Relaxations and heuristics for the multiple non-linear separable knapsack problem

Relaxations and Heuristics for the Multiple Non-Linear Separable Knapsack Problem

Claudia D'Ambrosio[1] Silvano Martello[2] Luca Mencarelli[1]

November 20, 2017

Abstract

We consider the multiple non-linear knapsack problem with separable non-convex functions. The problem, which can be modeled as a (mixed) integer non-linear program, is extremely difficult to solve in practice. We present a fast heuristic algorithm, based on constructive techniques, surrogate relaxations, and local search improvements. Computational comparisons with exact and heuristic methods for general non-convex mixed integer non-linear programs show that the proposed approach provides good-quality solutions within small computing times.

Keywords. Multiple non-linear knapsack problem, Heuristic algorithms, Surrogate relaxation.

1 Introduction

Let x be an $m \times n$ array of non-negative real variables $x = [x_{ij}]$ $(i = 1, \ldots, m,\ j = 1, \ldots, n)$ and define $M = \{1, \ldots, m\}$ and $N = \{1, \ldots, n\}$. We consider a multiple non-linear knapsack problem in which

- the objective function and the capacity constraints are expressed by separable, continuously differentiable functions $f_j(x_{ij})$ and $g_j(x_{ij})$ $(i \in M,\ j \in N)$;
- the values of f and g do not depend on i, i.e., $f_j(x_{ij}) = f_j(x_{kj})$ and $g_j(x_{ij}) = g_j(x_{kj})$ when $x_{ij} = x_{kj}$ for $j \in N$ and $i, k \in M$;
- $f_j(x_{ij})$ and $g_j(x_{ij})$ are non-linear non-negative non-decreasing functions for $j \in N$ and $i \in M$;
- for each $j \in N$, the total value of x_{ij} over all $i \in M$ cannot exceed a given upper bound u_j;
- integrality requirements may be imposed on part of the variables.

Note that there is no further assumption on $f_j(x_{ij})$ and $g_j(x_{ij})$ which, in general, can be non-convex and non-concave.

[1]LIX CNRS (UMR7161), École Polytechnique, 91128 Palaiseau Cedex, France.
Email: {dambrosio,mencarelli}@lix.polytechnique.fr
[2]DEI "Guglielmo Marconi", University of Bologna, 40136 Bologna, Italy.
Email: silvano.martello@unibo.it

The *Multiple Non-Linear Knapsack Problem* (MNLKP) is:

$$\max \sum_{i \in M} \sum_{j \in N} f_j(x_{ij}) \tag{1}$$

$$\text{s.t.} \sum_{j \in N} g_j(x_{ij}) \leq c_i \qquad i \in M \tag{2}$$

$$\sum_{i \in M} x_{ij} \leq u_j \qquad j \in N \tag{3}$$

$$x_{ij} \geq 0 \qquad i \in M,\ j \in N \tag{4}$$

$$x_{ij} \text{ integer} \qquad i \in M,\ j \in \overline{N} \subseteq N, \tag{5}$$

which can be informally described as follows. We are given m knapsacks and n items. Each item j has a *profit function* $f_j(x_{ij})$ and a *weight function* $g_j(x_{ij})$ $(i \in M)$, and each knapsack i has a *capacity* c_i. For each item j we want to assign x_{ij} quantities (some restricted to integer values) to the knapsacks so that

- the overall assigned profit is maximized, see (1);
- for each knapsack i the overall assigned weight does not exceed the corresponding capacity, see (2);
- for each item j the overall assigned quantity does not exceed the corresponding upper bound, see (3).

When f and g are linear functions (i.e., (1) and (2) become $\max \sum_{i \in M} \sum_{j \in N} p_j x_{ij}$ and $\sum_{j \in N} w_j x_{ij} \leq c_i$, respectively), $u_j = 1$ for $j \in N$ and x_{ij} is restricted to binary variables for $i \in M$ and $j \in N$, the MNLKP becomes the classical 0-1 multiple knapsack problem, see, e.g., Martello and Toth [12] or Kellerer et al. [10]. It follows that the MNLKP is, at least, strongly $\mathcal{NP}$-hard. When $m = 1$ (i.e., (1) and (2) become $\max \sum_{j \in N} f_j(x_j)$ and $\sum_{j \in N} g_j(x_j) \leq c_1$, respectively), the MNLKP becomes the classical (single) *Non-Linear Knapsack Problem* (NLKP), see, e.g., D'Ambrosio and Martello [5].

Non-linear knapsack problems have a number of applications in different fields such as portfolio selection, production planning, and resource allocation (see, e.g., Ibaraki and Katoh [8], Li and Sun [11], and Bretthauer and Shetty [3]). Let us consider the case where one has to assign one or more (m) economical resources i to advertise n different products and assume that c_i is the advertising budget for resource i. The optimization problem consists in maximizing the overall expected sales from all categories. The sales may expect to sharply increase as the advertisements reach an increasing number of potential buyers. However, at some point there is a saturation effect by which increasing advertisements does not significantly increase the sales. In other words, the profit function $f_j(x_{ij})$ has the non-convex and non-concave shape depicted in Figure 1. The advertisement costs $g_j(x_{ij})$ can either be linear (e.g., $g_j(x_{ij}) = x_{ij}$, if a constant unit cost is assumed), or non-linear (if economies of scale are considered, i.e., unit costs decrease with size).

To the best of our knowledge no study of the MNLKP has been presented in the literature. Zhang and Hua [16] proposed an exact method for the minimization version of the NLKP for the case where objective and constraints functions are convex and all variables are continuous, i.e., $\overline{N} = \emptyset$. Exact and heuristic algorithms for the same problem with integer variables were presented by Zhang and Chen [15]. D'Ambrosio and Martello [5] provided heuristics for the NLKP. A survey chapter on NLKP with integer variables can be found in the book by Li and Sun [11].

In the next section we discuss surrogate and Lagrangian relaxations for the MNLKP. In Section 3 we describe a constructive heuristic algorithm, which extends the approach proposed by D'Ambrosio and Martello [5] for the NLKP, and we introduce two heuristics which derive a feasible solution from the usually infeasible solution provided by the surrogate relaxation. In Section 4 we describe a local search procedure that improves a feasible solution through variations of variable values of item pairs. The overall heuristic algorithm, summarized in Section 5, is experimentally evaluated through comparison with the solutions provided by open-source non-linear programming solvers and with the upper bounds produced by an exact global optimization solver. The computational results, reported in Section 6, show that

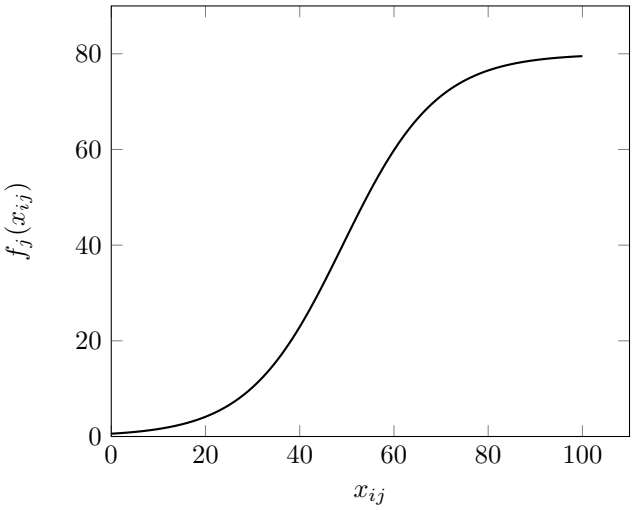

Figure 1: Example of profit function.

the proposed approach provides good-quality solutions within small CPU times. Conclusions follow in Section 7.

2 Relaxations

Let us consider the surrogate relaxation, S(MNLKP,π), of the MNLKP, defined as follows. Given a vector of m non-negative multipliers $(\pi_1, \ldots, \pi_m)$, we multiply the i-th constraint (2) by π_i and replace the resulting set of constraints (2) by their sum, i.e.,

$$\max \sum_{i \in M} \sum_{j \in N} f_j(x_{ij}) \tag{6}$$

$$\text{s.t.} \sum_{i \in M} \pi_i \sum_{j \in N} g_j(x_{ij}) \leq \sum_{i \in M} \pi_i c_i \tag{7}$$

$$\sum_{i \in M} x_{ij} \leq u_j \qquad j \in N \tag{8}$$

$$x_{ij} \geq 0 \qquad i \in M,\ j \in N \tag{9}$$

$$x_{ij} \text{ integer} \qquad i \in M,\ j \in \overline{N} \subseteq N. \tag{10}$$

Let $z(S(MNLKP, \pi))$ denote the optimal solution value of (6)-(10) under given multipliers π. The *surrogate dual problem*

$$\min_{\pi \geq 0} \{z(S(MNLKP, \pi))\} \tag{11}$$

consists in finding the optimal vector of multipliers, i.e., the one producing the minimum optimal value for the surrogate relaxation, and hence the tighter upper bound for the MNLKP. The analogous surrogate relaxation of the 0-1 multiple linear knapsack problem has a strong surrogate dual property: The optimal vector of multipliers is $\pi_i = k$ for all $i \in M$, where k is any positive constant (see Martello and Toth [12]). Unfortunately, such property does not hold for the MNLKP, as shown by the following example. Let $m = 2$, $n = 1$, $u_1 = 100$, $c_1 = 10$, $c_2 = 2$, and, for $i \in \{1, 2\}$, $f_1(x_{i1}) = x_{i1}$, $g_1(x_{i1}) = 80/\left(1 + 50\mathrm{e}^{-\frac{1}{10}(x_{i1}-10)}\right)$. (This is actually the function used as an example of $f(x)$ in Figure 1.) For $\pi_1 = \pi_2 = 1$ the optimal solution to S(MNLKP,π) is $x_{11} = x_{21} \simeq 24$ (with $g_1(x_{11}) = g_1(x_{21}) \simeq 6$) and

has value $\simeq 48$. For $\pi_1 = 1$ and $\pi_2 = 2$ such solution violates (7) and the optimal solution is $x_{11} \simeq 26$ and $x_{21} \simeq 18$ (with $g_1(x_{11}) \simeq 7.2$ and $g_1(x_{21}) \simeq 3.4$), of value $\simeq 44$. Intuitively, by increasing a multiplier, the right-hand side of (7) increases linearly while its left-hand side can increase more than linearly (as it locally happens for the shape of Figure 1).

Although the optimal surrogate dual solution cannot be immediately found, we will see in Section 3.2 that reasonably good multipliers can be heuristically obtained, and that the surrogate solution can be used to produce feasible solutions which can be conveniently used in the heuristic process introduced in the next sections.

Given a vector $(\lambda_1, \ldots, \lambda_m)$ of non-negative multipliers, a possible Lagrangian relaxation, L(MNLKP,λ), of the MNLKP can be obtained by relaxing (2):

$$\sum_{i \in M} \lambda_i c_i + \max \sum_{i \in M} \sum_{j \in N} (f_j(x_{ij}) - \lambda_i g_j(x_{ij})) \tag{12}$$

$$\text{s.t.} \sum_{i \in M} x_{ij} \leq u_j \qquad j \in N \tag{13}$$

$$x_{ij} \geq 0 \qquad i \in M,\ j \in N \tag{14}$$

$$x_{ij} \text{ integer} \qquad i \in M,\ j \in \overline{N} \subseteq N. \tag{15}$$

Generally speaking, relaxation (12)-(15) has the advantage of being decomposable into n subproblems (one for each item j) with a non-linear objective function and a single linear knapsack constraint. Preliminary computational experiments have shown, however, that these problems remain difficult to solve in practice. In addition, the experiments have shown that the heuristic solutions that can be obtained from this relaxation are considerably worse than those that can be obtained from the surrogate relaxation (see Section 3.2). Similar considerations apply to the relaxations that can be obtained by relaxing in a Lagrangian fashion constraints (3), or both constraints (3) and (2).

3 Heuristics

In this section we propose two heuristics for the MNLKP, one based on an iterated constructive procedure and one obtained by deriving a feasible solution from the surrogate relaxation.

3.1 Constructive heuristic

We first show how the constructive heuristics presented by D'Ambrosio and Martello [5] for the single knapsack case can be generalized to provide good quality feasible solutions to the MNLKP (see Mencarelli et al. [13]). In the following we assume without loss of generality that the knapsacks have been preliminary sorted so that $c_1 \geq c_2 \geq \cdots \geq c_m$.

We start by defining a number s of samplings and a sampling step $\delta_j = u_j/s$ for $j \in N$ (or $\delta_j = \max(1, \lfloor u_j/s \rfloor)$ if $j \in \overline{N}$). We use such values to discretize profit and weight functions. Define

$$r_{jk} = \frac{f_j(k\delta_j)}{g_j(k\delta_j)} \qquad (j \in N, k = 1, \ldots, s). \tag{16}$$

For every item j let $\mu_j = \arg\max_{k=1,\ldots,s}\{r_{jk}\}$ and assume that the items are sorted according to their maximum profit-to-weight ratio (16), i.e., so that $r_{1\mu_1} \geq r_{2\mu_2} \geq \cdots \geq r_{n\mu_n}$.

The algorithm considers one knapsack at a time and fills it completely. Algorithm 1 details it by referring to a generic knapsack i.

Initially the algorithm considers items 1 and 2, i.e., the two items with best largest profit-to-weight ratio, and finds the highest sampling point $\bar{\mu}_1$ where item 1 has a better ratio than the best ratio of item 2 (Step 4). It assigns $\bar{\mu}_1\delta_1$ units of item 1 to knapsack i and updates the upper bound of item 1 and the residual capacity (Steps 5 and 6). Assume by the moment that the sampling points corresponding to μ_2 and μ_3 remain feasible. The process is iterated by finding the highest feasible sampling point $\bar{\mu}_2$ where item 2 has a better ratio than the best ratio of item 3: the algorithm assigns $\bar{\mu}_2\delta_2$ units of item 2 to the knapsack (and so on with the next pairs of items). If instead (Step 7) for at least one of the next two items, say 2 and 3, the sampling point corresponding to μ_2 or μ_3 is infeasible, an update of the μ values is performed on items $2, 3, \ldots$ (and, consequently, the item order might change).

Algorithm 1 Procedure Construct(i).

1: $\bar{c}_i := c_i$;
2: $j := 1$;
3: **while** $j < n$ **and** $\bar{c}_i > 0$ **do**
4: $\quad\bar{\mu}_j := \max\{k : r_{jk} \geq r_{(j+1)\mu_{j+1}}, \mu_j \leq k \leq s\}$;
5: $\quad x_{ij} := \bar{\mu}_j\,\delta_j$;
6: $\quad\bar{u}_j := u_j - x_{ij}$, $\bar{c}_i = \bar{c}_i - g_j(x_{ij})$;
7: $\quad$**if** $(g_{j+1}(\mu_{j+1}\,\delta_{j+1}) > \bar{c}_i$ or $g_{j+2}(\mu_{j+2}\,\delta_{j+2}) > \bar{c}_i)$ **then** update μ for items $j+1, j+2, \ldots$ (and possibly update their order);
8: $\quad j := j + 1$;
9: **end while**
10: **if** $\bar{c}_i > 0$ **then** {**comment:** fill the residual capacity with item n}
11: $\quad x_{in} := \min(g_n^{-1}(\bar{c}_i), u_n)$;
12: $\quad\bar{u}_n = u_n - x_{in}$, $\bar{c}_i = \bar{c}_i - g_n(x_{in})$;
13: **end if**

Note that, in order to improve on the efficiency, whenever a new x_{ij} is defined, the ratios of the unscanned items change, but it is not necessary to update and re-sort all of them: indeed, we can just consider the two items providing the maximum and second maximum new ratios, which can be identified in linear time, and proceed with a new search.

Moreover, in order to guarantee the existence of the inverse g_j^{-1} (Step 11), we assume, for simplicity, that, for $j \in N$, functions g_j are strictly increasing and continuous. If it is not the case, we simply consider the pseudo-inverse of g_j with the largest value for the preimage. Note that, given a scalar $\bar{c}_i$, the evaluation of the (pseudo)inverse function at that point, i.e., $g_j^{-1}(\bar{c}_i)$, does not imply the need to have its explicit form. In fact, its evaluation at $\bar{c}_i$ reduces to computing the zeros of $g_j(x_{ij}) - \bar{c}_i$, and hence we can assume that this operation can be performed in a CPU time bounded by a constant independent of the instance size.

The algorithm can be improved through a refined search for $\bar{\mu}_j$. Once it has been obtained (at Step 4), the interval $[\bar{\mu}_j, \bar{\mu}_{j+1}]$ can be searched with a smaller sampling step and new, more precise, profit-to-weight ratios for item j can be computed. In this way a more precise point $\bar{\mu}_j$ is obtained, and the process can be iterated by further decreasing the sampling step.

Steps 4–8 are iterated at most n times. At each iteration, we find the maximum and second maximum ratios, which takes $O(n)$ time. As pointed out above, Step 7 as well can be implemented so as to take $O(n)$ time, by avoiding complete resorting of the unscanned items. If the number of refinements through smaller sampling steps is bounded by a constant (as it reasonably occurs in practical implementations), the time complexity of Construct(i) is $O(n^2)$.

A heuristic for the MNLKP can be obtained by executing procedure Construct(i) on knapsacks 1, 2, ... by considering, at each iteration, only those items for which the previous assignments did not reach the corresponding upper bound. The overall Constructive procedure, shown in Algorithm 2, is terminated by re-scanning knapsacks and items, and filling in a greedy fashion the residual capacities, if any.

Algorithm 2 Procedure Constructive.

for $i := 1$ **to** m **do** Construct(i) **comment:** optionally include the refined search;
for $i := 1$ **to** m **do**
  if $\bar{c}_i > 0$ **then**
    for $j := 1$ **to** n **do** {**comment:** increase x_{ij} as much as possible}
      $x_{ij} := x_{ij} + \min(g_j^{-1}(\bar{c}_i), \bar{u}_j)$;
      if $j \in \overline{N}$ **then** $x_{ij} := \lfloor x_{ij} \rfloor$;
      $\bar{u}_j := u_j - \sum_{k \in M} x_{kj}$, $\bar{c}_i := c_i - \sum_{k \in N} g_j(x_{ik})$;
    end for
  end if
end for

3.2 Surrogate heuristics

In this section we describe two simple heuristics based on the feasibility recovery of the surrogate relaxation (6)-(10). Let us first consider the problem of determining good surrogate multipliers π. A series of preliminary experiments was performed on the benchmark instances adopted for the computational experiments of Section 6, with

(i) π_i uniformly random in $[0.0, 3.0]$ for all $i \in M$;

(ii) π_i uniformly random in $[0.8, 1.2]$ for all $i \in M$;

(iii) π_i uniformly random in $[0.9, 1.1]$ for all $i \in M$,

and

(iv) $\pi_i = 1$ for all $i \in M$.

It turned out that the surrogate solutions produced by (i) were dominated by the other generations, those produced by (ii) and (iii) had about the same quality, and those produced by (iv) were, on average, clearly the best ones. Additional tests were performed using (easier) convex and concave objective functions, globally obtaining the same results. It was thus decided to always adopt option (iv). In Section 2 we have shown that identical multipliers (optimal solution of the surrogate dual for the linear case) are not optimal for the non-linear case. It is worth observing that they appear to be a good choice for such case too, at least for the objective functions we considered.

Let x^*_{ij} ($i \in M$, $j \in N$) be the surrogate solution. We assume without loss of generality that: (i) the knapsacks are sorted so that $c_1 \geq c_2 \geq \cdots \geq c_m$; (ii) the items are sorted according to non-increasing profit-to-weight ratios r_j relative to the surrogate solution, but evaluated for the original functions, i.e.,

$$r_j = \frac{\sum_{i \in M} f_j(x^*_{ij})}{\sum_{i \in M} g_j(x^*_{ij})} \qquad (j \in N). \tag{17}$$

The first heuristic based on feasibility recovery can be stated as shown in Algorithm 3.

Algorithm 3 Procedure Surrogate-feas-1(x^*).

1: **for** $j := 1$ **to** n **do** $\bar{u}_j := u_j$;
2: **for** $i := 1$ **to** m **do**
3: $\quad \bar{c}_i := c_i$;
4: $\quad$ **for** $j := 1$ **to** n **do** $x_{ij} := 0$;
5: **end for**
6: **for** $i := 1$ **to** m **do**
7: $\quad$ **for** $j := 1$ **to** n **do**
8: $\quad\quad$ **if** $g_j(x^*_{ij}) \leq \bar{c}_i$ **then** $x_{ij} := x^*_{ij}$, $\bar{c}_i := \bar{c}_i - g_j(x^*_{ij})$, $\bar{u}_j := \bar{u}_j - x^*_{ij}$;
9: $\quad$ **end for**
10: **end for**
11: $\bar{\imath} = \arg\max_{i \in M}\{\bar{c}_i\}$, $\bar{c}_{\max} := \bar{c}_{\bar{\imath}}$, $\bar{u}_{\max} := \max_{j \in N}\{\bar{u}_j\}$;
12: **while** $\bar{c}_{\max} > 0$ **and** $\bar{u}_{\max} > 0$ **do**
13: $\quad$ **for** $j := 1$ **to** n **do**
14: $\quad\quad$ $\bar{r}_j := \begin{cases} 0 & \text{if } \bar{u}_j = 0 \\ \frac{f_j(\min(x_{\bar{\imath}j}+\bar{u}_j, \max(0, \lfloor g_j^{-1}(\bar{c}_{\bar{\imath}}+g_j(x_{\bar{\imath}j}))\rfloor)))}{g_j(\min(x_{\bar{\imath}j}+\bar{u}_j, \max(0, \lfloor g_j^{-1}(\bar{c}_{\bar{\imath}}+g_j(x_{\bar{\imath}j}))\rfloor)))} & \text{if } \bar{u}_j > 0 \text{ and } i \in \bar{N} \\ \frac{f_j(\min(x_{\bar{\imath}j}+\bar{u}_j, \max(0, g_j^{-1}(\bar{c}_{\bar{\imath}}+g_j(x_{\bar{\imath}j})))))}{g_j(\min(x_{\bar{\imath}j}+\bar{u}_j, \max(0, g_j^{-1}(\bar{c}_{\bar{\imath}}+g_j(x_{\bar{\imath}j})))))} & \text{if } \bar{u}_j > 0 \text{ and } i \notin \bar{N} \end{cases}$;
15: $\quad$ **end for**
16: $\quad \bar{\jmath} = \arg\max_{j \in N}\{\bar{r}_j\}$;
17: $\quad x_{\bar{\imath}\bar{\jmath}} := \min(x_{\bar{\imath}\bar{\jmath}} + \bar{u}_{\bar{\jmath}}, \max(0, g_{\bar{\jmath}}^{-1}(\bar{c}_{\bar{\imath}} + g_{\bar{\jmath}}(x_{\bar{\imath}\bar{\jmath}}))))$;
18: $\quad$ **if** $\bar{\jmath} \in \overline{N}$ **then** $x_{\bar{\imath}\bar{\jmath}} := \lfloor x_{\bar{\imath}\bar{\jmath}} \rfloor$;
19: $\quad \bar{u}_{\bar{\jmath}} := u_{\bar{\jmath}} - \sum_{i \in M} x_{i\bar{\jmath}}$, $\bar{c}_{\bar{\imath}} := c_{\bar{\imath}} - \sum_{j \in N} g_j(x_{\bar{\imath}j})$;
20: $\quad \bar{\imath} = \arg\max_{i \in M}\{\bar{c}_i\}$, $\bar{c}_{\max} := \bar{c}_{\bar{\imath}}$, $\bar{u}_{\max} := \max_{j \in N}\{\bar{u}_j\}$;
21: **end while**

We start with an empty solution. In the first phase (Steps 6-10) the quantities x^*_{ij} are iteratively reassigned to knapsack i ($i = 1, \ldots, m$), as long as the resulting partial solution satisfies the capacity constraints. Due to the preliminary sortings, we first consider the more promising items, i.e., the ones with better profit-to-weight ratios. In the second phase we identify the knapsack $\bar{\imath}$ with the largest residual capacity (Step 11), we update the profit-to-weight ratios (Steps 13-15), and we determine the item $\bar{\jmath}$ with the best *residual* profit-to-weight ratio (Step 16). Knapsack $\bar{\imath}$ is then "filled" as much as possible with item $\bar{\jmath}$ (Steps 17-18). We iteratively update the upper bound and the residual capacity (Step 19), and we repeat the previous steps, as long as the the largest residual capacity or the largest residual upper bound are strictly positive.

Loops 2-5 and 6-10 take $O(mn)$ time. The **while** loop is executed at most $\max(m, n)$ times. By assuming, as for procedure Constructive, that the computation of the inverse of a weight function takes a CPU time bounded by a constant independent of the input size, each iteration takes $O(n)$ time. The overall time complexity of Algorithm 3 is thus $O(mn + n^2)$.

While Algorithm 3 starts with an empty solution and tries to construct a feasible solution that replicates as much as possible the (infeasible) surrogate solution, our second heuristic based on feasibility recovery (shown in Algorithm 4) starts with the surrogate solution and reduces the quantity x^*_{ij} of some items currently assigned to knapsack i ($i \in M$), until the capacity constraints (2) are satisfied. We consider the items in reverse order with respect to Algorithm 3, i.e., according to *non-decreasing* r_j values (see (17)) and we iteratively reduce the quantity of the item that has the worst profit-to-weight ratio, so as to undermine as little as possible the quality of the surrogate solution.

Algorithm 4 Procedure Surrogate-feas-2(x^*).

1: **for** $i := 1$ **to** m **do for** $j := 1$ **to** n **do** $x_{ij} := x^*_{ij}$;
2: **for** $i := 1$ **to** m **do**
3: $\quad \bar{c}_i := \max(0, \sum_{j \in N} g_j(x_{ij}) - c_i)$;
4: $\quad$ **for** $j := n$ **back to** 1 **do**
5: $\quad\quad x_{ij} := \max(0, g_j^{-1}(g_j(x_{ij}) - \bar{c}_i))$;
6: $\quad\quad$ **if** $j \in \overline{N}$ **then** $x_{ij} := \lfloor x_{ij} \rfloor$;
7: $\quad\quad \bar{c}_i := \max(0, \sum_{j \in N} g_j(x_{ij}) - c_i)$;
8: $\quad\quad$ **if** $\bar{c}_i \leq 0$ **then** break;
9: $\quad$ **end for**
10: **end for**

In this case $\bar{c}_i$ represents the amount of weight that exceeds the capacity of knapsack i. Note that the solution of the surrogate relaxation x^*_{ij} already satisfies constraints (3). Thus, as we are only decreasing the value of x_{ij}, such constraints remain satisfied.

The main steps 5-8 are executed mn times. With the usual assumption on the computation of the inverse of a weight function, the time complexity of Algorithm 4 is thus $O(mn)$.

4 Local search

The solutions produced by the heuristics of the previous sections can be improved through a local search algorithm that implements pairwise exchanges of the amounts of items assigned to the same knapsack i.

Let i be the current knapsack. Consider two items j and k, define a "small" incremental step $\varepsilon < \min(\delta_j, \delta_k)$ (see Section 3.1) and consider the effect of two potential variations, resulting from a simultaneous increase (resp. decrease) of x_{ij} and decrease (resp. increase) of x_{ik} by ε units, namely:

1. $\Delta^1 = (f_j(x_{ij} + \varepsilon) - f_j(x_{ij})) + (f_k(x_{ik} - \varepsilon) - f_k(x_{ik}))$;
2. $\Delta^2 = (f_j(x_{ij} - \varepsilon) - f_j(x_{ij})) + (f_k(x_{ik} + \varepsilon) - f_k(x_{ik}))$.

Further impose that a Δ^ℓ ($\ell = 1, 2$) takes the value 0 if the corresponding variation is infeasible, i.e., if either a right-hand side (u_j, u_k, or c_i) of inequalities (2)-(3) is exceeded or one of the two variables takes a negative value. Let $\Delta = \max(\Delta^1, \Delta^2)$:

- if $\Delta > 0$ the procedure, shown in Algorithm 5,
  - (i) performs the corresponding variation, producing a new solution with objective function value increased by Δ;
  - (ii) iterates the process, for the same item pair and ε, obviously by only computing the Δ^ℓ ($\ell = 1$ or 2) that produced Δ;
- if instead $\Delta \leq 0$, i.e., both variations either worsen the solution value or are infeasible, the next item pair is tested, or the next knapsack is considered (when $j = n - 1$ and $k = n$).

Variants of this procedure (obtained by randomizing the choice of the knapsacks and/or of the item pairs) were also tested, but they did not provide satisfactory results.

Algorithm 5 Procedure Local Search.

```
for i := 1 to m do
    for j := 1 to n − 1 do
        for k := j + 1 to n do
            define an appropriate value ε < min(δ_j, δ_k);
            repeat
                compute Δ^1, Δ^2, and Δ;
                if Δ > 0 then apply the variation corresponding to Δ;
            until Δ > 0
        end for
    end for
end for
```

The inner **repeat-until** loop is executed $O(mn^2)$ times. This loop can theoretically require a pseudopolynomial time but, in practice, it is executed a limited number of times and in any case the number of iterations can be limited by a constant. Over the 3,360 instances of the benchmark we tested (see Section 6), the number of iterations was normally between 1 and 2, and it never attained 10. By assuming again that the computation of the inverse of a function can be done in constant time, the procedure runs in practice in $O(mn^2)$ time.

5 Overall algorithm

Our overall heuristic algorithm for the MNLKP can be outlined as follows:

Algorithm 6 Algorithm DMM.

1: solve the surrogate relaxation (6)-(10) and let U and x^* be respectively the resulting upper bound and the corresponding solution;
2: execute procedure Constructive of Section 3.1 and let Z^h be the solution value;
3: **if** $Z^h = U$ **then** terminate;
4: execute procedures Surrogate-feas-1(x^*) and Surrogate-feas-2(x^*) of Section 3.2 and let Z^s be the best solution value;
5: **if** $Z^s = U$ **then** terminate **else** $Z := \max(Z^h, Z^s)$;
6: execute Local Search of Section 4 on the solution corresponding to Z and let Z^l be the solution value;
7: **return** the solution corresponding to Z^l.

6 Computational experiments

The algorithm of Section 5 was experimentally compared with open-source non-linear programming local solvers (Ipopt [9] for real instances and Bonmin [2], with option `bonmin.algorithm B-BB`, for integer instances) and with a global solver (Couenne [4], both for real and integer instances). Note that Ipopt and Bonmin are exact solvers for convex non-linear programs and convex mixed-integer non-linear programs, respectively, but they can be used as heuristics for non-convex problems like ours.

We produced a benchmark using the profit and weight functions adopted in D'Ambrosio and Martello [5]. The profits were always obtained from

$$f_j(x_{ij}) = \frac{c_j}{1 + b_j \mathrm{e}^{-a_j(x_{ij}+d_j)}} \tag{18}$$

by uniformly randomly generating a_j in $[0.1, 0.2]$, b_j and c_j in $[0, 100]$, and d_j in $[-100, 0]$. The upper bounds on the x_{ij} values were all set to $u_j := 100$. Observe that the random generation of the four parameters produces, for every test instance, a different (possibly very different) shape of the objective function. For example,

- $a = 0.2$, $b = 50$, $c = 50$, and $d = -100$ produce the convex function shown in Figure 2(a);
- $a = 0.2$, $b = 1$, $c = 50$, and $d = 0$ produce the concave function shown in Figure 2(b),
- $a = 0.1$, $b = 50$, $c = 80$, and $d = -10$ produce the non-concave, non-convex function shown in Figure 1.

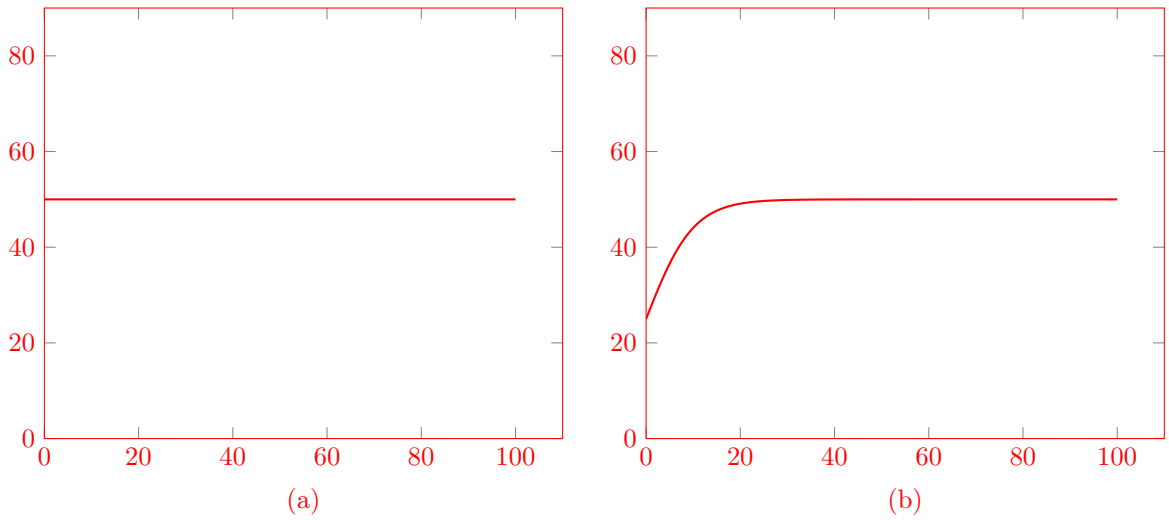

Figure 2: Different shapes produced by objective function (18).

Two test beds (*Non-linear weights* and *Linear weights*) were then obtained by generating the weights either according to the concave increasing function

$$g_j(x_{ij}) = \sqrt{p_j x_{ij} + q_j} - \sqrt{q_j}, \tag{19}$$

with p_j and q_j uniformly random in $[1, 20]$, or according to the linear increasing function

$$g_j(x_{ij}) = w_j x_{ij}, \tag{20}$$

with w_j uniformly random in $[1, 100]$.

For each test bed, two sets of instances were obtained by generating the capacities following the method adopted in Chapter 6 of Martello and Toth [12] to produce difficult instances: *Similar capacities*

$$c_i \text{ uniformly random in } \left[0.4 \sum_{j=1}^{n} \frac{g_j(u_j)}{m}, 0.6 \sum_{j=1}^{n} \frac{g_j(u_j)}{m}\right] \quad (i = 1, \ldots, m-1), \tag{21}$$

and *Dissimilar capacities*

$$c_i \text{ uniformly random in } \left[0, \left(0.5 \sum_{j=1}^{n} g_j(u_j) - \sum_{k=1}^{i-1} c_k\right)\right] \quad (i = 1, \ldots, m-1). \tag{22}$$

In both cases, the m-th capacity was set to:

$$c_m = 0.5 \sum_{j=1}^{n} g_j(u_j) - \sum_{i=1}^{m-1} c_i. \tag{23}$$

The experiments were performed for values of n in $\{10, 20, 50, 100, 200, 500, 1000\}$ and of m in $\{2, 5, 10\}$. For each pair (n, m), 20 real instances and 20 integer instances were produced. The total number of tested instances was thus 3 360. All instances, and the corresponding solutions are available at `http://or.dei.unibo.it/library/multiple-non-linear-knapsack-problem`.

All the experiments were performed on an Intel Core 2, CPU 6600, 2.4 GHz, 1.94 GB ram, using only one processor.

All algorithms (Ipopt, Bonmin, Couenne, and DMM) were run with a time limit of one CPU hour per instance. Couenne was executed with its default values (with an exception mentioned at the end of the present section) as the use of other options strongly increases its computing times. In addition, it was only executed for instances with $n \leq 50$, due to its poor performance for larger values. For DMM, the solution of the surrogate relaxation (6)-(10) was obtained by running Couenne with a time limit of $n/10$ seconds. If no feasible solution was found within the time limit, the surrogate heuristics were not executed. Moreover, each local search (Step 6) had a time limit of 5 CPU seconds assigned. In a non-convex environment, Ipopt and Bonmin produce different solutions depending on the starting point: we executed them both with one and ten random starting points (taking the best solution). We also tested Bonmin with ten random starting points at each branch-and-bound decision node, but this only resulted in few improvements for the small instances, so we do not report the corresponding computational results. Another not reported additional round of tests was executed with Scip [14], obtaining results significantly worse than the ones obtained by Couenne.

Procedure Construct(i) (within Constructive) was executed with four values of s (1, 10, 50, 100), and the best solution was selected. The refined search for $\bar{\mu}_j$ (see Section 3.1) was obtained: (i) by trying up to 5 consecutive refinement rounds, each time dividing the current sampling step by 2; (ii) by trying a single refinement round twice (dividing the initial sampling step by 5 and 10, respectively), and (iii) taking the best solution. The value of ε was set to $\min(\delta_j, \delta_k)/2$. All computations of the zero of a weight function needed by DMM were performed through a binary search over the definition range. (The impact on the overall CPU time was however negligible.)

Tables 1-4 report the results for non-linear weights, with real and integer variables x_{ij}. The entries give: number of knapsacks, number of items, and:

- for real variables, average values produced by DMM, Ipopt with a single starting point (Ipopt_1), Ipopt with 10 starting points (Ipopt_10), and Couenne;
- for integer variables, average values produced by DMM, Bonmin with a single starting point (Bonmin_1), Bonmin with 10 starting points at the root node (Bonmin_10), and Couenne.

Additional lines give the total values for each number of knapsacks and the overall total values. The tables with odd numbering provide the average solution values over the 20 generated instances, while those with even numbering provide the corresponding average CPU times (in seconds). In a number of cases the solvers were not able to produce a solution within the time limit: for such cases, the tables report the number of non-solved instances. The average solution values and CPU times were only computed over the instances for which a solution was produced.

Tables 5-8 report the same information for the case of linear weights.

Tables 1 and 3 clearly show that, on the non-linear instances, the proposed algorithm DMM almost always outperforms both the exact and the heuristic solvers, with few exceptions only occurring for the

Table 1: Non-linear weights, similar capacities. Average solution values over 20 instances (# no solution).

		Real Variables				Integer Variables			
m	n	DMM	Ipopt_1	Ipopt_10	Couenne	DMM	Bonmin_1	Bonmin_10	Couenne
2	10	350.49	313.50	351.70	362.63	350.47	327.21	313.87	364.45
2	20	641.73	569.20	635.35	593.43(12)	645.63	591.26	589.88	561.83
2	50	1,900.41	1,714.74	1,799.78	n/a(20)	1,902.65	1,769.76	1,800.63	1,481.36
2	100	3,741.92	3,341.04	3,548.40	–	3,742.14	3,413.98	3,454.54	–
2	200	7,167.80	6,302.04	6,683.39	–	7,168.07	6,429.80(5)	6,419.05	–
2	500	18,375.35	16,153.10	16,942.30	–	18,376.30	16,286.80(13)	16,404.90(5)	–
2	1000	37,051.16	32,117.80	33,899.30	–	37,058.30	n/a(20)	n/a(20)	–
2	total	69,228.86	60,511.42	63,860.22	–	69,243.56	–	–	–
5	10	322.76	271.61	312.63	299.06(5)	321.18	288.19	281.05	299.35
5	20	729.46	687.95	733.25	n/a(20)	727.24	706.88	716.23	620.54(2)
5	50	1,822.58	1,705.86	1,782.99	n/a(20)	1,821.57	1,753.76(2)	1,765.72	1,245.52(1)
5	100	3,865.64	3,701.10	3,818.14	–	3,868.62	3,719.86(1)	3,825.23	–
5	200	7,846.03	7,372.56	7,681.43	–	7,850.18	n/a(20)	7,677.54(16)	–
5	500	19,272.60	18,180.80	18,825.70	–	19,273.73	n/a(20)	n/a(20)	–
5	1000	38,540.16	36,391.00	37,485.80	–	38,541.14	n/a(20)	n/a(20)	–
5	total	72,399.23	68,310.88	70,639.94	–	72,403.66	–	–	–
10	10	216.19	187.40	218.59	212.07(2)	232.58	183.37(2)	201.60(1)	222.29
10	20	734.75	664.58	702.21	n/a(20)	727.85	719.17(3)	681.76(2)	489.35(5)
10	50	1,983.10	1,864.39	1,927.13	n/a(20)	1,983.72	1,834.39(16)	1,937.89	1,275.43(16)
10	100	3,952.78	3,732.51	3,843.88	–	3,957.84	n/a(20)	3,852.28(19)	–
10	200	7,652.05	7,311.37	7,431.36	–	7,655.04	n/a(20)	n/a(20)	–
10	500	19,640.75	18,788.40	19,097.90	–	19,645.34	n/a(20)	n/a(20)	–
10	1000	39,717.69	36,316.50	38,266.70	–	39,721.93	n/a(20)	n/a(20)	–
10	total	73,897.31	68,865.15	71,487.77	–	73,924.30	–	–	–
total	total	215,525.40	197,687.45	205,987.93	–	215,571.52	–	–	–

Table 2: Non-linear weights, similar capacities. Average CPU times over 20 instances (# no solution).

		Real Variables				Integer Variables			
m	n	DMM	Ipopt_1	Ipopt_10	Couenne	DMM	Bonmin_1	Bonmin_10	Couenne
2	10	1.08	0.05	0.51	1,113.88	1.09	3.71	1.47	847.17
2	20	2.02	0.10	1.18	3,601.81(12)	2.03	15.58	4.53	3,601.20
2	50	5.08	0.34	3.45	n/a(20)	5.11	637.51	78.25	3,603.19
2	100	10.23	0.93	10.24	–	10.26	2,363.94	978.34	–
2	200	20.88	2.44	28.34	–	20.72	3,089.98(5)	2,693.71	–
2	500	57.45	10.60	112.08	–	54.14	3,600.48(13)	3,496.83(5)	–
2	1000	113.54	32.84	327.93	–	116.11	n/a(20)	n/a(20)	–
2	total	210.28	47.30	483.73	–	209.46	–	–	–
5	10	1.20	0.16	1.72	3,600.61(5)	1.24	60.58	8.34	3,600.88
5	20	2.03	0.32	3.57	n/a(20)	2.05	601.01	185.12	3,602.25(2)
5	50	5.15	1.17	11.76	n/a(20)	5.18	2,765.24(2)	996.16	3,602.47(1)
5	100	10.25	3.53	31.76	–	10.41	3,071.08(3)	3,419.49	–
5	200	20.79	9.72	87.03	–	21.35	n/a(20)	3,600.40(16)	–
5	500	54.26	51.12	403.15	–	57.29	n/a(20)	n/a(20)	–
5	1000	125.86	151.13	1,207.48	–	126.70	n/a(20)	n/a(20)	–
5	total	219.54	217.15	1,746.47	–	224.22	–	–	–
10	10	1.49	0.39	3.78	3,600.54(2)	1.54	827.80(2)	181.15(1)	3,600.17
10	20	2.14	0.93	8.57	n/a(20)	2.17	2,570.88(3)	1059.40(2)	3,602.81(5)
10	50	5.14	3.35	30.70	n/a(20)	5.31	3,601.42(16)	3,464.19	3,665.76(16)
10	100	10.37	10.38	86.37	–	10.94	n/a(20)	3,600.19(19)	–
10	200	23.51	32.98	278.67	–	23.43	n/a(20)	n/a(20)	–
10	500	87.11	147.12	1,197.91	–	86.49	n/a(20)	n/a(20)	–
10	1000	375.57	348.52	3,108.31	–	371.94	n/a(20)	n/a(20)	–
10	total	505.33	543.67	4,714.31	–	501.82	–	–	–
total	total	935.15	808.12	6,944.51	–	935.50	–	–	–

Table 3: Non-linear weights, dissimilar capacities. Average solution values over 20 instances (# no solution).

		Real Variables				Integer Variables			
m	n	DMM	Ipopt_1	Ipopt_10	Couenne	DMM	Bonmin_1	Bonmin_10	Couenne
2	10	340.62	298.24	339.50	355.19	339.89	299.21	308.60	354.62
2	20	634.68	547.68	610.20	564.65(9)	636.54	569.67	573.25	555.90
2	50	1,870.83	1,619.80	1,737.00	n/a(20)	1,871.45	1,688.48	1,653.26	1,362.62
2	100	3,722.84	3,236.18	3,437.91	–	3,722.43	3,340.35(1)	3,294.41	–
2	200	7,074.48	5,908.90	6,313.01	–	7,072.53	6,188.82(3)	5,905.41	–
2	500	18,280.25	15,286.70	16,081.20	–	18,287.28	15,883.50(14)	15,862.40(5)	–
2	1000	37,089.65	31,168.30	32,825.20	–	37,086.84	n/a(20)	n/a(20)	–
2	total	69,013.35	58,065.80	61,344.02	–	69,016.96	–	–	–
5	10	321.28	285.28	318.68	329.51(1)	313.70	305.05(1)	299.28(1)	319.22(1)
5	20	728.42	670.57	717.91	659.84(17)	720.97	692.01(2)	664.84	598.87(5)
5	50	1,792.76	1,614.39	1,723.24	n/a(20)	1,772.33	1,600.30(4)	1,627.85	1,228.18(4)
5	100	3,810.80	3,392.78	3,625.14	–	3,805.91	3,406.69(2)	3,400.23	–
5	200	7,647.44	6,778.96	7,180.81	–	7,595.80	n/a(20)	n/a(20)	–
5	500	19,108.46	17,131.00	17,967.30	–	19,100.00	n/a(20)	n/a(20)	–
5	1000	38,447.52	34,559.20	36,099.60	–	38,472.11	n/a(20)	n/a(20)	–
5	total	71,856.68	64,432.18	67,632.68	–	71,780.82	–	–	–
10	10	343.95	314.88	345.18	354.52	321.47	315.20(1)	307.11	340.21(1)
10	20	757.28	678.59	741.04	699.31(11)	737.34	703.57(3)	681.37(1)	633.95(8)
10	50	1,972.11	1,807.00	1,909.65	n/a(20)	1,883.62	1,817.64(9)	1,810.51(1)	1,499.67(7)
10	100	3,873.07	3,441.87	3,684.99	–	3,615.12	n/a(20)	3,804.82(19)	–
10	200	7,543.03	6,708.00	7,094.40	–	7,395.62	n/a(20)	n/a(20)	–
10	500	19,271.73	17,027.60	17,950.80	–	19,242.96	n/a(20)	n/a(20)	–
10	1000	39,486.36	35,248.30	37,237.50	–	39,403.01	n/a(20)	n/a(20)	–
10	total	73,247.53	65,226.24	68,963.56	–	72,599.14	–	–	–
total	total	214,117.56	187,724.22	197,940.26	–	213,396.92	–	–	–

Table 4: Non-linear weights, dissimilar capacities. Average CPU times over 20 instances (# no solution).

		Real Variables				Integer Variables			
m	n	DMM	Ipopt_1	Ipopt_10	Couenne	DMM	Bonmin_1	Bonmin_10	Couenne
2	10	1.08	0.05	0.53	794.39	1.09	2.91	1.11	364.44
2	20	2.02	0.11	1.14	3,307.95(9)	2.03	20.41	8.18	3,430.29
2	50	5.08	0.34	3.36	n/a(20)	5.30	277.57	69.66	3,602.78
2	100	10.23	0.89	9.96	–	10.26	1,731.33(1)	813.99	–
2	200	20.87	2.40	26.51	–	20.71	3,254.23(3)	3,099.98	–
2	500	57.34	10.64	105.76	–	54.07	3,265.7(14)	3,456.18(5)	–
2	1000	113.39	33.71	318.10	–	115.93	n/a(20)	n/a(20)	–
2	total	210.01	48.14	465.36	–	209.39	–	–	–
5	10	1.20	0.14	1.53	2,790.28(1)	1.23	29.11(1)	196.66(1)	2,013.34(1)
5	20	2.03	0.34	3.23	3,600.18(17)	2.05	250.53(2)	588.38	3,600.92(5)
5	50	5.16	1.16	10.25	n/a(20)	5.17	2,716.96(4)	773.89	3,604.77(4)
5	100	10.25	3.04	26.45	–	10.41	3,373.18(2)	2,341.63	–
5	200	20.77	8.70	74.45	–	21.25	n/a(20)	n/a(20)	–
5	500	54.14	45.04	356.04	–	57.02	n/a(20)	n/a(20)	–
5	1000	124.84	130.89	1,044.09	–	126.05	n/a(20)	n/a(20)	–
5	total	218.39	189.31	1,516.04	–	223.18	–	–	–
10	10	1.49	0.31	2.62	2,266.36	1.53	33.25(1)	53.09	1,753.20(1)
10	20	2.16	0.79	6.10	3,599.61(11)	2.16	650.11(3)	631.44(1)	3,600.00(8)
10	50	5.14	2.89	23.01	n/a(20)	5.28	3,177.66(9)	2,326.35(1)	3,597.48(7)
10	100	10.35	7.98	58.18	–	10.84	n/a(20)	3,600.9(19)	–
10	200	23.44	25.93	184.13	–	23.22	n/a(20)	n/a(20)	–
10	500	86.92	113.32	704.57	–	86.07	n/a(20)	n/a(20)	–
10	1000	374.20	328.10	2,268.40	–	370.22	n/a(20)	n/a(20)	–
10	total	503.70	479.32	3,247.01	–	499.32	–	–	–
total	total	932.10	716.77	5,228.41	–	931.89	–	–	–

Table 5: Linear weights, similar capacities. Average solution values over 20 instances (# no solution).

		Real Variables				Integer Variables			
m	n	DMM	Ipopt_1	Ipopt_10	Couenne	DMM	Bonmin_1	Bonmin_10	Couenne
2	10	343.63	318.66	350.62	343.65(6)	343.95	332.91	351.35	335.53
2	20	780.54	739.70	786.05	n/a(20)	780.03	752.42	784.46	676.70
2	50	1,943.44	1,841.93	1,921.17	n/a(20)	1,941.38	1,896.66	1,939.18	1,494.77(10)
2	100	3,873.34	3,585.02	3,727.55	–	3,872.29	3,681.92	3,740.08	–
2	200	7,966.74	7,401.99	7,645.14	–	7,966.32	7,510.38(4)	7,574.34(1)	–
2	500	20,111.10	18,685.00	19,222.20	–	20,107.13	18,438.10(18)	18,438.10(18)	–
2	1000	39,562.06	36,752.90	37,448.70	–	39,563.97	n/a(20)	n/a(20)	–
2	total	74,580.85	69,325.20	71,101.43	–	74,575.07	–	–	–
5	10	374.58	364.54	392.81	243.95(1)	375.28	366.75	386.39	286.99(2)
5	20	762.51	739.22	782.22	n/a(20)	761.99	750.92	768.77	485.02(4)
5	50	2,010.44	1,990.01	2,038.93	n/a(20)	2,013.13	1,991.96	1,997.33	1,366.98(11)
5	100	3,942.02	3,808.03	3,955.77	–	3,942.90	3,978.83(15)	3,900.36(14)	–
5	200	8,135.86	7,927.23	8,125.15	–	8,141.77	n/a(20)	n/a(20)	–
5	500	20,883.19	20,123.60	20,424.60	–	20,900.04	n/a(20)	n/a(20)	–
5	1000	41,748.25	39,406.90	40,231.00	–	41,755.37	n/a(20)	n/a(20)	–
5	total	77,856.85	74,359.53	75,950.48	–	77,890.48	–	–	–
10	10	240.61	224.04	243.62	211.05(15)	235.48	229.52	238.99	170.29(4)
10	20	790.00	778.73	816.61	n/a(20)	787.61	804.24(1)	807.22	329.71(4)
10	50	2,095.67	2,042.56	2,122.72	1,572.96(18)	2,097.90	2,249.09(19)	2,249.09(19)	486.79(6)
10	100	4,040.29	3,860.72	4,008.91	–	4,043.65	n/a(20)	n/a(20)	–
10	200	8,376.55	8,063.39	8,224.59	–	8,382.21	n/a(20)	n/a(20)	–
10	500	21,309.28	20,091.20	20,535.40	–	21,321.39	n/a(20)	n/a(20)	–
10	1000	42,766.36	36,853.00	39,276.10	–	42,782.19	n/a(20)	n/a(20)	–
10	total	79,618.76	71,913.64	75,227.95	–	79,650.43	–	–	–
total	total	232,056.46	215,598.37	222,279.86	–	232,115.98	–	–	–

Table 6: Linear weights, similar capacities. Average CPU times over 20 instances (# no solution).

		Real Variables				Integer Variables			
m	n	DMM	Ipopt_1	Ipopt_10	Couenne	DMM	Bonmin_1	Bonmin_10	Couenne
2	10	1.06	0.04	0.56	2,250.10(6)	1.07	3.67	2.91	3,592.36
2	20	2.02	0.13	1.40	n/a(20)	2.02	16.76	12.94	3,602.05
2	50	5.06	0.50	6.02	n/a(20)	5.08	145.37	195.66	3,602.23(10)
2	100	10.21	1.69	17.64	–	10.22	1,499.39	1,312.30	–
2	200	20.71	5.90	58.31	–	20.81	2,978.50(4)	2,666.15(1)	–
2	500	56.59	26.18	273.87	–	58.50	1,4671.80(18)	11,718.9(18)	–
2	1000	115.69	82.53	857.74	–	118.58	n/a(20)	n/a(20)	–
2	total	211.34	116.97	1,215.54	–	216.28	–	–	–
5	10	1.11	0.13	1.46	3,600.77(1)	1.13	350.92	330.21	3,601.28(2)
5	20	2.03	0.36	3.78	n/a(20)	2.04	550.75	675.76	3,601.41(4)
5	50	5.10	1.41	14.74	n/a(20)	5.14	3,110.13	3,182.53	3,608.20(11)
5	100	10.34	5.72	47.38	–	10.32	3,585.95(15)	3,292.52(14)	–
5	200	20.85	15.15	145.10	–	21.18	n/a(20)	n/a(20)	–
5	500	55.02	82.17	834.01	–	57.23	n/a(20)	n/a(20)	–
5	1000	119.42	229.58	2,357.48	–	128.10	n/a(20)	n/a(20)	–
5	total	213.87	334.52	3,403.95	–	225.14	–	–	–
10	10	1.21	0.34	3.36	3,602.44(15)	1.27	1,088.11	934.47	3,601.56(4)
10	20	2.04	0.81	8.00	n/a(20)	2.06	3,130.17(1)	3,097.31	3,602.08(4)
10	50	5.11	3.46	35.50	3,596.06(18)	5.18	3,583.36(19)	3,417.98(19)	3,610.75(6)
10	100	10.31	13.30	148.41	–	10.57	n/a(20)	n/a(20)	–
10	200	21.09	46.59	497.50	–	22.18	n/a(20)	n/a(20)	–
10	500	56.17	195.58	1,999.56	–	63.64	n/a(20)	n/a(20)	–
10	1000	170.69	398.56	4,219.18	–	163.09	n/a(20)	n/a(20)	–
10	total	266.62	658.64	6,911.51	–	267.99	–	–	–
total	total	691.83	1,110.13	11,531.00	–	709.41	–	–	–

Table 7: Linear weights, dissimilar capacities. Average solution values over 20 instances (# no solution).

		Real Variables				Integer Variables			
m	n	DMM	Ipopt_1	Ipopt_10	Couenne	DMM	Bonmin_1	Bonmin_10	Couenne
2	10	333.39	296.56	342.03	342.20(3)	331.75	312.05	339.32	336.78
2	20	760.08	696.13	758.70	674.85(16)	757.24	705.37	763.36	652.22(1)
2	50	1,916.44	1,789.24	1,854.16	n/a(20)	1,920.41	1,866.97	1,887.10	1,407.08(5)
2	100	3,859.95	3,451.60	3,625.93	–	3,864.18	3,614.56	3,696.08	–
2	200	7,856.57	7,044.65	7,263.35	–	7,854.79	7,209.46(2)	7,348.76(1)	–
2	500	19,806.13	17,817.50	18,240.70	–	19,870.13	19,164.10(18)	18,296.70(14)	–
2	1000	39,193.30	35,220.40	35,900.90	–	39,183.30	31,516.00(19)	n/a(20)	–
2	total	73,725.86	66,316.08	67,985.77	–	73,781.80	–	–	–
5	10	394.60	371.40	404.20	434.82(10)	394.12	379.34	401.04	378.30(3)
5	20	749.42	699.44	757.70	600.46(18)	746.29	708.66	739.91	481.27(6)
5	50	1,985.20	1,907.99	1,965.24	n/a(20)	1,986.83	1,953.95	1,978.85	1,030.52(8)
5	100	3,906.82	3,657.64	3,776.51	–	3,899.92	3,657.99(8)	3,733.00(9)	–
5	200	7,991.03	7,535.88	7,739.56	–	8,000.10	6,768.74(19)	7,021.51(18)	–
5	500	20,598.05	19,076.70	19,540.10	–	20,597.93	n/a(20)	n/a(20)	–
5	1000	41,112.47	37,430.50	38,623.60	–	41,144.61	n/a(20)	n/a(20)	–
5	total	76,737.59	70,679.55	72,806.91	–	76,769.80	–	–	–
10	10	308.99	279.20	310.72	313.65(11)	307.06	283.12	304.86	273.38(1)
10	20	816.72	814.43	854.19	n/a(20)	817.87	824.66(1)	844.13	647.17(8)
10	50	2,060.70	1,951.85	2,040.01	n/a(20)	2,064.86	1,865.52(6)	1,921.84(6)	1,117.49(6)
10	100	3,957.53	3,640.15	3,793.31	–	3,956.12	3,426.41(13)	3,711.03(16)	–
10	200	8,287.68	7,549.40	7,859.10	–	8,299.41	n/a(20)	n/a(20)	–
10	500	20,984.56	18,858.90	19,445.60	–	20,898.03	n/a(20)	n/a(20)	–
10	1000	41,989.27	37,114.20	38,332.10	–	41,963.22	n/a(20)	n/a(20)	–
10	total	78,405.45	70,208.13	72,635.03	–	78,306.57	–	–	–
total	total	228,868.90	207,203.76	213,427.71	–	228,858.17	–	–	–

Table 8: Linear weights, dissimilar capacities. Average CPU times over 20 instances (# no solution).

		Real Variables				Integer Variables			
m	n	DMM	Ipopt_1	Ipopt_10	Couenne	DMM	Bonmin_1	Bonmin_10	Couenne
2	10	1.06	0.04	0.58	1613.09(3)	1.07	2.24	1.85	2,017.11
2	20	2.02	0.12	1.46	3,602.01(16)	2.02	9.99	10.64	3,601.04(1)
2	50	5.06	0.48	5.78	n/a(20)	5.08	530.30	137.54	3,602.40(5)
2	100	10.21	1.55	16.23	–	10.21	1,699.74	1,571.01	–
2	200	20.69	5.13	52.37	–	20.77	2,655.79(2)	2,601.08(1)	–
2	500	56.61	22.90	237.33	–	58.30	1,493.78(18)	4,318.67(14)	–
2	1000	115.26	70.27	741.25	–	118.16	50,892.30(19)	n/a(20)	–
2	total	210.91	100.49	1,055.00	–	215.61	–	–	–
5	10	1.10	0.14	1.41	3,269.85(10)	1.13	316.61	475.27	3,600.87(3)
5	20	2.02	0.34	3.59	3,601.70(18)	2.04	477.62	793.09	3,601.03(6)
5	50	5.10	1.40	14.21	n/a(20)	5.13	2,356.92	2,872.45	3,602.52(8)
5	100	10.41	4.03	40.96	–	10.30	3,240.82(8)	2,624.47(9)	–
5	200	20.80	11.91	112.03	–	21.12	3,589.85(19)	3,411.551(18)	–
5	500	54.68	58.92	554.51	–	56.87	n/a(20)	n/a(20)	–
5	1000	118.28	192.46	1,736.15	–	126.88	n/a(20)	n/a(20)	–
5	total	212.39	269.20	2,462.86	–	223.47	–	–	–
10	10	1.24	0.28	2.90	2,546.81(11)	1.27	768.94	763.07	3,085.93(1)
10	20	2.04	0.71	7.21	n/a(20)	2.06	1,693.68(1)	2,091.58	3,601.56(8)
10	50	5.10	2.55	28.40	n/a(20)	5.17	2,634.70(6)	2,361.36(6)	3,601.53(6)
10	100	10.27	8.98	92.64	–	10.53	3,211.55(13)	2,752.26(16)	–
10	200	20.94	26.05	295.69	–	22.04	n/a(20)	n/a(20)	–
10	500	55.14	139.96	1,351.50	–	62.25	n/a(20)	n/a(20)	–
10	1000	172.99	350.02	3,629.18	–	159.23	n/a(20)	n/a(20)	–
10	total	267.72	528.55	5,407.52	–	262.55	–	–	–
total	total	691.02	898.24	8,925.38	–	701.63	–	–	–

smaller instances with $n = 10$ (and a single case for $n = 20$). Coming to the CPU times, Tables 2 and 4 show that on the integer instances DMM is always the fastest method. For the real instances Ipopt_1 is generally faster, but on the other hand the solution values it provides are definitely worse (by over 10% on average). Overall, DMM appears to be an excellent trade-off between effectiveness and quality of the provided solution.

Similar considerations apply to the results on the linear instances. Ipopt_10 and Bonmin_10 often provide better solutions for smaller instances (see Tables 5 and 7), while DMM always performs better for $n \geq 200$. Concerning the average CPU times (Tables 6 and 8), DMM is again the clear winner on the integer instances. For the real instances, Ipopt_1 is faster for $m = 2$ (but at the expenses of worse solution values), while it is always outperformed by DMM for $m \geq 5$ (especially on the larger instances).

It turns out that the instances with linear weights are more difficult to solve to optimality than those with non-linear weights. Although this can appear surprising, there is no theoretical result implying that one case must be easier than the other. Couenne, for example, transforms the objective function so as it becomes linear, while its non-linear terms become additional constraints. It follows that, in any case, the feasible region is defined by non-linear constraints, and the evolution of the branching search is unpredictable.

Coming to the relative merits of the two approaches we presented, separate executions showed that the constructive heuristic has in general a better performance than the surrogate heuristic. However, the addition of the latter produces a number of benefits to the overall DMM algorithm:

- in a number of cases, especially for small-size instances, it finds a solution better than that of the constructive heuristic. For example **** citare casi favorevoli ****;
- *** se ricordo bene, in passato abbiamo fatto esperimenti SENZA il surrogate heuristic. Si possono recuperare? Sarebbero utili per poter dire (se vero):
  - di quanto aumenta il valore medio delle soluzioni trovate;
  - di quanto aumenta il numero di soluzioni ottime trovate;
  - altro?
- the CPU time it takes is comparatively limited (*** Ë vero?), amounting, on average, at *** % of the total CPU time;
- it produces an upper bound that can allow DMM to prematurely terminate with an optimal solution;
- *** altro?

Worth is also noting that all heuristic methods (DMM, Ipopt_1, and Ipopt_10) can produce a feasible solution for all the real instances (both for the case of linear and non-linear weights). Instead, for the integer instances DMM is the only method capable of finding a feasible solution for the larger instances.

We have seen that objective function (18) takes, according to the values of the four parameters, very different shapes. In order to further evaluate the robustness of the behavior of DMM, we computed the average ratio R = (upper bound produced by Couenne)/(solution value provide by DMM) on (18) and on totally different objective functions, obtaining (****tutto da controllare e specificare *****):

- for (18): $R = ****$ (computed over all *** generated instances ??? vero?);
- for the sinusoidal function $10\, a\, x_{ij}\, \sin \frac{x_{ij}}{50}$: $R = ***??$ (computed over ??? with a uniformly random in ???);
- for the convex quadratic function $10\, a^2\, x_{ij}^2 + b\, x_{ij}/5$: $R = ***??$ (computed over ??? with a uniformly random in ??? and b uniformly random in ???),

which confirms the good behavior of the proposed algorithm.

We conclude our experimental analysis by evaluating the quality of the solutions provided by DMM (for profit functions (18)) with respect to the global optimum. To this end, we only considered those instances for which Couenne could produce the optimal solution within the time limit. Table 9 refers to non-linear weights, Table 10 to linear weights. For each instance of a specific type (similar/dissimilar and real/integer) the tables give the best solution value provided by DMM and the value of the global optimum provided by Couenne. The very satisfactory performance of DMM is shown by the average error of 3.65%, with a minimum of 0 and a maximum of 18%. We observe that Couenne optimally solved more instances with non-linear weights (5.6%) than with linear weights (1.8%).

Table 9: Non-linear weights. Solution values for instances globally solved by Couenne.

			Similar Capacities				Dissimilar Capacities			
			Real Variables		Integer Variables		Real Variables		Integer Variables	
m	n	instance	DMM	Couenne	DMM	Couenne	DMM	Couenne	DMM	Couenne
2	10	0	223.20	230.69	205.37	228.65	211.03	230.62	211.98	228.33
		1	462.02	470.98	457.49	471.87	–	–	457.04	471.50
		2	338.81	349.23	–	–	340.36	349.29	339.04	349.21
		3	373.14	381.21	368.56	379.98	367.99	382.72	367.83	382.82
		4	309.95	317.61	–	–	309.04	316.53	312.20	316.17
		5	553.69	584.99	553.92	583.28	553.66	587.48	554.02	585.63
		6	–	–	286.42	303.57	250.55	264.43	250.55	263.71
		7	286.71	304.76	285.50	305.06	217.68	221.78	217.83	221.64
		8	–	–	443.08	455.53	–	–	438.36	455.49
		9	–	–	440.18	454.90	–	–	407.35	455.29
		10	314.74	330.25	314.74	326.33	327.94	330.40	322.26	326.86
		11	355.74	396.79	353.70	395.00	374.36	395.14	369.83	393.58
		12	291.93	305.05	294.18	302.82	298.81	299.82	292.96	299.39
		13	328.83	353.59	328.73	352.30	329.20	349.62	329.08	349.50
		14	374.85	390.88	375.30	389.56	361.39	390.92	362.07	389.32
		15	–	–	440.29	440.79	409.60	428.26	411.79	422.74
		16	244.25	244.25	244.25	244.25	244.25	244.25	244.25	244.25
		17	388.22	401.22	391.21	401.16	350.11	352.39	351.32	352.11
		18	358.20	362.85	357.65	362.05	332.11	359.07	330.58	357.38
		19	218.75	227.50	217.83	227.50	227.50	227.50	227.50	227.50
2	20	17	–	–	–	–	525.58	530.82	524.70	530.70
5	10	0	–	–	–	–	–	–	221.56	227.47
		6	–	–	–	–	293.89	306.33	294.76	305.67
		7	–	–	–	–	–	–	293.04	303.33
		8	–	–	–	–	383.54	392.58	367.80	390.95
		10	–	–	–	–	–	–	226.83	232.73
		14	–	–	–	–	302.48	309.32	302.99	308.46
		16	–	–	–	–	238.84	255.36	239.23	254.70
		18	–	–	–	–	259.90	264.89	259.17	264.10
		19	–	–	–	–	476.79	486.37	464.06	483.98
10	10	1	–	–	–	–	449.44	453.39	448.38	452.76
		3	–	–	–	–	261.67	267.20	260.86	266.68
		4	–	–	–	–	–	–	327.73	399.69
		6	–	–	–	–	469.96	486.00	476.02	483.23
		7	–	–	–	–	263.76	266.95	263.76	265.78
		9	–	–	–	–	232.63	234.31	233.57	233.59
		12	–	–	–	–	176.41	178.04	146.23	162.55
		13	–	–	–	–	288.57	288.79	287.89	288.67
		15	–	–	–	–	267.69	274.38	259.68	274.01
		16	–	–	–	–	431.69	437.85	432.11	437.98

We finally mention that, for 10 of the instances globally solved by Couenne, it turned out that the software behaved incorrectly, as our heuristics found a better solution than the one it had obtained. We therefore contacted the developer of Couenne, who suggested to disable some of the default options, namely: aggressive_fbbt, optimality_bt, and redcost_bt. We ran the modified version for the 10 instances, and the results were finally correct. For example, the solution value of the instance 6 with $m = 5$, $n = 10$, dissimilar capacities, integer variables and non-linear weights, produced by Couenne with the default options was 286.86, while our heuristic found a feasible solution of value 294.76. When re-run with the suggested options, the solution value was 305.67.

Table 10: Linear weights. Solution values for instances globally solved by Couenne.

			Similar Capacities				Dissimilar Capacities			
			Real Variables		Integer Variables		Real Variables		Integer Variables	
m	n	instance	DMM	Couenne	DMM	Couenne	DMM	Couenne	DMM	Couenne
2	10	1	–	–	–	–	213.78	217.12	213.41	217.03
		2	286.74	291.88	286.94	292.11	281.86	291.32	281.86	290.34
		3	269.21	271.67	–	–	262.49	271.61	250.59	271.29
		4	–	–	–	–	–	–	287.51	296.59
		7	–	–	–	–	246.83	250.45	247.66	250.22
		10	279.41	296.13	–	–	282.88	297.53	278.65	297.07
		12	–	–	–	–	462.93	484.80	463.05	484.84
		13	–	–	–	–	452.91	484.65	454.44	485.26
		15	351.95	354.69	–	–	332.08	349.80	332.26	349.28
		17	597.89	626.07	–	–	451.64	488.11	451.23	492.90
		18	–	–	–	–	481.08	494.04	–	–
		19	250.94	252.75	–	–	–	–	–	–
5	10	17	–	–	–	–	481.44	496.10	–	–
10	10	1	–	–	–	–	477.93	516.43	479.13	517.08
		11	–	–	–	–	–	–	363.21	365.08
		13	–	–	–	–	288.66	295.05	288.69	294.78
		19	–	–	–	–	195.25	206.44	–	–

7 Conclusions

We have considered a very difficult multiple non-linear knapsack problem, that non-linear solvers are unable to handle for instances of realistic size. We have studied a surrogate relaxation and we have investigated the quality of the lower bounds possibly achieved. We have presented a solution approach based on a fast constructive heuristic algorithm, two heuristics based on the feasibility recovery of the surrogate solution, and a local search post-optimization procedure. We have compared our approach with exact and heuristic solvers for general non-convex mixed integer non-linear programs. Extensive computational comparisons have shown that the proposed approach provides, especially for large-size instances, quick solutions of unexpected quality (i.e., within a few percent from the global optimum) with CPU times that are orders of magnitude smaller. The simplicity of the proposed algorithms makes them promising candidates for an extension to other nonlinear problems. We thank the three referees of this paper for useful comments.

Acknowledgements

This research was supported by MIUR-Italy (Grant PRIN 2015) and by Air Force Office of Scientific Research (under award number FA9550-17-1-0067). We thank Master student Angelo Di Zio for the implementation of the constructive heuristics [6, 7] and Pietro Belotti for the technical support on the use of Couenne [1, 4]. The last author acknowledges the financial support provided by MINO Initial Training Network (ITN) under the Marie Curie 7th European Framework Programme.

References

[1] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter. Branching and Bounds Tightening Techniques for Non–convex MINLP. *Optimization Methods & Software*, 24(4–5):567–634, 2009.

[2] Bonmin. URL `https://projects.coin-or.org/Bonmin`.

[3] K.M. Bretthauer and B. Shetty. The Nonlinear Knapsack Problem – Algorithms and Applications. *European Journal of Operational Research*, 138(3):459–472, 2002.

[4] Couenne. URL `https://projects.coin-or.org/Couenne`.

[5] C. D'Ambrosio and S. Martello. Heuristic Algorithms for the General Nonlinear Separable Knapsack Problem. *Computers & Operations Research*, 38(2):505–513, 2011.

[6] C. D'Ambrosio, A. Di Zio, S. Martello, and L. Mencarelli. A Heuristic Algorithm for the General Multiple Nonlinear Knapsack Problem. Technical report, DEI, University of Bologna, Italy and LIX, École Polytechnique, Palaiseau, France, 2015.

[7] A. Di Zio. Design and Development of Heuristic Algorithms for Multiple Nonlinear Knapsack Problems. Master's thesis, Department of Electronics, Computer Sciences and Systems (DEIS), University of Bologna, 2012.

[8] T. Ibaraki and N. Katoh. *Resource Allocation Problems*. Cambridge, MA: MIT Press, 1998.

[9] Ipopt. URL `https://projects.coin-or.org/Ipopt`.

[10] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Berlin, Germany: Springer, 2004.

[11] D. Li and X. Sun. *Nonlinear Integer Programming*, volume 84 of *International Series in Operations Research & Management Science*. Berlin, Germany: Springer, 2006.

[12] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Chichester, New York: John Wiley & Sons, 1990.

[13] L. Mencarelli, C. D'Ambrosio, A. Di Zio, and S. Martello. Heuristics for the General Multiple Non-linear Knapsack Problem. *Electronic Notes in Discrete Mathematics*, 55:59–62, 2016.

[14] Scip. URL `http://scip.zib.de`.

[15] B. Zhang and B. Chen. Heuristic and Exact Solution Method for Convex Nonlinear Knapsack Problem. *Asia-Pacific Journal of Operational Research*, 29(5):1250031, 2012. doi: 10.1142/S0217595912500315.

[16] B. Zhang and Z. Hua. A Unified Method for a Class of Convex Separable Nonlinear Knapsack Problems. *European Journal of Operational Research*, 191(1):1–6, 2008.