The Core Concept for the Multidimensional Knapsack Problem *

Jakob Puchinger[1], Günther R. Raidl[1], and Ulrich Pferschy[2]

[1] Institute of Computer Graphics and Algorithms
Vienna University of Technology, Vienna, Austria
{puchinger|raidl}@ads.tuwien.ac.at
[2] Institute of Statistics and Operations Research
University of Graz, Austria
pferschy@uni-graz.at

Abstract. We present the newly developed core concept for the Multidimensional Knapsack Problem (MKP) which is an extension of the classical concept for the one-dimensional case. The core for the multidimensional problem is defined in dependence of a chosen efficiency function of the items, since no single obvious efficiency measure is available for MKP. An empirical study on the cores of widely-used benchmark instances is presented, as well as experiments with different approximate core sizes. Furthermore we describe a memetic algorithm and a relaxation guided variable neighborhood search for the MKP, which are applied to the original and to the core problems. The experimental results show that given a fixed run-time, the different metaheuristics as well as a general purpose integer linear programming solver yield better solution when applied to approximate core problems of fixed size.

1 Introduction

The Multidimensional Knapsack Problem (MKP) is a well-studied, strongly NP-hard combinatorial optimization problem occurring in many different applications. It can be defined by the following ILP:

$$\text{(MKP)} \qquad \text{maximize} \quad z = \sum_{j=1}^{n} p_j x_j \tag{1}$$

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

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

Given are n items with profits $p_j > 0$ and m resources with capacities $c_i > 0$. Each item j consumes an amount $w_{ij} \geq 0$ from each resource i. The goal is to

* This work is supported by RTN ADONET under grant 504438 and the Austrian Science Fund (FWF) under grant P16263-N04.

select a subset of items with maximum total profit, see (1); chosen items must, however, not exceed resource capacities, see (2). The 0–1 decision variables x_j indicate which items are selected.

A comprehensive overview on practical and theoretical results for the MKP can be found in the monograph on knapsack problems by Kellerer et al. [8]. Besides exact techniques for solving small to moderately sized instances, see [8], many kinds of metaheuristics have already been applied to the MKP. To our knowledge, the method currently yielding the best results, at least for commonly used benchmark instances, was described by Vasquez and Hao [18] and has recently been refined by Vasquez and Vimont [19]. Various other metaheuristics have been described for the MKP [5, 3], including several variants of hybrid evolutionary algorithms (EAs); see [16] for a survey and comparison of EAs for the MKP.

We first introduce the core concept for KP, and then expand it to MKP with respect to different efficiency measures. We then give some results of an empirical study of the cores of widely-used benchmark instances and present the application of the general ILP-solver CPLEX to MKP cores of fixed sizes. Furthermore we present a Memetic Algorithm (MA) and a Relaxation Guided Variable Neighborhood Search (RGVNS) applied to cores of hard to solve benchmark instances. We finally conclude by summarizing our work.

2 The Core Concept

The core concept was first presented for the classical 0/1-knapsack problem [1], which led to very successful KP algorithms [9, 11, 12]. The main idea is to reduce the original problem by only considering a core of items for which it is hard to decide if they will occur in an optimal solution or not, whereas the variables for all items outside the core are fixed to certain values.

2.1 The Core Concept for KP

The one-dimensional 0/1-knapsack problem (KP) considers items $j = 1, \ldots, n$, associated profits p_j, and weights w_j. A subset of these items has to be selected and packed into a knapsack having a capacity c. The total profit of the items in the knapsack has to be maximized, while the total weight is not allowed to exceed c. Obviously, KP is the special case of MKP with $m = 1$.

If the items are sorted according to decreasing efficiency values

$$e_j = \frac{p_j}{w_j}, \tag{4}$$

it is well known that the solution of the LP-relaxation consists in general of three consecutive parts: The first part contains variables set to 1, the second part consists of at most one split item s, whose corresponding LP-values is fractional, and finally the remaining variables, which are always set to zero, form the third part. For most instances of KP (except those with a very special

structure of profits and weights) the integer optimal solution closely corresponds to this partitioning in the sense that it contains most of the highly efficient items of the first part, some items with medium efficiencies near the split item, and almost no items with low efficiencies from the third part. Items of medium efficiency constitute the so called core.

Balas and Zemel [1] gave the following precise definition of the core of a one-dimensional 0/1-knapsack problem, based on the knowledge of an optimal integer solution x^*. Assume that the items are sorted according to decreasing efficiencies and let

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

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.

The KP Core (KPC) problem is defined as

$$\text{(KPC)} \qquad \text{maximize} \quad z = \sum_{j \in C} p_j x_j + \tilde{p} \tag{6}$$

$$\text{subject to} \quad \sum_{j \in C} w_j x_j \leq c - \tilde{w}, \tag{7}$$

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

with $\tilde{p} = \sum_{j=1}^{a-1} p_j$ and $\tilde{w} = \sum_{j=1}^{a-1} w_j$. The solution of KPC would suffice to compute the optimal solution of KP, which, however, has to be already partially known to determine C. Pisinger [12] reported experimental investigations of the exact core size. He moreover studied the hardness of core problems, giving also a model for their expected hardness in [13].

The first class of core algorithms is based on solving a core problem with an approximate core of fixed size $c = \{s - \delta, \ldots, s + \delta\}$ with various choices of δ, e.g. with δ being a constant or $\delta = \sqrt{n}$. An example is the MT2 algorithm by Martello and Toth [9].

Since it is impossible to estimate the core size in advance, Pisinger proposed two expanding core algorithms. Expknap [11] uses branch and bound for enumeration, whereas Minknap [12] (which enumerates at most the smallest symmetrical core) uses dynamic programming. For more details on core algorithms for KP we refer to Kellerer et al. [8].

2.2 The Core Concept for MKP

The previous definition of the core for KP can be expanded to MKP without major difficulties. The main problem, however, lies in the fact that there is no obvious efficiency measure.

Efficiency measures for MKP. Consider the most obvious form of efficiency for the MKP which is a direct generalization of the one-dimensional case:

$$e_j(\text{simple}) = \frac{p_j}{\sum_{i=1}^{m} w_{ij}}. \tag{9}$$

Different orders of magnitude of the constraints are not considered and a single constraint may dominate the others. This drawback can easily be avoided by scaling:

$$e_j(\text{scaled}) = \frac{p_j}{\sum_{i=1}^{m} \frac{w_{ij}}{c_i}}. \tag{10}$$

Taking into account the relative contribution of the constraints Senju and Toyoda [17] get:

$$e_j(\text{st}) = \frac{p_j}{\sum_{i=1}^{m} w_{ij}(\sum_{j=1}^{n} w_{ij} - c_i)}. \tag{11}$$

For more details on efficiency values we refer to Kellerer et al. [8] where a general form of efficiency is defined by introducing relevance values r_i for every constraint:

$$e_j(\text{general}) = \frac{p_j}{\sum_{i=1}^{m} r_i w_{ij}}. \tag{12}$$

The relevance values r_i can also be seen as kind of surrogate multipliers. Pirkul calculates good multipliers heuristically [10]. Fréville and Plateau [4] suggested setting

$$r_i = \frac{\sum_{j=1}^{n} w_{ij} - c_i}{\sum_{j=1}^{n} w_{ij}}, \tag{13}$$

giving the efficiency value $e_j(\text{fp})$. Setting the relevance values r_i to the values of an optimal solution to the dual problem of the MKP's LP-relaxation was a successful choice in [3], yielding the efficiency value $e_j(\text{duals})$.

The MKP Core. Since there are several possibilities of defining efficiency measures for MKP, the core and the core problem have to be defined depending on a specific efficiency measure e. Let x^* be an optimal solution and assume that the items are sorted according to decreasing efficiency e, then let

$$a_e := \min\{j \mid x_j^* = 0\}, \qquad b_e := \max\{j \mid x_j^* = 1\}. \tag{14}$$

The core is given by the items in the interval $C_e := \{a_e, \ldots, b_e\}$, and the core problem is defined as

$$(\text{MKPC}_\text{e}) \qquad \text{maximize} \quad z = \sum_{j \in C} p_j x_j + \tilde{p} \tag{15}$$

$$\text{subject to} \quad \sum_{j \in C} w_{ij} x_j \leq c_i - \tilde{w}_i, \quad i = 1, \ldots, m \tag{16}$$

$$x_j \in \{0, 1\}, \quad j \in C, \tag{17}$$

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 in general does not consist of a single fractional split item. But up to m fractional values give rise to a whole *split interval* $S_e := \{s_e, \ldots, t_e\}$, where s_e and t_e are the first and the last index of variables with fractional values after sorting by efficiency e. Note that depending on the choice of the efficiency measure, the split interval can also contain variables with integer values. Moreover, the sets S_e and C_e can have almost any relation to each other, from inclusion to disjointness. For a "reasonable" choice of e, we expected them, however, to overlap to a large extent.

If the dual solution values of the LP-relaxation are taken as relevance values, the split interval S_e resulting from the corresponding efficiency values $e_j(\text{duals})$ can be precisely characterized. Let x^{LP} be the optimal solution of the LP-relaxation of MKP.

Theorem 1.

$$x_j^{LP} = \begin{cases} 1 & \textit{if } \ e_j > 1\,, \\ \in [0,1] & \textit{if } \ e_j = 1\,, \\ 0 & \textit{if } \ e_j < 1\,. \end{cases} \tag{18}$$

Proof. The dual LP associated with the LP-relaxation of MKP is given by

$$\text{(D(MKP))} \qquad \text{minimize} \quad \sum_{i=1}^{m} c_i u_i + \sum_{j=1}^{n} v_j \tag{19}$$

$$\text{subject to} \quad \sum_{i=1}^{m} w_{ij} u_i + v_j \geq p_j, \quad j = 1, \ldots, n \tag{20}$$

$$u_i, v_j \geq 0, \quad i = 1, \ldots, m,\ j = 1, \ldots, n, \tag{21}$$

where u_i are the dual variables corresponding to the capacity constraints (2) and v_j correspond to the inequalities $x_j \leq 1$. For the optimal primal and dual solutions the following complementary slackness conditions hold (see any textbook on linear programming, e.g. [2]):

$$x_j \left(\sum_{i=1}^{m} w_{ij} u_i + v_j - p_j \right) = 0 \tag{22}$$

$$v_j (x_j - 1) = 0 \tag{23}$$

Recall that $e_j(\text{duals}) = \frac{p_j}{\sum_{i=1}^{m} u_i w_{ij}}$. Hence, $e_j > 1$ implies $p_j > \sum_{i=1}^{m} w_{ij} u_i$, which means that (20) can only be fulfilled by $v_j > 0$. Now, (23) immediately yields $x_j = 1$, which proves the first part of the theorem.

If $e_j < 1$, there is $p_j < \sum_{i=1}^{m} w_{ij} u_i$ which together with $v_j \geq 0$ makes the second factor of (22) strictly positive and requires $x_j = 0$. This proves the theorem since nothing has to be shown for $e_j = 1$. □

It follows from Theorem 1 that $S_e \subseteq \{j \mid e_j = 1,\ j = 1, \ldots, n\}$. It should be noted that the theorem gives only a structural result which does not yield any direct algorithmic advantage to compute the primal solution x^{LP} since it requires knowing the dual optimal solution.

3 Experimental Study of MKP Cores and Core Sizes

3.1 MKP Cores and Efficiency Measures

In order to analyze the core sizes in dependence on different efficiency values, we performed an empirical in-depth examination on smaller instances of Chu and Beasley's benchmark library[3]. Chu and Beasley [3] generated the instances as suggested by Fréville and Plateau [4]. The instance classes consist of ten instances each with $n \in \{100, 250, 500\}$ items, $m \in \{5, 10, 30\}$ constraints, and tightness ratios $\alpha = c_i / \sum_{j=1}^{n} w_{ij}$, $\alpha \in \{0.25, 0.5, 0.75\}$.

For the empirical results presented in this section, we used the smaller instances, which could be solved to proven optimality in reasonable time using the ILP-solver CPLEX, with $n = 100$ and $m \in \{5, 10\}$, and $n = 250$ and $m = 5$.

In Table 1 we examine cores devised using the scaled efficiency e(scaled), the efficiency e(st), the efficiency e(fp) as defined in equations (12) and (13), and finally the efficiency e(duals) setting the relevance values r_i of equation (12) to the optimal dual variable values of the MKP's LP-relaxation. Listed are average values of the sizes (in percent of the number of items) of the split interval ($|S_e|$) and of the exact core ($|C_e|$), the percentage of how much the split interval covers the core (ScC) and how much the core covers the split interval (CcS), and the distance (in percent of the number of items) between the center of the split interval and the center of the core (C_{dist}).

As expected from Theorem 1, the smallest split intervals, consisting of the fractional variables only are derived with e(duals). They further yield the smallest cores. Using one of the other efficiency measures results in significantly larger split intervals and cores. Furthermore, the smallest distances between the centers of the split intervals and the cores are produced by e(duals) for almost all the subclasses. The most promising information for devising approximate cores are therefore available from the split intervals generated with e(duals), on which we will concentrate our further investigations.

3.2 A Fixed Core Approach

In order to evaluate the influence of core sizes on solution quality and run-times, we propose a fixed core size algorithm, where we solve approximate cores using the general purpose ILP-solver CPLEX 9.0. We performed the experiments on a 2.4 GHz Pentium 4 computer.

In analogy to KP, the approximate core is generated by adding δ items on each side of the center of the split interval. We created the cores with the e(duals) efficiency. The values of δ were chosen in accordance with the results of the previous section, where an average core size of about $0.2n$ was observed. Table 2 lists average objective values and run-times for the original problem, and percentage gaps to the optimal solution ($\overline{\%_{\text{opt}}} = 100 \cdot (z^* - z)/z^*$), the number of times the optimum was reached (#), as well as the average run-times $\overline{\%t}$ (in percent of the run-time required for solving the original problem) for cores of different sizes.

[3] `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`

			e(scaled)					e(st)				
n	m	α	$\lvert S_e\rvert$	$\lvert C_e\rvert$	ScC	CcS	C_{dist}	$\lvert S_e\rvert$	$\lvert C_e\rvert$	ScC	CcS	C_{dist}
100	5	0.25	23.40	30.50	72.69	94.71	4.05	27.20	30.20	78.85	88.11	4.80
		0.5	29.50	37.60	71.93	88.45	5.95	27.00	35.60	69.88	89.01	5.90
		0.75	24.30	27.00	72.61	83.13	5.05	22.80	25.20	77.72	84.08	4.30
250	5	0.25	17.44	22.40	77.20	97.38	1.88	17.12	22.20	76.91	94.62	2.46
		0.5	22.88	29.44	71.71	94.25	3.44	23.76	30.88	74.95	94.69	4.04
		0.75	11.44	17.84	56.14	88.45	4.60	11.96	16.64	63.82	85.86	3.62
100	10	0.25	42.60	38.30	92.62	84.39	4.35	43.30	38.20	88.78	79.36	5.55
		0.5	39.40	45.20	80.80	91.20	5.30	44.40	46.50	85.43	88.49	5.65
		0.75	37.50	34.80	94.29	86.42	2.55	38.60	36.20	93.04	87.16	2.10
Average			27.61	31.45	76.67	89.82	4.13	28.46	31.29	78.82	87.93	4.27

			e(fp)					e(duals)				
n	m	α	$\lvert S_e\rvert$	$\lvert C_e\rvert$	ScC	CcS	C_{dist}	$\lvert S_e\rvert$	$\lvert C_e\rvert$	ScC	CcS	C_{dist}
100	5	0.25	24.70	30.10	75.50	91.94	4.20	5.00	20.20	28.12	100.00	3.30
		0.5	27.10	35.80	70.36	89.74	6.35	5.00	22.10	27.49	100.00	3.45
		0.75	23.20	26.10	74.47	84.22	4.55	5.00	19.60	26.95	100.00	3.20
250	5	0.25	16.92	21.72	76.87	95.63	2.24	2.00	12.68	18.16	100.00	2.46
		0.5	22.96	29.68	74.79	95.02	3.56	2.00	12.20	18.45	100.00	1.38
		0.75	11.40	17.12	59.00	87.27	4.06	2.00	10.40	20.18	100.00	1.56
100	10	0.25	42.10	38.20	90.41	83.74	4.75	10.00	23.20	46.57	100.00	2.90
		0.5	41.90	45.60	84.52	90.85	5.15	9.80	25.70	48.17	95.00	3.15
		0.75	37.90	35.30	94.55	86.96	2.40	9.70	18.80	55.74	99.00	2.75
Average			27.58	31.07	77.83	89.49	4.14	5.61	18.32	32.20	99.33	2.68

Table 1. Split intervals, core sizes and their mutual coverages and distances for different efficiency values (average percent values taken from 10 instances and average over all problem classes).

Observing the results of CPLEX applied to cores of different sizes, we see that smaller cores can be solved substantially faster and the obtained solution values are only slightly worse than the optimal ones given by the *no core* column. The best results with respect to average run-times were achieved with $\delta = 0.1n$, the run-time could be reduced by factors going from 3 to 1000, whereas, most importantly, the obtained objective values are very close to the respective optima (0.1% on average). Solving the bigger cores needs more run-time, but almost all of the optimal results could be reached, with still significant time savings.

4 Applying Metaheuristics to the Core

The question of how the reduction to MKP cores influences the performance of metaheuristics arises due to the observed differences in run-times and solution qualities of the previous section. Furthermore the core concept might enable us to find better solutions for larger instances which cannot be solved to optimality. We therefore study a memetic algorithm and a relaxation guided variable neighborhood search for solving the MKP, applied to MKP cores. The results of Section 3.2 indicate that the obtained solutions should be good approximations of the overall MKP optimum.

n	m	α	no core		$\delta = 0.1n$			$\delta = 0.15n$		
			$\overline{z}$	$\overline{t}[s]$	$\overline{\%_{\text{opt}}}$	$\#$	$\overline{\%t}$	$\overline{\%_{\text{opt}}}$	$\#$	$\overline{\%t}$
100	5	0.25	24197	21	0.097	5	1	0.034	7	9
		0.5	43253	27	0.053	4	1	0.018	6	6
		0.75	60471	6	0.038	5	4	0.021	7	17
250	5	0.25	60414	1474	0.008	7	36	0.003	9	81
		0.5	109293	1767	0.002	8	21	0.000	10	63
		0.75	151560	817	0.000	10	17	0.000	10	47
100	10	0.25	22602	189	0.473	1	0	0.152	4	1
		0.5	42661	97	0.234	3	0	0.084	5	1
		0.75	59556	29	0.036	6	0	0.015	8	3
Average			63778	492	0.105	5.4	9	0.036	7.3	25

n	m	α	$\delta = 0.2n$			$\delta = 2m + 0.1n$			$\delta = 2m + 0.2n$		
			$\overline{\%_{\text{opt}}}$	$\#$	$\overline{\%t}$	$\overline{\%_{\text{opt}}}$	$\#$	$\overline{\%t}$	$\overline{\%_{\text{opt}}}$	$\#$	$\overline{\%t}$
100	5	0.25	0.015	9	32	0.015	9	32	0.000	10	62
		0.5	0.002	9	24	0.002	9	24	0.002	9	64
		0.75	0.001	9	39	0.001	9	39	0.000	10	61
250	5	0.25	0.000	10	82	0.003	9	69	0.000	10	91
		0.5	0.000	10	67	0.000	10	59	0.000	10	73
		0.75	0.000	10	72	0.000	10	40	0.000	10	61
100	10	0.25	0.002	9	10	0.000	10	46	0.000	10	66
		0.5	0.030	8	13	0.022	8	60	0.000	10	75
		0.75	0.011	9	22	0.000	10	54	0.000	10	70
Average			0.007	9.2	40	0.005	9.3	47	0.000	9.9	69

Table 2. Solving cores of different sizes exactly (average over 10 instances and average over all problem classes).

4.1 A Memetic Algorithm

The MA which we consider here is based on Chu and Beasley's principles and includes some improvements suggested in [15, 6, 16]. The framework is steady-state and the creation of initial solutions is guided by the LP-relaxation of the MKP, as described in [6]. Each new candidate solution is derived by selecting two parents via binary tournaments, performing uniform crossover on their characteristic vectors x, flipping each bit with probability $1/n$, performing repair if a capacity constraint is violated, and always performing local improvement. If such a new candidate solution is different from all solutions in the current population, it replaces the worst of them.

Both, repair and local improvement, are based on greedy first-fit strategies and guarantee that any resulting candidate solution lies at the boundary of the feasible region, where optimal solutions are always located. The repair procedure considers all items in a specific order Π and removes selected items ($x_j = 1 \rightarrow x_j = 0$) as long as any capacity constraint is violated. Local improvement works vice-versa: It considers all items in the reverse order $\overline{\Pi}$ and selects items not yet appearing in the solution as long as no capacity limit is exceeded.

Crucial for these strategies to work well is the choice of the ordering Π. Items that are likely to be selected in an optimal solution must appear near the end of Π. Following the results of Section 3.1 we determine Π by ordering the items according to $e(\text{duals})$, as it has also been previously done in [3].

4.2 A Relaxation Guided VNS for the MKP

Relaxation Guided Variable Neighborhood Search (RGVNS) [14] is a recently developed Variable Neighborhood Search (VNS) [7] variant where the neighborhood-order of Variable Neighborhood Descent (VND) is dynamically determined by solving relaxations of the neighborhoods. The RGVNS used here for the MKP, is a slightly improved version of the approach described in [14].

Representation and Initialization. Solutions are directly represented by binary strings, and all our neighborhoods are defined on the space of feasible solutions only. We denote by $I_1(x^f) = \{j \mid x_j^f = 1\}$ the index-set of the items contained in the knapsack of a current solution x^f and by $I_0(x^f) = \{j \mid x_j^f = 0\}$ its complement. The initial solution for the RGVNS is generated using a greedy first-fit heuristic, considering the items in a certain order, determined by sorting the items according to decreasing values of the solutions to the MKP's LP-relaxation; see [16].

ILP Based Neighborhoods. We want to force a certain number of items of the current feasible solution x^f to be removed from or added to the knapsack. This is realized by adding neighborhood-defining constraints depending on x^f to the ILP formulation of the MKP.

In the first neighborhood, *ILP-Remove-and-Fill* $IRF(x^f, k)$, we force precisely k items from I_1 to be removed from the knapsack, and any combination of items from I_0 is allowed to be added to the knapsack as long as the solution remains feasible. This is accomplished by adding the equation $\sum_{j \in I_1(x^f)} x_j = \sum_{j \in I_1(x^f)} x_j^f - k$ to (1)–(3).

In the second neighborhood, *ILP-Add-and-Remove* $IAR(x^f, k)$, we force precisely k items not yet packed, i.e. from I_0, to be included in the knapsack. To achieve feasibility any combination of items from I_1 may be removed. This is achieved by adding $\sum_{j \in I_0(x^f)} x_j = k$ to (1)–(3).

As relaxations $IRF^R(x^f, k)$ and $IAR^R(x^f, k)$ we use the corresponding LP-relaxations in which the integrality constraints (3) are replaced by $0 \leq x_j \leq 1$, $j = 1, \ldots, n$. For searching the (integer) neighborhoods we use a general purpose ILP-solver (CPLEX) with a certain time limit.

Classical Neighborhoods. As first neighborhood we use a simple swap $SWP(x^f)$, where a pair of items (x_i^f, x_j^f), with $i \in I_1$ and $j \in I_0$, is exchanged, i.e. $x_i^f := 0$ and $x_j^f := 1$. Infeasible solutions are discarded. Note that this neighborhood is contained in both, $IRF(x^f, 1)$ and $IAR(x^f, 1)$. Its main advantage is that it can be explored very fast.

Based on the ideas of Chu and Beasley [3] and as another simplification of IRF and IAR but an extension of SWP, we define two additional neighborhoods based on greedy concepts. In the first case, the *Remove-and-Greedy-Fill* neighborhood $RGF(x^f, k)$, k items are removed from x^f; i.e. a k-tuple of variables from $I_1(x^f)$ is flipped. The resulting solution is then locally optimized as

described in Section 4.1. In the second case, the *Add-and-Greedy-Repair* neighborhood $AGR(x^f, k)$, k items are added to x^f; i.e. k variables from $I_0(x^f)$ are flipped. The resulting solution, which is usually infeasible, is then repaired and locally improved as previously described.

Relaxation Guided VND. The Relaxation Guided VND (RGVND) is based on the previously defined neighborhoods and follows a best neighbor strategy. The faster to solve neighborhoods are ordered as follows: $\mathcal{N}_1{:=}SWP(x^f)$, $\mathcal{N}_2{:=}RGF(x^f, 1)$, $\mathcal{N}_3{:=}AGR(x^f, 1)$. If none of these neighborhoods leads to an improved solution, we solve the LP-relaxations of $IRF(x^f, k)$ and $IAR(x^f, k)$ for $k = 1, \ldots, k_{\max}$, where $k_{\max}$ is a prespecified upper limit on the number of items we want to remove or add. The neighborhoods are then sorted according to decreasing LP-relaxation solution values. Ties are broken by considering smaller ks earlier. Only the first $\beta_{\max}$ ILP-based neighborhoods are explored before shaking.

Shaking. After RGVND has terminated, shaking is performed for diversification. It flips κ different randomly selected variables of the currently best solution and applies greedy repair and local improvement as previously described for the MA. As usual in general VNS, κ runs from 1 to some $\kappa_{\max}$ and is reset to 1 if an improved solution was found. Furthermore, the whole process is iterated until a termination criterion, in our case the CPU-time, is met.

5 Computational Experiments

We present several computational experiments where we evaluated the influence of differently sized cores on the performance of CPLEX, the presented MA, and RGVNS. The algorithms were given 500 seconds per run. Since the MA converges much earlier, it was restarted every 1 000 000 generations, always keeping the so-far best solution in the population. In RGVNS, CPLEX was given a maximum of 5 seconds for exploring the ILP-based neighborhoods, $k_{\max}$ and $\beta_{\max}$ were set to 10, and $\kappa_{\max} = n$. We used the hardest instances of the Chu and Beasley benchmark set, i.e. those with $n = 500$ items and $m \in \{5, 10, 30\}$ constraints. As before, CPLEX 9.0 was used and we performed the experiments on a 2.4 GHz Pentium 4 computer.

In Table 3 we display the results of CPLEX applied to cores of different sizes. For comparison CPLEX was also applied to the original problem with the same time limit. We list averages over ten instances of the percentage gaps to the optimal objective value of the LP-relaxation ($\overline{\%_{\mathrm{LP}}} = 100 \cdot (z^{\mathrm{LP}} - z)/z^{\mathrm{LP}}$), the number of times this core size yielded the best solution of this algorithm (#), and the number of explored nodes of the branch and bound tree.

First, it can be noticed that CPLEX applied to approximate cores of different sizes yields, on average, better results than CPLEX applied to the original problem. Second, the number of explored nodes increases with decreasing problem/core size. The best average results are obtained with higher core sizes.

m	α	no core			$\delta = 0.1n$			$\delta = 0.15n$			$\delta = 0.2n$		
		$\overline{\%_{\text{LP}}}$	#	$\overline{\text{Nnodes}}$	$\overline{\%_{\text{LP}}}$	#	$\overline{\text{Nnodes}}$	$\overline{\%_{\text{LP}}}$	#	$\overline{\text{Nnodes}}$	$\overline{\%_{\text{LP}}}$	#	$\overline{\text{Nnodes}}$
5	0.25	0.080	5	5.50E5	0.075	9	1.00E6	0.076	9	9.85E5	0.076	8	8.34E5
	0.5	0.040	6	5.06E5	0.039	7	1.05E6	0.039	9	1.00E6	0.039	9	8.38E5
	0.75	0.025	6	5.36E5	0.024	10	1.05E6	0.025	8	1.02E6	0.025	8	9.04E5
10	0.25	0.206	1	3.15E5	0.198	5	1.10E6	0.195	6	6.99E5	0.198	4	5.68E5
	0.5	0.094	4	3.01E5	0.088	8	1.11E6	0.090	6	6.95E5	0.092	5	5.73E5
	0.75	0.066	4	3.05E5	0.065	5	1.07E6	0.064	7	6.83E5	0.065	7	5.59E5
30	0.25	0.598	2	1.11E5	0.621	0	4.22E5	0.566	4	3.06E5	0.537	6	2.28E5
	0.5	0.258	2	1.15E5	0.246	3	4.50E5	0.243	4	3.28E5	0.250	2	2.38E5
	0.75	0.158	2	1.12E5	0.151	6	4.48E5	0.160	1	3.14E5	0.151	5	2.36E5
Average		0.169	3.6	3.17E5	0.167	5.9	8.55E5	0.162	6.0	6.70E5	0.159	6.0	5.53E5

Table 3. Solving cores of different sizes with CPLEX (avgerage over 10 instances and average over all problem classes, $n = 500$).

In Table 4 the results of the MA applied to approximate cores of different sizes are shown. In order to evaluate the benefits of using a core-based approach, we also applied the MA to the original problem. The table lists ($\overline{\%_{\text{LP}}}$), the number of times this core size yielded the best solution of this algorithm (#), and the average numbers of MA iterations.

As observed with CPLEX, the use of approximate cores consistently increases the achieved solution quality. The core size has a significant influence on the number of iterations performed by the MA, which can be explained by the smaller size of the problem to be solved. This also seems to be a reason for the better results, since more candidate solutions can be examined in the given runtime. Furthemore, the search space of the MA is restricted to a highly promising part of the original search space. The best average results were obtained with $\delta = 0.15n$. The smaller approximate cores yield better results on average.

m	α	no core			$\delta = 0.1n$			$\delta = 0.15n$			$\delta = 0.2n$		
		$\overline{\%_{\text{LP}}}$	#	$\overline{\text{Niter}}$	$\overline{\%_{\text{LP}}}$	#	$\overline{\text{Niter}}$	$\overline{\%_{\text{LP}}}$	#	$\overline{\text{Niter}}$	$\overline{\%_{\text{LP}}}$	#	$\overline{\text{Niter}}$
5	0.25	0.078	6	1.40E7	0.073	10	5.08E7	0.074	9	4.07E7	0.074	9	3.33E7
	0.5	0.040	6	1.35E7	0.039	9	5.07E7	0.039	9	4.07E7	0.040	7	3.33E7
	0.75	0.025	7	1.46E7	0.024	9	5.07E7	0.024	10	4.08E7	0.024	9	3.34E7
10	0.25	0.208	5	1.26E7	0.202	5	4.54E7	0.202	6	3.62E7	0.208	4	2.90E7
	0.5	0.099	2	1.21E7	0.093	6	4.51E7	0.091	8	3.59E7	0.093	5	2.89E7
	0.75	0.066	6	1.31E7	0.065	8	4.53E7	0.067	4	3.59E7	0.068	4	2.87E7
30	0.25	0.604	1	9.10E6	0.573	5	3.08E7	0.575	5	2.39E7	0.569	6	1.92E7
	0.5	0.254	3	8.10E6	0.257	1	3.08E7	0.246	7	2.37E7	0.253	3	1.90E7
	0.75	0.159	4	8.12E6	0.156	5	3.14E7	0.157	3	2.35E7	0.157	5	1.96E7
Average		0.170	4.4	1.17E7	0.165	6.4	4.23E7	0.164	6.8	3.35E7	0.165	5.8	2.72E7

Table 4. Solving cores of different sizes with the MA (avgerage over 10 instances and average over all problem classes, $n = 500$).

In Table 5, the results of RGVNS when applied to approximate cores of different sizes are shown together with the results of RGVNS on the original problem. The table also displays the average total number of iterations performed by RGVND inside RGVNS.

The results obtained by RGVNS applied to the smaller approximate cores clearly dominate the results obtained without core and with $\delta = 0.2n$. This can be explained by the fact that CPLEX is used in RGVNS, and that it is able to find better solutions when dealing with smaller problem sizes. Interestingly, the number of iterations stays about the same for the different settings. The reason is that CPLEX is given the same constant time limit for searching for the neighborhoods within RGVND.

m	α	no core			$\delta = 0.1n$			$\delta = 0.15n$			$\delta = 0.2n$		
		$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\mathrm{Niter}}$	$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\mathrm{Niter}}$	$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\mathrm{Niter}}$	$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\mathrm{Niter}}$
5	0.25	0.088	4	230	0.080	5	208	0.080	6	223	0.082	4	230
	0.5	0.043	5	236	0.040	7	215	0.040	8	226	0.040	7	239
	0.75	0.027	5	246	0.026	8	230	0.026	8	252	0.026	7	240
10	0.25	0.230	0	225	0.198	7	200	0.211	2	193	0.210	3	205
	0.5	0.108	1	209	0.096	5	201	0.096	3	199	0.100	1	205
	0.75	0.069	2	208	0.066	7	207	0.066	7	211	0.066	4	214
30	0.25	0.595	5	202	0.599	3	196	0.593	4	191	0.609	5	195
	0.5	0.263	3	197	0.260	0	198	0.254	6	189	0.261	3	197
	0.75	0.168	2	191	0.158	5	191	0.164	3	187	0.164	2	191
Average		0.177	3.0	216	0.169	5.2	205	0.170	5.2	208	0.173	4.0	213

Table 5. Solving cores of different sizes with RGVNS (avgerage over 10 instances and average over all problem classes, $n = 500$).

Comparing our results to the best known solutions [19], we are able to reach the best solutions for $m = 5$, and stay only 0.5% below these solutions for $m \in \{10, 30\}$, requiring 500 seconds, whereas in [19] up to 33 hours were needed.

6 Conclusions

We presented the new core concept for the multidimensional knapsack problem, extending the core concept for the classical one-dimensional 0/1-knapsack problem. An empirical study of the exact core sizes of widely used benchmark instances with different efficiency measures was performed. The efficiency value using dual-variable values as relevance factors yielded the smallest possible split-intervals and the smallest cores.

We further studied the influence of restricting problem solving to approximate cores of different sizes, and observed significant differences in terms of run-time when applying the general-purpose ILP-solver CPLEX to approximate cores or to the original problem, whereas the objective values remained very close to the respective optima.

We finally applied CPLEX and two metaheuristics to approximate cores of hard to solve benchmark instances and observed that using approximate cores of fixed size instead of the original problem clearly and consistently improves the solution quality when using a fixed run-time.

In the future, we want to further examine the MKP core concept and possibly extend it to other combinatorial optimization problems.

References

1. E. Balas and E. Zemel. An algorithm for large zero-one knapsack problems. *Operations Research*, 28:1130–1154, 1980.
2. D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization.* Athena Scientific, 1997.
3. P. C. Chu and J. Beasley. A genetic algorithm for the multiconstrained knapsack problem. *Journal of Heuristics*, 4:63–86, 1998.
4. A. Fréville and G. Plateau. An efficient preprocessing procedure for the multidimensional 0–1 knapsack problem. *Discrete Applied Mathematics*, 49:189–212, 1994.
5. F. Glover and G. Kochenberger. Critical event tabu search for multidimensional knapsack problems. In I. Osman and J. Kelly, editors, *Metaheuristics: Theory and Applications*, pages 407–427. Kluwer Academic Publishers, 1996.
6. J. Gottlieb. On the effectivity of evolutionary algorithms for multidimensional knapsack problems. In C. Fonlupt et al., editors, *Proceedings of Artificial Evolution: Fourth European Conference*, volume 1829 of *LNCS*, pages 22–37. Springer, 1999.
7. P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In S. Voss, S. Martello, I. Osman, and C. Roucairol, editors, *Metaheuristics, Advances and Trends in Local Search Paradigms for Optimization*, pages 433–458. Kluwer, 1999.
8. H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems.* Springer, 2004.
9. S. Martello and P. Toth. A new algorithm for the 0-1 knapsack problem. *Management Science*, 34:633–644, 1988.
10. H. Pirkul. A heuristic solution procedure for the multiconstraint zero-one knapsack problem. *Naval Research Logistics*, 34:161–172, 1987.
11. D. Pisinger. An expanding-core algorithm for the exact 0–1 knapsack problem. *European Journal of Operational Research*, 87:175–187, 1995.
12. D. Pisinger. A minimal algorithm for the 0–1 knapsack problem. *Operations Research*, 45:758–767, 1997.
13. D. Pisinger. Core problems in knapsack algorithms. *Operations Research*, 47:570–575, 1999.
14. J. Puchinger and G. R. Raidl. Relaxation guided variable neighborhood search. In P. Hansen, N. Mladenović, J. A. M. Pérez, B. M. Batista, and J. M. Moreno-Vega, editors, *Proceedings of the 18th Mini Euro Conference on Variable Neighborhood Search*, Tenerife, Spain, 2005.
15. G. R. Raidl. An improved genetic algorithm for the multiconstrained 0–1 knapsack problem. In D. Fogel et al., editors, *Proceedings of the 5th IEEE International Conference on Evolutionary Computation*, pages 207–211. IEEE Press, 1998.
16. 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 Journal*, 13(4), to appear 2005.
17. S. Senju and Y. Toyoda. An approach to linear programming with 0–1 variables. *Management Science*, 15:196–207, 1968.
18. M. Vasquez and J.-K. Hao. A hybrid approach for the 0–1 multidimensional knapsack problem. In *Proceedings of the Int. Joint Conference on Artificial Intelligence 2001*, pages 328–333, 2001.
19. M. Vasquez and Y. Vimont. Improved results on the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 165:70–81, 2005.