The Core Concept for 0/1 Integer Programming

Anonymous

Abstract

In this paper we examine an extension of the core concept for the 0/1 Multidimensional Knapsack Problem (MKP) towards general 0/1 Integer Programming (IP) by allowing negative profits, weights and capacities. The core concept provides opportunities for heuristically solving the MKP, achieving higher quality solutions and shorter runtimes than general IP methods. We provide the theoretical foundations of the extended core concept and further provide computational experiments showing that we can achieve similar computational behaviour for extended MKP instances with negative weights, profits and capacities.

1 Introduction

The core concept for the 0/1 Multidimensional Knapsack Problem (MKP) (13, 14) has been shown to be very effective in providing opportunities for heuristically solving the MKP, achieving higher quality solutions and shorter run-times than general IP methods. In this paper we will examine the possibilities of extending the core concept towards general 0/1 Integer Programming (IP).

The Multidimensional Knapsack Problem (MKP) is defined as:

$$\text{maximise} \quad z = \sum_{j=1}^{n} p_j x_j \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_{ij} x_j \leq c_i, \quad i = 1, \ldots, m \tag{2}$$

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

where the profits p_j, the weights w_{ij}, and the capacities c_i are all positive. Allowing negative values for those variables results in general 0/1 Integer Problems.

The aim of the core concept is to reduce the original problem to a *core* of items for which it is hard to decide whether or not they will occur in an optimal solution. All variables corresponding to items outside the core are fixed to their optimal values.

The *core* of a given MKP is defined with respect to some variable ordering where variables that are expected to be in the knapsack (set to one) occur before those which are not expected to be in the knapsack (set to zero). Given the optimal solution of the MKP the exact core is defined as the set of variables from the first variable that takes the value zero to the last variable that takes the value one. In order to devise an exact core for a given MKP, the optimal solution has to be known. However, being able to heuristically obtain good approximation to cores and solve those smaller problems, may lead to high quality solutions in short computational times.

The underlying concept of such a heuristic is to order the items of the MKP according to a specific efficiency measure. This ordering will allow to partition the items into three sections. The first section which contains items which are included in the knapsack (variables set to one). The second section, named the *approximate core*, containing the items which may or may not be included in the knapsack (variables set to either one or zero). Finally the third section contains the items which are not included in the knapsack (variables set to zero). The aim is to have the approximate core closely mimic the exact core.

The following example illustrates the core concept for a small 2-dimensional knapsack problem. The variables are ordered by an efficiency measure described later. The first line shows the optimal integer solution, while the second line shows the optimal solution to the LP-relaxation of the problem. The exact core is shown in bold in the first line, while an approximate core (adding 2 variables around the non 0-1 LP solution values) is shown in bold in the second line.

IP	1	1	**0**	**0**	**1**	**0**	**1**	0	0	0
LP	1	1	1	**1**	**1**	**0.96**	**0.23**	**0**	**0**	0

In the remainder of the paper, we first introduce the core concept in the context of its previous uses. We then extend the efficiency measure used for MKPs to general 0/1 integer programs, and prove that the efficiency measure is tightly related to the optimal solution. In Section 4 we show the result of experiments illustrating the effectiveness of the approximate core computations, the loss of precision that arises from restricting the problem

to an approximate core, and the improvement in best solutions found if we use approximate cores.

2 Background

2.1 The Multidimensional Knapsack Problem

A comprehensive overview of practical and theoretical results for the MKP can be found in the monograph on knapsack problems by Kellerer et al. (9). Solving the MKP with heuristic methods seems to be the method of choice for the bigger instances described in the literature, since no exact method is known for solving these instances to optimality. Besides exact techniques for solving small to moderately sized instances, based on dynamic programming (6, 19) and branch-and-bound (16, 5), many kinds of meta-heuristics have been applied to the MKP (7, 2). See (15) for a recent survey and comparison of evolutionary algorithms for the MKP. The hybrid tabu-search methods presented in (17, 18) are currently yielding the best known results for the commonly used benchmark instances.

2.2 The core concept for KP and MKP

The core concept was first presented for the classical 0/1-Knapsack Problem (KP) (1) and led to very successful knapsack algorithms (10, 11, 12). These ideas were also studied in the context of bicriteria knapsack problems in (8). The core concept was successfully extended to the MKP (13, 14), leading to highly competitive heuristic algorithms.

The classical greedy heuristic for KP packs the items into the knapsack in decreasing order of their *efficiencies* $\frac{p_j}{w_j}$ as long as the knapsack constraint is not violated. The same ordering also defines the solution structure of the LP-relaxation, which consists of three parts: The first part contains all variables set to one, the second part consists of at most one *split item* (s), whose corresponding LP-value is fractional, and finally the remaining variables, which are always set to zero, form the third part.

The precise definition of the core of KP introduced by (1) requires the knowledge of an optimal integer solution x^*. Assume that the items are sorted according to decreasing efficiencies and let

$$a := \min\{j \mid x_j^* = 0\}, \qquad b := \max\{j \mid x_j^* = 1\}. \tag{4}$$

The core is given by the items in the interval $C = \{a, \ldots, b\}$. It is obvious that the split item is always part of the core.

These ideas have been expanded to MKP without major difficulties (13, 14). The main difference is that the choice of the efficiency measure is not obvious for the MKP any more. The efficiency measure:

$$e_j = \frac{p_j}{s_j},$$

where $s_j = \sum_{i=1}^{m} u_i w_{ij}$, and u_i are the dual variable values of the LP-relaxation of the MKP, provided the best theoretical and practical results.

Let x^* be an optimal solution and assume that the items are sorted according to the decreasing efficiency measure e, then define

$$a := \min\{j \mid x_j^* = 0\} \quad \text{and} \quad b := \max\{j \mid x_j^* = 1\}. \tag{5}$$

The core is given by the items in the interval $C := \{a, \ldots, b\}$, and the core problem (MKPC) is defined as

$$\text{maximise} \quad z = \sum_{j \in C} p_j x_j + \tilde{p} \tag{6}$$

$$\text{subject to} \quad \sum_{j \in C} w_{ij} x_j \leq c_i - \tilde{w}_i, \; i = 1, \ldots, m \tag{7}$$

$$x_j \in \{0, 1\}, \quad j \in C, \tag{8}$$

with $\tilde{p} = \sum_{j=1}^{a-1} p_j$ and $\tilde{w}_i = \sum_{j=1}^{a-1} w_{ij}$, $i = 1, \ldots, m$.

In contrast to KP, the solution of the LP-relaxation of MKP does not consist of a single fractional split item, but its up to m fractional values give rise to a whole *split interval* $S := \{s, \ldots, t\}$, where s and t are the first and the last index of variables with fractional values after sorting by efficiency e.

The split interval S_e has been precisely characterized (13, 14). Let x^{LP} be the optimal solution of the LP-relaxation of MKP.

Theorem 1 ((13, 14)) *For efficiency values e_j we have:*

$$x_j^{\mathrm{LP}} = \begin{cases} 1 & \text{if } e_j > 1\,, \\ \in [0,1] & \text{if } e_j = 1\,, \\ 0 & \text{if } e_j < 1\,. \end{cases} \tag{9}$$

3 Extending the core concept

As mentioned above, the main goal of this paper is to extend the core concept to general 0/1 Integer Programs. We show how the efficiency measure for the classical MKP problem can be adapted in such a way that the ordering of the variables according to this measure remains valuable for devising good approximate cores. We further provide a characterization of the structure of the LP relaxation of the 0/1 Integer Program.

We extend the previously defined efficiency measure e_j to create a tuple based measure. We introduce ordering variables, o_j which takes values representing a section of the ordering. Variables x_j are sorted in decreasing (lexicographic) order of efficiency (o_j, e_j), in other words they are first sorted by section variable, o_j, and then efficiency value e_j.

These extensions are implemented as indicated in equation 10:

$$(o_j, e_j) = \begin{cases} (7, \frac{p_j}{s_j}) & \text{if } p_j > 0 \wedge s_j < 0 \\ (6, p_j) & \text{if } p_j > 0 \wedge s_j = 0 \\ (5, \frac{1}{s_j}) & \text{if } p_j = 0 \wedge s_j < 0 \\ (4, \frac{p_j}{s_j}) & \text{if } p_j > 0 \wedge s_j > 0 \\ (4, \frac{s_j}{p_j}) & \text{if } p_j < 0 \wedge s_j < 0 \\ (4, 1) & \text{if } p_j = 0 \wedge s_j = 0 \\ (3, \frac{1}{s_j}) & \text{if } p_j = 0 \wedge s_j > 0 \\ (2, p_j) & \text{if } p_j < 0 \wedge s_j = 0 \\ (1, \frac{p_j}{s_j}) & \text{if } p_j < 0 \wedge s_j > 0 \end{cases} \quad (10)$$

The ordering values are chosen to minimize the size of the IP core. The section $o_j = 4$ contains the center of the split interval. Our experiments in Section 4 show that the center of the core and the center of the split interval are close for our benchmark instances.

The sections $o_j \in \{7, 6, 5\}$ contain items that are purely beneficial: they either increase profit or "on average" remove weight from the multi-dimensional knapsack without decreasing profit. They are ordered to maximize profit and removal of weight. Similarly the sections $o_j \in \{1, 2, 3\}$ contain items that are purely detrimental to the knapsack: they either decrease profit or add weight. Again they are ordered to maximize profit and removal of weight.

In none of the benchmarks where we can actually compute the exact core do items in sections other than 4 appear in the core.

Using this efficiency measure the nature of the split interval can be characterized as follows. Let x^{LP} be the optimal solution of the LP-relaxation of MKP with negative profits and weights.

Theorem 2

$$x_j^{LP} = \begin{cases} 1 & \textit{if } e_j > 1 \textit{ or } o_j > 4, \\ \in [0, 1] & \textit{if } e_j = 1 \textit{ and } o_j = 4, \\ 0 & \textit{if } 0 \leq e_j < 1 \textit{ or } o_j < 4. \end{cases} \quad (11)$$

Proof The dual LP (DMKP) associated with the LP-relaxation of MKP is given by

$$\text{minimise} \sum_{i=1}^{m} c_i u_i + \sum_{j=1}^{n} v_j \quad (12)$$

$$\text{subject to} \sum_{i=1}^{m} w_{ij} u_i + v_j \geq p_j, j = 1, ..., n \quad (13)$$

$$u_i, v_j \geq 0, i = 1, ..., m, j = 1, ..., n, \quad (14)$$

where u_i are the dual variables corresponding to the knapsack constraints and each v_j corresponds to the inequality $x_j \leq 1$. For the optimal primal and dual solutions the following complementary slackness conditions hold for $j = 1, ..., n$.

$$x_j \left(\sum_{i=1}^{m} w_{ij} u_i + v_j - p_j \right) = 0 \quad (15)$$

$$v_j \left(x_j - 1 \right) = 0 \quad (16)$$

We illustrate the result for each section in the definitions of the ordered tuple (o_j, e_j).

Consider the top 3 sections ($o_j \in \{7, 6, 5\}$). Clearly in each case we have that $p_j > s_j$. Hence satisfying equation (13) requires that $v_j > 0$. Therefore equation (16) implies that $x_j = 1$.

Consider the bottom 3 sections ($o_j \in \{1, 2, 3\}$). Clearly in each case $p_j < s_j$. Hence the expression $\sum_{i=1}^{m} w_{ij} u_i + v_j - p_j$ or equivalently $s_j + v_j - p_j$ is greater than 0, since $v_j \geq 0$. In order to satisfy equation 15 $x_j = 0$.

For section $(4, \frac{p_j}{s_j})$ we consider two cases. If $e_j = \frac{p_j}{s_j} > 1$ then $p_j > s_j$ since $s_j > 0$ and the same reasoning as for case $(7, 2)$ applies, while if $e_j < 1$ then $p_j < s_j$ and the same reasoning as case $(2, 0)$ applies.

For section $(4, \frac{s_j}{p_j})$ we consider two cases. If $e_j = \frac{s_j}{p_j} > 1$ then $p_j > s_j$ since $p_j < 0$ and the same reasoning as for case $(7, 2)$ applies, while if $e_j < 1$ then $p_j < s_j$ and the same reasoning as case $(2, 0)$ applies.

For the remainder of the cases, $e_j = 1$ and $o_j = 4$, there is nothing to prove.

□

An illustration of the ordering (o_j, e_j) is given in Table 1. This example shows some of the sections described above. As predicted by the theorem, the split interval exists entirely within section $o_j = 4$.

4 Computational experiments

4.1 Benchmark Problems

In order to study the core concept on 0/1 integer programs we generated 0/1 integer programs using the Chu and Beasley (2) benchmark instances for the MKP, as the starting point.

These benchmark problems consist of classes of randomly created instances for each combination of $n \in \{100, 250, 500\}$ items, $m \in \{5, 10, 30\}$ constraints and tightness ratios 0.25, 0.50.75. The tightness ratio refers to the ratio between the constraint value and the sum of the corresponding weights.

$$\alpha = \frac{c_i}{\sum_{j=1}^{n} w_{ij}} \in \{0.25, 0.5, 0.75\} \quad (17)$$

Weights are integers between 0 and 1000. Profits are generated by the equation:

$$p_j = \sum_{i=1}^{m} \frac{w_{ij}}{m} + [500 r_i] \in \{0.25, 0.5, 0.75\}$$

where r_i is a random number generated from $(0, 1]$. For each permutation of n, $m$and α, 10 instances are provided.

In order to generate 0/1 integer programs with negative values we multiply a random percentage of the weights and profits of a given problem by -1. The percentages used are $5\%, 10\%, 20\%$. Larger percentages of negative values were tried,

$Weight_1$	$Weight_2$	Profit	s_j	(o_j, e_j)	LP	IP
-1	-1	1	-0.89	(7,0.89)	1.00	1
-9	12	7	-5.17	(7,0.74)	1.00	1
0	0	8	0.00	(6,8.00)	1.00	1
-2	13	12	0.24	(4,0.25)	1.00	1
5	0	19	3.77	(4,5.04)	1.00	1
-5	-4	-3	-4.31	(4,1.44)	1.00	1
16	21	18	14.88	(4,1.21)	1.00	1
13	15	14	11.81	(4,1.19)	1.00	**0**
-6	-10	-5	-5.87	(4,1.17)	1.00	**1**
20	-8	14	14.00	(4,1.00)	0.71	**1**
-6	-11	-6	-6.00	(4,1.00)	0.85	0
20	16	15	17.22	(4,0.87)	0.00	0
10	8	6	8.61	(4,0.70)	0.00	0
-1	-14	-4	-2.63	(4,0.66)	0.00	0
14	9	5	11.76	(4,0.43)	0.00	0
16	-3	3	11.66	(4,0.26)	0.00	0
-7	23	-9	-2.19	(4,0.24)	0.00	0
-1	10	0	0.59	(3,1.69)	0.00	0
7	0	-5	5.28	(1,0.95)	0.00	0
24	10	-9	19.43	(1,0.46)	0.00	0

Table 1: Example 2-Dimensional Knapsack Problem. The 3 sections are separated based upon the IP solution. This example shows an exact core using the efficiency measure defined in Theorem 2.

however the problems quickly became optimally solvable in short run-times.

The set of profits and the set of all weights are operated on separately. This ensures that there is a set percentage of negative profits and a set percentage of negative weights. Combinations of different percentages of negative weights and profits were tried, however almost invariably this made the problems easier and thus faster to solve.

This process will change the tightness ratio. In order to maintain the tightness ratio the capacities have to be adjusted:

$$\hat{c}_i = \frac{c_i * \sum_{j=1}^{n} \hat{w}_{i,j}}{\sum_{j=1}^{n} w_{i,j}},$$

$w_{i,j}$ represents the original weights, $\hat{w}_{i,j}$ represents the adjusted weights, and $\hat{c}_i$ represents the new constraint value. Since the sum of the adjusted weights may become negative, it is possible that the new capacity $\hat{c}_i$ will also be negative.

4.2 MKP Core Analysis

We provide empirical results supporting our adaptation of the efficiency measure e, (see Table 2). This table shows information about actual cores when the above efficiency function is utilized. The problems shown in these tables are based upon the smaller instances in Chu and Beasley's benchmark library (2). Specifically these problems use $n = 100, m \in 5, 10$, and $n = 250, m = 5$. These problems were chosen because they are solvable in reasonable runtime. This means that the optimum solutions can be found, and the size of the core can be determined.

The tables show the averaged values over 10 problem instances. Average values listed include size of the split interval ($|S_e|$), size of the exact core ($|C_e|$), percentage that the split interval covers the exact core (ScC), percentage that the exact core covers the split interval (ScC), and the distance between the centre of the split interval and the centre of the exact core ($|C_{dist}|$) as a percentage of the number of items in the problem. We only show results where the percent of negative weights and negative profits are the same, since this tended to be where the harder problems arose.

The table entries for 0% negative coefficients shows that the newly defined efficiency value provides equivalent results for standard MKP problems as those reported in (14). As expected from Theorem 2, negative values do not increase the size of the split interval or the core.

The size of the split interval and the core actually decreases as the number of negative weights increases. The centre of the core remains close to the centre of the split interval.

4.3 Approximate core algorithm

In order to evaluate the influence of negative values on solution quality and run-times an approximate core algorithm was implemented. This algorithm is similar to the algorithm implemented by Puchinger et al. (14). The approximate core is generated by adding δ items to either side of the center of the split interval. The values of δ were chosen to approximately reflect the size of the actual cores detected in the previous section: $\delta \in 0.1n, 0.15n, 0.2n, 0.1n + 2m, 0.2n + 2m$.

The problems shown in these tables are based upon the smaller instances in Chu and Beasley's benchmark library (2). They are the same set of problems used to investigate the actual core sizes in the previous section.

			e - 0% negative weights 0% negative profits					e - 5% negative weights 5% negative profits				
n	m	α	$\|S_e\|$	$\|C_e\|$	ScC	CcS	C_{dist}	$\|S_e\|$	$\|C_e\|$	ScC	CcS	C_{dist}
100	5	0.25	5.00	20.20	28.12	100.00	3.30	5.00	20.00	31.58	100.00	2.90
		0.5	5.00	22.10	27.49	100.00	3.45	5.00	15.90	28.33	86.00	2.65
		0.75	5.00	20.00	26.32	100.00	3.40	5.00	14.80	35.91	98.00	3.50
250	5	0.25	2.00	12.68	18.16	100.00	2.46	2.00	13.36	17.35	100.00	3.12
		0.5	2.00	12.20	18.45	100.00	1.38	2.00	9.60	21.47	100.00	1.20
		0.75	2.00	10.40	20.18	100.00	1.56	2.00	10.96	21.04	100.00	1.92
100	10	0.25	10.00	23.20	46.57	100.00	2.90	9.90	25.80	42.74	96.67	3.45
		0.5	9.80	25.80	48.17	96.00	3.10	9.70	23.70	44.06	100.00	3.00
		0.75	9.70	18.30	54.36	94.00	3.00	9.20	16.90	60.09	93.19	2.45
Average			5.61	18.32	31.98	98.89	2.73	5.53	16.75	33.62	97.10	2.69
			e - 10% negative weights 10% negative profits					e - 20% negative weights 20% negative profits				
n	m	α	$\|S_e\|$	$\|C_e\|$	ScC	CcS	C_{dist}	$\|S_e\|$	$\|C_e\|$	ScC	CcS	C_{dist}
100	5	0.25	5.00	23.60	22.31	100.00	4.30	5.00	18.60	29.55	100.00	2.70
		0.5	5.00	19.40	27.11	100.00	3.00	4.60	9.80	57.68	95.50	1.20
		0.75	4.80	12.40	47.28	98.00	1.60	2.50	16.90	30.34	86.67	7.00
250	5	0.25	2.00	10.36	20.27	100.00	1.22	2.00	10.52	20.22	98.00	2.02
		0.5	2.00	11.72	18.79	100.00	2.22	2.00	8.84	24.31	100.00	1.94
		0.75	2.00	7.28	30.13	98.00	1.24	1.56	6.08	40.80	100.00	1.74
100	10	0.25	9.70	24.20	42.19	97.89	4.05	9.70	28.00	37.91	100.00	4.15
		0.5	9.50	20.10	49.20	97.00	3.20	8.40	20.10	47.86	98.89	3.35
		0.75	8.80	14.30	65.99	92.70	2.05	4.30	13.80	35.51	92.50	4.05
Average			5.42	15.92	35.92	98.18	2.54	4.45	14.74	36.02	96.84	3.13

Table 2: Split intervals, core sizes, mutual coverage of the split interval and cores, distances of the centers for various percentages of negative values. (Values are averaged over 10 instances of each problem.)

The results of this experiment are shown in Table 3. It shows the average values over 10 problems with the same tightness ratio. Values shown for the original problem include the average optimal IP solution for the problem ($\bar{z}$), then the average amount of CPU-time taken to produce the optimal IP solution in seconds (t[s]). Values shown for each core include the average percentage difference between the optimal IP solution(z^*) and the IP solution produced by the core problem(z), ($\%_{opt} = 100*(z^*-z)/z^*$), the number of times the optimal solution was reached (#), and the average CPU-time taken to solve the core IP, as a percentage of the CPU-time taken to solve the original IP problem, $\%t = 100 * (t_{core}/t_{original})$.

The solution to the core of the problem is always within 0.7 % of the optimal solution. The results shown in Table 3 show that smaller approximate core sizes produce a significant increase in speed. However they are less likely to produce the optimal solution, and on average produce less optimal solutions than the larger cores. As the percentage of negative values increases the problems become faster to solve. Larger negative percentages were examined, however run-times were too small to see any benefit from the core concept.

4.4 Larger MKPs with Fixed Time Runs

We now investigate fixed-time runs over larger instances. These tests are performed over instances which are currently very hard or not at all solvable to optimality. The problems we tested over are based on the hardest benchmarks provided by Chu and Beasley (2), $n = 500,\ m \in 5, 10, 30$

Again these problems were adjusted to contain negative values in a manner similar to the problems above. All of the results shown here are performed over problems with 10% negative weights, and 10% negative profits. The constraints are also adjusted accordingly.

Table 4 shows the best feasible solution for the original problem and the core problems as a percentage of the LP solution, ($\%_{LP} = 100 * (LP - IP)/LP$). These values are averaged over 10 instances of similar problems. Standard deviations are provided as subscripts. The smallest values for each row are highlighted in bold. This table also shows the number of times a particular core size has lead to the best solution for a problem, (#). The final column for each core size is the average number of nodes explored in the branch and bound tree.

The experiments show that for the considered time limits the results obtained on the core problems are, on average, better that the results obtained from the original problem. There is also a inverse relationship between the size of the core and the number of nodes explored. As the size of the core decreases the number of nodes explored increases. The best average results for a time limit of 500 seconds is $\delta = 0.2$. It can be seen that smaller time limits provide best results with smaller approximate core sizes.

0 % negative weights, 0 % negative profits																		
			no core		$\delta = 0.1n$			$\delta = 0.15n$			$\delta = 0.2n$			$\delta = 0.1n + 2m$			$\delta = 0.2n + 2m$		
n	m	α	$\bar{z}$	t[s]	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$
100	5	0.25	24197	4.81	0.097	5	3	0.034	7	24	0.015	9	51	0.015	9	51	0.000	10	69
		0.50	43253	5.77	0.053	4	2	0.007	8	16	0.002	9	42	0.002	9	42	0.002	9	71
		0.75	60471	1.78	0.038	5	6	0.019	7	35	0.001	9	53	0.001	9	53	0.000	10	72
250	5	0.25	60414	277.22	0.008	7	29	0.003	9	70	0.000	10	66	0.003	9	51	0.000	10	59
		0.50	43253	5.77	0.053	4	2	0.007	8	16	0.002	9	42	0.002	9	42	0.002	9	71
		0.75	60471	1.78	0.038	5	6	0.019	7	35	0.001	9	53	0.001	9	53	0.000	10	72
100	10	0.25	22602	61.85	0.473	1	0	0.061	7	2	0.002	9	17	0.000	10	53	0.000	10	75
		0.50	43253	5.77	0.053	4	2	0.007	8	16	0.002	9	42	0.002	9	42	0.002	9	71
		0.75	60471	1.78	0.038	5	6	0.019	7	35	0.001	9	53	0.001	9	53	0.000	10	72
Average			46487	40.73	0.095	4.4	6	0.020	7.6	28	0.003	9.1	47	0.003	9.1	49	0.001	9.7	70

5 % negative weights, 5 % negative profits																		
			no core		$\delta = 0.1n$			$\delta = 0.15n$			$\delta = 0.2n$			$\delta = 0.1n + 2m$			$\delta = 0.2n + 2m$		
n	m	α	$\bar{z}$	t[s]	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$
100	5	0.25	26603	1.91	0.113	3	4	0.014	8	31	0.004	9	50	0.004	9	50	0.000	10	82
		0.50	44666	1.88	0.072	5	5	0.001	9	17	0.000	10	41	0.000	10	41	0.000	10	69
		0.75	60387	0.46	0.025	7	15	0.013	8	50	0.011	9	63	0.011	9	63	0.000	10	72
250	5	0.25	68598	65.52	0.007	7	33	0.004	8	69	0.003	9	89	0.004	8	55	0.003	9	93
		0.50	44666	1.88	0.072	5	5	0.001	9	17	0.000	10	41	0.000	10	41	0.000	10	69
		0.75	60387	0.46	0.025	7	15	0.013	8	50	0.011	9	63	0.011	9	63	0.000	10	72
100	10	0.25	24211	16.24	0.614	1	0	0.133	6	3	0.026	7	16	0.000	10	59	0.000	10	75
		0.50	44666	1.88	0.072	5	5	0.001	9	17	0.000	10	41	0.000	10	41	0.000	10	69
		0.75	60387	0.46	0.025	7	15	0.013	8	50	0.011	9	63	0.011	9	63	0.000	10	72
Average			48286	10.08	0.114	5.2	11	0.021	8.4	34	0.007	9.1	52	0.005	9.3	53	0.000	9.9	75

10 % negative weights, 10 % negative profits																		
			no core		$\delta = 0.1n$			$\delta = 0.15n$			$\delta = 0.2n$			$\delta = 0.1n + 2m$			$\delta = 0.2n + 2m$		
n	m	α	$\bar{z}$	t[s]	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$
100	5	0.25	28582	2.04	0.136	2	6	0.067	6	38	0.000	10	58	0.000	10	58	0.000	10	78
		0.50	45222	1.84	0.051	4	5	0.004	8	29	0.000	10	46	0.000	10	46	0.000	10	83
		0.75	59180	0.15	0.003	8	22	0.000	10	50	0.000	10	68	0.000	10	68	0.000	10	86
250	5	0.25	74521	45.59	0.000	10	29	0.000	10	43	0.000	10	74	0.000	10	43	0.000	10	72
		0.50	45222	1.84	0.051	4	5	0.004	8	29	0.000	10	46	0.000	10	46	0.000	10	83
		0.75	59180	0.15	0.003	8	22	0.000	10	50	0.000	10	68	0.000	10	68	0.000	10	86
100	10	0.25	25581	33.94	0.673	1	0	0.201	5	3	0.038	8	20	0.000	10	59	0.000	10	73
		0.50	45222	1.84	0.051	4	5	0.004	8	29	0.000	10	46	0.000	10	46	0.000	10	83
		0.75	59180	0.15	0.003	8	22	0.000	10	50	0.000	10	68	0.000	10	68	0.000	10	86
Average			49099	9.73	0.108	5.4	13	0.031	8.3	36	0.004	9.8	55	0.000	10.0	56	0.000	10.0	81

20 % negative weights, 20 % negative profits																		
			no core		$\delta = 0.1n$			$\delta = 0.15n$			$\delta = 0.2n$			$\delta = 0.1n + 2m$			$\delta = 0.2n + 2m$		
n	m	α	$\bar{z}$	t[s]	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$
100	5	0.25	34071	0.72	0.117	4	12	0.008	9	35	0.000	10	61	0.000	10	61	0.000	10	83
		0.50	46970	0.11	0.008	9	29	0.000	10	55	0.000	10	75	0.000	10	75	0.000	10	90
		0.75	56538	0.02	0.029	7	64	0.006	8	64	0.006	8	72	0.006	8	72	0.006	8	80
250	5	0.25	85206	33.74	0.005	8	32	0.000	10	46	0.000	10	70	0.000	10	49	0.000	10	80
		0.50	46970	0.11	0.008	9	29	0.000	10	55	0.000	10	75	0.000	10	75	0.000	10	90
		0.75	56538	0.02	0.029	7	64	0.006	8	64	0.006	8	72	0.006	8	72	0.006	8	80
100	10	0.25	29090	13.37	0.604	1	0	0.190	5	5	0.048	7	19	0.001	9	64	0.000	10	81
		0.50	46970	0.11	0.008	9	29	0.000	10	55	0.000	10	75	0.000	10	75	0.000	10	90
		0.75	56538	0.02	0.029	7	64	0.006	8	64	0.006	8	72	0.006	8	72	0.006	8	80
Average			50988	5.36	0.093	6.8	36	0.024	8.7	49	0.007	9.0	66	0.002	9.2	68	0.002	9.3	84

Table 3: Solving different sized cores for various percentages of negative values to optimality. (All values shown are averaged over 10 problem instances)

Time Limit = 5 Seconds													
			original problem			$\delta = 0.1n$			$\delta = 0.15n$			$\delta = 0.2n$		
n	m	α	$\%_{LP}$	#	Nnodes	$\%_{LP}$	#	Nnodes	$\%_{LP}$	#	Nnodes	$\%_{LP}$	#	Nnodes
500	5	0.25	$0.120_{0.021}$	2	16421	$\mathbf{0.114}_{0.025}$	4	31847	$0.116_{0.022}$	3	26955	$0.117_{0.014}$	4	24337
		0.50	$0.067_{0.011}$	2	15976	$\mathbf{0.051}_{0.012}$	9	31901	$0.061_{0.011}$	4	27587	$0.061_{0.017}$	3	23958
		0.75	$0.041_{0.004}$	5	18598	$\mathbf{0.041}_{0.004}$	8	32791	$0.042_{0.006}$	6	29194	$0.042_{0.005}$	6	27454
500	10	0.25	$0.438_{0.024}$	0	8061	$\mathbf{0.352}_{0.048}$	6	23208	$0.383_{0.051}$	3	15496	$0.364_{0.037}$	6	13707
		0.50	$0.067_{0.011}$	2	15976	$\mathbf{0.051}_{0.012}$	9	31901	$0.061_{0.011}$	4	27587	$0.061_{0.017}$	3	23958
		0.75	$0.041_{0.004}$	5	18598	$\mathbf{0.041}_{0.004}$	8	32791	$0.042_{0.006}$	6	29194	$0.042_{0.005}$	6	27454
500	30	0.25	$1.220_{0.115}$	2	2977	$\mathbf{1.191}_{0.105}$	3	10262	$1.230_{0.044}$	3	7881	$1.214_{0.113}$	2	6014
		0.50	$0.067_{0.011}$	2	15976	$\mathbf{0.051}_{0.012}$	9	31901	$0.061_{0.011}$	4	27587	$0.061_{0.017}$	3	23958
		0.75	$0.041_{0.004}$	5	18598	$\mathbf{0.041}_{0.004}$	8	32791	$0.042_{0.006}$	6	29194	$0.042_{0.005}$	6	27454
Average			$0.333_{0.031}$	2.0	9642	$0.310_{0.034}$	5.3	22019	$0.323_{0.023}$	3.4	17433	$0.315_{0.034}$	4.2	15259
Time Limit = 50 Seconds													
			original problem			$\delta = 0.1n$			$\delta = 0.15n$			$\delta = 0.2n$		
n	m	α	$\%_{LP}$	#	Nnodes	$\%_{LP}$	#	Nnodes	$\%_{LP}$	#	Nnodes	$\%_{LP}$	#	Nnodes
500	5	0.25	$0.103_{0.016}$	3	172471	$0.100_{0.015}$	5	330280	$0.100_{0.015}$	6	272642	$\mathbf{0.099}_{0.015}$	7	249998
		0.50	$0.049_{0.011}$	6	181177	$\mathbf{0.046}_{0.008}$	9	328066	$0.048_{0.011}$	8	287341	$0.047_{0.009}$	8	260145
		0.75	$0.038_{0.004}$	6	188492	$0.038_{0.004}$	7	322998	$\mathbf{0.038}_{0.004}$	8	289610	$0.038_{0.004}$	9	275409
500	10	0.25	$0.331_{0.026}$	2	85187	$0.301_{0.019}$	4	231031	$\mathbf{0.296}_{0.024}$	6	150726	$0.312_{0.031}$	4	132856
		0.50	$0.049_{0.011}$	6	181177	$\mathbf{0.046}_{0.008}$	9	328066	$0.048_{0.011}$	8	287341	$0.047_{0.009}$	8	260145
		0.75	$0.038_{0.004}$	6	188492	$0.038_{0.004}$	7	322998	$\mathbf{0.038}_{0.004}$	8	289610	$0.038_{0.004}$	9	275409
500	30	0.25	$1.108_{0.076}$	1	33855	$\mathbf{1.045}_{0.066}$	6	102129	$1.098_{0.077}$	2	78250	$1.094_{0.080}$	2	62556
		0.50	$0.049_{0.011}$	6	181177	$\mathbf{0.046}_{0.008}$	9	328066	$0.048_{0.011}$	8	287341	$0.047_{0.009}$	8	260145
		0.75	$0.038_{0.004}$	6	188492	$0.038_{0.004}$	7	322998	$\mathbf{0.038}_{0.004}$	8	289610	$0.038_{0.004}$	9	275409
Average			$0.289_{0.023}$	3.0	103422	$0.272_{0.022}$	5.7	222198	$0.279_{0.023}$	5.3	174850	$0.281_{0.025}$	4.8	155927
Time Limit = 250 Seconds													
			original problem			$\delta = 0.1n$			$\delta = 0.15n$			$\delta = 0.2n$		
n	m	α	$\%_{LP}$	#	Nnodes	$\%_{LP}$	#	Nnodes	$\%_{LP}$	#	Nnodes	$\%_{LP}$	#	Nnodes
500	5	0.25	$0.093_{0.012}$	8	839328	$\mathbf{0.092}_{0.011}$	10	1262581	$\mathbf{0.092}_{0.011}$	10	1260524	$0.094_{0.014}$	8	1115971
		0.50	$0.046_{0.009}$	8	789650	$0.045_{0.008}$	8	1344004	$\mathbf{0.044}_{0.007}$	10	1222077	$0.046_{0.008}$	8	1112724
		0.75	$\mathbf{0.038}_{0.004}$	10	780168	$\mathbf{0.038}_{0.004}$	10	965154	$\mathbf{0.038}_{0.004}$	10	948057	$\mathbf{0.038}_{0.004}$	10	926572
500	10	0.25	$0.305_{0.021}$	1	397584	$0.284_{0.021}$	6	1101566	$0.285_{0.018}$	4	640335	$\mathbf{0.283}_{0.023}$	5	568891
		0.50	$0.046_{0.009}$	8	789650	$0.045_{0.008}$	8	1344004	$\mathbf{0.044}_{0.007}$	10	1222077	$0.046_{0.008}$	8	1112724
		0.75	$\mathbf{0.038}_{0.004}$	10	780168	$\mathbf{0.038}_{0.004}$	10	965154	$\mathbf{0.038}_{0.004}$	10	948057	$\mathbf{0.038}_{0.004}$	10	926572
500	30	0.25	$1.030_{0.082}$	2	164246	$\mathbf{0.988}_{0.033}$	5	507929	$1.014_{0.055}$	1	382731	$1.007_{0.084}$	3	306841
		0.50	$0.046_{0.009}$	8	789650	$0.045_{0.008}$	8	1344004	$\mathbf{0.044}_{0.007}$	10	1222077	$0.046_{0.008}$	8	1112724
		0.75	$\mathbf{0.038}_{0.004}$	10	780168	$\mathbf{0.038}_{0.004}$	10	965154	$\mathbf{0.038}_{0.004}$	10	948057	$\mathbf{0.038}_{0.004}$	10	926572
Average			$0.268_{0.022}$	4.8	472658	$0.257_{0.018}$	7.0	955127	$0.262_{0.017}$	5.9	744910	$0.260_{0.022}$	6.0	664661
Time Limit = 500 Seconds													
			original problem			$\delta = 0.1n$			$\delta = 0.15n$			$\delta = 0.2n$		
n	m	α	$\%_{LP}$	#	Nnodes	$\%_{LP}$	#	Nnodes	$\%_{LP}$	#	Nnodes	$\%_{LP}$	#	Nnodes
500	5	0.25	$\mathbf{0.092}_{0.011}$	10	1468306	$\mathbf{0.092}_{0.011}$	10	2156859	$\mathbf{0.092}_{0.011}$	10	2038741	$0.092_{0.011}$	9	1844037
		0.50	$0.045_{0.008}$	9	1474390	$\mathbf{0.044}_{0.007}$	10	2282445	$\mathbf{0.044}_{0.007}$	10	2184819	$0.045_{0.008}$	9	1968420
		0.75	$\mathbf{0.038}_{0.004}$	10	1051111	$\mathbf{0.038}_{0.004}$	10	1170739	$\mathbf{0.038}_{0.004}$	10	1193311	$\mathbf{0.038}_{0.004}$	10	1133267
500	10	0.25	$0.291_{0.022}$	1	752103	$\mathbf{0.266}_{0.027}$	4	1973083	$0.269_{0.024}$	5	1190391	$0.274_{0.018}$	4	1070741
		0.50	$0.045_{0.008}$	9	1474390	$\mathbf{0.044}_{0.007}$	10	2282445	$\mathbf{0.044}_{0.007}$	10	2184819	$0.045_{0.008}$	9	1968420
		0.75	$\mathbf{0.038}_{0.004}$	10	1051111	$\mathbf{0.038}_{0.004}$	10	1170739	$\mathbf{0.038}_{0.004}$	10	1193311	$\mathbf{0.038}_{0.004}$	10	1133267
500	30	0.25	$0.982_{0.088}$	3	312874	$0.982_{0.026}$	3	990696	$0.974_{0.025}$	2	759324	$\mathbf{0.952}_{0.073}$	3	610611
		0.50	$0.045_{0.008}$	9	1474390	$\mathbf{0.044}_{0.007}$	10	2282445	$\mathbf{0.044}_{0.007}$	10	2184819	$0.045_{0.008}$	9	1968420
		0.75	$\mathbf{0.038}_{0.004}$	10	1051111	$\mathbf{0.038}_{0.004}$	10	1170739	$\mathbf{0.038}_{0.004}$	10	1193311	$\mathbf{0.038}_{0.004}$	10	1133267
Average			$0.258_{0.021}$	5.3	840346	$0.253_{0.016}$	6.3	1678296	$0.253_{0.015}$	6.1	1308292	$0.251_{0.020}$	5.9	1162165

Table 4: Fixed time runs of larger benchmark instances. Various core sizes are shown for 10% negative coeffecients. All values shown are averaged over 10 problem instances.

5 Related Work

The most closely related work to this paper is the application of the core concept to the MKP (13, 14). We extend the results therein to general 0/1 Integer Programs, and show that the core concept continues to be valuable in the more general case.

Recently, very interesting results have been achieved with heuristics for 0/1 Mixed Integer Programming Problems with the goal of devising better feasible solutions earlier in the optimization process. Local Branching (4) combines local search and general branch-and-bound by introducing local branching constraints forcing the search to explore the neighborhoods of current feasible solutions first.

In Relaxation Induced Neighborhood Search (RINS) (3) subproblems for finding better feasible solutions are solved at some nodes the branch-and-bound tree. The subproblems are obtained by fixing the variables having identical values in the current best feasible solution and in the current solution of the LP-relaxation, leaving the remaining variables free.

RINS and local branching are local-search based ideas, reducing the subproblems to certain neighborhoods around a currently feasible solution. Our approach requires an LP solution only, and does not make use of feasible solutions at all.

6 Conclusions

We have extended the core concept, previously successfully used for finding better solutions to Multiple Knapsack Problems to general 0/1 Integer Programs. We provided an ordering of the variables using dual information, which results in a compact split interval just as for the standard MKP. This ordering is used to reduce the size of the tackled instances and obtain near-optimal solutions in shorter run-times. Our computational experiments show, that for challenging 0/1 Integer Programs with a large number of variables compared to the number of constraints, the core concept provides better solutions than directly solving the original problem. In the future we plan to test our approach on other 0/1 MIP test instances.

References

[1] E. Balas and E. Zemel. An algorithm for large zero-one knapsack problems. *Operations Research*, 28(5):1130–1154, 1980.

[2] P.C. Chu and J.E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4(1):63–86, 1998.

[3] E. Danna, E. Rothberg, and C. Le Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming, Series A*, 102:71–90, 2005.

[4] Matteo Fischetti and Andrea Lodi. Local Branching. *Math. Programming Series B*, 98:23–47, 2003.

[5] B. Gavish and H. Pirkul. Efficient algorithms for solving the multiconstraint zero-one knapsack problem to optimality. *Mathematical Programming*, 31:78–105, 1985.

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

[7] F. Glover and G.A. Kochenberger. Critical event tabu search for multidimensional knapsack problems. In I.H. Osman and J.P. Kelly, editors, *Metaheuristics: Theory and Applications*, pages 407–427. Kluwer Academic Publishers, 1996.

[8] Carlos Gomes da Silva, João Clímaco, and José Figueira. Core problems in bi-criteria {0,1}-knapsack: new developments. Technical Report 12/2005, INESC-Coimbra, 2005.

[9] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[10] S. Martello and P. Toth. A new algorithm for the 0–1 knapsack problem. *Management Science*, 34:633–644, 1988.

[11] D. Pisinger. An expanding-core algorithm for the exact 0-1 knapsack problem. *European Journal of Operational Research*, 87(1):175–187, 1995.

[12] D. Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45(5):758–767, 1997.

[13] J. Puchinger, G.R. Raidl, and U. Pferschy. The core concept for the multidimensional knapsack problem. In *Evolutionary Computation in Combinatorial Optimization - EvoCOP 2006*, volume 3906 of *LNCS*, pages 195–208. Springer, 2006.

[14] J. Puchinger, G.R. Raidl, and U. Pferschy. The multidimensional knapsack problem: Structure and algorithms. Technical Report 006149, National ICT Australia, Melbourne, Australia, March 2007. submitted for publication.

[15] G.R. Raidl and J. Gottlieb. Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem. *Evolutionary Computation*, 13(4):441–475, 2005.

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

[17] M. Vasquez and J.K. Hao. A hybrid approach for the 0–1 multidimensional knapsack problem. *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 328–333, 2001.

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

[19] H. M. Weingartner and D. N. Ness. Methods for the solution of the multidimensional 0/1 knapsack problem. *Operations Research*, 15:83–103, 1967.