Revisiting Surrogate Relaxation for the Multidimensional Knapsack Problem

Trivikram Dokka* Adam N. Letchford† M. Hasan Mansoor†

To appear in *Operations Research Letters*

Abstract

The *multidimensional knapsack problem* (MKP) is a classic problem in combinatorial optimisation. Several authors have proposed to use *surrogate relaxation* to compute upper bounds for the MKP. In their papers, the surrogate dual is solved heuristically. We show that, using a modern dual simplex solver as a subroutine, one can solve the surrogate dual exactly in reasonable computing times. On the other hand, the resulting upper bound tends to be strong only for relatively small MKP instances.

Keywords: knapsack problems; integer programming; surrogate relaxation

1 Introduction

The *multidimensional knapsack problem* (MKP) is a classic problem in combinatorial optimisation, with a wide range of practical applications. It is defined as follows. We have n items and m resources. The profit of item j is p_j. The amount of resource i available is b_i. Item j uses a_{ij} units of resource i. The goal is to select a set of items of maximum total profit, while respecting the availabilities of the resources.

There is a vast literature on the MKP. For reviews of the literature up to 2004 or so, see the surveys [14, 15, 27]. For recent examples of exact algorithms, heuristics and upper-bounding procedures, see [29, 35], [1, 11, 32] and [22, 23], respectively.

The MKP has a natural formulation as a 0-1 linear program (0-1 LP). One simple way to compute an upper bound for any 0-1 LP is to solve its continuous relaxation. A more sophisticated approach is to use *surrogate*

*Management Section, Queen's Management School, Riddel Hall, 185 Stranmillis Road, Belfast BT9 5EE, Northern Ireland. E-mail: t.dokka@qub.ac.uk

†Department of Management Science, Lancaster University, Lancaster LA1 4YX, UK. E-mail: {A.N.Letchford,H.Mansoor}@lancaster.ac.uk

relaxation [19, 20]. In the 1980s, several authors experimented with the application of surrogate relaxation to the MKP, with quite promising results (see, e.g., [16, 18, 26, 31, 33]).

To obtain a strong upper bound with surrogate relaxation, one must solve an auxiliary optimisation problem, called the *surrogate dual*. It was shown in 1986 that the dual can be solved exactly in pseudo-polynomial time [5]. In the five above-mentioned papers, however, the dual was solved only approximately. Indeed, up to now, no paper has presented computational results obtained by solving the surrogate dual exactly for benchmark MKP instances.

To address this gap in the literature, we present a simple algorithm for solving the dual exactly. The algorithm exploits the fact that excellent simplex-based LP solvers are now incorporated into many modern optimisation packages (such as `CPLEX` and `Gurobi`). We then test the algorithm on several families of benchmark MKP instances. It turns out that the algorithm can solve the dual in very reasonable computing times, even for large instances. Surprisingly, however, the resulting upper bound tends to be strong only for relatively small MKP instances. Indeed, for many of the larger instances, the upper bounds from continuous and surrogate relaxation turn out to be virtually identical.

For interest, we also present a simple "primal heuristic", which enables one to generate lower as well as upper bounds during our algorithm. The results from this heuristic are rather promising.

The paper has the following structure. The literature is reviewed in Section 2. The algorithm for the dual is described in Section 3, and the primal heuristic is given in Section 4. The computational results are in Section 5. Finally, some concluding remarks are made in Section 6.

Throughout the paper, we assume that the reader is familiar with the basics of linear, integer and dynamic programming (see [7, 9]). We write "SR" and "SD" for *surrogate relaxation* and *surrogate dual*, respectively. We let N denote $\{1, \ldots, n\}$. We assume that the p_j, b_i and a_{ij} are positive integers. Finally, given a vector $v \in \mathbb{R}^p_+$, we let $|v|$ denote $\sum_{i=1}^{p} v_i$.

2 Literature Review

We now review the literature. For brevity, we mention only papers of direct relevance.

2.1 The KP

When $m = 1$, the MKP reduces to the standard 0-1 knapsack problem (KP). Although the KP is itself $\mathcal{NP}$-hard [24], it is often easy in practice (see, e.g., [27]). Moreover, the KP can be solved in pseudo-polynomial time via *dynamic programming* (DP). The original DP algorithm, due to Bellman

[3], takes $O(nb_1)$ time. By reversing the roles of the profits and weights in Bellman's algorithm, one can solve the KP in $O(nP)$ time, where P is the profit of the optimal solution (see Section 2.3 of [27]).

2.2 The MKP

The MKP is $\mathcal{NP}$-hard in the strong sense [17], but solvable in pseudo-polynomial time for fixed m [27]. It has a natural formulation as a 0-1 LP. For all $j \in N$, define a binary variable x_j, taking the value 1 if and only if item j is selected. We then have:

$$\max\{p^T x : Ax \leq b,\ x \in \{0,1\}^n\}. \tag{1}$$

The *continuous relaxation* of the 0-1 LP is obtained by replacing the binary condition with the weaker condition $x \in [0,1]^n$. The resulting LP can be solved extremely quickly in practice. We will let $x^* \in [0,1]^n$ denote the LP solution, U_{LP} the corresponding upper bound, $\pi \in \mathbb{Q}_+^m$ the vector of dual prices, and $\rho \in \mathbb{Q}_+^n$ the vector of reduced costs.

Several exact and heuristic algorithms for the MKP attempt to exploit information in the LP solution, by giving "priority" to variables with large x_j^* and/or small ρ_j (e.g., [1, 11, 29, 35]).

2.3 Surrogate relaxation and duality

Now consider an arbitrary 0-1 LP of the form (1), where negative coefficients are permitted. In SR, we pick a vector $\mu \in \mathbb{Q}_+^m$ of *surrogate multipliers*, and solve the following simpler 0-1 LP [19, 20]:

$$\max\left\{p^T x : (\mu^T A)x \leq \mu^T b,\ x \in \{0,1\}^n\right\}. \tag{2}$$

We will let $U(\mu)$ denote the resulting upper bound.

The SD is the problem of finding the vector μ that minimises $U(\mu)$. It is shown in [19, 20] that $U(\mu)$ is a quasi-convex function of μ. This fact is used in various heuristic algorithms for the SD (e.g., [4, 25, 28, 33]), most of which are variants of the subgradient method.

It has been proved that the SD can be solved in pseudo-polynomial time [5, 12]. However, the proofs in [5, 12] rely on the equivalence of separation and optimisation, which in turn relies on the ellipsoid method [21]. Given that the ellipsoid method is very slow in practice, the results in [5, 12] are of theoretical interest only.

Now recall the definition of π from the previous subsection. It was shown in [19, 20] that $U(\pi) \leq U_{LP}$. Thus, in theory at least, the upper bound from the SD is at least as strong as the one from the LP.

2.4 Surrogate relaxation for the MKP

Observe that, if we apply SR to the MKP, all coefficients are non-negative in (2). Thus, in this case, (2) is a KP. For this reason, SR may be an attractive option for the MKP.

In the 1980s, several papers applied SR to the MKP (e.g., [16, 18, 26, 31, 33]). In those papers, the SD was solved heuristically, via iterative approaches. The resulting upper bounds were good, but the test instances were rather small by today's standards (see again [11, 29]).

Crama & Mazzola [10] proved a negative result concerning the quality of the upper bounds that we can expect to obtain by applying SR to the MKP. It states that, for any $\mu \in \mathbb{Q}_+^m$, we have $U(\mu) \geq \max\left\{U_{LP}/2,\, U_{LP} - p^+\right\}$, where p^+ is the maximum of p_j over all $j \in N$.

myredFinally, we remark that SR was applied to the MKP more recently, in [6, 30]. In both papers, however, only very simple heuristics were used to select the multipliers. Moreover, in [30], SR was used to guide branching decisions in a branch-and-bound algorithm, rather than to improve the upper bound from the LP.

3 Solving the Surrogate Dual

As mentioned above, the exact algorithms in [5, 12] for solving the SD are of theoretical interest only, since the ellipsoid method is very slow in practice. Given the fact that excellent simplex-based LP solvers are now available, we consider using a simplex-based method instead.

Observe that, for any given $\mu \in \mathbb{Q}_+^m$, the upper bound $U(\mu)$ will be integral. We then use the following result, proved in [12]. For any given $\theta \in \mathbb{Z}_+$, there exists a vector $\mu \in \mathbb{Q}_+^m$ such that $U(\mu) \leq \theta$ if and only if the following LP is feasible:

$$\min \quad |\mu|_1 \tag{3}$$

$$\text{s.t.} \quad (A\bar{x} - b)^T \mu \geq 1 \quad \left(\forall \bar{x} \in \{0,1\}^n : p^T \bar{x} > \theta\right) \tag{4}$$

$$\mu \in \mathbb{R}_+^m. \tag{5}$$

We call the LP (3)-(5) the "master" LP. Note that the number of constraints (4) is typically exponential in n. Thus, to solve the master LP (for a given θ), one must use a specialised algorithm. In [12], we used the ellipsoid method. Here, however, we propose to use a simple cutting-plane algorithm based on the dual simplex method. This is because, as mentioned in the introduction, several excellent simplex-based LP solvers are now available.

In each iteration of our cutting-plane algorithm, we solve a relaxation of the master LP that contains only a subset of the constraints (4). The solution to the relaxed master LP yields a "tentative" multiplier vector, say $\bar{\mu}$. One then solves a *separation problem* to check whether $\bar{\mu}$ satisfies all of

the constraints (4) (see [21] for a thorough treatment of separation problems in convex and combinatorial optimisation).

Intuitively, the separation problem attempts to find a binary vector $\bar{x}$ which has large profit, but satisfies the current "tentative" surrogate constraint. The separation problem can itself be formulated as the following 0-1 LP:

$$\min \Big\{ \big(\bar{\mu}^T A\big)x : \; x \in \{0,1\}^n, \; p^T x > \theta \Big\}. \tag{6}$$

The problem (6) is itself a KP (of minimisation type). It is not hard to show that it can be solved exactly in $O(n\theta)$ time. (This can be done using a modified version of the $O(nP)$-time algorithm for the standard KP, mentioned in Subsection 2.1.) Moreover, since the profit of the optimal MKP solution cannot exceed $|p|$, we can assume without loss of generality that $\theta \leq |p|$. Thus, we can solve (6) in $O\big(n\,|p|\big)$ time. Note that this running time is pseudo-polynomial.

The above-mentioned cutting-plane algorithm takes a given value of $\theta \in \mathbb{Z}_+$ as input, but we do not know the optimal value of θ in advance. Thus, we perform a binary search on θ, solving the LP (3)-(5) in each major iteration.

Now, let μ^* denote the (as yet unknown) optimal solution to the SD, and let $\theta^* = U(\mu^*)$ denote the corresponding upper bound (also unknown) for the MKP. For the binary search, we need initial lower and upper bounds on θ^*. We will call these ℓ and u. To obtain ℓ, we simply run a greedy heuristic for the MKP (inserting items in non-increasing order of profit), and then set ℓ to the profit of the resulting MKP solution. As for u, we simply use $\lfloor U_{LP} \rfloor$.

The overall approach is described in Algorithm 1. Note that, towards the end of the outer "while" loop, we set θ to $\lfloor 0.9u + 0.1\ell \rfloor$ rather than to $\lfloor (u + \ell)/2 \rfloor$. This is because, in our experience, θ^* tends to be closer to u than to ℓ.

We remark that the algorithm as stated returns only the upper bound θ^*. One can easily modify the algorithm so that it also returns the optimal multiplier vector μ^*, and/or the associated x vector. We omit details for brevity.

4 Primal Heuristic

Observe that, each time we solve the separation problem (6), we obtain a vector $\bar{x} \in \{0,1\}^n$. Typically, $\bar{x}$ is not a feasible solution to the MKP. For interest, we designed a simple primal heuristic, which attempts to "repair" $\bar{x}$, to make it feasible for the MKP.

Before running the heuristic for the first time, we sort the items in order of "attractiveness" and create a sorted list. In more detail, when we solve the LP relaxation at the start of Algorithm 1, we store the vectors x^* and ρ. We then sort the items in N according to two criteria. First, we sort

Algorithm 1: Solving the Surrogate Dual of the MKP

input: positive integers m, n; non-negative integer matrix A;
positive integer vectors p and b.

`// initialisation`

Solve the LP relaxation of the MKP and set u to $\lfloor U_{LP} \rfloor$;

Run a greedy heuristic for the MKP to get initial lower bound ℓ;

Set up initial "master" LP $\min\left\{|\mu| : |\mu| \geq 1,\ \mu \in \mathbb{R}_+^m\right\}$;

Solve the master LP via primal simplex;

Set θ to u;

`// binary search`

while $u \neq \ell$ **do**

  `// cutting-plane algorithm`

  while *the master LP is feasible* **do**

    Let $\bar{\mu}$ be the current solution to the master LP;

    Solve the 0-1 LP (6), yielding a solution $\bar{x}$;

    if $\left(\bar{\mu}^T A\right)\bar{x} < 1 + \bar{\mu}^T b$ **then**

      `// we have found a violated constraint`

      Add the constraint (4) to the master LP;

      Re-optimise the master LP via dual simplex;

    else

      Break;

    end

  end

  `// check whether` $\theta^* \leq \theta$

  if *the master LP remains feasible* **then**

    Set u to θ;

    Clean the master LP by deleting all cuts with positive slack;

  end

  else

    Set ℓ to $\theta + 1$;

    Let S denote the set of newly-added cutting-planes;

    Clean the master LP by deleting the cutting planes in S;

  end

  Set θ to $\lfloor 0.9u + 0.1\ell \rfloor$;

end

Output θ and stop.

them in non-increasing order of x_j^* value. Then, if there are any items with $x_j^* = 0$, we sort those in non-decreasing order of ρ_j. We call the sorted list "SL".

Our primal heuristic is described in Algorithm 2. The idea is that, instead of relying entirely on the list SL, we give priority to items that have $\bar{x}_j = 1$. Since the separation problem is solved many times during Algorithm 1, this gives our primal heuristic several chances to find a good solution. Of course, the best primal solution found so far is stored in memory.

Algorithm 2: Primal Heuristic

input : positive integers m, n; positive integer vectors p and b, non-negative integer matrix A, sorted list SL, current solution $\bar{x}$ of the subproblem (6).
output: feasible MKP solution $S \subset N$.
Let N^1 be the set of items for which $\bar{x}_j = 1$;
Let $N^2 = N \setminus N^1$;
Set $S = \emptyset$;
for $k = 1, 2$ **do**
  for $t = 1$ *to* $|N^k|$ **do**
    Let j be the t-th item in SL that belongs to N^k;
    if *item j fits into the knapsack* **then**
      Place item j into S;
    end
  end
end

We remark that, if Algorithm 2 obtains a solution whose profit is larger than ℓ, we can update ℓ in Algorithm 1. This can be accomplished by adding the following line to Algorithm 1, immediately after the cutting-plane phase: "if the primal heuristic has found a new best MKP solution, then set ℓ to the profit of that solution". We found, however, that the performance of Algorithm 2 is already satisfactory in practice, making this change unnecessary.

5 Computational Results

In this section, we present our computational results. Our algorithm was implemented in C++ and compiled with Microsoft Visual Studio (v. 15.0) in Windows 10. To solve the master LPs, we used the CPLEX (v. 12.7.1) dual simplex solver, under default settings. All experiments were conducted on a 2.80GHz, 4-core Intel i7-7700HQ laptop, with 16GB RAM.

5.1 Test instances

Several hundred benchmark MKP instances are available at the `OR-Library` [2]. Most of those, however, are very small by modern standards. Of the remaining instances, we have selected the following three sets:

- Instances proposed in 1967 by Weingartner & Ness [36]. All of these instances have $m = 2$. There are six instances with $n = 28$ and two with $n = 105$, making 8 instances in total. We will call these 'WN' instances.
- Instances proposed in 1979 by Shi [34]. All of these instances have $m = 5$. There are five instances with $n = 30$ and five with $n = 90$. For each value of $n \in \{40, 50, \ldots, 80\}$, there are only four instances. This makes 30 instances in total.
- Large-scale random instances proposed in 1998 by Chu & Beasley [8]. There are thirty instances for each combination of $n \in \{100, 250, 500\}$ and $m \in \{5, 10, 30\}$, making 270 instances in total. We will call these 'CB' instances.

The optimal values for the WN and Shi instances are available in the `OR-Library`. For the CB instances, the situation is more complicated. For the instances with $n = 100$ and/or $m \leq 10$, the optimal values are known [29]. The remaining 60 instances, with $n \in \{250, 500\}$ and $m = 30$, remain unsolved. The best known lower bounds for those instances can be found in [29].

5.2 Results from LP and SD

Table 1 presents a summary of the results obtained with our upper-bounding procedures. The first three columns give the source of the instances, the number of items n and number of constraints m. The columns headed "UB %gap" show the average gap between the LP and SD upper bounds and the optimum, expressed as a percentage of the optimum. If a number is in parenthesis, it is because the optimal values are not known for the corresponding instances. (For those instances, the true gaps could be smaller, but cannot be larger.) The next two columns show the average number of major iterations in the binary search procedure, and the average number of times the separation problem had to be solved, respectively. Finally, the last column shows the average time taken by our SD algorithm, in seconds. (The time taken to solve the LP was negligible, being less than 0.1 seconds, in every case.)

Each entry in the last five columns is an average taken over the instances of the given size. The full results will be made available at the Lancaster

			UB %gap		Iterations		
Set	n	m	LP	SD	Bin	Sep	Time
WN	28	2	1.685	0.571	23.50	39.0	0.902
1967	105	2	0.369	0.251	19.50	41.0	2.423
	30	5	0.923	0.196	8.00	18.6	0.58
	40	5	0.445	0.184	8.25	25.0	0.52
Shi	50	5	1.193	0.782	10.50	28.5	0.79
1979	60	5	0.493	0.109	10.25	24.5	1.46
	70	5	0.428	0.195	8.25	22.0	0.65
	80	5	0.400	0.149	10.25	26.3	0.83
	90	5	0.276	0.054	9.20	24.4	0.92
	100	5	0.590	0.529	10.70	87.5	3.32
	250	5	0.135	0.126	8.10	110.3	4.46
	500	5	0.044	0.043	6.90	134.0	8.59
CB	100	10	0.956	0.943	6.60	183.7	6.27
1998	250	10	0.276	0.275	5.97	274.2	12.37
	500	10	0.100	0.100	6.23	313.3	20.55
	100	30	1.714	1.713	5.67	290.4	10.67
	250	30	(0.607)	(0.606)	5.80	572.3	31.52
	500	30	(0.275)	(0.274)	6.17	933.1	101.73

Table 1: Results obtained with upper-bounding procedures.

University Data Repository.)[1]

We see that, for the WN and Shi instances, the SD bound is much stronger than the LP bound. For the CB instances, however, the improvement is small. In fact, closer inspection of the output revealed that, for many of the larger CB instances, the SD bound was equal to the LP bound rounded down to the nearest integer. Thus, for large-scale random instances, one may as well just solve the continuous relaxation, without bothering to use SR.

Another observation is that both integrality gaps tend to decrease as n increases, but increase as m increases. The effort required to solve the SD, however, seems to be an increasing function of both n and m.

On the positive side, the running times of our exact algorithm are reasonable, being measured in second rather than minutes. We remark that some additional implementation "tricks" could potentially improve the speed of our algorithm. For example, one could begin by running a local-search heuristic for the MKP, in order to obtain a better initial lower bound ℓ. One could also use a heuristic to solve the separation problem (6), and only use an exact separation algorithm when the heuristic fails.

[1] `http://www.research.lancs.ac.uk/portal/en/datasets/search.html`

Set	n	m	Gr	LP	SD
WN	28	2	1.708	4.243	0.494
1967	105	2	2.668	1.728	0.280
	30	5	2.343	0.840	0.176
	40	5	4.416	0.829	0.157
Shi	50	5	6.734	0.326	0.051
1979	60	5	3.872	0.446	0.094
	70	5	5.971	1.030	0.018
	80	5	6.160	1.399	0.193
	90	5	7.667	0.226	0.000
	100	5	11.23	0.999	0.401
	250	5	11.20	0.397	0.227
	500	5	10.49	0.230	0.146
CB	100	10	9.441	1.303	0.600
1998	250	10	8.016	0.578	0.413
	500	10	7.598	0.300	0.261
	100	30	6.612	1.722	1.341
	250	30	(5.74)	(0.876)	(0.835)
	500	30	(5.22)	(0.480)	(0.480)

Table 2: Average percentage gaps for primal heuristics.

5.3 Results from primal heuristics

The results obtained with the primal heuristics are summarised in Table 2. Each entry in the last three columns shows the average gap between a lower bound and the optimum, expressed as a percentage of the optimum. The column "Gr'" corresponds to the greedy heuristic, in which items are inserted in non-increasing order of profit. The column "LP" corresponds to the case in which items are first inserted in non-increasing order of x_j^*, and then in non-decreasing order of ρ_j. (This is equivalent to running Algorithm 2 with N^1 set to N.) The column "SD" corresponds to our primal heuristic.

We do not report average running times in the table, because the time taken by all primal heuristics was negligible compared to the time taken to solve the SD.

We see that the LP-based primal heuristic performs much better than the greedy heuristic in almost all cases, and our SD-based primal heuristic performs much better still. Of course, one could attempt to improve the lower bounds further using local search or some other meta-heuristic. This is however beyond the scope of this paper.

6 Conclusion

Although surrogate relaxation has been applied many times to the MKP, we are the first to solve the surrogate dual exactly for instances of meaningful size. To do this, we used an approach based on the dual simplex method, rather than one of the classical heuristics, which are all essentially variations of the subgradient method.

The results indicate that one can solve the dual quite quickly in practice, using a modern implementation of the dual simplex method to solve the intermediate linear programs. Moreover, the upper bound from the surrogate dual was very strong for the instances with no more than 100 items. For the larger instances, however, the upper bound from the surrogate dual was often no better than the standard linear programming bound.

Finally, our simple primal heuristic, based on "repairing" the infeasible solutions found during the course of our algorithm, produced solutions of very good quality. This gives us hope that surrogate relaxation could be a useful tool to "drive" heuristics for various combinatorial optimisation problems. We explore this topic in a follow-up paper [13].

References

[1] E. Angelelli, R. Mansini & M.G. Speranza (2010) Kernel search: a general heuristic for the multi-dimensional knapsack problem. *Comput. Oper. Res.*, 37, 2017–2026.

[2] J.E. Beasley (1990) OR-Library: distributing test problems by electronic mail. *J. Oper. Res. Soc.*, 41, 1069–1072.

[3] R. Bellman (1957) *Dynamic Programming.* Princeton, NJ: Princeton University Press.

[4] N. Boland, A.C. Eberhard & A. Tsoukalas (2015) A trust region method for the solution of the surrogate dual in integer programming. *J. Optim. Th. Appl.*, 167, 558–584.

[5] E. Boros (1986) On the complexity of the surrogate dual of 0–1 programming. *Zeit. Oper. Res.*, 30, A145–A153.

[6] V. Boyer, M. Elkihel & D. El Baz (2009) Heuristics for the 0-1 multidimensional knapsack problem. *Eur. J. Oper. Res.*, 199, 658–664.

[7] D.S. Chen, R.G. Batson & Y. Dang (2010) *Applied Integer Programming.* Hoboken, NJ: Wiley.

[8] P.C. Chu & J.E. Beasley (1998) A genetic algorithm for the multidimensional knapsack problem. *J. Heur.*, 4, 63–86.

[9] M. Conforti, G. Cornuéjols & G. Zambelli (2015) *Integer Programming*. Graduate Texts in Mathematics, vol. 271. Springer.

[10] Y. Crama & J. Mazzola (1994) On the strength of relaxations of multidimensional knapsack problems. *INFOR*, 32, 219–225.

[11] F. Della Croce & A. Grosso (2012) Improved core problem based heuristics for the 0/1 multi-dimensional knapsack problem. *Comput. Oper. Res.*, 39, 27–31.

[12] T. Dokka, A.N. Letchford & M.H. Mansoor (2021) On the complexity of surrogate and group relaxation for integer linear programs. *Oper. Res. Lett.*, 49, 530–534.

[13] T. Dokka, A.N. Letchford & M.H. Mansoor (2022) Surrogate relaxation as a matheuristic. *Working paper*, Department of Management Science, Lancaster University, UK.

[14] A. Fréville (2004) The multidimensional 0–1 knapsack problem: an overview. *Eur. J. Oper. Res.*, 155, 1–21.

[15] A. Fréville & S. Hanafi (2005) Multidimensional 0–1 knapsack problem: bounds and computational aspects. *Ann. Oper. Res.*, 139, 195–227.

[16] A. Fréville & G. Plateau (1993) An exact search for the solution of the surrogate dual of the 0-1 bidimensional knapsack problem. *Eur. J. Oper. Res.*, 68, 413–421.

[17] M.R. Garey & D.S. Johnson (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: Freeman.

[18] B. Gavish & H. Pirkul (1985) Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality. *Math. Program.*, 31, 78–105.

[19] F. Glover (1975) Surrogate constraint duality in mathematical programming. *Oper. Res.*, 23, 434–451.

[20] H.J. Greenberg & W.P. Pierskalla (1970) Surrogate mathematical programming. *Oper. Res.*, 18, 924–939.

[21] M. Grötschel, L. Lovász & A.J. Schrijver (1988) *Geometric Algorithms and Combinatorial Optimization*. New York: Wiley.

[22] H. Gu (2018) Local cuts for 0–1 multidimensional knapsack problems. In: R. Sarker *et al.* (eds.) *Data and Decision Sciences in Action*. Cham, Switzerland: Springer.

[23] K. Kaparis & A.N. Letchford (2008) Local and global lifted cover inequalities for the multidimensional knapsack problem. *Eur. J. Oper. Res.*, 186, 91–103.

[24] R.M. Karp (1972) Reducibility among combinatorial problems. In R.E. Miller *et al.* (eds.) *Complexity of Computer Computations*, pp. 85–103. New York: Plenum.

[25] M.H. Karwan & R.L. Rardin (1980) Searchability of the composite and multiple surrogate dual functions. *Oper. Res.*, 28, 1251–1257.

[26] M.H. Karwan & R.L. Rardin (1984) Surrogate dual multiplier search procedures in integer programming. *Oper. Res.*, 32, 52–69.

[27] H. Kellerer, U. Pferschy & D. Pisinger (2004) *Knapsack Problems.* Berlin: Springer.

[28] S.-L. Kim & S. Kim (1998) Exact algorithm for the surrogate dual of an integer programming problem: subgradient method approach. *J. Optim. Th. Appl.*, 96, 363–375.

[29] R. Mansini & M.G. Speranza (2012) CORAL: an exact algorithm for the multidimensional knapsack problem. *INFORMS J. Comput.*, 24, 399–415.

[30] M.A. Osorio, F. Glover & P. Hammer (2002) Cutting and surrogate constraint analysis for improved multidimensional knapsack solutions. *Ann. Oper. Res.*, 117, 71–93.

[31] H. Pirkul (1987) A heuristic solution procedure for the multiconstraint zero-one knapsack problem. *Nav. Res. Logist.*, 34, 161–172.

[32] J. Puchinger, G.R. Raidl & U. Pferschy (2010) The multidimensional knapsack problem: structure and algorithms. *INFORMS J. Comput.*, 22, 250–265.

[33] S. Sarin, M.H. Karwan & R.L. Rardin (1987) A new surrogate dual multiplier search procedure. *Nav. Res. Logist.*, 34, 431–450.

[34] W. Shi (1979) A branch and bound method for the multiconstraint zero one knapsack problem. *J. Oper. Res. Soc.*, 30, 369–378.

[35] Y. Vimont, M. Boussier, M. Vasquez (2008) Reduced costs propagation in an efficient implicit enumeration for the 0–1 multidimensional knapsack problem. *J. Combin. Optim.*, 15, 165–178.

[36] H.M. Weingartner & D.N. Ness (1967) Methods for the solution of the multi-dimensional 0/1 knapsack problem. *Oper. Res.*, 15, 83–103.