ARCHIVIO ISTITUZIONALE
DELLA RICERCA

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

Lagrangian matheuristics for the Quadratic Multiple Knapsack Problem

This is the final peer-reviewed author's accepted manuscript (postprint) of the following publication:

Published Version:
Galli L., Martello S., Rey C., Toth P. (2023). Lagrangian matheuristics for the Quadratic Multiple Knapsack Problem. DISCRETE APPLIED MATHEMATICS, 335, 36-51 [10.1016/j.dam.2022.06.033].

Availability:
This version is available at: https://hdl.handle.net/11585/983176 since: 2024-09-13

Published:
DOI: http://doi.org/10.1016/j.dam.2022.06.033

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/).
When citing, please refer to the published version.

(Article begins on next page)

07 November 2026

Lagrangian matheuristics for the Quadratic Multiple Knapsack Problem

Laura Galli* Silvano Martello † Carlos Rey † Paolo Toth†

Draft January 31, 2022

Abstract

The *Quadratic Multiple Knapsack Problem* (QMKP) is a challenging combinatorial optimization problem combining the well-known Quadratic Knapsack Problem with the Multiple Knapsack Problem. After a long stream of research devoted to metaheuristic approaches for large-scale instances, only recently some authors started to study the mathematical properties of the QMKP and proposed exact solution methods for benchmark instances of smaller size. In this paper, we propose the first matheuristic approach for solving the QMKP approximately. The proposed method exploits the strength of a Lagrangian relaxation for the natural quadratic formulation of the QMKP to derive heuristic solutions. Postoptimization local search procedures are embedded in the final framework. Experimental studies show that the resulting deterministic matheuristic approach consistently delivers solutions of very good quality in short computing times.

Keywords: Combinatorial optimization; Knapsack problems; Matheuristics; Lagrangian relaxation.

1 Introduction

In the *Quadratic Multiple Knapsack Problem* (QMKP), one is given n items, each having a *profit* p_i and a *weight* w_i $(i = 1, \ldots, n)$, m knapsacks, each having a *capacity* c_k $(k = 1, \ldots, m)$, and an additional $n \times n$ symmetric profit matrix $[p_{ij}]$. The QMKP calls for packing m disjoint subsets of items into the knapsacks so that the total weight in any knapsack does not exceed its capacity, with the objective of maximizing the overall profit given by: (i) the profit p_i of each selected item i, plus (ii) a profit p_{ij} for each pair of items (i, j) packed into the same knapsack. A natural mathematical formulation of the

*Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo 3, 56127 Pisa, Italy. E-mail: laura.galli@unipi.it

†DEI "Guglielmo Marconi", Università di Bologna, Viale Risorgimento 2, 40136 Bologna, Italy. E-mail: {silvano.martello,carlos.rey2,paolo.toth}@unibo.it

problem is the following binary quadratic program:

$$\max \quad \sum_{i=1}^{n}\sum_{k=1}^{m} p_i x_{ik} + \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\sum_{k=1}^{m} p_{ij} x_{ik} x_{jk} \tag{1}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_{ik} \leq c_k \qquad (k = 1, \ldots, m) \tag{2}$$

$$\sum_{k=1}^{m} x_{ik} \leq 1 \qquad (i = 1, \ldots, n) \tag{3}$$

$$x_{ik} \in \{0, 1\} \qquad (i = 1, \ldots, n;\ k = 1, \ldots, m), \tag{4}$$

where x_{ik} takes the value one iff item i is packed into knapsack k ($i = 1, \ldots, n$; $k = 1, \ldots, m$). Note that the objective function (1) is the sum of a linear term and a quadratic term expressing linear and pairwise profits, respectively. Inequalities (2) express the traditional knapsack capacity constraints, while inequalities (3) forbid an item to be packed into more that one knapsack. We assume throughout the paper that p_i, w_i, and c_k are positive integers ($i = 1, \ldots, n,\ k = 1, \ldots, m$), and that p_{ij} is a nonnegative integer with $p_{ij} = p_{ji}$ ($i, j = 1, \ldots, n$).

The QMKP comes from the combination of two intensively studied combinatorial optimization problems: the *Quadratic Knapsack Problem* (QKP) and the *Multiple Knapsack Problem* (MKP). Both the QKP and the MKP are special cases of the QMKP, arising when $m = 1$ (so constraints (3) become redundant) and when $p_{ij} = 0$ ($i, j = 1, \ldots, n$) (so the quadratic term in (1) vanishes), respectively. When both restrictions hold, the resulting problem is the famous *0-1 Knapsack Problem* (01KP). For the vast literature on QKP, MKP, and 01KP, the reader is referred to the classical monographs by Martello and Toth [15] and Kellerer et al. [14], as well as to the recent survey by Cacchiani et al. [3].

While the 01KP is weakly $\mathcal{NP}$-hard and can be solved in pseudo-polynomial time, both the QKP and the MKP are strongly $\mathcal{NP}$-hard, implying strong $\mathcal{NP}$-hardness for the QMKP as well. In addition to its theoretical intractability, the QMKP (which arises in a number of real-world situations like, e.g., project management, capital budgeting, product-distribution system design) is very difficult to solve in practice.

The QMKP was defined in 2006 by Hiley and Julstrom [12], who studied the case (commonly encountered in the literature) in which all capacities are equal. They proposed three heuristic algorithms for its approximate solution and a benchmark set of instances with $n \in \{100, 200\}$ and $m \in \{3, 5, 10\}$. In the following decade, a number of meta-heuristic approaches was computationally tested on these instances. We mention in particular those by Singh and Baghel [18] (genetic algorithm), Sundar and Singh [20] (artificial bee colony), Soak and Lee [19] (memetic algorithm), Garcia-Martinez et al. [10, 11] (strategic oscillation and a Tabu search), Chen et al. [5, 6] (threshold search and evolutionary path relinking). The last approaches were later improved by Qin et al. [16] and Tlili et al. [21]. A local branching approach was recently proposed by Aïder et al. [1].

The study of exact methods for the QMKP started in 2019, when Bergman [2] presented the first exact algorithm for the problem, obtained from an exponentially-sized

formulation, solved through Branch-and-Price. He used the generation scheme of [12] to produce new (smaller) instances with $n \in \{20, 25, 30, 35\}$ and $m \in \{3, 5, 10\}$. More recently, Galli et al. [9] studied effective polynomial-size formulations and relaxations of the problem leading to a decomposable structure, and proposed new medium-size benchmark sets (produced with the instance generator of [2]) with $n \in \{40, 45, 50, 55, 60\}$ and $m \in \{3, 5, 10\}$. Computational experiments showed that off-the-shelf solvers like CPLEX applied to the decomposable model are competitive with the Bergman algorithm for small-size instances. A combinatorial branch-and-bound algorithm was very recently proposed by Fleszar [7] for the case where all knapsack capacities are equal, computationally improving on the previous methods. In the following, we consider such a case. (Worth is mentioning that all benchmark instances from the literature consider the same capacity value for each knapsack.)

This paper is devoted to the study of matheuristic methods that can be derived from the mathematical models of the problem. In Section 2, we introduce the Lagrangian relaxation of (1)-(4) and its dual (see [9] for more details), which are used in Section 3 to obtain a matheuristic approach based on the solution of a variant of the set packing problem. A postoptimization stage is introduced in Section 4, and the overall deterministic algorithm is presented in Section 5. The outcome of computational experiments on small and medium size instances is reported in Section 6. Conclusions follow in Section 7.

2 Lagrangian relaxation

Let us define a vector λ of nonnegative Lagrangian multipliers λ_i $(i = 1, \ldots, n)$, and relax in a Lagrangian fashion cardinality constraints (3). We get

$$L(\lambda) = \sum_{i=1}^{n} \lambda_i + \max \sum_{i=1}^{n} \sum_{k=1}^{m} (p_i - \lambda_i) x_{ik} + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{k=1}^{m} p_{ij} x_{ik} x_{jk} \tag{5}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_{ik} \leq c_k \qquad (k = 1, \ldots, m) \tag{6}$$

$$x_{ik} \in \{0, 1\} \qquad (i = 1, \ldots, n;\ k = 1, \ldots, m). \tag{7}$$

Note that the dualization of constraints (3) decomposes the model into m independent (single) QKPs, sharing the same input for what concerns the items, with capacities $c_1, c_2, \ldots c_m$, respectively. Further observe that, if two knapsacks have the same capacity value, then the corresponding QKPs are identical. Therefore, for a given vector λ of multipliers, upper bound $L(\lambda)$ can be obtained by solving a single QKP for each different capacity value, i.e., just one single QKP when all capacities are identical. Also note that the Lagrangian linear profits $(p_i - \lambda_i)$ $(i = 1, \ldots, n)$ can take negative values.

Coming to the Lagrangian dual problem of finding $L(\lambda^\star) = \min_{\lambda \geq 0} L(\lambda)$, while the optimal multipliers cannot be computed efficiently, "good" approximations of $\lambda^\star$ can be obtained by traditional subgradient optimization. A more sophisticated technique, known as the *proximal bundle* method (see Frangioni [8]), which can be halted after a prefixed number of iterations, yields either the optimal multiplier vector $\lambda^\star$ or a good approximation of it.

The computer code `ProximalBundle`, implemented by Frangioni [8] and freely available at the `GitLab` repository, produces, at each iteration, a possibly improved vector λ of Lagrangian multipliers and the corresponding upper bound $L(\lambda)$. We denote as $\widetilde{\lambda}$ the best Lagrangian multiplier vector determined during the execution of the bundle procedure, and with $L(\widetilde{\lambda})$ the corresponding upper bound. In our case, each bundle iteration requires the optimal solution of a QKP: this is obtained by executing the `C` code `quadknap`, available at the home page of D. Pisinger. This code implements the QKP algorithm by Caprara et al. [4], and is currently the most effective code for optimally solving the QKP when the linear profits can take negative values. Since the QKP is a strongly $\mathcal{NP}$-hard problem, it might be necessary to halt the execution of `quadknap` after a certain elapsed time (in which case the upper bound provided in the current iteration of the bundle procedure may no longer be valid). In the next section, we show that, in any case, the set of QKP solutions provided by the bundle procedure can be used to derive an effective matheuristic algorithm for the QMKP.

3 Matheuristic scheme

In this section, we introduce the basic idea underlying the Lagrangian deterministic matheuristic for the QMKP. Our starting point is that of creating a family $\mathcal{F}$ of *promising subsets* of items that can be used to construct a feasible partial solution to the QMKP by filling "some" of the knapsacks. A promising subset is one that is likely to appear in a good quality (possibly optimal) solution to the QMKP. To do that, we take advantage of the subsets of items produced during the bundle iterations (as solutions to the QKP subproblems). In particular, our criterion for inserting a subset in $\mathcal{F}$ is that the upper bound $L(\lambda)$ produced at the corresponding bundle iteration be smaller than a given threshold $UBM = \alpha\, L(\widetilde{\lambda})$ (where α is a prefixed parameter with $\alpha > 1$). For the case of identical subsets, only one copy is retained. We denote the selected subsets as

$$B_h \subset \{1, \ldots, n\} \qquad (h = 1, \ldots, s), \tag{8}$$

where $s = |\mathcal{F}|$. Let

$$a_{hi} = \begin{cases} 1 & \text{if item } i \in B_h \\ 0 & \text{otherwise} \end{cases} \quad (h = 1, \ldots, s; i = 1, \ldots, n) \tag{9}$$

be the corresponding incidence matrix, and observe that subsets B_h will normally have nonempty intersections. Note that only "some" knapsacks will be filled according to the subsets in $\mathcal{F}$. (In fact, there may not even exist m disjoint subsets in $\mathcal{F}$.) More precisely, we define a parameter $r_{\max} < m$ representing the maximum number of (disjoint) subsets of $\mathcal{F}$ that can be selected to fill the knapsacks. Our approach consists of two phases:

1. Select *at most* $r_{\max}$ *disjoint* subsets B_h from family $\mathcal{F}$.
   Let $\mathcal{S}^\star$ denote the family of selected subsets, and $\overline{r} = |\mathcal{S}^\star|$.

2. *Pack* the subsets in $\mathcal{S}^\star$ into knapsacks $1, \ldots, \overline{r}$.
   Then iteratively *pack* into knapsacks $\overline{r}+1, \ldots, m$ subsets of unpacked items obtained by solving a series of (single) QKPs.

3.1 Phase 1

This phase requires to solve a combinatorial problem that can be seen as a generalization of the well-known *Set Packing Problem* (SPP), obtained by limiting the number of selected subsets. We denote such problem as the *Cardinality Constrained Set Packing Problem* (CCSPP). The CCSPP can then be formally defined as

$$\text{(CCSPP)} \quad \max \quad \sum_{h=1}^{s} y_h \tag{10}$$

$$\text{s.t.} \quad \sum_{h=1}^{s} a_{hi} y_h \leq 1 \qquad (i = 1, \ldots, n) \tag{11}$$

$$\sum_{h=1}^{s} y_h \leq r_{\max} \tag{12}$$

$$y_h \in \{0, 1\} \qquad (h = 1, \ldots, s), \tag{13}$$

where y_h takes the value one if subset B_h is chosen and the value zero otherwise. Constraints (11) ensure that the selected subsets are disjoint, while constraint (12) imposes the maximum cardinality to the family of subsets. The pseudocode implementing Phase 1 is given in Algorithm 1.

Algorithm 1: Procedure Bundle

input : A QMKP instance I
output: A family $\mathcal{S}^\star \subseteq \mathcal{F}$ of disjoint item subsets
A Lagrangian multiplier vector $\tilde{\lambda}$

1 **execute** `ProximalBundle` on (5)-(7);
2 let B_h $(h = 1, \ldots, s)$ be the subsets in family $\mathcal{F}$;
3 **solve** the CCSPP (10)-(13);
4 $y^\star \leftarrow$ CCSPP optimal solution;
5 $\mathcal{S}^\star \leftarrow \{B_h : h \in \{1, \ldots, s\}, y_h^\star = 1\}$.

In the special case where $r_{\max} = s$ the decision version of the CCSPP reduces to the SPP, one of the famous Karp's 21 $\mathcal{NP}$-complete problems [13]. This proves that the CCSPP is $\mathcal{NP}$-hard.

The solution to the CCSPP in procedure Bundle is obtained through an MILP solver (CPLEX 20.0 in our case).

3.2 Phase 2

The second phase of the matheuristic approach consists of either one or two steps, depending on the $\overline{r}$ value produced by the previous phase. If $\overline{r} < m - 1$, we iteratively solve $m - \overline{r} - 1$ (single) QKPs on the unpacked items, with Lagrangian linear profits $(p_i - \tilde{\lambda}_i)$. In any case, the m-th knapsack is assigned a set obtained by solving a QKP with the original linear profits p_i. We denote as $QKP(T)$ the (single) QKP induced by a

subset T of items with knapsack capacity c and appropriately defined item profits. The corresponding procedure is shown in Algorithm 2.

Algorithm 2: Procedure Pack

input : A family $\mathcal{S}^\star \subseteq \mathcal{F}$ of disjoint item subsets
A Lagrangian multiplier vector $\tilde{\lambda}$
output: A feasible solution x to the QMKP

```
1  r̄ := |S*|;
2  pack the r̄ subsets of S* into knapsacks 1, ..., r̄;
3  T ← set of the remaining (unpacked) items;
4  if r̄ < m − 1 then
5  |  for ℓ := r̄ + 1 to m − 1 do
6  |  |  execute quadknap(T) with linear profits p_i − λ̃_i (i ∈ T);
7  |  |  Q ← set of items in the QKP(T) solution;
8  |  |  pack the items of Q into knapsack ℓ;
9  |  |  T := T \ Q
10 |  end
11 end
12 execute quadknap(T) with linear profits p_i (i ∈ T);
13 Q ← set of items in the QKP(T) solution;
14 pack the items of Q into knapsack m.
```

The core matheuristic scheme consists thus of consecutively executing procedures `Bundle` and `Pack`.

4 Postoptimization

In this section, we show how to heuristically improve a feasible solution to the QMKP. We adopt the following notation:

- $z(x)$ = objective function value of a feasible solution x;
- x^{best} = incumbent solution, UB = upper bound on the optimal solution value. These quantities are assumed to be global variables;
- $K(i)$ = knapsack where item i is currently packed ($i = 1, \ldots, n$). We assume that an unpacked item i is packed into a *dummy knapsack* $m + 1$;
- $X(k)$ = set of items currently packed into knapsack k, with $W(k) = \sum_{i \in X(k)} w_i$ ($k = 1, \ldots, m$) and $W(m + 1) = \infty$;

The postoptimization approach makes use of two procedures:

- a procedure `Delete` which, given a feasible solution, removes, according to two possible criteria, some of the currently packed items;

- a procedure `LocalSearch` which improves a feasible solution through classical local search moves.

4.1 Item removal

For each knapsack k ($k = 1, \ldots, m$) and each item $i \in X(k)$, we define a *global profit* $\overline{p}_i$ given by its profit p_i plus the overall pairwise profit it currently produces. The items of $X(k)$ can be sorted either according to nondecreasing global profit, or according to global profit per unit weight, and a prefixed percentage γ of the first sorted items is then removed from $X(k)$. The corresponding procedure is provided in Algorithm 3.

Algorithm 3: Procedure Delete

input : A feasible solution x to an instance I
a parameter $d \in \{1, 2\}$, a positive parameter $\gamma < 1$
output: A feasible solution x^- to I with some items removed

```
1 x⁻ := x ;                                                    /* initialize */
2 for k := 1 to m do
3 |   foreach i ∈ X(k) do p̄_i := p_i + Σ_{j∈X(k)\{i}} p_ij;      /*global profit*/
4 |   if d = 1 then sort the items i ∈ X(k) by nondecreasing p̄_i values
5 |   else sort the items i ∈ X(k) by nondecreasing p̄_i/w_i values;
6 |   nd := ⌊γ · |X(k)|⌋;
7 |   remove the first nd items i from X(k) and set the corresponding x⁻_ik to 0
8 end
```

4.2 Local search

In order to improve the solution produced by the Lagrangian matheuristic described in the previous section, we embedded some classical local search basic moves from the literature (item exchanges and relocations). The framework is inspired by a similar one successfully adopted by Qin et al. [16] within a randomized, non-deterministic Tabu search algorithm for the QMKP. The main characteristic of such scheme is that of alternating moves that maintain feasibility and moves that can produce infeasible solutions, and repairing the final solution if infeasible.

The deterministic local search method, presented in Algorithm 4, consists of two main steps:

- **Step 1** performs a local search that preserves feasibility. It iteratively invokes two procedures:
  - `BestExchange` tries to improve a feasible solution by swapping pairs of items;
  - `BestReloc` tries to improve a feasible solution by moving items to a different knapsack.

Algorithm 4: Procedure Local Search

input : A feasible QMKP solution x
output: A possibly improved QMKP solution x^{best}

$x' := x$;
/*Step 1. `Feasible Local Search` */
$d := 1$;
while $d \leq 2$ **do**
  if $d = 1$ **then** $\overline{x} \leftarrow$ `BestExchange`(x') **else** $\overline{x} \leftarrow$ `BestReloc`(x');
  if $z(\overline{x}) > z(x')$ **then** $x' := \overline{x}$, $d := 1$ **else** $d := d + 1$;
end
$x^{\text{best}} := x'$;
/*Step 2. `Infeasible Local Search` */
Stop := *Feas* := `false`;
while *Stop* = `false` **do**
  if *Feas* = `true` **then**
    $\tilde{x} \leftarrow$ `BestExchange`(x');
    $\overline{x} \leftarrow$ `BestReloc`(x');
    if $z(\tilde{x}) > z(\overline{x})$ **then** $\overline{x} := \tilde{x}$;
    if $z(\overline{x}) > z(x')$ **then** $x' := \overline{x}$, *Feas* := `true` **else** *Feas* := `false`;
    $x^{\text{best}} := x'$
  else
    $\hat{x} := x'$;
    for $k := 1$ **to** m **do**
      if $W(k) < c_k$ **then** $\overline{x} \leftarrow$ `InfRelocA`(x', k) **else** $\overline{x} \leftarrow$ `InfRelocB`(x', k);
      if $z(\overline{x}) > z(\hat{x})$ **then** $\hat{x} := \overline{x}$
    end
    if $z(\hat{x}) = z(x')$ **then**
      Stop := `true`
    else
      Stop := `false`, $x' := \hat{x}$;
      if x' is feasible **then** $x^{\text{best}} := x'$, *Feas* := `true` **else** *Feas* := `false`
    end
  end
end
if x' is infeasible **then** $x' \leftarrow$ `Repair`(x');
if $z(x') > z(x^{\text{best}})$ **then** $x^{\text{best}} := x'$

Step 1 iterates `BestExchange` until no improvement is obtained, and then switches to `BestReloc` (possibly returning later to `BestExchange`);

- **Step 2** consists of a main **while** loop which alternates between a stage that preserves feasibility and a stage that can produce infeasible solutions. The former stage invokes in sequence both procedures above, iterating until no improvement is obtained. The latter stage is then executed by invoking two procedures which can produce an infeasible solution by relocating items:

  – procedures `InfRelocA` and `InfRelocB` try to move a currently packed item to a different knapsack, irrespective of a possible capacity violation.

  If such relocating procedures are unable to produce an improved (even if infeasible) solution, the local search terminates. Otherwise, one of the two stages is performed, depending on the feasibility of the improved solution. When the **while** loop is ended, if the incumbent solution is infeasible,

  – procedure `Repair` removes, exchanges, or relocates items until a feasible solution is obtained.

A detailed description of the local search procedures follows.

4.2.1 Local search procedures

In this section, we provide more details on the inner local search procedures. Despite being based on classical moves from the literature, for the sake of completeness the corresponding pseudocodes are given in the Appendix.

Procedure `BestExchange`, shown in Algorithm 7, tries to exchange pairs of items (i,j) (with $j > i$) not packed into the same knapsack, provided the exchange is both feasible and profitable. (Note that one of the two items can currently be unpacked.) An $i-j$ exchange is *feasible* if both capacity constraints of knapsacks $K(i)$ and $K(j)$ are preserved. The profit variation produced by the exchange is:

$$\Delta = \begin{cases} \sum\limits_{\ell \in X(K(j)) \setminus \{j\}} p_{i\ell} + \sum\limits_{\ell \in X(K(i)) \setminus \{i\}} p_{j\ell} - \sum\limits_{\ell \in X(K(i))} p_{i\ell} - \sum\limits_{\ell \in X(K(j))} p_{j\ell} \text{ if } K(i) \leq m \text{ and } K(j) \leq m \\ p_j + \sum\limits_{\ell \in X(K(i)) \setminus \{i\}} p_{j\ell} - p_i - \sum\limits_{\ell \in X(K(i))} p_{i\ell} \quad \text{if } K(i) \leq m \text{ and } K(j) = m+1 \\ p_i + \sum\limits_{\ell \in X(K(j)) \setminus \{j\}} p_{i\ell} - p_j - \sum\limits_{\ell \in X(K(j))} p_{j\ell} \quad \text{if } K(i) = m+1 \text{ and } K(j) \leq m \end{cases}$$

where we assume that $p_{ii} = 0 \ \forall i$. An exchange is thus profitable if $\Delta > 0$.

Procedure `BestReloc`, shown in Algorithm 8, shares part of the structure of `BestExchange`. It tries to move an item i (possibly unpacked) to a different (non dummy) knapsack k, provided the move is both feasible and profitable. In this case, the profit variation produced

by the relocation is:

$$\Delta = \begin{cases} \sum_{\ell \in X(k)} p_{i\ell} - \sum_{\ell \in X(K(i))} p_{i\ell} & \text{if } K(i) \leq m \\ p_i + \sum_{\ell \in X(k)} p_{i\ell} & \text{if } K(i) = m+1 \end{cases}$$

and the move is profitable if $\Delta > 0$.

The next two procedures, `InfRelocA` and `InfRelocB`, can produce infeasible solutions in which the capacity constraint of some knapsack is violated. Procedure `InfRelocA` is given in Algorithm 9. It tries to move to a specified knapsack k an item currently packed into a more filled knapsack. The item i (if any) producing the highest *normalized* objective function increase is selected, irrespective of a possible violation of the capacity constraint of knapsack k. The normalization is performed as in Qin et al. [16], i.e., the objective function increase is divided by w_i^β with $\beta \in (0, 1)$. Procedure `InfRelocB`, shown in Algorithm 10, shares part of the structure of `InfRelocA`, the main difference being that the item is (possibly) moved to a less (or equally) filled knapsack.

The last procedure, `Repair`, shown in Algorithm 11, is only executed if the incumbent solution produced by the previous steps turns out to be infeasible. It tries to iteratively remove an item, exchange a pair of items, or relocate an item with the main objective of reducing the *weight surplus*

$$\sigma = \sum_{k=1}^{m} \max\{0, W(k) - c\}, \tag{14}$$

where c is the (common) knapsack capacity, breaking ties by the largest profit of the resulting solution. This strategy differs from the one adopted in the repair procedure by Qin et al. [16] which privileges the largest resulting profit. At each iteration, the best move is performed and the execution terminates as soon as a feasible solution is obtained.

4.3 Overall postoptimization

The overall postptimization starts by performing a local search on the current solution. It then executes twice the sequence `Delete` – `LocalSearch`, for the two removal criteria previously defined. We assume in the following that the incumbent solution x^{best} is a global variable. The pseudocode is given in Algorithm 5.

5 Complete algorithm

In this section, we introduce our complete deterministic algorithmic framework, which combines the matheuristic approach of Section 3, a "weighted" matheuristic variant, and the postoptimization method of Section 4.

The basic idea for an effective implementation of the matheuristic approach is to explore a large number of "high quality" combinations of the item subsets produced by procedure `Bundle` and to avoid exploring previously generated solutions multiple times. To this end, we define a weighted variant of the CCSPP in which:

Algorithm 5: Procedure Postopt

input : A feasible solution x to an instance I
output: A possibly improved solution x^{best} to I

```
x ← LocalSearch(x);
for d := 1 to 2 do
    del := true;
    while del = true do
        x⁻ ← Delete(x,d);
        x⁻ ← LocalSearch(x⁻);
        if z(x⁻) > z(x) then x := x⁻ else del := false
    end
end
if z(x) > z(x^best) then
    x^best := x
end
```

(i) each subset $B_h \in \mathcal{F}$ is *weighted* by a measure g_h $(h = 1, \ldots, s)$ of its "quality" (to be defined later);

(ii) a *fixed* number r of disjoint subsets of $\mathcal{F}$ *must* be selected to fill the knapsacks;

(iii) a collection $\mathcal{S}^-$ of subset families is *forbidden*.

The corresponding model is

$$\text{(CCSPP1)} \quad \max \quad \sum_{h=1}^{s} g_h y_h \tag{15}$$

$$\text{s.t.} \quad \sum_{h=1}^{s} a_{hi} y_h \leq 1 \qquad (i = 1, \ldots, n) \tag{16}$$

$$\sum_{h=1}^{s} y_h = r \tag{17}$$

$$\sum_{h \in S} y_h \leq |S| - 1 \qquad (S \in \mathcal{S}^-) \tag{18}$$

$$y_h \in \{0, 1\} \qquad (h = 1, \ldots, s), \tag{19}$$

where points (i)-(iii) above are implemented by equations (15), (17), and (18), respectively.

The overall algorithm, procedure `GMRT` shown in Algorithm 6,

- starts by generating, through procedure `Bundle`, a family $\mathcal{S}^\star$ of item subsets, and setting $\overline{r} = |\mathcal{S}^\star|$;
- defines, for each subset B_h $(h = 1, \ldots, s)$, two different types of weights:
  g_h^1, the cardinality of subset B_h,
  g_h^2, the sum of all linear and pairwise profits produced by subset B_h;

Algorithm 6: Procedure GMRT

input : A QMKP instance I
output: A feasible solution x^{best} to I

$\mathcal{S}^\star$, $\tilde{\lambda} \leftarrow$ `Bundle(`I`)`;
$\overline{r} := |\mathcal{S}^\star|$, $x^{\text{best}} = \emptyset$; `/* initialize */`
for $h := 1$ **to** s **do**
  $g_h^1 := |B_h|$, $g_h^2 := \sum_{i \in B_h} p_i + \sum_{i \in B_h} \sum_{j \in B_h : j > i} p_{ij}$; `/* weights */`
end
for $r := 1$ **to** $\overline{r}$ **do**
  $\mathcal{S}^- := \emptyset$, $\ell := 1$; `/* initialize */`
  repeat
    if ℓ is odd **then**
      $g_h := g_h^1$ $(h = 1, \ldots, s)$
    else
      $g_h := g_h^2$ $(h = 1, \ldots, s)$
    end
    solve the CCSPP1 (15)-(19) with an MILP solver;
    $y^\star \leftarrow$ CCSPP1 optimal solution;
    $\mathcal{S}^\star \leftarrow \{B_h : h \in \{1, \ldots, s\}, y_h^\star = 1\}$;
    if $\mathcal{S}^\star \neq \emptyset$ **then**
      $x \leftarrow$ `Pack(`$\mathcal{S}^\star$`,`$\tilde{\lambda}$`)`;
      $x \leftarrow$ `Postopt(`x`)`;
      if $z(x) > z(x^{\text{best}})$ **then**
        $x^{\text{best}} := x$;
        if $z(x^{\text{best}}) = L(\tilde{\lambda})$ **then**
          terminate ; `/* Optimal solution found */`
        end
      end
      $\mathcal{S}^- := \mathcal{S}^- \cup \mathcal{S}^\star$
    end
    $\ell := \ell + 1$
  until $\ell = \ell_{\max}$ or $\mathcal{S}^\star = \emptyset$;
end

- executes a series of iterations, one for each value $r = 1, \ldots, \overline{r}$. Each iteration consists of repeatedly, in turn using weights g_h^1 and g_h^2,
  - generating a new QMKP solution from the sequence CCSPP1–`Pack`, and postoptimizing it through `Postopt`;
  - adding the new family $\mathcal{S}^\star$ produced by CCSPP1 to the current collection $\mathcal{S}^-$ of forbidden families.

  An iteration ends when either CCSPP1 is no longer able to produce a solution (due to the current $\mathcal{S}^-$) or a prefixed maximum number $\ell_{\max}$ of repetitions is reached: $\mathcal{S}^-$ is then reset to empty, and the next iteration is performed.

The process is terminated if an optimal QMKP solution is found.

Preliminary computational experiments showed that larger instances ($n \geq 45$ and $m \geq 5$) can require high computing times for the execution of the `quadknap` code at each bundle iteration. For this reason, in the final version of the algorithm, we use two time limits within the `ProximalBundle` code executed by the `Bundle` procedure invoked by `GMRT`:

- TLB = time limit for the global execution of the `ProximalBundle` code. If TLB is reached, the best Lagrangian multiplier vector $\tilde{\lambda}$ and the best upper bound $L(\tilde{\lambda})$ found by the bundle code could be improved;
- TLQ = time limit for the execution of the `quadknap` code at each iteration of the bundle code. If TLQ is reached, the upper bound computed at the current iteration is not a valid upper bound for the current instance.

In addition, it turns out that by executing the `GMRT` algorithm with different TLB and/or TLQ values, one can obtain different QMKP solutions that are not dominating each other. Therefore, our final solution approach to the QMKP consists of running `GMRT` twice, with two different pairs of time limits, (TLB^1, TLQ^1) and (TLB^2, TLQ^2), as follows:

- **Run 1**: during the first run of `GMRT`, we store in $UB1$ the best upper bound value (and the corresponding Lagrangian multiplier vector) found at the bundle iterations for which the CPU time of `quadknap` did not reach time limit TLQ^1. Note that $UB1$ represents a valid upper bound for the current instance;
- **Run 2**: this run of `GMRT` is only performed when at least one of the two time limits was reached in the first run. We set TLB^2 and TLQ^2 as the new time limits, perform a "warm" start from the best valid upper bound $UB1$ (and the corresponding Lagrangian multiplier vector) found in **Run 1**, and retain the best QMKP solution found.

6 Computational experiments

The deterministic matheuristic algorithm presented in the previous sections was implemented in `C++` and computationally evaluated by comparing its results with those produced by the fastest exact algorithm for the QMKP, namely the branch-and-bound algorithm recently proposed by Fleszar [7]. The comparison was performed using small and

medium size benchmark instances from the literature. (No computational experiments of other heuristic algorithms on these instances are reported in the literature.) The experiments were performed on a single thread of an Intel© Core™ i7-8700K running at 3.70GHz with 32 GB RAM.

The following software was used:

- `C++` code of the branch-and-bound algorithm by Fleszar [7], available at `https://sites.google.com/view/kfleszar/research`;
- `C++` code `ProximalBundle` by Frangioni [8], available at `https://gitlab.com/frangio68/ndosolver_fioracle_project`;
- `C` code `quadknap`, available at `http://hjemmesider.diku.dk/~pisinger/codes.html`;
- MILP commercial software CPLEX 20.0.

The parameter values used for all our experiments are as follows:

- $\alpha = 1.02$;
- $\beta = 0.7$;
- $\gamma = 0.25$;
- $\ell_{\max} = 300$;
- $r_{\max} = \begin{cases} \min\{3, \lceil m/2 \rceil\} & \text{if } n \leq 35 \\ \lceil m/2 \rceil & \text{if } n > 35; \end{cases}$
- $TLB^1 = 500$ seconds, $TLQ^1 = 5$ seconds;
- $TLB^2 = 200$ seconds, $TLQ^2 = 3$ seconds.

The benchmark instances consist of:

- two sets of small-size instances:
  - **HJ**, based on the generating scheme of Hiley and Julstrom [12]. For each triple (n, m, d) with $n \in \{20, 25, 30, 35\}$, $m \in \{3, 5, 10\}$, and $d \in \{0.25, 0.5, 0.75\}$ (d being the *density*, defined as the fraction of non-zero pairwise profits), five instances were generated, for a total of 180 instances.
    The linear and pairwise item profits and the weights are uniformly random integers in $(0, 100)$ and $(1, 50)$, respectively.
    The knapsack capacities are set to $\lfloor 0.8 \sum_{i=1}^{n} w_i/m \rfloor$;
  - **SS**, 45 groups of five instances each, based on the generating scheme of Saraç and Sipahioglu [17]. The resulting 225 instances have $n = 30$, $m \in \{3, 5, 10\}$, $d = 0.5$.
    The linear and pairwise profits are randomly generated using a parameter

$\rho \in \{1, 2, 3\}$ which produces different degrees of correlation with the weights. The item weights are drawn from uniform distributions with five different upper limits depending on a parameter $\psi \in \{1, 2, 3, 4, 5\}$. The knapsack capacities are as in **HJ**;

- **HJM**, one set of medium-size instances generated by Galli et al. [9] according to the **HJ** scheme, using the instance generator by Bergman [2]. For each triple (n, m, d) with $n \in \{40, 45, 50, 55, 60\}$, $m \in \{3, 5, 10\}$, and $d \in \{0.25, 0.5, 0.75\}$, five instances were generated, for a total of 225 instances.

Benchmarks **HJ** and **SS** by Bergman [2], can be downloaded at the INFORMS address: `https://pubsonline.informs.org/doi/suppl/10.1287/ijoc.2018.0840/suppl_file/ijoc.2018.0840-instances.sm2.zip`.

Benchmark **HJM** by Galli et al. [9] can be downloaded from: `http://or.dei.unibo.it/instances/quadratic-multiple-knapsack-problem-instances`.

Tables 1 and 2 provide the results for the small-size instances. The branch-and-bound algorithm by Fleszar [7] was run with a time limit of 3600 seconds. All CPU times are expressed in seconds. Each line refers to a triple (n, m, d) (Table 1) or to a 4-tuple (Table 2) and provides the average values over the corresponding five instances:

- the first group of four columns refers to the Fleszar [7] algorithm: the entries report the best solution value found ($\texttt{Best}^{\texttt{F}}$), the upper bound value (`UB`), the percentage gap ($\texttt{\%Gap}^{\texttt{F}}$) (computed as $100\,(\texttt{UB - Best}^{\texttt{F}})/\texttt{Best}^{\texttt{F}}$), and the CPU time (`Time`);

- the second group of five columns refers to the "pure" metaheuristic algorithm of Section 3, i.e., before the postoptimization phase: the entries report the best solution value found ($\texttt{Best}^{\texttt{M}}$), the percentage gap (`%Gap`) (computed as $100\,(\texttt{Best}^{\texttt{F}} - \texttt{Best}^{\texttt{M}})/\texttt{Best}^{\texttt{F}}$), the CPU time (`Time`), the upper bound value (`UB`) computed through the Lagrangian relaxation of Section 2, and the CPU time (`T-UB`) spent for its computation;

- the third group of three columns refers to the complete algorithm `GMRT` of Section 5: the entries report the best solution value found ($\texttt{Best}^{\texttt{MP}}$), the percentage gap (`%Gap`) (computed as $100\,(\texttt{Best}^{\texttt{F}} - \texttt{Best}^{\texttt{MP}})/\texttt{Best}^{\texttt{F}}$), and the CPU time (`Time`) spent in total for its computation.

For these instances, time limits TLB^1 and TLQ^1 were never reached in **Run 1**, so the code never invoked **Run 2**. Tables 1 and 2 show that, on small-size instances, the `GMRT` algorithm consistently delivers solutions of very good quality in short computing times. In particular, on all **HJ** and **SS** instances, the percentage gap of `GMRT` is either zero or very close to, and the computing times for $n \geq 30$ are at least one order of magnitude smaller than those taken by the exact algorithm by Fleszar [7]. The upper bound values produced by `GMRT` are not very far from the tighter values obtained by such algorithm. A comparison between the percentage gaps of the last two groups of columns reveals the importance of the postoptimization phase, which significantly improves on the matheuristic solution, with a fairly low impact on the overall CPU time.

Instance			Fleszar				Matheuristic					Matheur.+Postopt		
n	m	d	Best$^{\mathrm{F}}$	UB	%Gap$^{\mathrm{F}}$	Time	Best$^{\mathrm{M}}$	%Gap	Time	UB	T-UB	Best$^{\mathrm{MP}}$	%Gap	Time
20	3	25	2289.20	2289.20	0.00	0.03	2286.00	0.16	0.07	2295.40	0.06	2289.20	0.00	0.09
20	5	25	1916.20	1916.20	0.00	0.07	1911.00	0.23	0.06	1918.20	0.05	1916.20	0.00	0.13
20	10	25	1199.00	1199.00	0.00	0.01	1198.00	0.10	0.05	1199.00	0.04	1199.00	0.00	0.05
20	3	50	3079.80	3079.80	0.00	0.09	3076.60	0.11	0.07	3093.60	0.07	3079.80	0.00	0.10
20	5	50	2296.60	2296.60	0.00	0.21	2268.80	1.16	0.07	2297.00	0.06	2296.60	0.00	0.25
20	10	50	1427.00	1427.00	0.00	0.01	1304.00	7.76	0.05	1433.60	0.04	1427.00	0.00	0.08
20	3	75	3553.80	3553.80	0.00	0.17	3530.60	0.62	0.07	3592.40	0.07	3543.80	0.27	0.12
20	5	75	2640.60	2640.60	0.00	0.16	2633.40	0.28	0.08	2653.80	0.06	2640.60	0.00	0.34
20	10	75	1639.60	1639.60	0.00	0.01	1591.80	2.66	0.05	1639.60	0.04	1639.60	0.00	0.06
Avg			2226.87	2226.87	0.00	0.08	2200.02	1.45	0.06	2235.84	0.06	2225.76	0.03	0.14
25	3	25	3002.40	3002.40	0.00	0.20	2981.20	0.73	0.14	3014.20	0.13	3000.20	0.08	0.20
25	5	25	2559.00	2559.00	0.00	0.67	2559.00	0.00	0.13	2565.80	0.11	2559.00	0.00	0.32
25	10	25	1781.20	1781.20	0.00	0.08	1771.00	0.60	0.12	1784.40	0.09	1773.80	0.44	0.53
25	3	50	4275.00	4275.00	0.00	1.08	4275.00	0.00	0.18	4294.80	0.17	4275.00	0.00	0.27
25	5	50	3272.80	3272.80	0.00	2.49	3272.80	0.00	0.13	3282.80	0.11	3272.80	0.00	0.33
25	10	50	2093.20	2093.20	0.00	0.14	2087.20	0.29	0.09	2093.20	0.08	2093.20	0.00	0.10
25	3	75	5155.00	5155.00	0.00	3.46	5137.00	0.35	0.26	5178.40	0.25	5155.00	0.00	0.39
25	5	75	3774.80	3774.80	0.00	5.99	3774.80	0.00	0.21	3807.20	0.17	3774.80	0.00	1.26
25	10	75	2420.20	2420.20	0.00	0.16	2408.80	0.47	0.14	2427.00	0.13	2417.40	0.12	2.14
Avg			3148.18	3148.18	0.00	1.58	3140.76	0.27	0.16	3160.87	0.14	3146.80	0.07	0.61
30	3	25	4194.80	4194.80	0.00	0.73	4161.60	0.79	0.41	4202.80	0.39	4194.80	0.00	0.52
30	5	25	3550.40	3550.40	0.00	6.51	3540.20	0.29	0.26	3557.60	0.22	3550.40	0.00	0.58
30	10	25	2558.40	2558.40	0.00	10.86	2558.40	0.00	0.19	2562.40	0.17	2558.40	0.00	3.34
30	3	50	5858.60	5858.60	0.00	19.09	5829.20	0.49	0.83	5903.20	0.79	5850.80	0.13	1.07
30	5	50	4609.40	4609.40	0.00	21.39	4595.80	0.27	0.28	4611.20	0.27	4609.40	0.00	1.12
30	10	50	2963.20	2963.20	0.00	6.35	2890.80	2.08	0.21	2964.60	0.19	2963.20	0.00	6.61
30	3	75	7210.20	7210.20	0.00	42.83	7207.00	0.04	1.02	7287.80	0.97	7210.20	0.00	1.30
30	5	75	5362.00	5362.00	0.00	30.57	5347.20	0.28	0.35	5375.80	0.26	5362.00	0.00	1.03
30	10	75	3283.40	3283.40	0.00	2.16	3260.40	0.73	0.24	3291.80	0.14	3281.00	0.08	1.02
Avg			4398.93	4398.93	0.00	15.61	4376.73	0.55	0.42	4417.47	0.38	4397.80	0.02	1.84
35	3	25	5277.80	5277.80	0.00	10.74	5272.40	0.11	2.34	5304.00	2.26	5272.40	0.11	2.75
35	5	25	4463.00	4463.00	0.00	87.35	4416.60	1.06	0.63	4472.80	0.51	4462.80	0.00	1.87
35	10	25	3318.60	3318.60	0.00	45.29	3309.60	0.28	0.51	3323.60	0.27	3317.60	0.03	10.96
35	3	50	7730.60	7730.60	0.00	443.02	7704.00	0.34	3.57	7768.00	3.47	7723.00	0.09	3.87
35	5	50	6000.20	6000.20	0.00	1412.79	5955.60	0.73	0.83	6021.00	0.80	6000.00	0.00	3.02
35	10	50	4062.20	4062.20	0.00	86.65	4028.80	0.84	0.33	4068.60	0.26	4041.80	0.50	3.18
35	3	75	9944.40	9944.40	0.00	271.80	9904.60	0.38	3.03	9993.40	2.98	9944.40	0.00	3.46
35	5	75	7204.20	7204.20	0.00	731.79	7189.20	0.20	1.52	7226.60	1.03	7202.60	0.02	5.82
35	10	75	4482.60	4482.60	0.00	260.15	4449.60	0.73	1.83	4498.40	0.40	4479.60	0.07	26.77
Avg			5831.51	5831.51	0.00	372.18	5803.38	0.52	1.62	5852.93	1.33	5827.13	0.09	6.86
Over. Avg			3901.37	3901.37	0.00	97.36	3880.22	0.70	0.57	3916.78	0.48	3899.37	0.05	2.36

Table 1: **HJ** instances. Average values over 5 instances.

Instance				Fleszar				Matheuristic					Matheur.+Postopt		
n	m	ψ	ρ	Best^{F}	UB	%Gap^{F}	Time	Best^{M}	%Gap	Time	UB	T-UB	Best^{MP}	%Gap	Time
30	3	1	1	1610.20	1610.20	0.00	65.83	1609.80	0.03	0.66	1618.00	0.63	1610.20	0.00	1.17
30	3	1	2	1425.60	1425.60	0.00	60.97	1425.00	0.04	0.64	1427.40	0.50	1425.60	0.00	0.89
30	3	1	3	1818.00	1818.00	0.00	135.44	1818.00	0.00	0.98	1821.80	0.82	1818.00	0.00	1.56
30	3	2	1	3530.40	3530.40	0.00	63.08	3529.80	0.02	0.78	3544.00	0.64	3530.40	0.00	1.20
30	3	2	2	3032.00	3032.00	0.00	96.54	3030.00	0.07	0.64	3039.00	0.56	3030.00	0.07	1.16
30	3	2	3	3714.00	3714.00	0.00	340.36	3707.60	0.18	1.02	3720.40	0.95	3710.80	0.09	1.83
30	3	3	1	5263.20	5263.20	0.00	159.68	5260.20	0.06	1.09	5289.80	0.98	5263.20	0.00	1.92
30	3	3	2	4188.80	4188.80	0.00	282.75	4188.20	0.01	0.59	4193.60	0.54	4188.20	0.01	0.98
30	3	3	3	5340.80	5340.80	0.00	533.46	5336.00	0.09	0.77	5360.40	0.68	5338.80	0.03	1.65
30	3	4	1	6971.80	6971.80	0.00	69.28	6971.80	0.00	0.98	6990.80	0.90	6971.80	0.00	1.68
30	3	4	2	5964.40	5964.40	0.00	628.57	5960.60	0.07	0.80	5975.80	0.72	5963.60	0.01	1.37
30	3	4	3	7120.80	7120.80	0.00	436.66	7120.40	0.01	0.74	7143.00	0.69	7120.80	0.00	1.35
30	3	5	1	9039.00	9039.00	0.00	62.49	9033.60	0.06	1.02	9059.20	0.91	9037.00	0.02	1.79
30	3	5	2	6566.80	6566.80	0.00	229.49	6565.20	0.02	0.50	6570.20	0.46	6565.40	0.02	0.65
30	3	5	3	8242.80	8242.80	0.00	240.64	8234.80	0.10	0.82	8274.20	0.73	8235.20	0.09	1.62
Avg				4921.91	4921.91	0.00	227.02	4919.40	0.05	0.80	4935.17	0.71	4920.60	0.02	1.39
30	5	1	1	1509.60	1509.60	0.00	354.06	1505.80	0.26	0.46	1514.40	0.41	1509.40	0.01	4.56
30	5	1	2	1382.00	1382.00	0.00	159.25	1381.20	0.05	0.57	1384.20	0.22	1382.00	0.00	1.75
30	5	1	3	1740.00	1740.00	0.00	805.66	1739.60	0.02	2.07	1743.60	0.45	1740.00	0.00	11.05
30	5	2	1	3342.60	3342.60	0.00	637.74	3339.60	0.09	1.92	3355.40	0.39	3340.20	0.07	6.42
30	5	2	2	2933.00	2933.00	0.00	345.75	2929.40	0.13	0.82	2941.00	0.26	2932.80	0.01	3.19
30	5	2	3	3511.60	3511.60	0.00	723.43	3511.20	0.01	0.58	3520.40	0.32	3511.20	0.01	5.92
30	5	3	1	4984.80	4984.80	0.00	594.26	4983.60	0.02	0.51	4996.20	0.41	4983.60	0.02	3.60
30	5	3	2	4041.40	4041.40	0.00	901.17	4040.60	0.02	0.52	4048.40	0.24	4040.60	0.02	2.79
30	5	3	3	5036.80	5036.80	0.00	1877.67	5033.60	0.06	1.18	5048.20	0.37	5036.80	0.00	8.09
30	5	4	1	6546.60	6546.60	0.00	758.69	6530.60	0.23	0.93	6569.80	0.42	6540.80	0.08	6.08
30	5	4	2	5726.60	5736.20	0.16	2127.03	5719.40	0.13	0.95	5741.60	0.28	5720.60	0.11	4.62
30	5	4	3	6704.00	6704.80	0.01	1905.60	6703.20	0.01	0.86	6726.20	0.34	6703.20	0.01	9.43
30	5	5	1	8517.00	8517.00	0.00	1209.14	8508.60	0.11	0.95	8549.80	0.47	8513.60	0.04	7.24
30	5	5	2	6274.80	6287.00	0.20	2129.79	6274.20	0.01	0.49	6295.40	0.24	6274.20	0.01	3.48
30	5	5	3	7762.80	7762.80	0.00	1431.44	7756.40	0.07	1.58	7794.00	0.35	7757.20	0.06	9.38
Avg				4667.57	4669.08	0.02	1064.05	4663.80	0.08	0.96	4681.91	0.34	4665.75	0.03	5.84
30	10	1	1	1366.40	1366.40	0.00	286.16	1357.80	0.63	0.80	1369.80	0.25	1365.60	0.05	6.48
30	10	1	2	1290.20	1290.20	0.00	4.49	1288.40	0.14	0.49	1291.00	0.18	1289.80	0.03	4.09
30	10	1	3	1640.00	1640.00	0.00	96.81	1636.80	0.18	1.67	1645.80	0.20	1638.00	0.11	11.80
30	10	2	1	3069.40	3069.40	0.00	649.28	3063.00	0.21	0.68	3071.80	0.22	3068.80	0.02	9.10
30	10	2	2	2746.80	2746.80	0.00	17.91	2746.20	0.02	0.56	2753.80	0.15	2746.20	0.02	8.92
30	10	2	3	3265.60	3265.60	0.00	74.16	3265.60	0.00	0.59	3276.40	0.20	3265.60	0.00	18.75
30	10	3	1	4575.40	4575.40	0.00	187.47	4572.60	0.07	0.46	4582.40	0.29	4575.20	0.00	15.82
30	10	3	2	3776.40	3776.40	0.00	8.13	3774.00	0.06	0.23	3778.60	0.12	3774.60	0.04	4.96
30	10	3	3	4678.00	4678.00	0.00	439.50	4673.20	0.12	0.53	4685.80	0.21	4673.20	0.12	18.63
30	10	4	1	5989.80	5989.80	0.00	509.02	5981.20	0.13	0.28	5994.00	0.22	5989.80	0.00	4.80
30	10	4	2	5319.20	5319.20	0.00	99.11	5316.60	0.05	0.63	5322.20	0.16	5319.20	0.00	12.26
30	10	4	3	6259.20	6259.20	0.00	310.73	6257.60	0.02	0.17	6267.20	0.14	6259.20	0.00	9.59
30	10	5	1	7863.00	7882.00	0.21	902.33	7840.20	0.29	0.87	7883.20	0.27	7860.40	0.03	12.25
30	10	5	2	5866.80	5866.80	0.00	45.38	5866.80	0.00	0.13	5868.60	0.10	5866.80	0.00	2.33
30	10	5	3	7151.20	7151.20	0.00	40.58	7127.20	0.30	0.29	7158.00	0.16	7151.20	0.00	14.19
Avg				4323.83	4325.09	0.01	244.74	4317.81	0.15	0.56	4329.91	0.19	4322.91	0.03	10.27
Over. Avg				4637.77	4638.69	0.01	511.93	4633.67	0.09	0.77	4649.00	0.42	4636.42	0.03	5.83

Table 2: **SS** instances. Average values over 5 instances.

Table 3 provides the results for the medium-size instances **HJM**. In this case too, the algorithm by Fleszar [7] was run with a time limit of 3600 seconds. Each line refers to a triple (n, m, d) and provides the average values over the corresponding five instances:

- the first group of four columns refers to the Fleszar [7] algorithm, and reports the same information as in the previous tables;
- the second group of five columns refers to **Run 1**. The entries report the best upper bound value `UB` found and the CPU time `T-UB` needed to obtain it, the best solution value found ($\texttt{Best}^{\text{MP}}$), the percentage gap (`%Gap`), computed as $100\,(\texttt{Best}^{\text{F}} - \texttt{Best}^{\text{MP}})/\texttt{Best}^{\text{F}}$ and the related CPU time (`Time`);
- the third group of two columns refers to **Run 2**, and reports the best upper bound value `UB` found by the two runs and the CPU time `T-UB` of **Run 2**;
- the fourth and fifth group of three columns each summarize the overall results (best solution value, percentage gap and CPU time) produced by the matheuristic approach before and after postoptimization, respectively.

The results in Table 3 show that on larger HJ instances Fleszar's exact method consistently hits the time limit of 3600 seconds, with few exceptions, mostly on the smaller ($n = 40$) instances. For such cases, columns $\texttt{Best}^{\text{F}}$ and `UB` in the first group provide the best solution and upper bound values, respectively, computed by the branch-and-bound algorithm before the time limit. Already **Run 1** of `GMRT` (second group of columns) regularly provides solutions of better quality with computing times that are one or two orders of magnitude smaller. The next group of two columns is only reported to examine the effect of **Run 2** on the upper bound computations: we can observe that: (i) the second run is only executed for relatively large values of n; (ii) a moderate increase of computing time occasionally allows to improve the upper bound values.

The overall performance of the proposed approach is summarized in the last two groups of columns, before and after the postoptimization phase, respectively. The results show that: (i) *before* postoptimization the best solutions found are worse than those obtained by Fleszar's algorithm; (ii) *after* postoptimization, the overall algorithm favorably compares with Fleszar's algorithm (negative percentage gap values), providing good solutions in CPU times of few hundred seconds; (iii) the execution of **Run 2** further improves the results obtained after the execution of **Run 1**, with a moderate increase of CPU time.

7 Conclusions

Recent literature on the QMKP shows an increasing attention by different authors to develop exact methods for this challenging combinatorial optimization problem. We propose the first deterministic Lagrangian matheuristic for solving the QMKP approximately. Our method relies on the strength of the Lagrangian relaxation for the natural quadratic formulation of the QMKP, which can be exploited to generate "high quality" subsets of

Instance			Fleszar				Matheur. + Postopt Run 1					Bundle Run 2		Overall Matheur.			Over. Matheur.+Postopt		
n	m	d	Best^{F}	UB	%Gap^{F}	Time	UB	T-UB	Best^{MP}	%Gap	Time	UB	T-UB	Best^{M}	%Gap	Time	Best^{MP}	%Gap	Time
40	3	25	6789.80	6789.80	0.00	384.65	6797.00	10.70	6788.80	0.01	11.27	6797.00	0.00	6729.40	0.86	10.86	6788.80	0.01	11.27
40	5	25	5755.20	5853.80	1.69	3108.10	5777.20	1.67	5747.00	0.14	6.24	5777.20	0.00	5630.00	2.13	2.21	5747.00	0.14	6.24
40	10	25	4381.60	4481.00	2.29	3215.70	4400.00	0.50	4372.40	0.21	29.91	4400.00	0.00	4236.20	3.24	2.25	4372.40	0.21	29.91
40	3	50	9869.40	10706.40	8.62	3599.86	9969.40	23.78	9876.00	-0.07	25.39	9969.40	0.00	9629.20	2.46	24.33	9876.00	-0.07	25.39
40	5	50	7659.60	8231.60	7.67	3600.13	7684.20	3.02	7656.00	0.05	7.43	7684.20	0.00	7610.60	0.63	3.21	7656.00	0.05	7.43
40	10	50	5173.20	5359.20	3.78	2973.77	5207.00	0.60	5166.60	0.13	49.90	5207.00	0.00	4945.80	4.26	3.32	5166.60	0.13	49.90
40	3	75	12170.80	12876.60	5.94	3600.14	12221.80	25.01	12176.00	-0.05	26.14	12221.80	0.00	11937.00	1.94	25.36	12176.00	-0.05	26.14
40	5	75	8863.20	9379.20	5.86	3600.15	8917.00	3.66	8851.00	0.13	11.05	8917.00	0.00	8612.60	2.78	4.32	8851.00	0.13	11.05
40	10	75	5695.60	5808.00	1.96	2886.33	5729.00	0.58	5701.00	-0.08	50.42	5729.00	0.00	5531.00	2.91	0.87	5701.00	-0.08	50.42
Avg			7373.16	7720.62	4.20	2996.54	7411.40	7.72	7370.53	0.05	24.20	7411.40	0.00	7206.87	2.36	8.53	7370.53	0.05	24.20
45	3	25	8290.00	8498.00	2.52	3100.39	8308.80	77.69	8280.80	0.11	79.51	8308.80	0.00	8206.20	0.99	78.40	8280.80	0.11	79.51
45	5	25	6894.60	7652.60	10.99	3600.42	6955.80	9.21	6933.00	-0.55	17.81	6955.80	0.00	6895.60	0.00	11.34	6933.00	-0.55	17.81
45	10	25	5252.60	5650.20	7.58	3600.59	5285.00	0.79	5256.80	-0.08	76.75	5285.00	0.00	5139.20	2.18	5.40	5256.80	-0.08	76.75
45	3	50	12282.40	13935.20	13.59	3600.39	12381.40	137.43	12288.20	-0.04	140.18	12381.40	36.42	12034.40	2.05	175.68	12295.00	-0.10	178.33
45	5	50	9443.40	10659.80	12.95	3600.53	9513.20	11.42	9436.80	0.07	22.68	9513.20	0.00	9105.60	3.53	11.92	9436.80	0.07	22.68
45	10	50	6453.60	6858.00	6.30	3600.52	6492.80	1.16	6459.80	-0.10	95.16	6492.80	0.00	6346.60	1.67	3.75	6459.80	-0.10	95.16
45	3	75	15722.20	17126.00	9.17	3600.00	15777.80	118.61	15722.20	0.00	123.12	15777.80	34.55	15699.40	0.15	154.85	15722.20	0.00	159.76
45	5	75	11288.20	12316.00	9.38	3600.17	11446.60	12.05	11383.80	-0.86	35.96	11446.60	0.00	11232.60	0.54	15.07	11383.80	-0.86	35.96
45	10	75	7118.40	7495.40	5.48	3599.79	7229.60	1.32	7188.60	-1.04	119.72	7229.60	0.00	6995.80	1.71	10.01	7188.60	-1.04	119.72
Avg			9193.93	10021.24	8.66	3544.76	9265.67	41.08	9216.67	-0.28	78.99	9265.67	7.89	9072.82	1.43	51.82	9217.42	-0.28	87.30
50	3	25	9684.40	10558.00	8.97	3600.51	10081.40	292.54	9684.40	0.00	295.88	10081.40	149.26	9573.20	1.15	443.46	9684.40	0.00	449.16
50	5	25	8095.40	9435.40	16.51	3600.58	8116.60	34.82	8089.20	0.08	52.60	8116.60	0.00	8002.40	1.11	36.11	8089.20	0.08	52.60
50	10	25	6105.20	6832.40	11.90	3600.63	6156.00	2.09	6130.80	-0.40	94.21	6156.00	0.00	5915.20	3.09	10.74	6130.80	-0.40	94.21
50	3	50	14636.40	17705.40	20.98	3600.59	15808.60	439.76	14618.60	0.12	448.53	15808.60	202.07	14391.40	1.59	652.40	14636.40	0.00	662.74
50	5	50	11116.00	13327.20	19.89	3600.64	11238.60	57.55	11119.60	-0.04	90.41	11238.60	0.00	10884.20	2.13	68.15	11119.60	-0.04	90.41
50	10	50	7568.20	8336.40	10.16	3600.61	7641.80	2.54	7621.40	-0.69	117.98	7641.80	0.00	7449.00	1.65	16.35	7621.40	-0.69	117.98
50	3	75	19162.60	21961.80	14.82	3600.44	20313.60	335.63	19162.60	-0.01	339.69	20313.60	176.59	19087.80	0.38	513.31	19193.20	-0.17	520.37
50	5	75	13852.60	15718.60	13.54	3600.63	13993.00	63.75	13872.80	-0.15	102.14	13993.00	0.00	13615.40	1.70	74.96	13872.80	-0.15	102.14
50	10	75	8807.80	9448.00	7.31	3600.33	8887.20	3.74	8819.20	-0.12	179.94	8887.20	0.00	8521.40	3.18	19.26	8819.20	-0.12	179.94
Avg			11003.18	12591.47	13.79	3600.55	11359.64	136.93	11013.18	-0.13	191.27	11359.64	58.66	10826.67	1.78	203.86	11018.56	-0.17	252.17
55	3	25	11215.60	12929.40	15.27	3600.68	12563.40	472.84	11207.00	0.08	480.87	12563.40	199.94	11128.00	0.81	680.70	11211.40	0.04	693.62
55	5	25	9230.00	11534.40	24.94	3600.69	9310.40	181.40	9231.60	-0.01	229.49	9310.40	0.00	9056.80	1.87	197.88	9231.60	-0.01	229.49
55	10	25	6978.60	8136.40	16.59	3600.88	7042.80	3.14	6987.80	-0.12	122.51	7042.80	0.00	6840.80	1.99	22.03	6987.80	-0.12	122.51
55	3	50	16923.00	21780.20	28.70	3600.09	20005.00	475.56	16876.40	0.28	482.78	20005.00	200.91	16681.80	1.42	701.03	16899.80	0.13	727.22
55	5	50	12913.00	16323.60	26.42	3600.11	13010.60	289.78	12916.20	-0.02	382.36	13010.60	40.18	12769.00	1.11	360.58	12916.20	-0.02	431.98
55	10	50	8860.20	10052.20	13.47	3600.76	8936.20	6.18	8862.00	-0.01	169.61	8936.20	0.00	8610.20	2.82	21.29	8862.00	-0.01	169.61
55	3	75	22295.60	26525.20	19.01	3600.21	27062.80	342.30	22088.40	0.91	346.24	25397.20	201.67	22041.60	1.13	546.07	22217.40	0.35	553.12
55	5	75	16039.40	18822.40	17.39	3600.17	16195.20	204.70	16043.40	-0.03	313.01	16195.20	31.11	15649.00	2.49	277.52	16053.00	-0.09	350.52
55	10	75	10175.40	11177.80	9.90	3600.72	10296.60	8.63	10223.20	-0.47	242.87	10296.60	0.00	10045.20	1.27	31.18	10223.20	-0.47	242.87
Avg			12736.76	15253.51	19.08	3600.48	13824.78	220.50	12715.11	0.07	307.75	13639.71	74.87	12535.82	1.66	315.36	12733.60	-0.02	391.22
60	3	25	12743.20	15695.40	23.15	3600.66	15965.20	502.94	12716.60	0.21	510.50	15875.80	201.59	12632.80	0.87	732.18	12759.40	-0.13	743.24
60	5	25	10489.60	14093.80	34.37	3600.98	11497.20	484.73	10484.60	0.05	566.59	11497.20	201.32	10242.80	2.34	760.69	10491.00	-0.01	818.03
60	10	25	7965.60	9772.40	22.72	3600.80	8070.20	12.23	8014.00	-0.64	151.85	8070.20	0.00	7835.20	1.61	30.38	8014.00	-0.64	151.85
60	3	50	19807.40	26315.60	32.93	3600.62	27676.00	501.54	19804.60	0.01	507.71	26356.80	201.45	19523.00	1.39	712.40	19822.40	-0.09	760.38
60	5	50	15056.00	19655.40	30.57	3600.66	16338.00	499.98	14936.00	0.80	535.05	16337.00	199.29	14146.60	6.07	727.04	14936.00	0.80	747.21
60	10	50	10395.00	12159.20	17.02	3600.81	10479.40	24.68	10401.80	-0.06	254.62	10479.40	0.00	10084.60	3.03	77.81	10401.80	-0.06	254.62
60	3	75	26791.80	32373.60	20.87	3600.55	33517.60	501.97	26628.40	0.61	506.84	33203.20	201.24	26211.00	2.16	703.67	26713.80	0.29	720.37
60	5	75	19310.20	22905.40	18.71	3600.72	20378.40	384.42	19105.20	1.08	414.80	20378.40	154.54	18739.80	2.98	549.07	19238.20	0.39	575.69
60	10	75	12346.60	13594.00	10.27	3600.90	12469.20	20.25	12365.60	-0.14	324.48	12469.20	0.00	12140.00	1.70	57.18	12365.60	-0.14	324.48
Avg			14989.49	18507.20	23.40	3600.75	17376.80	325.86	14939.64	0.21	419.16	17185.24	128.83	14617.31	2.46	483.38	14971.36	0.05	566.21
Over. Avg			11059.30	12818.81	13.83	3468.61	11847.66	146.42	11051.03	-0.02	204.27	11772.33	45.04	10851.90	1.94	212.59	11062.29	-0.07	264.22

Table 3: **HJM** instances. Average values over 5 instances.

items to be packed into the knapsacks. Experimental studies show that the method is competitive with the currently best performing exact algorithm by Fleszar [7] on small **HJ** instances with $n \geq 30$, and on all **SS** instances, consistently delivering solutions of very good quality in much shorter computing times. On the medium-size **HJM** instances, the proposed method outperforms the algorithm by Fleszar [7] both in terms of solution quality and computing time. The results show that the QMKP is very difficult to solve, especially when n and/or m grow. Directions for future research could thus be the development of faster exact algorithms to solve large problem instances. In particular, we believe that our Lagrangian matheuristic could be embedded within an exact framework to quickly generate high quality solutions and upper bounds.

Acknowledgments

This research was supported by the Air Force Office of Scientific Research under Grants no. FA8655-20-1-7012 and FA8655-20-1-7019 and by National Agency for Research and Development (ANID)/ Scholarship Program/Doctorado Becas Chile/2018-72190600. The authors also gratefully acknowledge partial financial support from the University of Pisa, under the project "Analisi di reti complesse: dalla teoria alle applicazioni" (grant PRA_2020_61). We thank Antonio Frangioni and David Pisinger for the support in the use of their codes.

References

[1] M. Aïder, O. Gacem, and M. Hifi. Branch and solve strategies-based algorithm for the quadratic multiple knapsack problem. *Journal of the Operational Research Society*, pages 1–18, 2020.

[2] D. Bergman. An exact algorithm for the quadratic multiknapsack problem with an application to event seating. *INFORMS Journal on Computing*, 31(3):477–492, 2019.

[3] V. Cacchiani, M. Iori, A. Locatelli, and S. Martello. Knapsack problems - an overview of recent advances. *Computers & Operations Research*, 2021. to appear.

[4] A. Caprara, D. Pisinger, and P. Toth. Exact solution of the quadratic knapsack problem. *INFORMS Journal on Computing*, 11(2):125–137, 1999.

[5] Y. Chen and J.K. Hao. Iterated responsive threshold search for the quadratic multiple knapsack problem. *Annals of Operations Research*, 226(1):101–131, 2015.

[6] Y. Chen, J.K. Hao, and F. Glover. An evolutionary path relinking approach for the quadratic multiple knapsack problem. *Knowledge-Based Systems*, 92:23–34, 2016.

[7] K. Fleszar. A branch-and-bound algorithm for the quadratic multiple knapsack problem. *European Journal of Operational Research*, 2021. to appear.

[8] A. Frangioni. Generalized Bundle Methods. *SIAM Journal on Optimization*, 13(1):117–156, 2002.

[9] L. Galli, S. Martello, C. Rey, and P. Toth. Polynomial-size formulations and relaxations for the quadratic multiple knapsack problem. *European Journal of Operational Research*, 291(3):871–882, 2021.

[10] C. García-Martínez, F. Glover, F.J. Rodriguez, M. Lozano, and R. Martí. Strategic oscillation for the quadratic multiple knapsack problem. *Computational Optimization and Applications*, 58(1):161–185, 2014.

[11] C. García-Martínez, F.J. Rodriguez, and M. Lozano. Tabu-enhanced iterated greedy algorithm: a case study in the quadratic multiple knapsack problem. *European Journal of Operational Research*, 232(3):454–463, 2014.

[12] A. Hiley and B.A. Julstrom. The quadratic multiple knapsack problem and three heuristic approaches to it. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 547–552, 2006.

[13] R.M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[14] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer Nature Book Archives Millennium. Springer, 2004.

[15] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., New York, NY, USA, 1990.

[16] J. Qin, X. Xu, Q. Wu, and T.C.E. Cheng. Hybridization of tabu search with feasible and infeasible local searches for the quadratic multiple knapsack problem. *Computers & Operations Research*, 66:199–214, 2016.

[17] T. Saraç and A. Sipahioglu. Generalized quadratic multiple knapsack problem and two solution approaches. *Computers & Operations Research*, 43(1):78–89, 2014.

[18] A. Singh and A.S. Baghel. A new grouping genetic algorithm for the quadratic multiple knapsack problem. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 210–218. Springer, 2007.

[19] SM. Soak and SW. Lee. A memetic algorithm for the quadratic multiple container packing problem. *Applied Intelligence*, 36(1):119–135, 2012.

[20] S. Sundar and A. Singh. A swarm intelligence approach to the quadratic multiple knapsack problem. In *International Conference on Neural Information Processing*, pages 626–633. Springer, 2010.

[21] T. Tlili, H. Yahyaoui, and S. Krichen. An iterated variable neighborhood descent hyperheuristic for the quadratic multiple knapsack problem. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing 2015*, pages 245–251. Springer, 2016.

Appendix: Local Search procedures

Algorithm 7: Procedure BestExchange

input : A feasible QMKP solution x
output: A possibly improved QMKP solution x'

```
1  Stop := false;
2  while Stop = false do
3  |   Stop := true;
4  |   for i := 1 to n − 1 do
5  |   |   Δ_i := 0;
6  |   |   for j := i + 1 to n do
7  |   |   |   if K(j) ≠ K(i) and the i − j exchange is feasible and profitable
8  |   |   |   then Δ_i := Δ, j* := j;
9  |   |   end
10 |   |   if Δ_i > 0 then
11 |   |   |   exchange items i and j* and update the relative variables;
12 |   |   |   Stop := false;
13 |   |   end
14 |   end
15 end
```

Algorithm 8: Procedure BestReloc

input : A feasible QMKP solution x
output: A possibly improved QMKP solution x'

```
1  Stop := false;
2  while Stop = false do
3  |  Stop := true;
4  |  for i := 1 to n do
5  |  |  Δ_i := 0;
6  |  |  foreach k ∈ {1,...,m} \ {K(i)} do
7  |  |  |  if the relocation of item i to knapsack k is feasible and profitable
8  |  |  |  then Δ_i := Δ, k* := k;
9  |  |  end
10 |  |  if Δ_i > 0 then
11 |  |  |  relocate item i to knapsack k* and update the relative variables;
12 |  |  |  Stop := false;
13 |  |  end
14 |  end
15 end
```

Algorithm 9: Procedure InfRelocA

input : A feasible or infeasible QMKP solution x',
a knapsack index $k \in \{1,\dots,m\}$
output: A possibly improved (feasible or infeasible) QMKP solution $\overline{x}$

```
1  x̄ := x', Maxgain := 0;
2  for κ := 1 to m do
3  |  if κ ≠ k and W(κ) ≥ W(k) then
4  |  |  foreach i ∈ X(κ) do
5  |  |  |  Δ^κ := Σ_{ℓ∈X(κ)} p_iℓ,  Δ^k := Σ_{ℓ∈X(k)} p_iℓ,  gain := (Δ^k − Δ^κ)/w_i^β;
6  |  |  |  if gain > Maxgain then Maxgain := gain, i* := i
7  |  |  end
8  |  end
9  end
10 if Maxgain > 0 then
11 |  relocate item i* to knapsack k and define the relative solution x̄
12 end
```

Algorithm 10: Procedure InfRelocB

input : A feasible or infeasible QMKP solution x',
a knapsack index $k \in \{1, \ldots, m\}$
output: A possibly improved (feasible or infeasible) QMKP solution $\overline{x}$

```
1  x̄ := x', Maxgain := 0;
2  foreach i ∈ X(k) do
3  |  Δ^k := Σ_{ℓ∈X(k)} p_{iℓ};
4  |  for κ := 1 to m do
5  |  |  if κ ≠ k and W(κ) < W(k) then
6  |  |  |  Δ^κ := Σ_{ℓ∈X(κ)} p_{iℓ},   gain := (Δ^κ − Δ^k) / w_i^β;
7  |  |  |  if gain > Maxgain then Maxgain := gain, i* := i, k* := κ
8  |  |  end
9  |  end
10 end
11 if Maxgain > 0 then
12 |  relocate item i* to knapsack k* and define the relative solution x̄
13 end
```

Algorithm 11: Procedure Repair

input : An infeasible QMKP solution x'
output: A possibly worse feasible QMKP solution $\overline{x}$

```
1  while x' is infeasible do
2      σ* := +∞, Δ* := 0;
3      foreach i ∈ {1,...,n} \ X(m+1) do
4          compute surplus σ and profit variation Δ produced by removing i;
5          if (σ < σ*) or (σ = σ* and Δ > Δ*)
6          then σ* := σ, Δ* := Δ, move := remove(i);
7      end
8      foreach item pair (i,j) with i,j ∈ {1,...,n}, j > i and K(j) ≠ K(i) do
9          compute surplus σ and profit variation Δ produced by exchanging i and j;
10         if (σ < σ*) or (σ = σ* and Δ > Δ*)
11         then σ* := σ, Δ* := Δ, move := exchange(i,j);
12     end
13     foreach item-knapsack pair (i,k) with i ∈ {1,...,n} \ X(m+1)
14                                       and k ∈ {1,...,m} \ K(i)      do
15         compute surplus σ and profit variation Δ produced by relocating i into k;
16         if (σ < σ*) or (σ = σ* and Δ > Δ*)
17         then σ* := σ, Δ* := Δ, move := relocate(i,k);
18     end
19     update solution x' by executing move
20 end
21 x̄ := x'
```