A procedure-based heuristic for 0-1 Multiple Knapsack Problems

Mohamed Esseghir Lalami, Moussa Elkihel, Didier El Baz* and Vincent Boyer

CNRS, LAAS, 7, avenue du Colonel Roche,
Université de Toulouse; UPS, INSA, INP, ISAE, LAAS,
F-31077 Toulouse, France
E-mail: mlalami@laas.fr
E-mail: elkihel@laas.fr
E-mail: elbaz@laas.fr
E-mail: vboyer@laas.fr
*Corresponding author

Abstract: In this paper, we present a heuristic which derives a feasible solution for the Multiple Knapsack Problem (MKP). The proposed heuristic called RCH, is a recursive method that performs computation on the core of knapsacks. The RCH heuristic is compared with the MTHM heuristic of Martello and Toth. Computational results on randomly generated instances show that the proposed approach gives better gap and smaller restitution times.

Keywords: MKP; multiple knapsack problems; heuristic; dynamic programming; subset sum problem; operational research.

Reference to this paper should be made as follows: Lalami, M.E., Elkihel, M., El Baz, D. and Boyer, V. (2012) 'A procedure-based heuristic for 0-1 Multiple Knapsack Problems', *Int. J. Mathematics in Operational Research*, Vol. 4, No. 3, pp.214–224.

Biographical notes: Mohamed Esseghir Lalami received his Engineer Degree in Control Theory, from the University of Science and Technology of Algiers in 2007 and his Master Degree in Automatic Systems from University Paul Sabatier of Toulouse, France in 2008. He is currently PhD Student in the team Distributed Computing and Asynchronism of LAAS-CNRS, Toulouse. His research interests are in combinatorial optimisation, GPU computing and parallel computing.

Moussa Elkihel received the PhD in Applied Mathematics from University of Sciences and Technologies of Lille, France in 1984. He is Associate Professor at IUT Toulouse. His research interest includes knapsack type problems and parallel computing. He was cofounder of the working group Knapsack and Optimisation of the French Operations Research Group (GDR RO).

Didier El Baz received the Doctor Engineer Degree in Control Theory from INSA Toulouse in January 1984. He was Visiting Scientist in the Laboratory for Information and Decision Systems, MIT Cambridge

Massachusetts, USA, in 1984. He received the HDR in Computer Sciences from INPT in 1998. He is founder and head of the team Distributed Computing and Asynchronism at LAAS-CNRS. His fields of interest are in parallel and distributed computing, peer to peer computing, grid computing, GPU computing, optimisation, control theory, robotics and numerical analysis.

Vincent Boyer received the Engineer Degree in Automatic Systems and Industrial Data Processing from ENSEEIHT Toulouse in 2004. He received the PhD in Automatic Systems from INSA Toulouse in 2007. He has had several teaching positions in Toulouse and has made his research work in the team Distributed Computing and Asynchronism of LAAS-CNRS. Today, he has a postdoctoral position in the Centre Interuniversitaire de Recherche sur les Réseaux d'Entreprise, la Logistique et le Transport (CIRRELT), in Montréal, Canada. His research interests include combinatorial optimisation and parallel computing.

1 Introduction

The Multiple Knapsack Problem (MKP) is a generalisation of the standard 0-1 Knapsack Problem where instead of considering only one knapsack, one tries to fill m knapsacks of different capacities. Let $N = \{1,, \dots, n\}$ be the set of items where each item j has a corresponding profit p_j and weight w_j. We consider m knapsacks of capacity c_i, $i \in \{1, \dots, m\}$, then the MKP consists in filling all knapsacks so that the total profit is maximised and the sum of weights in each knapsack i does not exceed the capacity c_i.

We denote the binary decision variables by x_{ij} which take value: 1 if item j is assigned to knapsack i and 0 otherwise. The MKP is formulated as the following 0-1 integer programming problem:

$$\text{maximise} : \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij}, \tag{1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} w_j x_{ij} \leq c_i, \; i \in \{1, \dots, m\}, \tag{2}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1, \quad j \in \{1, \dots, n\}, \tag{3}$$

$x_{ij} \in \{0, 1\}$, $i \in \{1, \dots, m\}$, $j \in \{1, \dots, n\}$; where p_j, c_i and w_j are positive integers and constraints (2) and (3), respectively, ensure that the filling of knapsack i does not exceed its corresponding capacity c_i and every selected item is assigned only to one knapsack, respectively.

In order to avoid any trivial case, we make the following assumptions.

- All items have a chance to be packed (at least in the largest knapsack):

$$\max_{j\in N} w_j \leq \max_{i\in\{1,\dots,m\}} c_i. \tag{4}$$

- The smallest knapsack can be filled at least by the smallest item:

$$\min_{i\in\{1,\dots,m\}} c_i \geq \min_{j\in N} w_j. \tag{5}$$

- There is no knapsack which can be filled with all items of N:

$$\sum_{j=1}^{n} w_j \geq c_i, \forall i \in \{1,\dots,m\}. \tag{6}$$

We assume also that the items and the knapsacks are sorted as follows:

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \cdots \geq \frac{p_n}{w_n}, \tag{7}$$

$$c_1 \leq c_2 \leq \cdots \leq c_m. \tag{8}$$

Cargo loading is a real world application of the MKP, see Eilon and Christofides (1971). The problem is to choose some containers in a set of n containers to be loaded in m vessels with different loading capacities for the shipment of the containers. Other industrial applications are the loading of n tanks with m liquids that cannot be mixed, see Martello and Toth (1980), vehicle loading, see Hifi (2009), task assignment and multiprocessor scheduling, see Labbé et al. (2003).

The MKP problem is strongly NP-complete and the need for algorithms that give a good heuristic solution is justified by the computational complexity of this problem. Reference is made to Hung and Fisk (1978), Martello and Toth (1990) and Kellerer et al. (2004) for contributions to this problem.

In this paper, we present a heuristic which yields a feasible solution within a reasonable computing time, this is done by exploiting efficiently the core of each knapsack. To the best of our knowledge, this paper is the first paper to deal with a heuristic since Martello and Toth (1980).

Section 2 deals with principle of the Martello and Toth heuristic. The proposed method is presented in Section 3. The Section 4 is devoted to the presentation and analysis of computational results on randomly generated instances. Finally, in Section 5, we give some conclusions and perspectives of our work.

2 Martello and Toth heuristic (MTHM)

In Martello and Toth (1980), the so-called MTHM heuristic is given for the MKP. This algorithm consists of three phases. The details are available on pages 179–181 in (Martello and Toth, 1990, pp. 179–181).

The first phase of MTHM obtains an initial feasible solution by applying the Greedy algorithm (Procedure 1) to the first knapsack; a set of remaining items is obtained, then the same procedure is applied for the second knapsack; this is continued till the mth knapsack.

The second phase tries to improve the initial solution by swapping every pairs of items assigned to different knapsacks and trying to insert a new item so that the total profit is increased.

The last phase tries to exclude in turn each selected item if it is possible to replace it by one or more remaining items so that the total profit sum is increased.

The advantage of the MTHM heuristic is that some items can be exchanged from a knapsack to another or excluded from the solution set so that total profit increases. This can lead to an efficient and fast solution when the solution given by the first phase is good. The main drawback of MTHM heuristic is that it considers only the exchanges between a pairs of items instead of combinations of items.

3 RCH, a Recursive Core Heuristic for the MKP

The aim of the proposed approach consists in packing the maximum number of 'best' items with regards to relation (4) in a small computation time. For that purpose, an efficient lower bound $\underline{z}$ of the MKP is built as follows. We consider the series of knapsack problems:

$$(\mathrm{KP}_i)\left\{\max \sum_{j\in N_i} p_j x_j \mid \sum_{j\in N_i} w_j x_j \le c_i,\ x_j \in \{0,1\},\ j\in N_i\right\},\quad i=1,\dots,m, \tag{9}$$

with $N_1 = N$, $N_i = N_{i-1} - \{l \in N_{i-1} \mid x_l = 1\}$ and $\underline{z}_i$ the lower bound of (KP_i) such that $\underline{z}_i = \sum_{j\in N_i} p_j x_j$. Then, a lower bound of MKP is given as follows.

$$\underline{z} = \sum_{i=1}^{m} \underline{z}_i.$$

In order to simplify notation and without loss of generality, we consider that the elements of N_i are indexed from 1 to $Card(N_i) = n_i$, i.e., $N_i = \{1,\dots,n_i\}$.

3.1 Deriving lower bounds with the first knapsacks

The $m-1$ first knapsack problems (KP_i) are considered successively. At step i, a Greedy algorithm is applied to fill the knapsack (KP_i) and derives a lower bound $\underline{z}_i = \sum_{j\in N_i} p_j x_j$. If the knapsack is entirely filled i.e., the residual capacity $\bar{c}_i = c_i - \sum_{j\in N_i} w_j x_j = 0$, then we consider the next knapsack; otherwise, we solve a subset sum problem on the core of the considered knapsack that allows us to derive a lower bound $\underline{z}_i = \sum_{j\in N_i} p_j x_j$.

Finally, a dynamic programming method is used to derive a lower bound of the last knapsack (KP_m).

3.1.1 *Computing a lower bound of a knapsack with a Greedy algorithm*

Assuming that the items and knapsacks, respectively, are sorted according to equations (7) and (8), respectively, the following Greedy algorithm is applied to knapsack (KP$_i$), $i \neq m$.

Procedure 1. Greedy.

```
    Input: $N_i, (p_j), (w_j), (x_j), c_i$;
    Output: $\underline{z}_i, \bar{c}_i$;
    $\underline{z}_i := 0$;
    $\bar{c}_i := c_i$;
    for $j := 1$ to $j := n_i$ do
        if $w_j \leq \bar{c}_i$ then
            $x_j := 1$;
            $\bar{c}_i := \bar{c}_i - w_j$;
            $\underline{z}_i := \underline{z}_i + p_j$;
        end if
    end for
    end;
```

3.1.2 *Reducing residual capacity via computation on the core*

- *The core of the knapsack problem (*KP$_i$*)*

We recall that items are sorted according to inequality (7). We denote by s the index of the split item of (KP$_i$) which satisfies:

$$\sum_{j=1}^{s-1} w_j \leq c_i < \sum_{j=1}^{s} w_j. \tag{10}$$

The core of the problem (KP$_i$) denoted by C is defined as follows:

$$C = \{j \in N_i \,|\, s - r \leq j \leq s + r - 1\}, \tag{11}$$

where $2r$ is a constant which denotes the size of the core. Several choices have been proposed for r. Balas and Zemel (1980) have proposed $2r = 25$ for large instances. Martello and Toth have proposed $r = \sqrt{n}$ in Martello and Toth (1988) and $r = \sqrt{n}/2$ in Martello and Toth (1990), see also Pisinger (1995).

For the proposed heuristic, best results were obtained with $r = \sqrt{n}$.

- *Solving the Subset Sum Problem (*SSP*)*

The residual capacity of the knapsack problem (KP$_i$) can be reduced by computing the Subset Sum Problem (SSP) on the core.

In the sequel, we use the following notation $\bar{c}_i = c_i - \sum_{j=1}^{s-r-1} w_j$. We solve:

$$\text{(SSP)} \left\{ \max \sum_{j \in C} w_j x_j \,|\, \sum_{j \in C} w_j x_j \leq \bar{c}_i,\ x_j \in \{0, 1\} \right\}, \tag{12}$$

we denote by v(SSP) the optimal value of problem (SSP). Problem (SSP) is solved via the dynamic programming method proposed by Plateau and Elkihel (1985).

In this method, items are taken alternatively according to each direction starting from index s. That is, we consider $s,\ s-1, s+1, s-2, \ldots$ (see also Pisinger, 1995, Martello and Toth, 1997). More precisely, the method consists in building iteratively the lists L_1^k and L_2^q as follows:

$$L_1^k = \left\{ w \mid w = \sum_{j=k}^{s-1} w_j x_j \le \bar{c}_i,\ x_j \in \{0,1\} \right\}, \qquad k = s-1, s-2, \ldots, s-r, \tag{13}$$

$$L_2^q = \left\{ w' \mid w' = \sum_{j=s}^{q} w_j x_j \le \bar{c}_i,\ x_j \in \{0,1\} \right\}, \qquad q = s, s+1, \ldots, s+r-1. \tag{14}$$

The states are sorted in the lists according to increasing weights w. Dominance techniques are applied: when 2 states are equal, then one of them is removed. Lists L_1^k and L_2^q are merged two by two after each step. The building lists process is stopped when $max\ \ w + w' = \bar{c}_i$ or $k = s - r$ and $q = s + r - 1$. Thus, we take

$$\underline{z}_i = \sum_{j=1}^{s-r-1} p_j + \sum_{j \in C} p_j x_j^*, \tag{15}$$

where the x_j^* correspond to the solution of SSP i.e.,. $v(\text{SSP}) = \sum_{j \in C} w_j x_j^*$. Thus, we can write this algorithm as follows:

```
**Procedure 2.** Solve SSP.
**Input:** $C, (w_j), s, c_i, r$;
**Output:** $N_{i+1}, \underline{z}_i$;
**for** $l := 1$ **to** $l := r$ **do**
    $k := s - l$;
    $q := s + l - 1$;
    Build $L_1^k$;
    Build $L_2^q$;
    Merge ($L_1^k$, $L_2^q$);
    **if** max $w + w' = \bar{c}_i$ **then** STOP
**end for**
Update: $\underline{z}_i$;
Update: $N_{i+1}$;
**end**
```

3.2 Deriving a lower bound with the last knapsack

Finally, dynamic programming is applied to the core of the last knapsack (KPC). We solve the problem

$$(\text{KPC}) \left\{ \max \sum_{j \in C} p_j x_j \mid \sum_{j \in C} w_j x_j \le \bar{c}_m,\ x_j \in \{0,1\} \right\}, \tag{16}$$

we denote by $v(\text{KPC})$ the optimal value of problem (KPC). Problem (KPC) is solved via the dynamic programming method with dominance techniques (see Ahrens and Finke, 1975; El Baz and Elkihel, 2005; Boyer et al., 2009). In this method, items are considered successively from $s-r$ to $s+r-1$. More precisely, the dynamic programming method consists in building iteratively the lists L_k as follows:

$$L^k = \left\{ (w,p) \,|\, w = \sum_{j=s-r}^{k} w_j x_j \leq \bar{c}_m, x_j \in \{0,1\} \right\}, \\ k = s-r, \ldots, s+r-1, \tag{17}$$

where $p = \sum_{j=s-r}^{k} p_j x_j$.

The states are sorted in the lists L_k according to increasing profits p. Dominance techniques are applied: if (w,p) and (w',p') are two states such that $w \leq w'$ and $p \geq p'$, then the state (w',p') is dominated and can be removed. We take

$$\underline{z}_m = \sum_{j=1}^{s-r-1} p_j + v(\text{KPC}), \tag{18}$$

with $v(\text{KPC}) = \sum_{j \in C} p_j x_j^* = max\{p \,|\, (w,p) \in L_{s+r-1}\}$. The computational procedure is detailed in the following algorithm:

```
**Procedure 3.** Solve KP.
**Input:** $C, (w_j), (p_j), s, c_m, r$;
**Output:** $\underline{z}_m$;
**for** $k := s-r$ **to** $l := s+r-1$ **do**
    Build $L^k$;
**end for**
Update: $\underline{z}_m$;
**end**
```

4 Computational experiences

We present now computational results for RCH and compare the obtained results with the MTHM heuristic and CPLEX Solver 12.1. The algorithms RCH and MTHM have been written in C. All tests have been carried out on an Intel core 2 Duo 2.2 Ghz with 2 GB of RAM. We have considered random generated problems. We display average results obtained with 10 instances. We have fixed a time limit of 600 s (10 min) for each instance.

We have considered three types of problems: *uncorrelated*, *weakly correlated* and *strongly correlated* problems; this has permitted us to consider from relatively easy to very difficult problems. The weights w_j are uniformly random distributed over the integer interval $[1, 1000]$ and profits p_j are computed as follows:

- *Uncorrelated* case:
  Uniformly random over $[1, 1000]$, independent of w_j.
- *Weakly correlated* case: Uniformly random over $[w_j - 100, w_j + 100]$.
- *Strongly correlated* case: $p_j = w_j + 100$.

The capacities c_i are uniformly random in $\left[0.4\sum_{j=1}^{n} w_j/m, 0.6\sum_{j=1}^{n} w_j/m\right]$ for $i = 1, \ldots, m-1$.

The capacity of the mth knapsack is set to

$$c_m = 0.5\sum_{j=1}^{n} w_j - \sum_{i=1}^{m-1} c_i,$$

if an instance does not satisfy conditions (4)–(6), then a new instance is generated. The MKP problems are available at:http://www.laas.fr/laas09/CDA-EN/45-31328-MKP.php. In the sequel, the gap of a given method is computed as follows:

$$\text{gap} = 100(U - \underline{z})/U,$$

where U is the best upper bound given by CPLEX and the surrogate relaxation of the problem MKP, see Martello and Toth (1980).

We have considered two orderings of items in the core i.e., the natural ordering whereby items are sorted according to decreasing ratio price over weight, see equation (7) and a different ordering whereby items are sorted according to the decreasing weight order i.e., $w_{s-r} \geq w_{s-r+1} \geq \cdots \geq w_{s-1}$, $w_s \geq w_{s+1} \geq \cdots \geq w_{s+r}$. We note that both orderings give good results in terms of gap. Nevertheless, the latter is slightly better in terms of time; this is the ordering we have used in the computational tests we present.

Tables 1–3 display computational results for the different types of problems i.e., uncorrelated, weakly correlated and strongly correlated problems.
According to the results presented in Tables 1–3, RCH is in general better than CPLEX and MTHM in terms of gap and processing time.

We remark that the processing time of RCH is not very sensitive to the number of knapsacks and the size of problems. We recall that the size of the solved cores is equal to $2\sqrt{n}$. We can remark also that greater is n, better is the gap.

We note that for 10 and 100 knapsacks, CPLEX exceeds the time limit of 10 min and the difference of gap between CPLEX and RCH can reach 3 orders of magnitude, e.g., see Table 1 ($m = 100, n = 5000$). We note also that for problems with 100 knapsacks and more than 50000 items, CPLEX exceeds the memory capacity of the machine.

The computing time of RCH method never exceeds 3 s while the computing time of MTHM method can be 80 s. The good results we have obtained with RCH as compared with MTHM can be explained as follows:

- The exchange of a combination of items between two adjacent knapsacks in a recursive way as proposed in RCH is more efficient than swapping pairs of items between different knapsacks like in MTHM.

Table 1 Time and gap for uncorrelated problems

		CPLEX		*MTHM*		*RCH*		
m	n	*Time* (s)	*GAP*(%)	*Time* (s)	*GAP*(%)	*Time* (s)	*GAP*(%)	U
2	5000	4.5200	0.00000	0.1438	0.00063	0.0064	0.00005	○
	10000	7.7600	0.00000	0.5751	0.00023	0.0079	0.00002	○
	50000	29.010	0.00000	14.546	0.00002	0.0157	0.00000	○
	100000	138.32	0.00000	58.530	0.00001	0.0220	0.00000	○
10	5000	600.00	0.01076	0.1766	0.00204	0.0110	0.00020	○
	10000	600.00	0.00864	0.6842	0.00065	0.0125	0.00011	○
	50000	600.00	0.00454	17.116	0.00006	0.0265	0.00001	○
	100000	600.00	0.00850	69.091	0.00003	0.0359	0.00000	○
100	5000	600.00	0.12789	0.1860	0.02493	0.0657	0.00193	○
	10000	600.00	0.69317	0.7341	0.00722	0.0828	0.00058	○
	50000	–	–	18.016	0.00051	0.0359	0.00008	•
	100000	–	–	71.758	0.00016	0.2704	0.00003	•

U, the best upper bound given by:
○ CPLEX; • the surrogate relaxation of MKP.
–: CPLEX exceeds memory capacity of the machine.

Table 2 Time and gap for weakly correlated problems

		CPLEX		*MTHM*		*RCH*		
m	n	*Time* (s)	*GAP*(%)	*Time* (s)	*GAP*(%)	*Time* (s)	*GAP*(%)	U
2	5000	5.1100	0.00000	0.1139	0.00067	0.0093	0.00000	○
	10000	8.9600	0.00000	0.4530	0.00025	0.0108	0.00000	○
	50000	31.450	0.00000	11.359	0.00002	0.0157	0.00000	○
	100000	143.27	0.00000	42.371	0.00000	0.0154	0.00000	○
10	5000	600.00	0.01249	0.1375	0.00541	0.0157	0.00072	○
	10000	600.00	0.00623	0.5422	0.00174	0.0078	0.00008	○
	50000	600.00	0.00149	13.568	0.00012	0.0375	0.00000	○
	100000	600.00	0.00544	54.250	0.00005	0.0420	0.00000	○
100	5000	600.00	0.15531	0.1453	0.09919	0.0766	0.00072	○
	10000	600.00	0.55200	0.5658	0.03841	0.0888	0.00016	○
	50000	–	–	13.879	0.00301	0.2092	0.00001	•
	100000	–	–	55.446	0.00088	0.3219	0.00000	•

U, the best upper bound given by:
○ CPLEX; • the surrogate relaxation of MKP.
–: CPLEX exceeds memory capacity of the machine.

Table 3 Time and gap for strongly correlated problems

		CPLEX		*MTHM*		*RCH*		
m	n	*Time* (s)	$GAP(\%)$	*Time* (s)	$GAP(\%)$	*Time* (s)	$GAP(\%)$	U
2	5000	5.8700	0.00000	0.1609	0.00000	0.1095	0.00000	○
	10000	9.3700	0.00000	0.6625	0.00000	0.2079	0.00000	○
	50000	38.780	0.00000	15.995	0.00000	1.0127	0.00000	○
	100000	146.54	0.00000	65.911	0.00000	2.0140	0.00000	○
10	5000	600.00	0.01924	0.1919	0.00173	0.1235	0.00087	○
	10000	600.00	0.01755	0.7626	0.00250	0.2267	0.00171	○
	50000	600.00	0.01000	19.185	0.00023	1.0533	0.00021	○
	100000	600.00	0.00556	76.602	0.00010	2.0406	0.00003	○
100	5000	600.00	0.46629	0.2030	0.18344	0.7330	0.04310	○
	10000	600.00	0.56466	0.8143	0.07015	1.3720	0.00457	○
	50000	–	–	20.109	0.00306	1.8797	0.00025	●
	100000	–	–	80.058	0.00096	2.8470	0.00013	●

U, the best upper bound given by:
○ CPLEX; ● the surrogate relaxation of MKP.
–: CPLEX exceeds memory capacity of the machine.

- The list building procedure of RCH combines items in the core with smallest weight, i.e., at left of index s, with items with largest weight, i.e., at right of s. This results in a better way to fill knapsacks.

5 Conclusion

In this paper, we have proposed a heuristic RCH for the MKP which consists in solving recursively each core of the different knapsacks. For the $m-1$ first cores, a subset sum problem is solved via dynamic programming and the last core is solved by using the classical dynamic programming. The ordering whereby items are sorted according to the decreasing weights in the two sets $\{s-r,\dots,s-1\}$ and $\{s,\dots,s+r-1\}$ of the $m-1$ first cores and the natural ordering in the last core are used. Computational results show that the RCH heuristic generally yields better gaps than the MTHM heuristic of Martello and Toth and CPLEX Solver in better computing time and that the proposed approach is able to solve large instances in a reasonable time. In future work, we plan to test other orderings in order to improve computing time.

References

Ahrens, J.H. and Finke, G. (1975) 'Merging and sorting applied to the zero-one knapsack problem', *Operations Research*, Vol. 23, pp.1099–1109.

Balas, E. and Zemel, E. (1980) 'An algorithm for large zero-one knapsack problems', *Operations Research*, Vol. 28, pp.1130–1154.

Boyer, V., Elkihel, M. and El Baz, D. (2009) 'Heuristics for the 0–1 multidimensional knapsack problem', *European Journal of Operational Research*, Vol. 199, pp.658–664.

Eilon, S. and Christofides, N. (1971) 'The loading problem', *Management Science*, Vol. 17, pp.259–268.

El Baz, D. and Elkihel, M. (2005) 'Load balancing methods and parallel dynamic programming algorithm using dominance technique applied to the 0-1 knapsack problem', *Journal of Parallel and Distributed Computing*, Vol. 65, pp.74–84.

Hung, M.S. and Fisk, J.C. (1978) 'An algorithm for the 0-1 multiple knapsack problems', *Naval Research Logistics Quarterly*, Vol 24, pp.571–579.

Hifi, M. (2009) 'Approximate algorithms for the container loading problem', *Operations Research*, Vol. 9, No. 6, pp.747–774.

Kellerer, H., Pisinger, D. and Pferschy, U. (2004) *Knapsack problems*, Springer, Berlin, Heidelberg New York.

Labbé, M., Laporte, G. and Martello, S. (2003) 'Upper bounds and algorithms for the maximum cardinality bin packing problem', *European Journal of Operational Research*, Vol. 149, pp.490–498.

Martello, S. and Toth, P. (1980) 'Solution of the zero-one multiple knapsack problem', *European Journal of Operational Research*, Vol. 4, pp.276–283.

Martello, S. and Toth, P. (1988) 'A new algorithm for the 0-1 knapsack problems', *Management Science*, Vol. 34, pp.633–644.

Martello, S. and Toth, P. (1990) *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley, Chichester.

Martello, S. and Toth, P. (1997) 'Upper bounds and algorithms for hard 0-1 knapsack problems', *Operations Research*, Vol. 45, pp.768–778.

Pisinger, D. (1995) 'An expanding-core algorithm for the exact 0-1 knapsack problems', *European Journal of Operational Research*, Vol. 87, pp.175–187.

Plateau, G. and Elkihel, M. (1985) 'A hybrid method for the 0-1 knapsack problem', *Methods of Operations Research*, Vol. 49, pp.277–293.

Website

Multiple Knapsack Problem Benchmarks, http://www.laas.fr/laas09/CDA-EN/45-31328-Multiple-knapsack-problem.php