new cuts we are proposing.

First, we solve the LP relaxation of $P(T_0)$, find the candidate solution vector and the integrality gap (δ) associated with this vector. Then, we apply the above algorithm to obtain T_1 and $P(T_1)$. We have,

$$T_1 \subset N \setminus T_0 \quad \text{and } |T_1| - \sum_{j \in T_1} \max [x_j^*, (1 - x_j^*)] < 1.$$

And $P(T_1)$ is defined as,

$$\text{maximize} \sum_{j \in N \setminus T_0 \cup T_1} c_j x_j ,$$

$$\text{subject to} \sum_{j \in N \setminus T_0 \cup T_1} a_{ij} x_j \leq b_i - \sum_{j \in T_0 \cup T_1} a_{ij} x_j' \quad , i = 1, \ldots, m,$$

$$x_j \in \{0, 1\}, j \in N \setminus T_0 \cup T_1$$

Then, we solve $P(T_1)$,and when either it is found infeasible or an optimal solution to it is found, we add a cut to $P(T_0)$. This cut will be

$$\sum_{j \in T_1 \cap S_1} x_j + \sum_{j \in T_1 \cap S_2} (1 - x_j) \leq |T_1| - 1$$

If $P(T_1)$ has a large number of variables, instead of solving it directly, we can repartition it to $P(T_2)$ and attempt to solve $P(T_2)$ in order to add a cut to $P(T_1)$.

Now, suppose that we have used partitioning until we reached $P(T_i)$ and we are able to solve it easily because it doesn't have a lot of variables. After solving $P(T_i)$ we will go back to $P(T_{i-1})$ and add a new cut to it. Then, we will solve the LP relaxation of $P(T_{i-1})$ plus the cuts added so far and redefine $P(T_i)$ using the solution of the LP relaxation of the new $P(T_{i-1})$. This process continues until $P(T_{i-1})$ is solved and a cut is generated for $P(T_{i-2})$. When either an optimal solution is found to $P(T_{i-2})$ or it is declared infeasible, we will append a cut to $P(T_{i-3})$ and continue to move back and forth recursively until $P(T_0)$ is solved by the aid of new cuts generated. After $P(T_0)$ is solved, we will add a new cut to our original problem, solve its LP relaxation and generate a new candidate solution. Then, we will redefine T_0 and start partitioning again to solve $P(T_0)$. After $P(T_0)$ is resolved, we will append one more cut to our original problem. The algorithm stops when sufficient cuts are added to solve the original problem.

3.3 An example

Consider the following 0-1 MDKP,

$$\text{maximize } 167 x_1 + 207 x_2 + 48 x_3 + 142 x_4 + 112 x_5$$
$$\text{subject to } 121 x_1 + 46 x_2 + 17 x_3 + 91 x_4 + 85 x_5 \leq 72$$
$$31 x_1 + 330 x_2 + 8 x_3 + 77 x_4 + 22 x_5 \leq 93$$
$$x_j \in \{0, 1\}, j = 1, \ldots, 5$$

The LP relaxation of this problem has the solution: (0.369832, 0.222834, 1, 0, 0) with the objective value, 155.8885. The candidate integer solution associated with this vector is (0, 0, 1, 0, 0). This solution is feasible and becomes the first incumbent solution with $Z_{inc}^1 = 48$. The integrality gap (δ) is 0.5926657. Since $\delta < 1$, we add the following cut to the problem:

$$- x_1 - x_2 + x_3 - x_4 - x_5 \leq 0$$

When we resolve the LP relaxation of our original problem together with the new cut we added, the solution is (0, 0.224820, 0.791843, 0, 0.567023) with the objective value, 148.0528. The candidate integer solution vector is (0, 0, 1, 0, 1) which is infeasible. The integrality gap associated with the solution is 0.8659547 and we append one more cut to the original problem:

$$- x_1 - x_2 + x_3 - x_4 + x_5 \leq 1$$

Now our problem becomes,

maximize $167\ x_1 + 207\ x_2 + 48\ x_3 + 142\ x_4 + 112\ x_5$

subject to $121\ x_1 + 46\ x_2 + 17\ x_3 + 91\ x_4 + 85\ x_5 \leq 72$

$$31\ x_1 + 330\ x_2 + 8\ x_3 + 77\ x_4 + 22\ x_5 \leq 93$$

$$- x_1 - x_2 + x_3 - x_4 - x_5 \leq 0$$

$$- x_1 - x_2 + x_3 - x_4 + x_5 \leq 1$$

$$x_j \in \{0, 1\},\ j = 1, \ldots, 5.$$

The solution to the LP relaxation of the above problem is (0.049426, 0.225066, 0.774492, 0, 0.5) with the objective value, 148.0185 and the candidate solution vector, (0, 0, 1, 0, 1) which is infeasible. The integrality gap associated with the solution to the LP relaxation is 1 and we partition the problem with $T_0 = 1, 4$; fixing $x_1 = 0$, $x_4 = 0$.

We define $P(T_0)$ as,

maximize $207\ x_2 + 48\ x_3 + 112\ x_5$

subject to $46\ x_2 + 17\ x_3 + 85\ x_5 \leq 72$

$$330\ x_2 + 8\ x_3 + 22\ x_5 \leq 93$$

$$- x_2 + x_3 - x_5 \leq 0$$

$$- x_2 + x_3 + x_5 \leq 1$$

$$x_j \in \{0, 1\},\ j = 2, 3, 5.$$

and solve the LP relaxation. The solution is (0.226676, 0.627862, 0.598815). The candidate solution for this sub-problem is (0, 1, 1) and it is infeasible. Since the integrality gap is 1, we repartition the problem with $T_1 = 2$, fixing $x_2 = 0$.

Now, $P(T_1)$ is,

maximize $48\ x_3 + 112\ x_5$

subject to $17\ x_3 + 85\ x_5 \leq 72$

$8\ x_3 + 22\ x_5 \leq 93$

$x_3 - x_5 \leq 0$

$x_3 + x_5 \leq 1$

$x_j \in \{0, 1\}, j = 3, 5.$

The solution to the LP relaxation of the above problem is (0.191176 , 0.808824). The candidate solution is (0, 1) which is infeasible. The integrality gap is 0.3823529 and we add the following cut to $P(T_1)$:

$$- x_3 + x_5 \leq 0$$

At this point $P(T_1)$ becomes,

maximize $48\ x_3 + 112\ x_5$

subject to $17\ x_3 + 85\ x_5 \leq 72$

$8\ x_3 + 22\ x_5 \leq 93$

$x_3 - x_5 \leq 0$

$x_3 + x_5 \leq 1$

$- x_3 + x_5 \leq 0$

$x_j \in \{0, 1\}, j = 3, 5.$

The solution to the LP relaxation of the above problem is (0.5, 0.5) with the associated candidate solution (1, 1) which is infeasible. Since the integrality

gap associated with the solution to the LP relaxation is 1, we repartition the problem with T_2 =3, fixing $x_3 = 1$.

Now, $P(T_2)$ is defined as,

maximize $112\ x_5$

subject to $85\ x_5 \leq 55$

$22\ x_5 \leq 85$

$x_5 \geq 1$

$x_5 \leq 0$

$x_5 \in \{0, 1\}$.

Since the above problem is infeasible, we add $x_3 \leq 0$ to $P(T_1)$.

The solution to the LP relaxation of the new $P(T_1)$ is (0, 0). Since the solution is integer with objective value equal to 0, we add the cut $x_2 \geq 1$ to $P(T_0)$.

The new $P(T_0)$ is infeasible, so we add the cut $x_1 + x_4 \geq 1$ to the original problem.

Now our original problem becomes,

maximize $167\ x_1 + 207\ x_2 + 48\ x_3 + 142\ x_4 + 112\ x_5$

subject to $121\ x_1 + 46\ x_2 + 17\ x_3 + 91\ x_4 + 85\ x_5 \leq 72$

$31\ x_1 + 330\ x_2 + 8\ x_3 + 77\ x_4 + 22\ x_5 \leq 93$

$-x_1 - x_2 + x_3 - x_4 - x_5 \leq 0$

$-x_1 - x_2 + x_3 - x_4 + x_5 \leq 1$

$x_1 + x_4 \geq 1$

$x_j \in \{0, 1\}, j = 1, \ldots, 5.$

The above problem is infeasible, therefore we conclude that $Z_{inc}^{1} = 48$ is optimal.

3.4 Computational Experimentation Results

In order to check the efficiency of the partitioning algorithm, we have designed 3 computational experiments; the first experiment involved the application of the algorithm to small-sized test problems from the literature, the second experiment involved the application of the algorithm to randomly generated 0-1 MDKP with different numbers of variables (n) and different numbers of constraints (m) and finding out the largest values of n and m for which the algorithm could solve problems to optimality, and the third experiment involved the application of the algorithm to large-sized test problems from the literature.

For the first experiment, we have chosen 30 small-sized test problems (Shih (1979)) from the literature which have been used in the computational experimentations of almost all of the algorithms designed for solving the 0-1 MDKP. We were able to solve all of these problems to optimality using the partitioning algorithm, coded in C language. Our computational results regarding these problems are shown in the table below,

Table 3-4: Computational results for standard test problems (Shih (1979))

Problem name	# of var.	# of cons.	optimal solution value	iteration	time(s)
weish01	30	5	4554	250	1.78
weish02	30	5	4536	144	1.15
weish03	30	5	4115	90	0.64
weish04	30	5	4561	15	0.1
weish05	30	5	4514	6	0.06
weish06	40	5	5557	142	1.28
weish07	40	5	5567	121	1.03
weish08	40	5	5605	103	0.92
weish09	40	5	5246	6	0.04
weish10	50	5	6339	824	9.16
weish11	50	5	5643	209	2.04
weish12	50	5	6339	190	1.77
weish13	50	5	6159	315	3.44
weish14	60	5	6954	317	3.04
weish15	60	5	7486	76	0.75
weish16	60	5	7289	76	0.75
weish17	60	5	8633	94	0.97
weish18	70	5	9580	173	1.8
weish19	70	5	7698	318	3.74
weish20	70	5	9450	129	1.28
weish21	70	5	9074	98	1.19
weish22	80	5	8947	241	2.67
weish23	80	5	8344	252	2.91
weish24	80	5	10220	87	0.97
weish25	80	5	9939	150	1.61
weish26	90	5	9584	483	6.77
weish27	90	5	9819	28	0.28
weish28	90	5	9492	65	0.7
weish29	90	5	9410	40	0.48
weish30	90	5	11191	10	0.09
			average	130	0.94

For the second experiment we randomly generated 0-1 MDKP as follows:

- Each coefficient of the constraint matrix a_{ij} (i = 1, . . . , m and j = 1, . . . , n) and each coefficient of the objective function c_j (j = 1, . . . , n) is an integer randomly chosen between 1 and 1000.

- The right-hand side coefficients (b_i 's) are generated using

$$b_i = 0.5 \sum_{j=1}^{n} a_{ij}$$

We made experiments for different values of *m* and *n* and found out that if we set *m* at 5, the largest value of *n* was 90 for which the algorithm could solve problems to optimality, this value was 40 if we set *m* at 10 and 25 if we set *m* at 20. The reason for this is that we limited our algorithm to partitioning the original problem at most 10 times, that is if the first sub-problem is $P(T_0)$, then the algorithm stops when it reaches $P(T_{10})$ (see section 3.2), due to computational and programming difficulties and to solve larger problems requires using the partitioning procedure more than 10 times.

Note that, the largest values of *m* and *n* for which the algorithm solves problems to optimality will be different if one generates easier or harder problems than the ones we generated using the procedure described above.

Since the easiest large-sized test problems in the literature are harder than the problems we generated, we conclude that with its current design the partitioning algorithm is not an efficient algorithm to solve large-sized 0-1 MDKP. Therefore, we tried to modify it in such a way that it does not require to use the partitioning procedure a large number of times to solve a

problem. For this purpose, we chose to take advantage of the IP solver of CPLEX to solve sub-problems. Our computational experiments with larger problems showed us that this may be an efficient way to solve large-sized 0-1 MDKP. As it can be seen in the Chapter 5, by this way we are able to find better quality solutions than CPLEX to 0-1 MDKP with large number of variables and few constraints.

The description of the modified partitioning algorithm is given in the next section.

3.5 One Step Partitioning Algorithm

Consider the following 0-1 integer programming problem:

$$\text{maximize} \sum_{j=1}^{n} c_j x_j ,$$

$$\text{subject to} \sum_{j=1}^{n} a_{ij} x_j \leq b_i , \; i = 1, \ldots , m,$$

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

1. solve the LP relaxation of the above problem plus any cuts generated so far. Let $X^* = (x_1^*, x_2^*, \ldots ,x_n^*)$ denote the solution to the LP relaxation. Stop, if either the problem is infeasible or the objective function value for X^* is less than the best feasible integer solution value found so far (Z_{inc}) plus one, declaring Z_{inc} as optimal.

2. generate $X_{int} = (x_1', x_2', \ldots ,x_n')$ as candidate solution by the following:

$x_j' = 1$ if $x_j^* \geq 0.5$, $x_j' = 0$ if $x_j^* < 0.5$.

Check if this solution is feasible and update the incumbent solution if it is feasible and the objective function value associated with it is greater than Z_{inc}.

3. calculate the value of the integrality gap associated with the solution to the LP relaxation as follows:

$$\delta = n - \sum_{j=1}^{n} \max [x_j^*, (1 - x_j^*)]$$

3.a. If the value of the integrality gap (δ) is less than 1, add the following cut to the problem:

$$\sum_{j \in S_1} x_j + \sum_{j \in S_2} (1 - x_j) \leq n - 1$$

where $S_1 = \{ j \mid x_j' = 1\}$ and $S_2 = \{ j \mid x_j' = 0\}$.

Go to **1.**

3.b. If the value of the integrality gap (δ) is greater than 1,

3.b.1. partition the problem with T defined as:

$T \subset N = \{1, 2, \ldots, n\}$, and $j \in T$ if $\max [x_j^*, (1 - x_j^*)] = 1$, $j = 1, \ldots, n$,

and solve the associated problem P(T):

$$\text{maximize} \sum_{j \in N \setminus T} c_j x_j ,$$

$$\text{subject to} \sum_{j \in N \setminus T} a_{ij} x_j \leq b_i - \sum_{j \in T} a_{ij} x_j' \quad , i = 1, \ldots, m,$$

$$x_j \in \{0, 1\}, j \in N \setminus T$$

by CPLEX v8.1 in MIP mode.

3.b.2. check the candidate solution for feasibility and if it is feasible and the objective function value associated with it is greater than Z_{inc}, update the incumbent solution.

3.b.3. append a new cut to the original problem as,

$$\sum_{j \in T \cap S_1} x_j + \sum_{j \in T \cap S_2} (1 - x_j) \leq |T| - 1$$

where $S_1 = \{ j \mid x_j' = 1 \}$ and $S_2 = \{ j \mid x_j' = 0 \}$.

Go to **1.**

As the number of iterations increases, the computational time increases rapidly at each iteration because the number of constraints increases and the sub-problems get larger. Since we use the IP solver of CPLEX to solve sub-problems, the computational time of the algorithm rapidly increases as the sub-problems get larger.

Chapter 4

Computational Analysis of the Search Based Cuts

4.1 Computational Analysis of the Search Based Cuts Applied to the 0-1 MDKP

We used the multidimensional 0-1 knapsack problems generated by Chu and Beasley (1998) for the computational analysis of the search based cuts. We made two experiments. In the first experiment, we fixed the number of constraints and for different numbers of variables we calculated the absolute reduction in the objective value per cut after 100 cuts (generated by the one step partitioning algorithm) are added to the original problem. We used 10 problems for each combination of m and n, while m is the number of constraints and n is the number of variables. The results for the first experiment are as follows:

Table 4-1a: Computational results for m=5, α=0.5 and n=100 OR-library instances (see [6])

pb	z(1)	z(2)	|z(1)-z(2)| / 100
Mkpcb1-11	42939.52	42830.36	1.0916
Mkpcb1-12	42706.7	42613.85	0.9285
Mkpcb1-13	42165.19	42095.35	0.6984
Mkpcb1-14	45347.07	45234.88	1.1219
Mkpcb1-15	42434.12	42328.64	1.0548
Mkpcb1-16	43082.23	42988.35	0.9388
Mkpcb1-17	42190.6	42069.72	1.2088
Mkpcb1-18	45265.47	45171.75	0.9372
Mkpcb1-19	43567.49	43507.01	0.6048
Mkpcb1-20	44796.63	44680.97	1.1566
		average	0.97414

Table 4-1b: Computational results for m=5, α=0.5 and n=250 OR-library instances (see [6])

pb	z(1)	z(2)	|z(1)-z(2)| / 100
mkpcb2-11	109220.6	109186.6	0.340
mkpcb2-12	109960.3	109917.4	0.429
mkpcb2-13	108648.8	108607.1	0.417
mkpcb2-14	109510.8	109472.7	0.381
mkpcb2-15	110834.2	110805.7	0.285
mkpcb2-16	110366.8	110336.1	0.307
mkpcb2-17	109152.6	109126	0.266
mkpcb2-18	109137.7	109113.1	0.246
mkpcb2-19	110123.1	110075.3	0.478
mkpcb2-20	107162.1	107136	0.261
		average	0.341

Table 4-1c: Computational results for m=5, α=0.5 and n=500 OR-library instances (see [6])

pb	z(1)	z(2)	|z(1)-z(2)| / 100
mkpcb3-11	218500.1	218483.1	0.170
mkpcb3-12	221272.4	221253.3	0.191
mkpcb3-13	217615.8	217600.9	0.149
mkpcb3-14	223653.2	223634.9	0.183
mkpcb3-15	219067.5	219042.9	0.246
mkpcb3-16	220617	220602.2	0.148
mkpcb3-17	220076.5	220054.4	0.221
mkpcb3-18	218282.7	218266.5	0.162
mkpcb3-19	217059.9	217043.3	0.166
mkpcb3-20	219812.8	219793.6	0.192
		average	0.1828

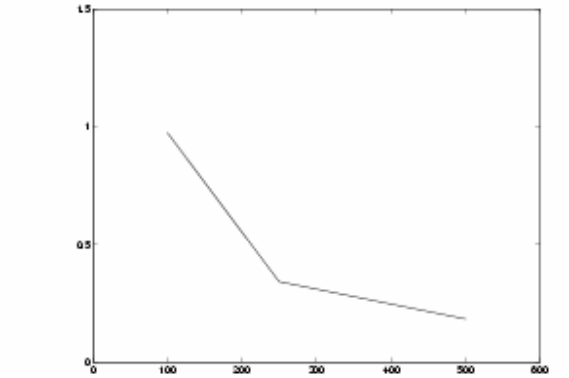
The first column of the tables above shows the problem name, in the second column the objective value for the LP relaxation of the original problem (z(1)) is given for each problem, z(2) is the objective value for the LP relaxation of the original problem plus 100 cuts generated by the one step partitioning algorithm and finally |z(1)- z(2)| /100 (absolute change in the objective value per cut) is given in column 4.

As seen from the tables above we fixed *m* at 5 and made computational experiments for *n* = 100, 250, 500. The average value of the absolute reduction in the objective value per cut after 100 cuts are added is 0.97414 for *n* = 100, 0.341 for *n* = 250 and 0.1828 for *n* = 500. These results are shown in the table and depicted in the graph below,

Table 4-1d: Average value of absolute reduction in objective value per cut for m=5, α=0.5

n	|z(1)- z(2)| / 100
100	0.97414
250	0.341
500	0.1828

Figure 4-1a: Absolute reduction in the objective value per cut as *n* increases



As it can be seen from the table, the absolute reduction in the objective value per cut after 100 cuts are added decreases as the number of variables increases. That is, the effectiveness of the search based cuts decrease as the number of variables increases.

In the second experiment, we fixed the number of variables and for different numbers of constraints we calculated the absolute reduction in the objective value per cut after 100 cuts (generated by the one step partitioning algorithm) are added to the original problem. We used 10 problems for each

combination of *m* and *n,* while *m* is the number of constraints and *n* is the number of variables. The results for the second experiment are as follows:

Table 4-1e: Computational results for n=100, α=0.5 and m=5 OR-library instances (see [6])

pb	z(1)	z(2)	|z(1)-z(2)| / 100
mkpcb1-11	42939.52	42830.36	1.0916
mkpcb1-12	42706.70	42613.85	0.9285
mkpcb1-13	42165.19	42095.35	0.6984
mkpcb1-14	45347.07	45234.88	1.1219
mkpcb1-15	42434.12	42328.64	1.0548
mkpcb1-16	43082.23	42988.35	0.9388
mkpcb1-17	42190.6	42069.72	1.2088
mkpcb1-18	45265.47	45171.75	0.9372
mkpcb1-19	43567.49	43507.01	0.6048
mkpcb1-20	44796.63	44680.97	1.1566
		average	0.97414

Table 4-1f: Computational results for n=100, α=0.5 and m=10 OR-library instances (see [6])

pb	z(1)	z(2)	|z(1)-z(2)| / 100
mkpcb4-11	41712.64	41622.68	0.8996
mkpcb4-12	42597.32	42504.68	0.9264
mkpcb4-13	42759.32	42643.13	1.1619
mkpcb4-14	45959.36	45862.18	0.9718
mkpcb4-15	42183.12	42076.02	1.0710
mkpcb4-16	43377.96	43265.22	1.1274
mkpcb4-17	43927.94	43820.5	1.0744
mkpcb4-18	43335.83	43220.3	1.1553
mkpcb4-19	42611.6	42494.93	1.1667
mkpcb4-20	41542.79	41433.04	1.0975
		average	1.0652

Table 4-1g: Computational results for n=100,α=0.5 and m=30 OR-library instances (see [6])

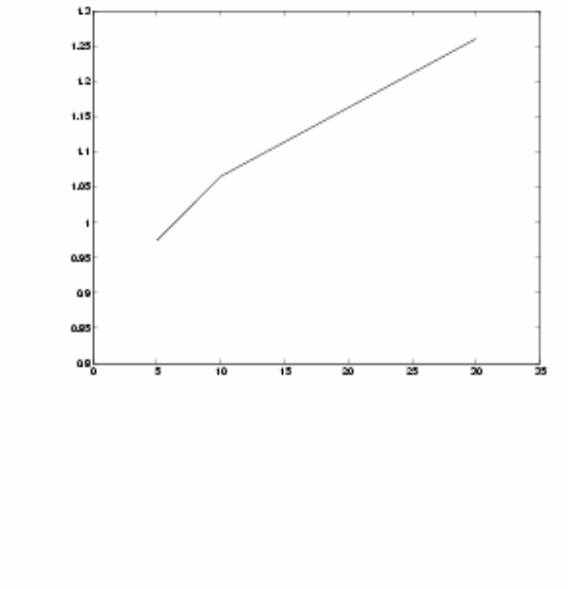
pb	z(1)	z(2)	|z(1)-z(2)| / 100
mkpcb7-11	41276.36	41173.16	1.0320
mkpcb7-12	41866.73	41740.82	1.2591
mkpcb7-13	42232.96	42124.72	1.0824
mkpcb7-14	41634.88	41510.52	1.2436
mkpcb7-15	41410.88	41323.66	0.8722
mkpcb7-16	41603.16	41468.19	1.3497
mkpcb7-17	41616.13	41500.70	1.1543
mkpcb7-18	43388.05	43207.30	1.8075
mkpcb7-19	42656.56	42533.10	1.2346
mkpcb7-20	42262.70	42105.70	1.5700
		average	1.26054

As seen from the tables above we fixed *n* at 100 and made computational experiments for *m* = 5, 10, 30. The average value of the absolute reduction in the objective value per cut after 100 cuts are added is 0.97414 for *m* = 5, 1.0652 for *m* = 10 and 1.26054 for *m* = 30. These results are shown in the table and depicted in the graph below,

Table 4-1h: Average value of absolute reduction in objective value per cut for n=100, α=0.5

m	|z(1)-z(2)| / 100
5	0.97414
10	1.0652
30	1.26054

Figure 4-1b: Absolute reduction in the objective value per cut as *m* increases



As it can be seen from the table, the absolute reduction in the objective value per cut after 100 cuts are added increases as the number of constraints increases. That is, the effectiveness of the search based cuts increase as the number of constraints increases.

4.2 Computational Analysis of the Search Based Cuts Applied to the Set Covering Problem

For the second part of the computational analysis of the search based cuts we used another well known combinatorial optimization problem which is the set covering problem. The set covering problem can be formulated as follows:

$$\text{minimize} \sum_{j=1}^{n} c_j x_j ,$$

$$\text{subject to} \sum_{j=1}^{n} a_{ij} x_j \geq 1 , \; i = 1, \ldots , m,$$

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

where $a_{ij} \in \{0, 1\}$ for i = 1, . . . , m; j = 1, . . . , n; and $c_j > 0$ for j = 1, . . .,n.

For our computational experiments, we randomly generated set covering problems by the following:

- Each coefficient of the constraint matrix a_{ij} (i = 1, . . . , m and j = 1, . . . ,n) is 1 with probability 0.5 and 0 with probability 0.5 .
- Each coefficient of the objective function c_j (j = 1, . . . , n) is 1.

In order to analyze the effectiveness of the search based cuts, we made two experiments. In the first experiment, we fixed the number of constraints and for different numbers of variables we calculated the absolute increase in the objective value per cut after 100 cuts (generated by the one step partitioning algorithm) are added to the original problem. We used 5 problems for each

combination of *m* and *n*, and the results for the first experiment are as follows:

Table 4-2a: Computational results for m=5 and n=100 set covering problems

pb	z(1)	z(2)	|z(1)-z(2)| / 100
sc1	1.25	1.661876	0.004119
sc2	1.25	1.705390	0.004554
sc3	1.25	1.735886	0.004859
sc4	1.25	1.710705	0.004607
sc5	1.25	1.771835	0.005218
		average	0.004671

Table 4-2b: Computational results for m=5 and n=150 set covering problems

Pb	z(1)	z(2)	|z(1)-z(2)| / 100
Sc6	1.25	1.580607	0.003306
Sc7	1.25	1.598384	0.003484
Sc8	1.25	1.597140	0.003471
Sc9	1.333333	1.538552	0.002052
Sc10	1.25	1.597115	0.003471
		average	0.003157

Table 4-2c: Computational results for m=5 and n=200 set covering problems

Pb	z(1)	z(2)	|z(1)-z(2)| / 100
Sc11	1.25	1.532358	0.002824
sc12	1.25	1.518742	0.002687
sc13	1.25	1.498249	0.002482
sc14	1.25	1.504127	0.002541
sc15	1.25	1.537601	0.002876
		average	0.002682

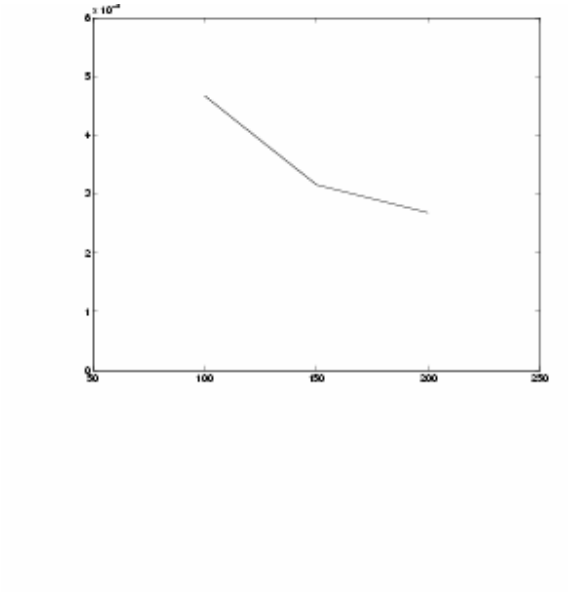
The first column of the tables above shows the problem name, in the second column the objective value for the LP relaxation of the original problem (z(1)) is given for each problem, z(2) is the objective value for the LP relaxation of the original problem plus 100 cuts generated by the one step partitioning algorithm and finally |z(1)- z(2)| /100 (absolute change in the objective value per cut) is given in column 4.

As seen from the tables above we fixed *m* at 5 and made computational experiments for *n* = 100, 150, 200. The average value of the absolute increase in the objective value per cut after 100 cuts are added is 0.004671 for *n* = 100, 0.003157 for *n* = 150 and 0.002682 for *n* = 200. These results are shown in the table and depicted in the graph below,

Table 4-2d: Average value of absolute increase in the objective value per cut for m=5

n	|z(1)- z(2)| / 100
100	0.004671
150	0.003157
200	0.002682

Figure 4-2: Absolute increase in the objective value per cut as *n* increases



As it can be seen from the table, the absolute increase in the objective value per cut after 100 cuts are added decreases as the number of variables increases. That is, the effectiveness of the search based cuts decreases as the number of variables increases as in the case of the 0-1 MDKP.

In the second experiment, we fixed the number of variables and for different numbers of constraints we calculated the absolute increase in the objective value per cut after 100 cuts (generated by the one step partitioning algorithm) are added to the original problem. We used 5 problems for each combination of *m* and *n,* and below are the results for the second experiment:

Table 4-2e: Computational results for n=100 and m=5 set covering problems

pb	z(1)	z(2)	|z(1)-z(2)| / 100
sc1	1.25	1.661876	0.004119
sc2	1.25	1.705390	0.004554
sc3	1.25	1.735886	0.004859
sc4	1.25	1.710705	0.004607
sc5	1.25	1.771835	0.005218
		Average	0.004671

Table 4-2f: Computational results for n=100 and m=10 set covering problems

pb	z(1)	z(2)	|z(1)-z(2)| /100
sc16	1.2	1.883667	0.006837
sc17	1.333333	1.889595	0.005563
sc18	1.25	1.764056	0.005141
sc19	1.2	1.821188	0.006212
sc20	1.285714	1.857548	0.005718
		Average	0.00589

As seen from the tables above, we fixed n at 100 and made computational experiments for m = 5, 10. The average value of the absolute increase in the objective value per cut after 100 cuts are added is 0.004671 for m = 5, and 0.00589 for m = 10. These results are shown in the table below,

Table 4-2g: Average value of absolute increase in the objective value per cut for n=100

m	|z(1)-z(2)| / 100
5	0.004671
10	0.00589

As it can be seen from the table, the absolute increase in the objective value per cut after 100 cuts are added increases as the number of constraints increases. That is, the effectiveness of the search based cuts increases as the number of constraints increases as in the case of the 0-1 MDKP.

Chapter 5

Application of the Partitioning Algorithm to the Multidimensional 0-1 Knapsack Problem

In order to check the efficiency of the partitioning algorithm, the one step partitioning algorithm described in Section 3.5 and coded in C language was applied to 60 randomly generated 0-1 MDKP and 30 large-sized 0-1 MDKP from the literature (Chu and Beasley (1998)). The computational results were compared with the implementation of CPLEX v8.1 in MIP mode and the results reported by Chu and Beasley (1998).

5.1 Problem generation

We generated a set of large 0-1 MDKP instances using the procedure suggested by Freville and Plateau (1994) and used by Chu and Beasley (1998) to generate the standard test problems in OR-library. The number of constraints *m* was set to 10 and the number of variables *n* was set to 1000 and 2000. We generated 30 problems for each m-n combination, giving a total of 60 problems.

Each problem instance is randomly generated as follows:

- Each coefficient of the constraint matrix a_{ij} (i = 1, . . . , m and j = 1, . . . , n) is an integer randomly chosen between 1 and 1000.

- The right-hand side coefficients (b_i 's) are found using

  $b_i = \alpha \sum_{j=1}^{n} a_{ij}$, where α is a tightness ratio and α = 0.25 for the first ten problems, α = 0.50 for the next ten problems and α = 0.75 for the remaining ten problems.

- The objective function coefficients (c_j 's) are correlated to a_{ij} and generated by:

  $$c_j = \sum_{j=1}^{n} a_{ij} /m + q_j/2$$

  where q_j is an integer randomly chosen between 1 and 1000. In general, correlated problems are more difficult to solve than uncorrelated problems (Gavish and Pirkul (1985), Pirkul (1987)).

5.2 Computational Results

The results obtained are shown on Tables 5-2a, 5-2b and 5-2c where, for each problem instance, the following information is given:

- *n* and *m*: number of variables and number of constraints.
- α: tightness ratio.
- z(lp): optimal value of the LP relaxation of 0-1 MDKP.
- initial value (partitioning algorithm): first feasible integer solution value found by one step partitioning algorithm.
- best value (partitioning algorithm): best feasible integer solution value found by one step partitioning algorithm in 225 iterations.

- iter.: the number of iterations required by one step partitioning algorithm to find the value in column 5.
- time (partitioning algorithm): computation time (in CPU seconds) required by one step partitioning algorithm to find the value in column 5.
- initial value (CPLEX): first feasible integer solution value found by CPLEX v8.1 in MIP mode.
- best value (CPLEX): best feasible integer solution value found by CPLEX v8.1 in MIP mode before memory tree size of 250 Mb is exceeded.
- time (CPLEX): computation time (in CPU seconds) required by CPLEX v8.1 in MIP mode until 250 Mb tree memory size is exceeded.

Table 5-2a: Computational results for n=500 and m=10 OR-library instances (see [6])

Prob. name	z(lp)	Partition Algorithm				CPLEX		
		initial	best	iter.	time	initial	best	time
mkpcb6-01	118019.5	117168	117779	72	9.6	116510	117712	603.98
mkpcb6-02	119437.3	118613	119165	172	135.2	118543	119158	650.96
mkpcb6-03	119405.7	118761	119194	210	443.87	118705	119211	532.23
mkpcb6-04	119066.1	118268	118813	156	82.35	118163	118813	631.62
mkpcb6-05	116698	115896	116509	188	171.01	115379	116423	557.77
mkpcb6-06	119710	118946	119463	188	197.7	118355	119448	577.63
mkpcb6-07	120033.3	119180	119777	168	114.77	118688	119777	576.10
mkpcb6-08	118545.7	117883	118323	128	63.12	117561	118266	611.74
mkpcb6-09	118001.6	117182	117776	170	85.31	116727	117779	561.42
mkpcb6-10	119440.6	118943	119163	160	80.45	118040	119191	613.79
mkpcb6-11	217552.9	217068	217341	190	336.27	216744	217312	633.38
mkpcb6-12	219255.2	218307	219030	164	255.73	218742	219027	536.46
mkpcb6-13	217987.8	217500	217792	146	57.75	216788	217792	537.22
mkpcb6-14	217040.7	216455	216851	154	75.83	215956	216851	678.05
mkpcb6-15	214010.3	212415	213830	200	229.71	213229	213827	590.07
mkpcb6-16	215261.3	214654	215041	224	260.1	214349	215034	582.36
mkpcb6-17	218109.2	217030	217899	194	307.45	217249	217875	622.92
mkpcb6-18	220175.6	219488	219984	102	27.49	218784	219965	606.38
mkpcb6-19	214561	213918	214329	212	449.42	213666	214312	508.71
mkpcb6-20	221083.6	220367	220852	204	186.4	219930	220846	596.21
mkpcb6-21	304555	304214	304334	156	70.76	303606	304344	582.11
mkpcb6-22	302553	301951	302333	176	49.2	302159	302326	461.06
mkpcb6-23	302581.5	300585	302416	82	12.65	302061	302386	581.82
mkpcb6-24	300956.7	300440	300747	128	56.9	300353	300719	455.72
mkpcb6-25	304584.7	303763	304349	196	371.82	303563	304346	548.79
mkpcb6-26	301952.5	301489	301767	148	81.18	301226	301742	537.43
mkpcb6-27	305139.7	304745	304949	128	29.1	304641	304949	538.1
mkpcb6-28	296636.6	296287	296441	158	50.21	296038	296441	551.31
mkpcb6-29	301547.6	301131	301326	164	211.01	300507	301353	606.75
mkpcb6-30	307250	306393	307072	218	138.5	306730	307038	564.21
			average	165	154.7		average	574.54

Table 5-2b: Computational results for n=1000 and m=10 randomly generated instances

Pb	α	z(lp)	Partition Algorithm				CPLEX		
			initial value	best value	iter.	time	initial value	best value	time
1	0.75	611260.7	610733	611104	136	79.73	610458	611033	654.36
2	0.75	603299.3	602722	603116	222	292.13	602872	603104	610.09
3	0.75	610002.4	609457	609824	190	117.3	609194	609803	589.94
4	0.75	613492	613063	613346	194	531.78	612456	613298	622.37
5	0.75	607796.8	607015	607637	210	373.53	606953	607588	641.2
6	0.75	611377.4	610879	611220	122	39.68	610527	611187	708.7
7	0.75	610871.2	610309	610694	224	496.85	610311	610692	643.92
8	0.75	612942.5	611738	612775	220	205.07	612089	612775	720.16
9	0.75	609705.2	607772	609552	192	272.24	609332	609517	715.11
10	0.75	610033.3	609417	609848	180	163.5	609324	609841	727.14
11	0.5	440789.7	439405	440606	198	283.72	439700	440561	715.01
12	0.5	435590.5	434665	435395	186	197.47	434741	435369	721.07
13	0.5	438296.2	437666	438118	174	164.03	437306	438065	752.99
14	0.5	441503.3	440737	441309	174	308.3	440288	441279	709.54
15	0.5	437539.3	436162	437371	156	139.95	436344	437334	869.74
16	0.5	440013.6	439282	439804	194	351.66	438524	439763	783.7
17	0.5	439492.5	438595	439309	174	101.88	438336	439247	762.93
18	0.5	439402.8	438253	439238	144	64.44	438806	439252	842.44
19	0.5	439067.8	437994	438863	198	386.46	438164	438808	795.31
20	0.5	439244	438606	439052	210	202.56	438350	438993	750.47
21	0.25	239227.7	238356	238975	128	47.36	238002	238936	735.02
22	0.25	237326.8	236201	237119	190	205.96	236229	237114	609.29
23	0.25	236982.5	236343	236797	212	368.34	235534	236720	712.88
24	0.25	239075.9	236964	238867	216	446.47	237708	238840	707.86
25	0.25	237069.9	236293	236855	160	88.65	235870	236806	780.25
26	0.25	238893.5	238040	238702	154	110.56	237622	238647	724.3
27	0.25	239103.9	238458	238867	170	179.93	237887	238804	648.67
28	0.25	238664	237857	238429	194	118.99	236818	238380	713.71
29	0.25	237864.1	237307	237658	204	137.08	236823	237626	777.48
30	0.25	237876.8	236893	237633	180	104.78	236798	237607	761.34
				average	184	219.35		average	716.9

Table 5-2c: Computational results for n=2000 and m=10 randomly generated instances

Pb	α	z(lp)	Partition Algorithm				CPLEX		
			initial value	Best value	iter.	time	initial value	Best value	time
31	0.75	1217182	1216465	1217041	192	263.05	1216480	1217032	909.09
32	0.75	1218369	1217382	1218217	106	55.14	1217596	1218162	967.93
33	0.75	1217932	1217381	1217775	225	569.67	1217183	1217773	997.9
34	0.75	1213535	1212941	1213368	196	301.3	1212806	1213319	898.43
35	0.75	1211499	1210662	1211360	168	308.95	1210750	1211292	989.95
36	0.75	1207297	1206708	1207146	198	617.6	1206424	1207084	949.73
37	0.75	1214952	1214411	1214805	190	132.25	1213721	1214744	985.58
38	0.75	1223266	1222627	1223132	174	297.32	1222621	1223078	1022.58
39	0.75	1212073	1211363	1211923	188	227.27	1211225	1211858	922.6
40	0.75	1212673	1212122	1212536	206	314.03	1211838	1212540	1211.06
41	0.5	877762.9	876999	877575	194	223.82	876561	877532	1126.78
42	0.5	877040.3	876306	876886	154	266.08	875810	876825	1103.52
43	0.5	878138	877731	877972	206	845.94	877047	877979	1007.01
44	0.5	876632.1	875584	876439	146	95.23	875721	876400	1107.26
45	0.5	872384.7	871817	872230	146	144.55	870854	872180	1219.81
46	0.5	869359.7	868817	869194	188	425.72	868595	869176	1071.94
47	0.5	874566.5	873179	874393	220	290.75	873285	874371	964.39
48	0.5	882383.8	881554	882192	164	108.44	881508	882164	949.76
49	0.5	873367.5	872514	873208	196	356.6	872206	873086	1176.84
50	0.5	874945	874552	874774	166	116.32	873514	874755	1089.03
51	0.25	476021.4	473825	475790	172	184.27	474626	475740	959.23
52	0.25	476476.5	474800	476286	72	18.55	475338	476215	1077.08
53	0.25	478913.5	478155	478693	196	283.68	477504	478641	941.51
54	0.25	478085.6	477089	477893	222	1402.7	476277	477890	939.87
55	0.25	472611.5	471963	472361	138	61.43	470667	472307	938.64
56	0.25	472503	471910	472300	168	153.44	470834	472267	904.93
57	0.25	474556.1	473957	474377	200	647.23	473167	474362	1239.40
58	0.25	480101.8	478685	479958	158	117.54	477861	479924	1044.22
59	0.25	474772.3	474115	474605	172	189.36	473334	474476	960.34
60	0.25	477055.2	476342	476864	224	475.96	475430	476761	959.49
				Average	179	316.47		average	1021.2

We terminated one step partitioning algorithm after 225 iterations and compared the best value of the objective function found so far with the best value of the objective function found by CPLEX before memory tree size of 250 Mb was exceeded. The reason for terminating the one step partitioning algorithm after 225 iterations is that after 225 iterations, the algorithm takes a lot of time at each iteration with a little probability of improving the best feasible solution value found so far.

For the first set of problems, which are from the literature, we also compared the best value of the objective function found by the one step partitioning algorithm with the genetic algorithm of Chu and Beasley (1998). The comparison is given in Tables 5-2d, 5-2e, 5-2f. Note that the one step partitioning algorithm does not take too much time. As it can be seen on the last rows of Tables 5-2a, 5-2b, 5-2c on average it takes 154.7 seconds for n = 500, 219.35 seconds for n = 1000 and 316.47 seconds for n = 2000 for the one step partitioning algorithm to find the best value in column 5.

The following information is given for the tables below:

- z(1): best feasible integer solution value found by one step partitioning algorithm in 225 iterations.
- z(2): best feasible integer solution value found by CPLEX v8.1 in MIP mode before memory tree size of 250 Mb was exceeded.
- z(3): best feasible integer solution value found by the genetic algorithm of Chu and Beasley (1998)

Table 5-2d: Comparison of one step partitioning alg. (z(1)) with Cplex (z(2)) and genetic algorithm of Chu and Beasley (z(3))

Prob. name	z(3)	z(2)	Z(1)	z(1)-z(3)	z(1)-z(2)
Mkpcb6-01	117726	117712	117779	53	67
Mkpcb6-02	119139	119158	119165	26	7
Mkpcb6-03	119159	119211	119194	35	-17
Mkpcb6-04	118802	118813	118813	11	0
Mkpcb6-05	116434	116423	116509	75	86
Mkpcb6-06	119454	119448	119463	9	15
Mkpcb6-07	119749	119777	119777	28	0
Mkpcb6-08	118288	118266	118323	35	57
Mkpcb6-09	117779	117779	117776	-3	-3
Mkpcb6-10	119125	119191	119163	38	-28
Mkpcb6-11	217318	217312	217341	23	29
Mkpcb6-12	219022	219027	219030	8	3
Mkpcb6-13	217772	217792	217792	20	0
Mkpcb6-14	216802	216851	216851	49	0
Mkpcb6-15	213809	213827	213830	21	3
Mkpcb6-16	215013	215034	215041	28	7
Mkpcb6-17	217896	217875	217899	3	24
Mkpcb6-18	219949	219965	219984	35	19
Mkpcb6-19	214332	214312	214329	-3	17
Mkpcb6-20	220833	220846	220852	19	6
Mkpcb6-21	304344	304344	304334	-10	-10
Mkpcb6-22	302332	302326	302333	1	7
Mkpcb6-23	302354	302386	302416	62	30
Mkpcb6-24	300743	300719	300747	4	28
Mkpcb6-25	304344	304346	304349	5	3
Mkpcb6-26	301730	301742	301767	37	25
Mkpcb6-27	304949	304949	304949	0	0
Mkpcb6-28	296437	296441	296441	4	0
Mkpcb6-29	301313	301353	301326	13	-27
Mkpcb6-30	307014	307038	307072	58	34
			Total	684	382
			Average	22.8	12.73

Table 5-2e: Comparison of one step partitioning alg. (z(1)) with Cplex (z(2)) for n=1000 randomly generated instances

Pb	Z(1)	z(2)	z(1)-z(2)
1	611104	611033	71
2	603116	603104	12
3	609824	609803	21
4	613346	613298	48
5	607637	607588	49
6	611220	611187	33
7	610694	610692	2
8	612775	612775	0
9	609552	609517	35
10	609848	609841	7
11	440606	440561	45
12	435395	435369	26
13	438118	438065	53
14	441309	441279	30
15	437371	437334	37
16	439804	439763	41
17	439309	439247	62
18	439238	439252	-14
19	438863	438808	55
20	439052	438993	59
21	238975	238936	39
22	237119	237114	5
23	236797	236720	77
24	238867	238840	27
25	236855	236806	49
26	238702	238647	55
27	238867	238804	63
28	238429	238380	49
29	237658	237626	32
30	237633	237607	26
		Total	1094
		Average	36.47

Table 5-2f: Comparison of one step partitioning alg. (z(1)) with Cplex (z(2)) for n=2000 randomly generated instances

Pb	Z(1)	z(2)	z(1)-z(2)
31	1217041	1217032	9
32	1218217	1218162	55
33	1217775	1217773	2
34	1213368	1213319	49
35	1211360	1211292	68
36	1207146	1207084	62
37	1214805	1214744	61
38	1223132	1223078	54
39	1211923	1211858	65
40	1212536	1212540	-4
41	877575	877532	43
42	876886	876825	61
43	877972	877979	-7
44	876439	876400	39
45	872230	872180	50
46	869194	869176	18
47	874393	874371	22
48	882192	882164	28
49	873208	873086	122
50	874774	874755	19
51	475790	475740	50
52	476286	476215	71
53	478693	478641	52
54	477893	477890	3
55	472361	472307	54
56	472300	472267	33
57	474377	474362	15
58	479958	479924	34
59	474605	474476	129
60	476864	476761	103
		Total	1360
		Average	45.33

As it is seen on the Table 5-2d, for 26 of 30 problems from the literature the best values found by the one step partitioning algorithm are greater than the best values reported by Chu and Beasley (1998), the average difference being 22.8. In addition, for 19 of these problems the best values found by the one step partitioning algorithm are greater than the best values found by CPLEX v8.1 in MIP mode, the average difference being 12.73.

For the second set of problems (n=1000) which are randomly generated, the best values found by the one step partitioning algorithm are on average 36.47 greater than the best values found by CPLEX v8.1 in MIP mode, and for 28 of these 30 problems, the best values found by the one step partitioning algorithm are greater than the best values found by CPLEX v8.1 in MIP mode.

For the third set of problems (n=2000), the best values found by the one step partitioning algorithm are on average 45.33 greater than the best values found by CPLEX v8.1 in MIP mode, and for 28 of these 30 problems, the best values found by the one step partitioning algorithm are greater than the best values found by CPLEX v8.1 in MIP mode.

We repeated one step partitioning algorithm for 8 problems for which the best values found by the one step partitioning algorithm were less than the best values found by CPLEX v8.1 in MIP mode and give more time to the one step partitioning algorithm than before. For 6 of those problems the one step partitioning algorithm found better quality solutions and for the remaining 2 problems (pb: mkpcb6-09, 40) there was no improvement in 2000 CPU seconds. These results are shown in the table below,

Table 5-2g: Computational results of the problems for which the experiment was repeated

Pb	z(lp)	Partition Algorithm				CPLEX		
		initial value	Best value (improved)	iter.	time	initial value	best value	time
mkpcb6-03	119405.7	118761	119211	250	1180.83	118705	119211	532.23
mkpcb6-10	119440.6	118943	119197	246	659.76	118040	119191	613.79
mkpcb6-21	304555	304214	304353	240	474	303606	304344	582.11
mkpcb6-29	301547.6	301131	301340	232	1222.99	300507	301353	606.75
18	439402.8	438253	439251	274	1663.08	438806	439252	842.44
43	878138	877731	877979	230	1442.33	877047	877979	1007.01

Chapter 6

Conclusion

In this study, we have considered the analysis of the search based cuts -a recently proposed cut for 0-1 programming problems by Oguz (2002)- applied to the multidimensional 0-1 knapsack problems and the application of the partitioning algorithm -a new algorithm based on the search based cuts- to the same problem.

In order to analyze the effectiveness of the search based cuts, we applied them to 60 multidimensional 0-1 knapsack problems from the literature as well as 25 randomly generated set covering problems with different numbers of variables and different numbers of constraints. As a result of our study, we showed that the effectiveness of the search based cuts decreases as the number of variables increases and increases as the number of constraints increases.

In order to check the efficiency of the partitioning algorithm, we made computational experiments with 30 small-sized test problems (Shih (1979)) from the literature and showed that it found the optimal solution to each of these problems. In addition, we made computational experiments on randomly generated problems with different numbers of variables (n) and different numbers of constraints (m) and found out that the largest value of n was 90 and m was 20 for which the algorithm could solve problems to optimality. The inconvenience of the algorithm is that due to computational and programming difficulties, we limited our algorithm to partitioning the

original problem at most 10 times, and to solve larger problems requires using the partitioning procedure more than 10 times.

Since the partitioning algorithm is not an efficient algorithm to solve the large-sized multidimensional 0-1 knapsack problems, we presented a modification of it, "the one step partitioning algorithm" for solving those problems.

We tested the one step partitioning algorithm on 90 multidimensional 0-1 knapsack problems (60 randomly generated and 30 from the literature) and compared its performance with CPLEX v8.1 in MIP mode. We showed that the one step partitioning algorithm performs better than CPLEX especially for the multidimensional 0-1 knapsack problems with a large number of variables and few constraints. One step partitioning algorithm found better quality solutions than CPLEX to the problems tested with less computational effort. The problems couldn't be solved to optimality, the solutions are approximate. We also compared our computational results with the ones reported by Chu and Beasley (1998) for the 30 problems from the literature and showed that for 87% of these problems the best values found by the one step partitioning algorithm were greater than the best values reported by Chu and Beasley (1998). One inconvenience of the one step partitioning algorithm is that as the number of constraints increases the computational time increases rapidly. The reason for this is that as the number of constraints increases the sub-problems get larger and since we use the IP solver of CPLEX to solve sub-problems, this takes more time and the computational time of the algorithm increases.

Finally, we think that the algorithms and cuts tested in this study should prove useful when applied to other similar 0-1 integer programming problems.

Bibliography

[1] Aboudi, R. and K. Jörnsten. (1994). *"Tabu Search for General Zero One Integer Programs Using the Pivot and* Complement Heuristic," *ORSA Journal on Computing* 6, 82–93.

[2] Balas, E. (1965). "An Additive Algorithm for Solving Linear Programs with Zero-One Variables," *Operations Research* 13, 517–546.

[3] Balas, E. and C.H. Martin. (1980). "Pivot and Complement-A Heuristic for 0-1 Programming," *Management Science* 26, 86–96.

[4] Barake, M., P. Chardaire and G. P. McKeown. (2001). "Application of PROBE to the Multiconstraint Knapsack Problem," *MIC 2001*, Kluwer.

[5] Cabot, A.V. (1970). "An Enumeration Algorithm for Knapsack Problems," *Operations Research* 18, 306–311.

[6] Chu, P. C. and J.E. Beasley. (1998). "A Genetic Algorithm for the MultidimensionalKnapsack Problem," *Journal of Heuristics* 4, 63-86.

[7] Crama, Y. and J.B. Mazzola. (1994). "On the Strength of Relaxations of Multidimensional Knapsack Problems," *INFOR* 32, 219–225.

[8] Dammeyer, F. and S. Voss. (1993). "Dynamic Tabu List Management Using Reverse Elimination Method," *Annals of Operations Research* 41, 31–46.

[9] Dantzig, G.B. (1957). "Discrete Variable Problems," *Operations Research* 5, 266–277.

[10] Drexl, A. (1988). "A Simulated Annealing Approach to the Multiconstraint Zero-One Knapsack Problem," *Computing* 40, 1–8.

[11] Everett, H. (1963). "Generalized Lagrange Multiplier Method for Solving Problems of Optimum Allocation of Resources," *Operations Research* 11, 399–417.

[12] Fontanari, J.F. (1995). "A Statistical Analysis of the Knapsack Problem," *Journal of Physics A—Mathematical and General* 28, 4751–4759.

[13] Fox, G.E. and G.D. Scudder. (1985). "AHeuristic with Tie Breaking for Certain 0-1 Integer Programming Models," *Naval Research Logistics Quarterly* 32, 613–623.

[14] Freville, A. and G. Plateau. (1986). "Heuristics and Reduction Methods for Multiple Constraints 0-1 Linear Programming Problems," *European Journal of Operational Research* 24, 206–215.

[15] Freville, A. and G. Plateau. (1994). "An Efficient Preprocessing Procedure for the Multidimensional 0-1 Knapsack Problem," *Discrete Applied Mathematics* 49, 189–212.

[16] Freville, A. and G. Plateau. (1997). "The 0-1 Bidimensional Knapsack Problem: Toward an Efficient High-Level Primitive Tool," *Journal of Heuristics* 2, 147–167.

[17] Gabrel, V. and M. Minoux. (2002). "A Scheme for Exact Separation of Extended Cover Inequalities and Application to Multidimensional Knapsack Problems," *Operations Research Letters* 30, 252-264.

[18] Garey, M. R. and D. S. Johnson. (1979). *Computers and Intractability:A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, San Fransisco.

[19] Gavish, B. and H. Pirkul. (1982). "Allocation of Databases and Processors in a Distributed Computing System." In J. Akoka (ed.) *Management of Distributed Data Processing*, North-Holland, pp. 215–231.

[20] Gavish, B. and H. Pirkul. (1985). "Efficient Algorithms for Solving Multiconstraint Zero-One Knapsack Problems to Optimality," *Mathematical Programming* 31, 78–105.

[21] Gilmore, P.C. and R.E. Gomory. (1966). "The Theory and Computation of Knapsack Functions," *Operations Research* 14, 1045-1075.

[22] Glover, F. and G.A.Kochenberger. (1996). *"Critical Event Tabu Search for Multidimensional Knapsack Problems."* In I.H. Osman and J.P. Kelly (eds.), *Meta-Heuristics: Theory and Applications*. Kluwer Academic Publishers, pp. 407–427.

[23] Hanafi, S. and A. Freville. (1997). "An Efficient Tabu Search Approach for the 0-1 Multidimensional Knapsack Problem," To appear in *European Journal of Operational Research*.

[24] Hillier, F.S. (1969). "Efficient Heuristic Procedures for Integer Linear Programming with an Interior," *Operations Research* 17, 600–637.

[25] Hoff, A., A. Løkketangen, and I. Mittet. (1996). "Genetic Algorithms for 0/1 Multidimensional Knapsack Problems." Working Paper, Molde College, Britveien 2, 6400 Molde, Norway.

[26] Khuri, S., T. Bäck, and J. Heitkötter. (1994). "The Zero/One Multiple Knapsack Problem and Genetic Algorithms," *Proceedings of the 1994 ACM Symposium on Applied Computing (SAC'94)*, ACM Press, pp. 188–193.

[27] Kochenberger, G.A., B.A. McCarl, and F.P. Wymann. (1974). "A Heuristic for General Integer Programming," *Decision Sciences* 5, 36–44.

[28] Lee, J.S. and M. Guignard. (1988). "An Approximate Algorithm for Multidimensional Zero-One Knapsack Problems—a Parametric Approach," *Management Science* 34, 402–410.

[29] Løkketangen, A. and F. Glover. (1996). "Probabilistic Move Selection in Tabu Search for Zero-One Mixed Integer Programming Problems." In I.H. Osman and J.P. Kelly (eds.), *Meta-Heuristics: Theory and Applications*. Kluwer Academic Publishers, pp. 467–487.

[30] Løkketangen, A. and F. Glover. (1997). "Solving Zero-One Mixed Integer Programming Problems Using Tabu Search," to appear in *European Journal of Operational Research*.

[31] Løkketangen, A., K. Jörnsten, and S. Storøy. (1994). "Tabu Search Within a Pivot and Complement Framework," *International Transactions of Operations Research* 1, 305–316.

[32] Loulou, R. and E. Michaelides. (1979). "New Greedy-Like Heuristics for the Multidimensional 0-1 Knapsack Problem," *Operations Research* 27, 1101–1114.

[33] Magazine, M.J. and O. Oguz. (1984). "A Heuristic Algorithm for the Multidimensional Zero-One Knapsack Problem," *European Journal of Operational Research* 16, 319–326.

[34] Martello, S. and P. Toth. (1987). "Algorithms for Knapsack Problems," *Annals of Discrete Mathematics* 31, 70-79.

[35] Martello, S. and P. Toth. (1990). *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons.

[36] Nemhauser, G.L. and Z. Ullmann. (1969). "Discrete Dynamic Programming and Capital Allocation," *Management Science* 15, 494–505.

[37] Oguz, O. (2002). "Search and Cut: New Class of Cutting Planes for 0-1 Programming," *http://www.optimization-online.org/DB_HTML/2002/05/484.html*

[38] Osorio, M. A., F. Glover, and P. Hammer. (2000). "Cutting and surrogate constraint analysis for improved multidimensional knapsack solutions," *Technical report, Hearin Center for Enterprise Science*, *Report HCES-08-00*.

[39] Pirkul, H. (1987). "A Heuristic Solution Procedure for the Multiconstraint Zero-One Knapsack Problem," *Naval Research Logistics* 34, 161–172.

[40] Raidl, Günther R. (1998). "An Improved Genetic Algorithm for the Multiconstrained 0–1 Knapsack Problem," *In Proceedings of the 5th IEEE International Conference on Evolutionary Computation,* 207-211.

[41] Schilling, K.E. (1990). "The Growth of m-Constraint Random Knapsacks," *European Journal of Operational Research* 46, 109–112.

[42] Senju, S. and Y. Toyoda. (1968). "An Approach to Linear Programming with 0-1 Variables," *Management Science* 15, 196–207.

[43] Shih,W. (1979). "A Branch and Bound Method for the Multiconstraint Zero-One Knapsack Problem," *Journal of the Operational Research Society* 30, 369–378.

[44] Soyster, A.L., B. Lev, and W. Slivka. (1978). "Zero-One Programming with Many Variables and Few Constraints," *European Journal of Operational Research* 2, 195–201.

[45] Szkatula, K. (1994). "The Growth of Multi-constraint Random Knapsacks with Various Right-hand Sides of the Constraints," *European Journal of Operational Research* 73, 199–204.

[46] Szkatula, K. (1997). "The Growth of Multi-constraint Random Knapsacks with Large Right-hand Sides of the Constraints," *Operations Research Letters* 21, 25–30.

[47] Thiel, J. and S. Voss. (1994). "Some Experiences on Solving Multiconstraint Zero-One Knapsack Problems with Genetic Algorithms," *INFOR* 32, 226–242.

[48] Toyoda, Y. (1975). "A Simplified Algorithm for Obtaining Approximate Solutions to Zero-One Programming Problems," *Management Science* 21, 1417–1427.

[49] Vasquez, M. and J.-K. Hao. (2001). "A Hybrid Approach for the 0-1 Multidimensional Knapsack Problem," *In Proc. of IJCAI-01*, 328-333.

[50] Volgenant, A. and J.A. Zoon. (1990). "An Improved Heuristic for Multidimensional 0-1 Knapsack Problems," *Journal of the Operational Research Society* 41, 963–970.

[51] Weingartner, H.M. (1967). *Mathematical Programming and the Analysis of Capital Budgeting Problems*. Chicago: Markham Publishing.

[52] Weingartner, H.M. and D.N. Ness. (1967). "Methods for the Solution of the Multidimensional 0/1 Knapsack Problem," *Operations Research* 15, 83–103.

[53] Zanakis, S.H. (1977). "Heuristic 0-1 Linear Programming: An Experimental Comparison of Three Methods," *Management Science* 24, 91–104.