Fast, Effective Heuristics for the 0–1 Multidimensional Knapsack Problem

Krzysztof Fleszar* and Khalil S. Hindi
Olayan School of Business
American University of Beirut (AUB)
P.O. BOX 11-0236
Riad El Solh, Beirut 1107 2020
Lebanon
Krzysztof.Fleszar@aub.edu.lb, Khalil.Hindi@aub.edu.lb

March 9, 2008

Abstract

The objective of the multidimensional knapsack problem (MKP) is to find a subset of items with maximum value that satisfies a number of knapsack constraints. Solution methods for MKP, both heuristic and exact, have been researched for several decades. This paper introduces several fast and effective heuristics for MKP that are based on solving the LP relaxation of the problem. Improving procedures are proposed to strengthen the results of these heuristics. Additionally, the heuristics are run with appropriate deterministic or randomly generated constraints imposed on the linear relaxation that allow generating a number of good solutions. All algorithms are tested experimentally on a widely used set of benchmark problem instances to show that they compare favourably with the best-performing heuristics available in the literature.

Keywords: multidimensional knapsack problem, heuristics, LP relaxation

*Corresponding author

1 Introduction

The multidimensional knapsack problem (MKP) concerns a set of items $N = \{1, \ldots, n\}$ to be packed in a knapsack that has a set of dimensions $M = \{1, \ldots, m\}$. The knapsack has a limited capacity in each dimension $i \in M$, denoted by b_i. Each item $j \in N$ has a weight in each dimension, denoted by a_{ij}, and a value, denoted by c_j. The objective is to find a subset of items that yields the maximum total value subject to the capacity constraints of the knapsack. Formally, the problem can be formulated as:

MKP:

$$\max \sum_{j \in N} c_j x_j \tag{1}$$

subject to

$$\sum_{j \in N} a_{ij} x_j \leq b_i \qquad i \in M \tag{2}$$

$$x_j \in \{0, 1\} \qquad j \in N, \tag{3}$$

where $a_{ij} \geq 0$ and $b_i > 0$. Following Freville and Plateau [8] we will also assume that $c_j > 0 \; \forall j$, $a_{ij} \leq b_i \; \forall i, j$, and $\sum_{j \in N} a_{ij} > b_i \; \forall i$. If some of these conditions are not satisfied, the problem can be easily reduced by eliminating constraints or fixing variables to 0 or 1. It can also be assumed without loss of generality that c_j, a_{ij}, and b_i are integer numbers $\forall i, j$.

Bertsimas and Demir [2] point out that the MKP arises in many real-world applications such as combinatorial auctions [18, 4], computer systems design [5], project selection [16] and cutting stock and cargo-loading [10]. Moreover, many complex problems can be transformed to MKPs or have close kinship with the MKP.

The practical and theoretical importance of the MKP has led to a large body of literature on both exact and approximate solution approaches. Freville [6] provides an excellent overview of the literature on the MKP and Freville and Hanafi [7] provide a survey of recently developed methods. Subsequently, Hanfi and Glover [11] offered an exploitation of nested inequalities and surrogate constraints on the MKP better than that proposed by Osorio et. al. [15], but did not offer computational results. Also, Akcay et. al. [1] proposed a greedy

heuristic ordering items by their value multiplied by the maximum number of copies of an item that could be accommodated with available resources.

2 Existing fast heuristics

Primal heuristics for the MKP begin with an empty solution and add items in a certain order without violating constraints. In contrast, dual heuristics begin with all items selected and remove items in a certain order until feasibility is achieved. A dual heuristic is usually followed by a primal heuristic that reintroduces items that have been removed unnecessarily.

Both primal and dual heuristics are greedy approaches. Many priority functions to order items have been suggested in the literature [6]. These are usually dynamically recalculated at each step of the heuristic, taking into account currently remaining slacks (or violations) of constraints. A static approach, i.e., computing priorities and ordering items once at the beginning, is faster but is thought to lead to inferior solutions.

Very good results are reported by Bertsimas and Demir [2] who introduce an adaptive-fixing heuristic based on the LP relaxation of the MKP. Initially, the heuristic fixes to zero or one all variables that in the LP relaxation have values equal to zero or one, respectively. To speed up computation, all variables that are smaller than a pre-set parameter, γ, are also fixed to zero. The linear relaxation is then repeatedly solved. At each iteration, all variables that have a zero value are fixed to zero, all variables that have a value of one are fixed to one, and additionally the variable with the smallest fractional value is fixed to zero. This process is repeated until all variables are fixed. Bertsimas and Demir compare the performance of their adaptive-fixing heuristic with the primal gradient heuristic of Toyoda [20], the dual gradient heuristic of Senju and Toyoda [19], the greedy-like heuristic of Loulou and Michaelides [13], and the incremental heuristic of Kochenberger et. al. [12]. They show that their heuristic is much more effective than all the others. Hence, comparing the performance of our heuristics with that Bertsimas and Demir's, in addition to others not considered by them, should suffice.

The dual heuristic of Magazine and Oguz [14] was not considered by Bertsi-

mas and Demir. It begins with all variables selected. Variables are then removed one at a time, according to a priority list based on Lagrange multipliers, until feasibility is achieved. Subsequently, variables are added if possible in the order of non-increasing c_j. Volgenant and Zoon [23] improved this heuristic by calculating more than one Lagrange multiplier at a time and Volgenant and Zwiers [24] further improved it by adding partial enumeration.

3 New fast heuristics

In this section, we present several new fast heuristics developed in the current work.

3.1 Primal LP-selection heuristic

This heuristic proceeds from the solution of the LP relaxation of MKP, with items ordered by non-increasing values of variables, breaking ties by non-increasing values of reduced costs. In effect, this amounts to selecting all the items that have a value of one, some of the items that have fractional values with priority given to larger values, and possibly some items with a value of zero with priority given to higher reduced costs. However, sorting all items in a definite order is important for a posteriori improvements that are based on removing items in the reverse order in which they were added (see section 3.3).

Since the reduced cost of a variable, x_j, is $\bar{c}_j = c_j - \sum_{i \in M} \lambda_i a_{ij}$, where λ_i is an optimal value of a dual variable associated with ith constraint, this order is similar to one achieved by sorting items by non-increasing $c_j / \sum_{i=1}^{m} \lambda_i a_{ij}$ (see [3], for example). However, our approach has the added advantage of ordering items with zero reduced cost by the fractional values of the corresponding variables. This is extremely important, since items for which $x_j = 1$ in the LP relaxation are all selected and resources become scarce only when the items with fractional x_j values are considered; making the order with which these items are considered what matters most.

3.2 Primal LP-adaptive-selection heuristic

Algorithm 1 below shows our heuristic. X denotes the set of indices of relaxed variables, X_0 and X_1 denote the set of indices of variables fixed to zero and one, respectively. Items in X_1 are also kept in list L in the order with which they are selected. Additionally, $a(X)$ is a vector of total resource consumption of items $j \in X$, and b is a vector of resource availability.

The LP relaxation of MKP is solved (SolveLP) first. All variables $x_j = 1$ are fixed to one and the respective items are added to list L. Moreover, all variables that cannot be fixed to one without violating constraints are fixed to zero. Then, repeatedly until all items are fixed to zero or one, the linear relaxation is resolved; the item with the highest variable value is added to L, breaking ties by the highest reduced cost, $\bar{c}_j$ and the corresponding variable is fixed to one; and variables that cannot be fixed to one without violating constraints are fixed to zero.

Algorithm 1: Primal LP-adaptive-selection heuristic

Output: List of items selected L
$X_0 = X_1 = \emptyset$;
$(x, \bar{c}) = \text{SolveLP}(X_0, X_1)$;
$X_1 = \{j \in N | x_j = 1\}$;
$X = N \setminus X_1$;
$L =$ items X_1 ordered by non-increasing $\bar{c}_j$;
foreach $j \in X | a(X_1 \cup \{j\}) \nleq b$ **do**
    $X = X \setminus \{j\}$;
    $X_0 = X_0 \cup \{j\}$;
while $X \neq \emptyset$ **do**
    $(x, \bar{c}) = \text{SolveLP}(X_0, X_1)$;
    $j = \text{arglexmax}_{j \in X}(x_j, \bar{c}_j)$;
    $X = X \setminus \{j\}$;
    $X_1 = X_1 \cup \{j\}$;
    Add j as the last element in L;
    foreach $j \in X | a(X_1 \cup \{j\}) \nleq b$ **do**
        $X = X \setminus \{j\}$;
        $X_0 = X_0 \cup \{j\}$;

This heuristic is similar to the one proposed by Bertsimas and Demir [2]. The most important difference is that instead of fixing at each stage the variable with the smallest value to zero, we fix the variable with the highest value to one. As a result, when our heuristic is finished, no item can be added to the

solution without violating constraints, while at the termination of the adaptive-fixing of Bertsimas and Demir, it may be possible to reintroduce items that were unnecessarily fixed to zero, which is something they do not mention.

3.3 Improving last added items

The simplest way to order items for a primal heuristic is by non-increasing values of c_j, but this approach usually leads to very poor results, since it ignores item sizes. However, if after adding some items, the remaining slacks are very tight, choosing the rest of the items by highest value may actually be the best approach. Therefore, we suggest the following procedure. At the termination of any primal heuristic, the solution may be improved at minimal CPU time by removing the last added item and filling the residual capacity by adding items, when possible, in non-increasing order of c_j. In the worst case, the item that has just been removed is reintroduced, but in many cases, it is replaced by one or several more valuable items, leading to an improvement of the solution value.

Loulou and Michaelides [13] introduce a similar idea. In their primal heuristic, which utilises a sophisticated priority function, they suggest switching to selecting items by non-increasing value when at least one of the resources becomes scarce. Our approach differs in that we first complete the primal heuristic and then backtrack by one item.

An extended version of our algorithm considers removing a number of items, ranging from one up to a certain limit. In every case, the solution is completed by adding items, when possible, by the non-increasing order of c_j and the best solution achieved is adopted.

A further extension is to perform enumerations of possible completions. First, the last added item is removed and all possible completions are enumerated, with items considered in the non-increasing order of c_js. Enumeration is then performed after two, three, ... items are removed, and the algorithm is terminated when a certain limit on the number of additions (i.e., items added that are later removed to enumerate combinations) is reached.

A similar partial enumeration has been applied by Volgenant and Zwiers [24]. However, their termination condition does not ensure termination in a reasonable time.

3.4 Cardinality constraint

Vasquez and Hao [21] and later Vasquez and Vimont [22] use, within their tabu-search algorithms for the MKP, an additional constraint on the cardinality of the set of selected items. In effect, they divide the search space into a number of subspaces, each with a different cardinality constraint, $\sum_{j \in N} x_j = k$, noting that this may be useful in guiding the search towards good solutions.

With either of the two LP-based heuristics described above, a cardinality constraint with different right-hand-side values may be used to determine several solutions. We start by solving the LP relaxation and calculating the base cardinality as $k_{\text{base}} = \lfloor \sum_{j \in N} x_j \rfloor$. The LP-based heuristic is then executed without a cardinality constraint, and its solution becomes the incumbent. Subsequently, the LP relaxation is solved and the heuristic is executed with a cardinality constraint having various values of k. The incumbent is updated whenever a better solution is found. In the first phase, k changes from $k_{\text{base}} + 1$ to $k_{\text{base}} + k_{\max}$, and in the second, k changes from k_{base} down to $k_{\text{base}} - k_{\max} + 1$. Either phase is terminated before k reaches the limit if the objective value of the cardinality-constrained LP relaxation drops below that of the incumbent.

Our approach differs from that of [21] and [22] in the way the values of k are chosen. In their work, k_{base} is calculated by rounding the sum of variable values to the nearest integer and then varying k from $k_{\text{base}} - k_{\max}$ to $k_{\text{base}} + k_{\max}$. However, our analysis of the quality of solutions for different values of k shows that our approach is better centred around good solutions.

With our primal LP-selection heuristic, using a cardinality constraint with a different right-hand-side value each time leads to different fractional values and reduced costs, and to different solutions. Similarly, with our primal LP-adaptive-selection heuristic, items are selected in a different order each time. However, in this case, the cardinality constraint may lead to an infeasible LP relaxation after some variables are fixed. If this occurs, the cardinality constraint is dropped and the heuristic is continued without it. Thus, in every attempt, a feasible solution is generated.

3.5 Random constraints

Use of cardinality constraints may be generalised to other integer-valued constraints; in particular to randomly generated constraints. Preliminary experimentation proved the following approach to be effective. The LP relaxation is solved and the values of the variables are recorded. To generate a constraint, every variable is included with a coefficient equal to 1 with a 50% probability and to -1 otherwise. k_{base} is then calculated by rounding down the fractional value of the left-hand-expression of the constraint to the nearest integer. The LP-based heuristic is then executed without the random constraint and subsequently repeatedly with the random constraint having the right-hand-side, k, change from $k_{\text{base}} + 1$ to $k_{\text{base}} + k_{\max}$ in the first phase, and from k_{base} down to $k_{\text{base}} - k_{\max} + 1$ in the second.

The advantage of using randomly generated constraints is that the approach may be repeated for more than one constraint, allowing for continued computation and further improvement of the incumbent. In fact, the algorithm may be executed as an anytime heuristic, terminating when a certain computation time elapses.

4 Computational study

4.1 Experimental setup

The benchmark problem instances of Chu and Beasley [3] are used in our numerical experiments. These were generated following the procedure suggested by Freville and Plateau [8]: $m = 5, 10, 30$, $n = 100, 250, 500$, a_{ij} are integer numbers drawn from $U(0, 1000)$, $b_i = \alpha \sum_{j \in N} a_{ij}$, where $\alpha = 1/4, 2/4, 3/4$ is a tightness ratio, and $c_j = \sum_{i=1}^{m} a_{ij}/m + 500 q_j$ where q_j is a real number drawn from $U(0, 1)$. For each combination of m, n, α, there are 10 problem instances, giving altogether 270 problem instances. All these were used in our computational experiments, unless otherwise stated specifically.

The problem instances were chosen for two reasons. First, they are correlated problems instances (for each item j, c_j is correlated to its a_{ij} values) which are generally more difficult to solve than uncorrelated problem instances

[17, 9] and have been experimentally shown to be very difficult to solve to optimality [3, 2]. Secondly, they are available from the well known OR-Library (`http://people.brunel.ac.uk/~mastjjb/jeb/info.html`), which makes it easy for researchers to compare their results with those presented here.

Computation has been carried out in C#.NET on a Dell Latitude D610 with a Pentium M 2GHz processor. The LPs have been solved using ILOG CPLEX 10.0.2, although any other linear programming solver could have been used. Processing times are reported in seconds, and deviations are calculated as 100(LP value − solution value)/(LP value), where 'LP value' is LP relaxation upper bound. Some authors use best-known solution values as the base for comparison; but these change over time, which makes future comparison difficult.

4.2 Heuristics not using LP relaxation

To put our LP-based heuristics in perspective and test the impact of the improving algorithms, we first present in Table 1 results on all 270 problem instances for some fast heuristics that do not involve solving the LP relaxation. The heuristics include dynamically applied primal heuristics ordering items by: value times maximum number of copies, $c_j \min_{i \in M | a_{ij} > 0} \lfloor s_i / a_{ij} \rfloor$, as suggested by Akcay et. al. [1]; value divided by the maximum relative item size, $c_j / \max_{i \in M} \{a_{ij}/s_i\}$; and value divided by the sum of relative item sizes, $c_j / \sum_{i \in M} a_{ij}/s_i$, where s_i denotes the current slack of the ith constraint. The latter heuristic is applied in a static and also in a partially dynamic manner. The dual heuristic of Volgenant and Zoon [23] is included for comparison, with the two versions of additional partial enumeration suggested by Volgenant and Zweirs [24]. All processing times reported are for Pentium M 2GHz processor, except for the heuristics of Volgenant and Zoon, and Volgenant and Zweirs that were tested on a Pentium 1.7GHz.

The heuristic of Akcay et. al. [1] and the second heuristic order items in a very similar manner and both achieve poor results compared to other methods. This is surprising given the fact that Akcay et. al. report robust results for the MKP, although their heuristic is mainly intended for the more general integer version of MKP. The heuristic of Volgenant and Zoon achieves slightly better

Table 1: Simple heuristics not using LP relaxation

heuristic	postproc.	dev. (%)		time (s.)	
		avg.	max.	avg.	max.
value×max.n.copies (dyn.) [1]	–	4.79	13.47	.0190	.11
value/max.rel.size (dyn.)	–	4.41	11.77	.0197	.13
Volgenant & Zoon [23]	–	4.03		$\ll 1$	$\ll 1$
	enum. [24]	3.76		< 1	< 1
	more enum. [24]	2.80		a few	13.60
value/sum.rel.size (static)	–	3.88	18.29	.0002	.02
	IL1	3.72	17.27	.0002	.02
	IL5	3.14	12.15	.0002	.02
value/sum.rel.size (dyn.)	–	1.53	8.47	.0146	.09
value/sum.rel.size (part.dyn.)	–	1.53	8.47	.0044	.03
	IL1	1.41	8.22	.0045	.03
	IL5	1.36	7.41	.0045	.03
	E1000	1.08	6.14	.0090	.05
	E10000	.99	6.14	.0256	.11

results, which are further improved by partial enumeration, albeit at a significant computation time. The surprising discovery here is that the first three heuristics (without enumeration) are on average worse than a simple primal heuristic based on ordering items in a static manner by their value divided by the sum of relative sizes. Note that the static application of a priority rule takes on average 0.2ms, which is extremely fast compared to dynamic priority rules that require almost 100 times more time on average.

The primal heuristic using the value per sum of relative sizes priority (value/sum.rel.size), dynamically recalculated, achieves overwhelmingly better solutions compared to all the above heuristics, even the ones with partial enumeration. The partial enumeration is effective, but cannot reach the quality of solutions achieved by a significantly better initial order of items.

Dynamic recalculation of priorities increases the processing time of the heuristic using value/sum.rel.size priority to 14.6ms compared to 0.2ms for the static approach. To save on computation, we apply the priority rule partially statically and partially dynamically, with the switch from static to dynamic occurring when the balance ratio, calculated as $(\min_i s_i/b_i)/(\max_i s_i/b_i)$, drops below 0.8. This approach decreases average processing time to 4.4ms, with the average quality of solutions remaining almost the same as in the fully dynamic case.

The static and partially dynamic heuristics using the value/sum.rel.size pri-

ority have also been tested with different versions of the postprocessing procedures described in section 3.3: improving last item by highest price (IL1), improving up to 5 last items by highest price (IL5), and partial enumeration with the number of item additions limited to 1000 (E1000) and 10000 (E10000). The parameters of postprocessing were chosen after preliminary experimentation to balance quality of solutions with processing times. It is noteworthy that the very simple IL1 postprocessing procedure significantly improves the quality of solutions with an infinitesimally small impact on processing time. IL5 improves the results even more, with the processing time still almost unaffected. Partial enumeration improves results considerably, but at a cost of increased processing time. Note that unlike the partial enumeration of Volgenant and Zweirs [24], the time taken by our partial enumeration is very well controlled by limiting the number of additions.

To summarise, the static priority rule of value/sum.rel.size followed by IL5 achieves very good results, with an average deviation of 3.14%, in only 0.2ms on average. The same heuristic applied partially dynamically and also followed by IL5 achieves 1.36% in 4.5ms on average. Following this heuristic by partial enumeration limited to 1000 additions decreased the average deviation to 1.08% while doubling the processing time. Increasing the limit on partial enumeration to 10000 additions achieved an average deviation of 0.99% in an average time of 25.6ms per instance.

4.3 Heuristics based on LP relaxation

Table 2 presents the results of three heuristics: the adaptive fixing of Bertsimas and Demir [2], our primal LP-selection, and our LP-adaptive-selection. All three have been tested on all 270 problem instances, without postprocessing and with four versions of postprocessing.

The choice of the value of γ for adaptive fixing is almost insignificant. It affects both the quality of solutions and the processing time only slightly. The best results are achieved when $\gamma = 0$ is chosen. Hence, this version of the heuristic is also tested with the postprocessing procedures and compared to other heuristics.

The processing times of all three heuristics are very similar, because most

Table 2: LP-relaxation-based heuristics

heuristic		postproc.	dev. (%)		time (s.)	
			avg.	max.	avg.	max.
adaptive fixing [2]	$\gamma = 0.25$	–	1.17	7.39	.0101	.16
	$\gamma = 0.1$	–	1.16	7.39	.0105	.16
	$\gamma = 0.05$	–	1.16	7.39	.0102	.16
	$\gamma = 0$	–	1.16	7.39	.0102	.16
		IL1	1.10	7.16	.0102	.16
		IL5	1.03	6.60	.0104	.16
		E1000	.80	5.04	.0135	.17
		E10000	.74	4.89	.0333	.17
LP-selection		–	1.27	7.75	.0078	.16
		IL1	1.13	7.23	.0078	.16
		IL5	1.05	6.84	.0084	.16
		E1000	.81	4.95	.0120	.17
		E10000	.74	4.67	.0314	.25
LP-adaptive-selection		–	1.19	7.15	.0111	.16
		IL1	1.06	6.73	.0111	.16
		IL5	1.00	5.69	.0111	.17
		E1000	.79	4.96	.0153	.17
		E10000	.73	4.45	.0347	.22

of the processing time is spent on the initial solution of the LP relaxation. The primal LP-selection heuristic is the fastest, since it solves the LP relaxation only once. The other two heuristics solve additionally several postoptimisations of the LP relaxation, with some variables fixed. In adaptive fixing, more variables are fixed in every postoptimisation, and, thus, the postoptimisations are slightly faster than in the LP-adaptive-selection heuristic. When no postprocessing is applied, adaptive fixing achieves better results than our LP-adaptive selection, but this is due to the fact that we complete the adaptive fixing solution by considering items for addition in the order of non-increasing prices. With any postprocessing procedure our LP-adaptive-selection heuristic finds on average better results than the adaptive fixing of Bertsimas and Demir.

It is interesting to note that all the LP-based heuristics presented here are on average faster and achieve on average better solutions than all the dynamically applied heuristics not using LP relaxation presented in section 4.2. Repeated computation of priorities is overall slower than solving the LP relaxation and postoptimising it several times. Thus, if a modern LP solver, like CPLEX, is available, LP-based heuristics would be the methods of choice.

4.4 Heuristics using constrained LP relaxation

Table 3 shows the results of our two LP-based heuristics and the adaptive fixing heuristic [2] applied with additional constraints. For each algorithm, the original LP-based heuristic is applied first without additional constraints, and then applied several times with an additional constraint, varying the right-hand-side (RHS) values. CC10 uses a cardinality constraint with 10 RHS values ranging from $k_{\text{base}} - 4$ to $k_{\text{base}} + 5$, RC10 uses a random constraint with the same 10 RHS values as CC10, while RC20 uses a random constraint with 20 RHS values ranging from $k_{\text{base}} - 9$ to $k_{\text{base}} + 10$. The postprocessing procedures are run for every solution computed and the best solution overall is reported.

Running an LP-based heuristic repeatedly with additional constraints improves solutions significantly, as can be seen by comparing the results in Table 2 with those in Table 3. Adaptive fixing and LP-adaptive-selection give comparable results that are significantly better than those achieved by LP-selection, albeit at the cost of longer computation time.

It is worth noting that in all heuristics the first LP-relaxation is solved fully by simplex method, whereas each of the following LP-relaxations is solved by dual simplex post-optimisation; a fact that explains why processing time does not increase α-fold for α repetitions. Thus, running the LP-selection heuristic with an additional constraint 10 or 20 times roughly doubles the processing time. However, the increase for the other two heuristics is significantly higher, which is caused by repeated fixing and unfixing of variables. As a result, the CC10 (RC10, RC20) LP-selection with E1000 postprocessing is faster and gives better results than CC10 (RC10, RC20) LP-adaptive-selection with IL1 or IL5.

When the cardinality constraint in CC10 is replaced by a random constraint in RC10 (as described in section 3.5) the quality of solutions is improved, while the processing times remain roughly the same. Further improvement at somewhat increased computation time is achieved when 20 distinct values of RHS are used instead of 10 (compare RC10 and RC20 heuristics). Overall, the RC20 LP-selection heuristic achieves slightly worse results than the RC20 LP-adaptive-selection or RC20 adaptive fixing, but has significantly shorter processing times.

Table 3: Heuristics based on constrained LP relaxation

heuristic	postproc.	dev. (%)		time (s.)	
		avg.	max.	avg.	max.
CC10 adaptive fixing $\gamma = 0$	IL1	.88	5.22	.0789	.34
	IL5	.85	5.17	.0791	.34
	E1000	.72	4.30	.1064	.41
	E10000	.67	4.01	.2586	1.16
CC10 LP-selection	IL1	.90	4.91	.0175	.17
	IL5	.86	4.91	.0197	.19
	E1000	.72	4.67	.0481	.19
	E10000	.67	4.20	.1972	.81
CC10 LP-adaptive-selection	IL1	.89	4.56	.0833	.45
	IL5	.84	4.56	.0809	.44
	E1000	.70	4.30	.1108	.50
	E10000	.66	3.92	.2627	1.19
RC10 adaptive fixing $\gamma = 0$	IL1	.80	4.87	.0940	.42
	IL5	.78	4.87	.0944	.36
	E1000	.66	3.73	.1370	.50
	E10000	.64	3.73	.3488	1.25
RC10 LP-selection	IL1	.85	5.72	.0128	.19
	IL5	.82	5.72	.0139	.19
	E1000	.69	4.02	.0545	.20
	E10000	.65	4.02	.2598	1.02
RC10 LP-adaptive-selection	IL1	.77	4.38	.1045	.55
	IL5	.76	4.38	.1043	.52
	E1000	.67	4.11	.1459	.61
	E10000	.64	3.92	.3605	1.38
RC20 adaptive fixing $\gamma = 0$	IL1	.78	4.67	.1785	.72
	IL5	.76	4.67	.1804	.75
	E1000	.65	3.73	.2618	.94
	E10000	.63	3.54	.6621	2.21
RC20 LP-selection	IL1	.82	5.49	.0177	.19
	IL5	.79	5.11	.0202	.19
	E1000	.67	4.02	.0959	.28
	E10000	.64	4.02	.4972	2.00
RC20 LP-adaptive-selection	IL1	.75	4.38	.1964	.84
	IL5	.74	4.38	.2005	.92
	E1000	.66	4.11	.2785	1.09
	E10000	.63	3.84	.6838	2.52

Table 4: Average percentage deviations with various time limits

instances	heuristic	time limit (s.)		
		2	10	60
all	Multi-run RC20 LP-selection + IL5	.576	.546	.536
	MIP in CPLEX 10.0.2	.611	.568	.545
$n = 100$	Multi-run RC20 LP-selection + IL5	1.115	1.083	1.075
	MIP in CPLEX 10.0.2	1.149	1.097	1.078
$n = 250$	Multi-run RC20 LP-selection + IL5	.412	.371	.359
	MIP in CPLEX 10.0.2	.451	.399	.372
$n = 500$	Multi-run RC20 LP-selection + IL5	.203	.185	.173
	MIP in CPLEX 10.0.2	.234	.206	.186

4.5 Anytime performance

As mentioned earlier (section 3.5), the heuristics employing random constraints can be executed as anytime heuristics, i.e., they may be run repeatedly, with a different random constraint each time. Even though the RC20 LP-adaptive-selection gives the best results in terms of average deviation, it takes significantly longer than the RC20 LP-selection. Hence, when both are used in multiple runs within a specified time limit, the latter produces on average better results. For the same reason the IL5 postprocessing has been chosen in preference to enumeration. In consequence, it is the RC20 LP-selection + IL5 that is adopted as the base of our multiple-run anytime heuristic.

Table 4 shows average deviations achieved by multiple runs of RC20 LP-selection, each followed by IL5. The heuristic is compared to the MIP formulation solved in CPLEX. Both algorithms were given 2, 10, and 60 seconds per instance. The results are presented for all instances and additionally partitioned into three subsets depending on the number of items.

The question of which parameters of CPLEX to use arises naturally. Short of carrying a full-factorial experiment, which is hardly feasible, the best that can be done is to test individual options. We have done this and found that the default CPLEX settings give the best results. In particular, changing 'emphasis' to 'feasibility' or 'hidden feasible solutions' had a marginal negative effect on the average deviation.

Our heuristic finds on average better solutions than CPLEX given the same time limit. In fact, our heuristic run for 10 seconds finds solutions that are on average of the same quality as found by CPLEX run for 60 seconds. CPLEX

Table 5: Comparison with best-known heuristics on $n = 500$ instances

algorithm	avg.dev.(%)	avg.time	processor
Multi-run RC20 LP-selection + IL5 (60s.)	.173	1m.	Pentium M 2GHz
GA of Chu & Beasley [3]	.178	34.67m.	SGI R4000 100MHz
Fix+Cuts of Osorio et al. [15]	.169	3h.	Pentium III 450MHz
LP+TS of Vasquez & Hao [21]	.160	8.67h.	Pentium IV 2GHz
Fix+LP+TS of Vasquez & Vimont [22]	.142	16.37h.	Pentium IV 2GHz

competes well with our heuristic only on small instances ($n = 100$), particularly within a 60-second time limit, because it manages to solve many of them to optimality. For larger instances, for which CPLEX terminates while far from proving optimality, our heuristic has significantly better performance.

For the set of the largest problem instances ($n = 500$), Table 5 compares our multi-run algorithm executed for 60 seconds with the best performing heuristics available in the literature, which are the ones that achieve an average deviation of less than 0.2%. Compared to our algorithm, the GA of Chu & Beasley [3] gives a slightly inferior average deviation. However, it is not possible to compare processing times. The other three algorithms achieve better results, but need significantly longer processing times, even taking into account the relative speed of the various processors. The best solutions are achieved by the Fix+LP+TS algorithm of Vasquez and Vimont [22]. However, their computation takes inordinately long times, with some instances solved for more than 3 days.

5 Conclusions

The multi-dimensional knapsack problem (MKP) is an eminently difficult combinatorial optimisation problem. Yet, the present work has succeeded in forging fast and effective simple heuristics based on priority rules. However, the success of this approach seems to be contingent upon the judicious choice of the priority rules. If the priority rule chosen is appropriate, then, as we have shown, it would outperform less effective rules, even if the former is applied in a static manner and the latter are applied dynamically.

An interesting aspect of the current work is that it shows how the implicit information provided by the LP relaxation can be used to fashion effective priority rules. This approach may very well prove to be useful in developing fast,

effective heuristics for other combinatorial optimisation problems.

We have shown that appropriately chosen deterministic constraints imposed on the LP relaxation can be used to partition the solution space effectively, leading to good solutions. Likewise, we have shown that randomly generated constraints, again appropriately designed and imposed on the LP relaxation, are exceedingly effective in randomising the search of the solution space. It would seem at this stage that this approach may indeed be generalisable in order to form the basis of a new metaheuristic.

An appropriate conclusion of the current work is that the power of modern MIP solvers, like CPLEX, is such that it is difficult to outperform them when they are used in an anytime heuristic mode, even though we have succeeded in doing so by a respectable margin for the MKP.

Acknowledgements

We are immensely grateful for the comments and queries of the referees that helped us improve content and presentation.

References

[1] Y. Akcay, H. Li, and S.H. Xu. Greedy algorithm for the general multidimensional knapsack problem. *Annals of Operations Research*, 150(1):17–29, 2007.

[2] D. Bertsimas and R. Demir. An approximate dynamic programming approach to multidimensional knapsack problems. *Management Science*, 48(4):550–565, 2002.

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

[4] S. DeVries and R. Vohra. Combinatorial auctions: A survey. Technical report, Northwestern University, Evanston, IL., 2000.

[5] C. Ferreira, M. Grotschel, S. Kiefl, C. Krispenz, A. Martin, and R. Weismantel. Some integer programs arising in the design of mainframe computers. *ZOR–Methods Models Oper. Res.*, 38(1):77–110, 1993.

[6] A. Freville. The multidimensional 0-1 knapsack problem: an overview. *European Journal of Operational Research*, 155:1–21, 2004.

[7] A. Freville and S. Hanafi. The multidimensional 0-1 knapsack problem-bounds and computational aspects. *Annals of Operations Research*, 139(1):195–227, 2005.

[8] A. Freville and G. Plateau. An efficient preprocessing procedure for the multidimensional 0-1 knapsack problem. *Discrete Applied Mathematics*, 49:189–212, 1994.

[9] B. Gavish and H. Pirkul. Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality. *Mathematical Programming*, 31(1):78–105, 1985.

[10] P.C. Gilmore and R.E. Gomory. The theory and computation of knapsack functions. *Operations Research*, 14:1045–1075, 1966.

[11] S. Hanafi and F. Glover. Exploiting nested inequalities and surrogate constraints. *European Journal of Operational Research*, 179(1):50–63, 2007.

[12] G.A. Kochenberger, B.A. McCarl, and F.P. Wyman. A heuristic for general integer programming. *Decision Sciences*, 5(1):36–44, 1974.

[13] R. Loulou and E. Michaelides. New greedy-like heuristics for the multidimensional 0–1 knapsack problem. *Operations Research*, 27(6):1101–1114, 1979.

[14] M.J. Magazine and O. Oguz. Heuristic algorithm for the multidimensional zero-one knapsack problem. *European Journal of Operational Research*, 16(3):319–326, 1984.

[15] M. A. Osorio, F. Glover, and P. Hammer. Cutting and surrogate constraint analysis for improved multidimensional knapsack solutions. *Annals of Operations Research*, 117(1-4):71–93, 2002.

[16] C.C. Peterson. Computational experience with variants of the balas algorithm applied to the selection of research and development projects. *Management Science*, 13:736–750, 1967.

[17] H. Pirkul. Heuristic solution procedure for the multiconstraint zero-one knapsack problem. *Naval Research Logistics*, 34(2):161–172, 1987.

[18] M.H. Rothkopf, A. Pekec, and R.M. Harstad. Computationally manageable combinatorial auctions. Technical report 95–09, Rutgers University, Piscataway, NJ, 1995.

[19] S. Senju and Y. Toyoda. An approach to linear programming with 0–1 linear variables. *Management Science*, 15:196–207, 1968.

[20] Y. Toyoda. A simplified algorithm for obtaining approximate solutions to zero-one programming problems. *Management Science*, 21:1417–1427, 1975.

[21] M. Vasquez and J.-K. Hao. A hybrid approach for the 0-1 multidimensional knapsack problem. In *IJCAI-01 Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pages 328–333, Seatle,Washington, 2001.

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

[23] A. Volgenant and J.A. Zoon. An improved heuristic for multidimensional 0–1 knapsack problem. *Journal of Operational Research Society*, 41(10):963–970, 1990.

[24] A. Volgenant and I.Y. Zwiers. Partial enumeration in heuristics for some combinatorial optimization problems. *Journal of the Operational Research Society*, 58(1):73–79, 2007.