Computers & Operations Research 173 (2025) 106873

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

The quadratic knapsack problem with setup

Laura Galli [a], Silvano Martello [b,*], Carlos Rey [c], Paolo Toth [b]

[a] Department of Mathematics, Alma Mater Studiorum Università di Bologna, Italy
[b] DEI "Guglielmo Marconi", Alma Mater Studiorum Università di Bologna, Italy
[c] Department of Industrial Engineering, Universidad del Bío-Bío, Concepción, 4030000, Chile

ARTICLE INFO

Keywords:
Quadratic knapsack problem
Setup constraints
Matheuristic algorithms
Local search

ABSTRACT

The Quadratic Knapsack Problem is a well-known generalization of the classical 0-1 knapsack problem, in which any pair of items produces a pairwise profit if both are selected. Another relevant generalization of the knapsack problem is the Knapsack Problem with Setup, in which the items are partitioned into classes, the items of a class can only be inserted into the knapsack if the corresponding class is activated, and activating a class involves a setup cost and a setup capacity reduction.

Despite a rich literature on these two problems, their obvious generalization, i.e., the Quadratic Knapsack Problem with Setup, was never investigated so far. We discuss applications, mathematical models, deterministic matheuristic algorithms, and computationally evaluate their performance.

1. Introduction

The *Quadratic Knapsack Problem* (QKP) has been intensively studied since its definition in 1980 (see Gallo et al. (1980)). As in the famous (0–1) *Knapsack Problem* (KP), we are given a knapsack with positive integer *capacity* c, a set of *items* $N = \{1, \dots, n\}$ having a non-negative integer *profit* p_i and a positive integer *weight* w_i ($i \in N$). In addition, in the QKP each unordered pair of distinct items $\{i, j\}$ produces a non-negative integer *pairwise profit* p_{ij} if both items are inserted into the knapsack. Both the KP and the QKP consist of selecting a subset of items, whose overall weight does not exceed the capacity, that maximizes an appropriate profit function. While in the KP such function is just the sum of the profits of the selected items, in the QKP it also includes the sum of their pairwise profits. The KP is thus the special case of the QKP arising when $p_{ij} = 0$ for all $i < j$. Several effective exact algorithms (see, e.g., Billionnet and Soutif (2004a,b), Caprara et al. (1999), Pisinger et al. (2007) and Fomeni et al. (2022)) and metaheuristics (see, e.g., Glover et al. (2002), Yang et al. (2013) and Chen and Hao (2017)) have been proposed for the QKP.

Another relevant generalization of the KP, which has been thoroughly investigated in the literature, is the *Knapsack Problem with Setup* (KPS), introduced in 1994 by Chajakis and Guignard (1994). In the KPS, the items are partitioned into r *classes* $R_h \subseteq N$, $h \in R = \{1, \dots, r\}$, and the items of a class can only be inserted into the knapsack if the corresponding class is activated. Activating a class h involves a non-negative integer *setup reduction* d_h of the knapsack capacity and a non-negative activation *setup cost* f_h. (Slightly different definitions can be encountered in the literature.) The objective is to select a subset of items maximizing the difference between item profits and class setup costs, such that their total weight does not exceed the difference between the capacity and the class setup reductions. Several effective exact algorithms (see, e.g., Chebil and Khemakhem (2015), della Croce et al. (2017) and Furini et al. (2018)) and heuristics (see, e.g., Khemakhem and Chebil (2016) and Amiri (2020)) have been proposed for the KPS.

As previously mentioned, there is a rich literature on both the QKP and the KPS. Quite surprisingly instead, the obvious generalization of both problems, i.e., the *Quadratic Knapsack Problem with Setup* (QKPS), to the best of our knowledge, was never investigated so far. The QKPS inherits all input parameters of the two problems above. The constraints are the same as the KPS, while the objective is to find a subset of items maximizing the difference between the total (individual and pairwise) item profit and the class setup cost.

The QKPS can be modeled as the following *0–1 Quadratic Program*

$$\text{(01QP)} \quad \max \sum_{i \in N} p_i x_i + \sum_{i \in N} \sum_{j \in N : i<j} p_{ij} x_i x_j - \sum_{h \in R} f_h y_h \tag{1}$$

$$\text{s.t.} \quad \sum_{i \in N} w_i x_i + \sum_{h \in R} d_h y_h \le c \tag{2}$$

$$x_i \le y_h \qquad (h \in R;\ i \in R_h) \tag{3}$$

$$x_i \in \{0, 1\} \qquad (i \in N) \tag{4}$$

$$y_h \in \{0, 1\} \qquad (h \in R), \tag{5}$$

* Corresponding author.
E-mail addresses: l.galli@unibo.it (L. Galli), silvano.martello@unibo.it (S. Martello), crey@ubiobio.cl (C. Rey), paolo.toth@unibo.it (P. Toth).

https://doi.org/10.1016/j.cor.2024.106873
Received 16 July 2024; Received in revised form 29 September 2024; Accepted 14 October 2024
Available online 21 October 2024

where x_i takes the value one iff item i is selected, and variable y_h takes the value one iff class h is activated. The objective function (1) is given by the sum of the individual and pairwise profits of the selected items, minus the setup costs of the classes they belong to. Inequality (2) imposes the capacity constraint on the total weight of the selected items, plus the setup reductions of their classes. Constraints (3) impose the activation of the necessary classes.

The QKP is the special case of (1)–(5) arising when $r = 1$ and $f_1 = d_1 = 0$. (Note that in this case the auxiliary variable y_1 and the corresponding linking constraints (3) are not needed.) It is known that the QKP is strongly $\mathcal{NP}$-hard (see, e.g., Gallo et al. (1980)), from which the same complexity applies to the QKPS.

Another special case is obtained when $p_{ij} = 0$ for $i, j = 1, \dots, n$ ($i < j$), in which case (1)–(5) corresponds to the KPS. If, in addition, $r = 1$ and $f_1 = d_1 = 0$, model (1), (2), (4) describes the KP. Both the KPS and the KP are weakly $\mathcal{NP}$-hard and can be solved in pseudo-polynomial time.

The QKP has a number of real-world applications, explored since the Seventies in different fields. Rhys (1970) modeled as a QKP the problem of selecting freight handling terminals to install, when the existence of a pair of terminals permits a service to be operated between them, and each service is associated with a revenue. Similar applications concerned electronic message handling stations, see Witzgall (1975). Typical applications of the KPS arise in machine scheduling, in particular when dealing with make-to-order production contexts in the management of different product categories (see, e.g., Chajakis and Guignard (1994)). The QKPS combines the QKP and the KPS, and can arise in applications like the one described in Page (2018), concerning the design of an auction to allocate the electromagnetic spectrum for cellular phones. The radio signal from a cell tower covers a geographic range, and the government sells licences for specific regions to different companies. The value of a licence clearly depends on the other licences acquired by the company, as the value of a licence for a certain region would be worth to a company that also acquires licences of neighboring regions. Indeed, holding neighboring licences implies, e.g., lower construction costs and common advertising media. The economists refer to these interdependent valuations as *externalities*. Let the classes correspond to regions (the setup costs being the licence prices), the items to infrastructures (e.g., cell towers), linear profits to the values of covering specific areas, and pairwise profits to their externalities. If the item weights measure the amount of resources (personnel, technological devices, etc.) needed by the cell towers, and the setup reductions and capacity represent, respectively, the amount of resources needed to open up the activities in a specific region and the overall resource budget available, the resulting optimization problem can be modeled as a QKPS.

The purpose of this paper is to provide a first study on the QKPS. The paper is structured as follows. Section 2 provides linear reformulations of the QKPS. Sections 3 and 4 present deterministic matheuristic methods, that proved to have a good computational performance, especially when the solutions they produce are refined through the local search procedures described in Section 5. Section 6 presents the results of extensive computational experiments. Finally, Section 7 provides some concluding remarks.

2. Reformulation linearization

The *Reformulation Linearization Technique* (RLT) was originally introduced by Adams and Sherali (1986) for 0–1 quadratic problems, and later extended by the same authors (Sherali and Adams, 1990) to general 0–1 problems. The basic underlying idea can however be found in earlier studies on linear formulations of the *quadratic assignment problem* dating back to the Sixties (see, e.g., the monograph by Burkard et al. (2012), Section 7.3, for a comprehensive treatment).

The RLT consists of a methodology to strengthen the continuous relaxation of 0–1 quadratic models through the addition of constraints obtained by multiplying the original (linear) constraints by the model binary variables (and, possibly, by their complement). The resulting quadratic constraints and the original quadratic objective function are then linearized in a standard way, by introducing new binary variables and the corresponding linking constraints to replace the quadratic terms. The method was adapted to the QKP by Caprara et al. (1999), and later extended to constrained 0–1 quadratic problems and generalized quadratic assignment problems by Caprara (2008) and Guignard (2020), respectively.

An RLT model for the QKPS can be obtained as follows. Let $h(i)$ denote the class index to which item i belongs ($i \in R_{h(i)}$). We consider capacity constraint (2), and obtain two groups of new constraints by multiplying it by both variables x_j ($j \in N$) and y_k ($k \in R$):

- the former group,

$$\sum_{i \in N; i<j} w_i z_{ij} + \sum_{i \in N; j<i} w_i z_{ji} + \sum_{h \in R \setminus \{h(j)\}} d_h q_{hj} \le (c - w_j - d_{h(j)}) x_j \quad (j \in N), \tag{6}$$

  contains two families of quadratic terms, respectively represented by auxiliary binary variables

  $z_{ij} = x_i x_j \quad (i, j \in N; i < j)$

  and

  $q_{hj} = y_h x_j \quad (h \in R, j \in N \setminus R_h).$

  Note that, for a binary variable x_j, we have $x_j^2 = x_j$, while $y_{h(j)} x_j = x_j$ follows from $x_j \le y_{h(j)}$, see (3);
- similarly, in the latter group,

$$\sum_{i \in N \setminus R_k} w_i q_{ki} + \sum_{h \in R; h<k} d_h t_{hk} + \sum_{h \in R; k<h} d_h t_{kh} \le (c - d_k) y_k - \sum_{i \in R_k} w_i x_i \quad (k \in R), \tag{7}$$

  the quadratic terms are represented, respectively, by auxiliary binary variables q_{hj} and

  $t_{hk} = y_h y_k \quad (h, k \in R; h < k)$

  (as $y_h^2 = y_h$).

By adding the linking constraints (9)–(11) we obtain the *Integer Linear Programming* (ILP) model

$$\text{(RLT)} \quad \max \sum_{i \in N} p_i x_i + \sum_{i,j \in N; i<j} p_{ij} z_{ij} - \sum_{h \in R} f_h y_h \tag{8}$$

$$\text{s.t.} \quad (2), (3), (4), (5), (6), (7)$$

$$z_{ij} \le x_i, \quad z_{ij} \le x_j, \quad z_{ij} \ge x_i + x_j - 1 \qquad (i, j \in N; i < j) \tag{9}$$

$$q_{hj} \le y_h, \quad q_{hj} \le x_j, \quad q_{hj} \ge y_h + x_j - 1 \qquad (h \in R; j \in N \setminus R_h) \tag{10}$$

$$t_{hk} \le y_h, \quad t_{hk} \le y_k, \quad t_{hk} \ge y_h + y_k - 1 \qquad (h, k \in R; h < k) \tag{11}$$

$$z_{ij} \in \{0, 1\} \qquad (i, j \in N; i < j) \tag{12}$$

$$q_{hj} \in \{0, 1\} \qquad (h \in R; j \in N \setminus R_h) \tag{13}$$

$$t_{hk} \in \{0, 1\} \qquad (h, k \in R; h < k). \tag{14}$$

We next present two ways to obtain alternative RLT models which computationally proved to be more effective for the QKPS.

2.1. Revised reformulation RLT1

First observe that, if $p_{ij} = 0$, then one can obtain an equivalent solution by setting $z_{ij} = 0$, and hence we can define variables z_{ij} only for those pairs $\{i, j\}$ $(i, j \in N)$ for which $i < j$ and $p_{ij} > 0$. The second observation is that the RLT-capacity constraints (6) and (7) are only introduced to strengthen the *Linear Programming* (LP) relaxation, but are redundant for the 01-LP RLT model, so they can be omitted in our analysis. Since variables z_{ij} appear with a strictly positive coefficient in objective function (8), in any optimal 01-LP solution (x^*, y^*, z^*) such variables will take the largest possible value allowed by (9), i.e., $z^*_{ij} = \min\{x^*_i, x^*_j\}$. Since $x^*_i \in \{0, 1\}$ $\forall i \in N$, then $\min\{x^*_i, x^*_j\} \geq x^*_i + x^*_j - 1$. This immediately proves that constraints $z_{ij} \geq x_i + x_j - 1$ in (9) are redundant, while $z_{ij} \in \{0, 1\}$ in (12) can be replaced with $z_{ij} \geq 0$. Note, however, that the above constraints may tighten the LP relaxation of the RLT model.

By adopting the notations

$$N^<(i) = \{j \in N : i < j \text{ and } p_{ij} > 0\},$$
$$N^>(i) = \{j \in N : i > j \text{ and } p_{ji} > 0\} \quad (i \in N),$$

we modify (6) as

$$\sum_{i \in N; j \in N^<(i)} w_i z_{ij} + \sum_{i \in N; j \in N^>(i)} w_i z_{ji} + \sum_{h \in R \setminus \{h(j)\}} d_h q_{hj} \leq (c - w_j - d_{h(j)}) x_j (j \in N) \quad (15)$$

thus obtaining the revised model RLT1

$$\text{(RLT1)} \quad \max \sum_{i \in N} p_i x_i + \sum_{i \in N; j \in N^<(i)} p_{ij} z_{ij} - \sum_{h \in R} f_h y_h \quad (16)$$

s.t. (2), (3), (4), (5), (7), (15)

$$z_{ij} \leq x_i, \quad z_{ij} \leq x_j, \qquad (i \in N;\ j \in N^<(i)) \quad (17)$$

$$q_{hj} \leq y_h, \quad q_{hj} \leq x_j, \quad q_{hj} \geq y_h + x_j - 1 \quad (h \in R;\ j \in N \setminus R_h) \quad (10)$$

$$t_{hk} \leq y_h, \quad t_{hk} \leq y_k, \quad t_{hk} \geq y_h + y_k - 1 \qquad (h, k \in R;\ h < k) \quad (11)$$

$$z_{ij} \geq 0 \qquad (i \in N;\ j \in N^<(i)) \quad (18)$$

$$q_{hj} \in \{0, 1\} \qquad (h \in R;\ j \in N \setminus R_h) \quad (13)$$

$$t_{hk} \in \{0, 1\} \qquad (h, k \in R;\ h < k). \quad (14)$$

2.2. Revised reformulation RLT2

We next show that reformulation RLT1 of the previous section can be further modified by:

(i) eliminating the first two inequalities in constraints (10) and (11), and

(ii) relaxing constraints (13) and (14) by imposing that variables q_{hj} and t_{hk} take non-negative values without changing the optimal value of the LP relaxation. Indeed,

- variables q_{hj} and t_{hk} do not appear in the objective function (16) and can only affect its value by increasing the left-hand-side of the RLT capacity constraints (15) and (7), respectively. It follows that it is superfluous to impose an upper bound on them as their increase reduces the solution space thus possibly decreasing the solution value;
- the previous consideration also shows that the third inequality in constraints (10) (resp. (11)) forces q_{hj} (resp. t_{hk}) to take at least the value one when $y_h = x_j = 1$ (resp. $y_h = y_k = 1$). It additionally follows that constraints (13) and (14) can be further relaxed to only impose non-negativity of variables q_{hj} and t_{hk}.

The new reformulation follows:

$$\text{(RLT2)} \quad \max \sum_{i \in N} p_i x_i + \sum_{i \in N; j \in N^<(i)} p_{ij} z_{ij} - \sum_{h \in R} f_h y_h \quad (16)$$

s.t. (2), (3), (4), (5), (7), (15)

$$z_{ij} \leq x_i, \quad z_{ij} \leq x_j \qquad (i \in N;\ j \in N^<(i)) \quad (17)$$

$$q_{hj} \geq y_h + x_j - 1 \qquad (h \in R;\ j \in N \setminus R_h) \quad (19)$$

$$t_{hk} \geq y_h + y_k - 1 \qquad (h, k \in R;\ h < k) \quad (20)$$

$$z_{ij} \geq 0 \qquad (i \in N;\ j \in N^<(i)) \quad (18)$$

$$q_{hj} \geq 0 \qquad (h \in R;\ j \in N \setminus R_h) \quad (21)$$

$$t_{hk} \geq 0 \qquad (h, k \in R;\ h < k). \quad (22)$$

It was pointed out in Section 2.1 that, due to the elimination of redundant constraints $z_{ij} \geq x_i + x_j - 1$ in (9), the LP relaxation of RLT1 can be weaker than that provided by the original RLT. It is worth observing that the solutions produced by the LP relaxations of RLT1 and RLT2 have the same value.

3. Basic matheuristic tools

Consider again model (1)–(5), and observe that, by fixing the value of the auxiliary variables y_h, the capacity is decreased by the setup reductions of the selected classes (i.e., those for which $y_h = 1$) and hence the solution to the resulting QKP instance provides a feasible solution to the QKPS.

This idea is exploited in the solution methods presented in the following. The main engine within the proposed procedures is indeed an algorithm for the optimal solution to the QKP, for which, as already mentioned, a number of exact methods are available in the literature.

One of the most competitive QKP exact algorithms is that proposed by Caprara et al. (1999), for which an effective C implementation, `quadknap`, can be freely downloaded from *David Pisinger's optimization codes* page

http://hjemmesider.diku.dk/~pisinger/quadknap.c

In the following, we will invoke such code through the statement

$$([\bar{x}], \bar{z}) \leftarrow \texttt{quadknap}(\bar{I}, \bar{c})$$

where

- $\bar{I}$ denotes a subset of input items (including their profits and weights);
- $\bar{c}$ is a capacity value ($\bar{c} \leq c$);
- $[\bar{x}]$ provides the solution array computed by `quadknap` (where, for $i \in \bar{I}$, $\bar{x}_i = 1$ if item i is selected, and $\bar{x}_i = 0$ otherwise), and $\bar{z}$ the corresponding solution value.

The main procedure based on `quadknap` determines a feasible solution to a QKPS instance by only considering a subset of classes and a capacity defined by the original capacity c minus the setup reductions of such classes. Let `Q` denote a QKPS instance (defined by all parameters introduced in Section 1). For a given $S \subseteq R$, procedure `QKnaP`, shown in Algorithm 1, defines a QKP instance consisting of the items belonging to the classes in S and the corresponding reduced capacity, solves it through `quadknap` (i.e., by disregarding the setup

costs), and reconstructs a QKPS solution by defining the solution arrays $([x], [y])$ and the solution value z induced by $[\bar{x}]$.

Algorithm 1: Procedure QKnaP

input : A QKPS instance Q, a set $S \subseteq R$
output: $([x], [y], z) \equiv$ A feasible solution $[x], [y]$ to Q, and its value z

$\bar{I} := \cup_{h \in S} R_h$, $\bar{c} := c - \sum_{h \in S} d_h$;
if $\min\{w_i : i \in \bar{I}\} > \bar{c}$ **then** $([\bar{x}] := 0, \bar{z} := 0)$ **else** $([\bar{x}], \bar{z}) \leftarrow$ quadknap$(\bar{I}, \bar{c})$;
define the QKPS solution $([x], [y])$ corresponding to $([\bar{x}])$;
$z := \bar{z} - \sum_{h \in S} f_h y_h$.

Besides being used in the greedy algorithms presented in Section 4, and in the local search procedures described in Section 5, procedure QKnap will be used to construct an initial feasible solution, as shown in the next section.

The time complexity of QKnaP is in general non polynomial, as it invokes quadknap to solve a QKP instance. It is however possible to execute quadknap with a prefixed (constant) limit on the CPU time (as we did in all our implementations). In our analysis of the time required by the presented procedures, we will thus evaluate the number of calls to quadknap.

3.1. Initialization

First observe that any instance of the QKPS can be initially reduced by eliminating (useless) classes h, if any, for which $\min\{w_i : i \in R_h\} > c - d_h$. A feasible solution to the reduced instance can then be obtained by invoking QKnaP for each individual class, and retaining the best solution found. Procedure Initialize, shown in Algorithm 2, computes in addition, for each class h, a profit-to-weight *score value*.

$$V_h = \frac{P_h}{\sum_{i \in R_h} w_i x_i^* + d_h} \tag{23}$$

given by the ratio between the profit P_h determined by QKnaP and the total weight (in the solution $[x^*]$ returned by QKnaP) of the selected items, increased by the corresponding setup reduction of class h. Initialize invokes quadknap $O(r)$ times.

In order to reduce the number of executions of quadknap, which is the most time consuming component of our algorithms, a globally defined family Σ stores all class sets for which QKnaP was invoked. Since the QKPS solution produced by QKnaP(Q, S) only depends on the composition of class set S, QKnaP is not executed if S belongs to family Σ, which is initialized in Algorithm 2 with all single classes and used/updated by all procedures presented in the following.

Algorithm 2: Procedure Initialize

input : A QKPS instance Q
output: $([x], [y], z, [V], [P]) \equiv$ A feasible solution $[x], [y]$ to Q, and its value z,
an array $[V]$ of profit-to-weight score values for each class
an array $[P]$ of solution values for each single class

$z := 0$, $[x] := \mathbf{0}$, $[y] := \mathbf{0}$;
for $h := 1$ **to** r **do**
  $([x^*], [y^*], P_h) \leftarrow$ QKnaP(Q, $\{h\}$);
  if $P_h > z$ **then** $z := P_h$, $[x] := [x^*]$, $[y] := [y^*]$;
  $V_h := P_h / (\sum_{i \in R_h} w_i x_i^* + d_h)$
end
$\Sigma := \{\{1\}, \{2\}, \ldots, \{r\}\}$.

4. Greedy algorithms

Once an instance has been initialized, an improved solution can be computed by procedure Greedy, shown in Algorithm 3. The classes are ordered by non-increasing value of their score V_h, and a set S of *solution classes* is

(i) initialized with the class having the highest score value, and

(ii) iteratively augmented by adding a class at a time and invoking QKnaP: classes which improve the incumbent solution value z are permanently added to S.

Procedure Greedy invokes Initialize once, and then, for $r - 1$ times, it invokes quadknap, i.e., the overall number of calls to quadknap is $O(r)$.

Algorithm 3: Procedure Greedy

input : A QKPS instance Q
output: $([x], [y], z) \equiv$ A feasible solution $[x], [y]$ to Q, and its value z

$([x], [y], z, [V], [P]) \leftarrow$ Initialize(Q);
sort the classes by non-increasing V_h value and update the indices;
$S := \{1\}$;
for $h := 2$ **to** r **do**
  $S^* := S \cup \{h\}$, $\Sigma := \Sigma \cup S^*$;
  $([x^*], [y^*], z^*) \leftarrow$ QKnaP(Q, S^*);
  if $z^* > z$ **then** $S := S^*$, $z := z^*$, $[x] := [x^*]$, $[y] := [y^*]$
end

Preliminary computational experiments proved that ordering the classes by non-increasing value of scores V_h provides better solutions than those obtained by ordering the classes by non-increasing value of profit P_h.

Note that, for the case of two classes ($r = 2$), procedure Greedy evaluates all possible subsets of classes, and hence it produces the optimal solution, provided the time limit assigned to the QKP code quadknap allows it to complete the three executions.

A refined, although more time consuming, algorithm, which can provide better solutions, is shown in Algorithm 4. Procedure Refined Greedy, determines the class h^* with maximum score value V_{h^*} and initializes a local incumbent solution value $\bar{z}$ to P_{h^*} and a set of "candidate" classes S to h^*. It then iteratively checks the effect of adding one additional class h at a time, by invoking QKnaP for the resulting family S^*. If the value Π_h of the corresponding solution improves the local incumbent solution value $\bar{z}$, a *relative score value* is computed as

$$W_h := \frac{\Pi_h}{\sum_{l \in S^*} \left(\sum_{i \in R_l} w_i x_i^* + d_l \right)} \tag{24}$$

and set to zero otherwise. (The overall incumbent solution, of value z, is also possibly updated.)

Once all classes have been checked, the one with maximum relative score value is permanently added to S, and the process is iterated for the remaining candidate classes $R \setminus S$. The procedure terminates when none of the candidate classes improves the local incumbent solution.

At each **while** iteration, the cardinality of $R \setminus S$ decreases by one (if not, a **return** occurs). Therefore, at most $O(r)$ iterations are performed, each requiring $O(r)$ calls to quadknap, for a total of $O(r^2)$ quadknap executions.

5. Local search procedures

The solutions provided by the greedy algorithms can be improved through local search. Traditional local search methods to knapsack-like problems rely on removals, insertions, and exchanges of *items*. The algorithms we propose in the present section consider instead

Algorithm 4: Procedure `Refined Greedy`

input : A QKPS instance $\mathtt{Q}$
output: $([x],[y],z) \equiv$ A feasible solution $[x],[y]$ to $\mathtt{Q}$, and its value z

$([x],[y],z,[V],[P]) \leftarrow$ `Initialize`($\mathtt{Q}$);
$h^* := \arg\max_{h\in R}\{V_h\}$, $S := \{h^*\}$;
if $P_{h^*} > 0$ **then** $\bar{z} := P_{h^*}$ **else** $\bar{z} := 0$;
while $R \setminus S \neq \emptyset$ **do**
  foreach $h \in R \setminus S$ **do**
    $S^* := S \cup \{h\}$;
    $\Sigma := \Sigma \cup S^*$;
    $([x^*],[y^*],\Pi_h) \leftarrow$ `QKnaP`($\mathtt{Q}$, S^*);
    if $\Pi_h > \bar{z}$ **then**
      $W_h := \Pi_h / \sum_{l\in S^*}(\sum_{i\in R_l} w_i x_i^* + d_l)$;
      if $\Pi_h > z$ **then** $z := \Pi_h$, $[x] := [x^*]$, $[y] := [y^*]$;
    else
      $W_h := 0$;
    end
  end
  $\bar{h} := \arg\max_{h\in R\setminus S}\{W_h\}$;
  if $\Pi_{\bar{h}} > \bar{z}$ **then** $\bar{z} := \Pi_{\bar{h}}$, $S := S \cup \{\bar{h}\}$ **else return**
end

removals and exchanges of *classes*. For each move, a series of calls to procedure `QKnaP` (and hence to code `quadknap`) looks for a QKPS solution improving the incumbent one: each execution of `quadknap` is halted whenever an upper bound computation excludes the possibility of improving on the incumbent solution.

5.1. Removals

Procedure `Remove`, shown in Algorithm 5, receives a feasible solution $[x]$, $[y]$, z (*global incumbent*) and initializes to it a *local incumbent* $[\hat{x}]$, $[\hat{y}]$, $\hat{z}$. It then iteratively excludes one class, $\bar{h}$, from the set of classes which are selected in the current solution, and computes a (possibly improved) solution by invoking `QKnaP` for the remaining selected classes. The local incumbent is updated whenever an improved solution is obtained. While the final local incumbent improves on the global incumbent, the process is iterated.

The **foreach** loop invokes procedure `QKnap` at most r times, each requiring a single call to `quadknap`. After every **foreach** execution, either the stop condition of the **while** loop becomes true or the number of classes h with $y_h = 1$ decreases by one. It follows that the **foreach** loop is executed $O(r)$ times, yielding an overall $O(r^2)$ calls to `quadknap`.

5.2. Exchanges

The general structure of procedure `Exchange` (shown in Algorithm 6), is similar to that of `Remove`. The input solution $[x]$, $[y]$, z (global incumbent) is used to initialize a local incumbent $[\hat{x}]$, $[\hat{y}]$, $\hat{z}$. For each different pair of classes such that one is currently selected and one not, the former (h^1) is excluded and replaced with the latter (h^0): `QKnaP` is then invoked to determine the corresponding QKPS solution, possibly updating the local incumbent. On exit from the nested **foreach** loop: if the global incumbent can be updated, the **while** loop iterates the process; otherwise, the procedure terminates.

The nested **foreach** loop invokes `QKnap` $O(r^2)$ times. Whenever the value of the global incumbent is increased, the **while** loop is iterated, and hence, in the worst case, `Exchange` can require a pseudo-polynomial number of `quadknap` executions (in which case, however, one should expect a very good solution, due to a large number of improvements).

Algorithm 5: Procedure `Remove`

input : A QKPS instance $\mathtt{Q}$, a feasible solution $([x],[y])$ to $\mathtt{Q}$, and its value z
output: A possibly improved QKPS solution $([x],[y])$, and its value z

$\hat{z} := z$, $[\hat{x}] := [x]$, $[\hat{y}] = [y]$;
Stop := `false`;
while *Stop* = `false` **do**
  foreach $\bar{h} \in R : y_{\bar{h}} = 1$ **do**
    $S := \{h \in R : y_h = 1\} \setminus \{\bar{h}\}$;
    if $S \notin \Sigma$ **then** $\Sigma := \Sigma \cup S$ **else continue**;
    $([\bar{x}],[\bar{y}],\bar{z}) \leftarrow$ `QKnaP`($\mathtt{Q}$, S);
    if $\bar{z} > \hat{z}$ **then** $\hat{z} := \bar{z}$, $[\hat{x}] := [\bar{x}]$, $[\hat{y}] := [\bar{y}]$;
  end
  if $\hat{z} > z$ **then** $z := \hat{z}$, $[x] := [\hat{x}]$, $[y] := [\hat{y}]$ **else** *Stop* := `true`
end

Algorithm 6: Procedure `Exchange`

input : A QKPS instance $\mathtt{Q}$, a feasible solution $([x],[y])$ to $\mathtt{Q}$ and its value z
output: A possibly improved QKPS solution $([x],[y])$ and its value z

Stop := `false`;
$\hat{z} := z$, $[\hat{x}] := [x]$, $[\hat{y}] := [y]$;
while *Stop* = `false` **do**
  foreach $h^1 \in R : y_{h^1} = 1$ **do**
    foreach $h^0 \in R : y_{h^0} = 0$ **do**
      $S := \{h \in R : y_h = 1\} \setminus \{h^1\} \cup \{h^0\}$;
      if $S \notin \Sigma$ **then** $\Sigma := \Sigma \cup S$ **else continue**;
      $([\bar{x}],[\bar{y}],\bar{z}) \leftarrow$ `QKnaP`($\mathtt{Q}$,S);
      if $\bar{z} > \hat{z}$ **then** $\hat{z} := \bar{z}$, $[\hat{x}] := [\bar{x}]$, $[\hat{y}] := [\bar{y}]$
    end
  end
  if $\hat{z} > z$ **then** $z := \hat{z}$, $[x] := [\hat{x}]$, $[y] := [\hat{y}]$ **else** *Stop* := `true`
end

6. Computational experiments

All algorithms presented in the previous sections were implemented in `C++` language and compiled with `g++`, but the `quadknap` solver, which was compiled with the GNU Compiler GCC. All experiments were executed on a single thread of a 13th Gen Intel Core i9-13900KF 64 GB Processor running at 3.00 GHz, and operating under Ubuntu 22.04 LTS. The models were solved using solvers CPLEX 22.11 and Gurobi 10.02.

6.1. Benchmarks

As the QKPS is a novel problem, we derived benchmarks instances by combining the main generation schemes adopted for its underlying combinatorial problems (the QKP and the KPS).

Following the classical scheme originally proposed by Gallo et al. (1980) for the QKP, we used four values $d \in (0,1]$ for the density of the non-zero quadratic terms: $d \in \{0.25, 0.50, 0.75, 1.00\}$. Linear profits p_i $(i = 1,\dots,n)$ and pairwise profits p_{ij} $(i = 1,\dots,n-1; j = i+1, n)$ were generated with probability d as random integer values uniformly drawn from $[1, 100]$, and set to 0 with probability $1-d$. Weights w_i $(i = 1,\dots,n)$ were generated as uniformly random integers from $[10, 100]$, while the capacity c was set to $[0.5\sum_{i=1}^{n} w_i]$.

Since the KPS is a weakly $\mathcal{NP}$-hard problem, "easy" to solve in practice even for very large-size instances, the number of items and the number of classes commonly used in the literature (with order of

Table 1
Models: small/medium size instances. 5 instances per entry; average Optimality gap and CPU time; number of optima.

Instance			01QP(CPLEX)			01QP(Gurobi)			RLT(CPLEX)			RLT(Gurobi)			RLT1(Gurobi)			RLT2(CPLEX)		
r	*n*	*d*	%Gap	Time	#opt	%Gap	Time	#opt	%Gap	Time	#opt	%Gap	Time	#opt	%Gap	Time	#opt	%Gap	Time	#opt
2	50	0.25	0.00	0.07	5	0.00	0.13	5	0.00	6.38	5	0.00	1.62	5	0.00	0.17	5	0.00	0.08	5
2	50	0.50	0.00	0.14	5	0.00	0.24	5	0.00	4.27	5	0.00	1.45	5	0.00	0.38	5	0.00	0.14	5
2	50	0.75	0.00	58.34	5	0.00	0.39	5	0.00	2.12	5	0.00	1.83	5	0.00	0.86	5	0.00	0.42	5
2	50	1.00	2.00	275.03	4	0.00	4.06	5	0.00	2.59	5	0.00	2.42	5	0.00	1.98	5	0.00	1.82	5
Avg/Sum			0.50	83.39	19	0.00	1.21	20	0.00	3.84	20	0.00	1.83	20	0.00	0.85	20	0.00	0.61	20
2	100	0.25	0.00	1.27	5	0.00	2.00	5	0.00	197.54	5	0.00	56.11	5	0.00	4.23	5	0.00	1.31	5
2	100	0.50	0.00	8.33	5	0.00	7.75	5	0.00	136.72	5	0.00	38.19	5	0.00	9.93	5	0.00	6.83	5
2	100	0.75	2.34	219.50	4	1.52	227.64	4	0.00	143.62	5	0.00	45.35	5	0.00	46.30	5	0.00	43.37	5
2	100	1.00	28.14	982.03	0	27.70	1000.00	0	0.00	49.90	5	0.00	28.33	5	0.00	27.72	5	0.00	35.32	5
Avg/Sum			7.62	302.78	14	7.31	309.35	14	0.00	131.95	20	0.00	42.00	20	0.00	22.04	20	0.00	21.71	20
2	150	0.25	0.51	416.42	3	0.00	191.56	5	3.12	999.55	0	0.12	716.46	2	0.00	243.82	5	0.00	168.53	5
2	150	0.50	0.38	363.23	4	0.54	322.81	4	0.42	922.58	3	0.06	505.81	3	0.00	133.04	5	0.00	125.41	5
2	150	0.75	5.12	784.80	3	33.72	1000.00	0	0.15	735.12	3	0.00	268.18	5	0.00	189.32	5	0.00	188.24	5
2	150	1.00	9.89	849.07	1	35.63	1000.00	0	0.30	506.74	4	0.25	328.73	4	0.00	359.76	5	0.14	510.25	3
Avg/Sum			3.97	603.38	11	17.47	628.59	9	1.00	791.00	10	0.11	454.79	14	0.00	231.48	20	0.04	248.11	18
2	200	0.25	0.58	328.04	4	0.00	212.44	5	4.89	999.45	0	0.94	995.21	1	0.78	336.67	4	0.67	401.78	4
2	200	0.50	7.41	941.44	2	36.95	1000.00	0	46.11	999.44	0	7.62	841.61	3	3.35	626.40	3	2.75	549.02	3
2	200	0.75	10.31	999.02	0	36.65	1000.00	0	0.22	729.57	3	0.02	503.49	4	0.03	515.23	4	0.24	646.46	4
2	200	1.00	15.21	999.29	0	37.26	1000.00	0	0.00	230.11	5	0.00	226.31	5	0.05	463.19	4	0.38	738.73	2
Avg/Sum			8.38	816.95	6	27.71	803.11	5	12.81	739.64	8	2.15	641.66	13	1.05	485.37	15	1.01	584.00	13
r-Avg/Sum			5.12	451.63	50	13.12	435.56	48	3.45	416.61	58	0.56	285.07	67	0.26	184.94	75	0.26	213.61	71
5	50	0.25	0.00	0.08	5	0.00	0.17	5	0.00	10.89	5	0.00	3.46	5	0.00	0.41	5	0.00	0.14	5
5	50	0.50	0.00	0.19	5	0.00	0.29	5	0.00	9.81	5	0.00	3.15	5	0.00	0.93	5	0.00	0.31	5
5	50	0.75	0.00	80.43	5	0.00	1.02	5	0.00	7.28	5	0.00	5.20	5	0.00	3.55	5	0.00	0.83	5
5	50	1.00	1.89	214.65	4	0.00	6.11	5	0.00	8.21	5	0.00	5.32	5	0.00	6.82	5	0.00	2.48	5
Avg/Sum			0.47	73.84	19	0.00	1.90	20	0.00	9.05	20	0.00	4.28	20	0.00	2.93	20	0.00	0.94	20
5	100	0.25	0.00	1.17	5	0.00	0.99	5	0.00	220.64	5	0.00	107.16	5	0.00	6.42	5	0.00	1.58	5
5	100	0.50	0.00	3.17	5	0.00	3.14	5	0.00	206.75	5	0.00	61.13	5	0.00	18.95	5	0.00	5.74	5
5	100	0.75	0.00	115.17	5	0.00	134.79	5	0.00	149.15	5	0.00	81.34	5	0.00	47.85	5	0.00	49.02	5
5	100	1.00	21.17	999.93	0	20.74	1000.00	0	0.00	157.39	5	0.00	40.73	5	0.00	86.90	5	0.00	49.29	5
Avg/Sum			5.29	279.86	15	5.18	284.73	15	0.00	183.48	20	0.00	72.59	20	0.00	40.03	20	0.00	26.41	20
5	150	0.25	0.44	322.37	4	0.00	149.01	5	10.94	999.47	0	0.10	818.88	3	1.60	445.27	3	0.00	195.66	5
5	150	0.50	0.00	351.05	5	0.00	138.92	5	14.09	999.55	0	0.14	911.96	2	0.31	422.52	4	0.00	251.87	5
5	150	0.75	2.61	659.07	2	26.52	1000.00	0	0.59	956.86	1	0.24	681.37	3	0.00	338.21	5	0.00	362.78	5
5	150	1.00	5.93	850.58	1	28.67	1000.00	0	0.63	696.69	2	0.02	518.04	3	0.16	534.31	3	0.21	603.01	3
Avg/Sum			2.25	545.77	12	13.80	571.98	10	6.56	913.14	3	0.12	732.56	11	0.52	435.08	15	0.05	353.33	18
5	200	0.25	0.00	258.06	5	0.00	75.24	5	14.07	999.37	0	42.86	980.87	1	0.51	496.32	4	0.00	255.15	5
5	200	0.50	4.61	847.03	1	31.32	1000.00	0	128.65	999.34	0	41.10	958.51	1	5.44	871.98	2	4.21	838.92	2
5	200	0.75	7.83	999.81	0	29.10	1000.00	0	0.44	999.40	0	0.14	892.63	3	0.35	626.97	3	1.02	752.85	2
5	200	1.00	12.20	999.61	0	29.50	1000.00	0	0.18	719.46	3	0.00	473.08	5	0.34	814.13	2	0.10	848.00	3
Avg/Sum			6.16	776.13	6	22.48	768.81	5	35.84	929.40	3	21.03	826.27	10	1.66	702.35	11	1.33	673.73	12
r-Avg/Sum			3.54	418.90	52	10.37	406.86	50	10.60	508.77	46	5.29	408.93	61	0.54	295.10	66	0.35	263.60	70
10	50	0.25	0.00	0.19	5	0.00	0.31	5	0.00	17.59	5	0.00	7.61	5	0.00	1.19	5	0.00	0.70	5
10	50	0.50	0.00	1.58	5	0.00	1.98	5	0.00	17.43	5	0.00	6.21	5	0.00	2.31	5	0.00	2.94	5
10	50	0.75	0.00	5.63	5	0.00	9.36	5	0.00	12.55	5	0.00	7.06	5	0.00	4.42	5	0.00	5.66	5
10	50	1.00	0.00	136.16	5	0.00	12.41	5	0.00	9.03	5	0.00	7.19	5	0.00	9.43	5	0.00	3.58	5
Avg/Sum			0.00	35.89	20	0.00	6.02	20	0.00	14.15	20	0.00	7.02	20	0.00	4.34	20	0.00	3.22	20
10	100	0.25	0.00	4.42	5	0.00	2.94	5	0.00	463.02	5	0.00	231.67	5	0.00	20.79	5	0.00	7.86	5
10	100	0.50	0.00	59.39	5	0.00	17.56	5	0.00	417.07	5	0.00	366.44	5	0.00	76.04	5	0.00	19.04	5
10	100	0.75	0.00	129.06	5	0.00	171.39	5	0.00	268.54	5	0.00	141.85	5	0.00	95.76	5	0.00	98.01	5
10	100	1.00	0.00	483.41	5	20.09	1000.00	0	0.00	339.86	5	0.00	220.66	5	0.00	188.41	5	0.00	84.78	5
Avg/Sum			0.00	169.07	20	5.02	297.97	15	0.00	372.12	20	0.00	240.15	20	0.00	95.25	20	0.00	52.43	20
10	150	0.25	0.00	93.39	5	0.00	43.86	5	21.44	999.70	0	1.88	845.35	2	1.20	558.71	3	0.00	119.68	5
10	150	0.50	0.00	240.26	5	0.00	271.72	5	110.38	999.65	0	0.17	801.61	3	0.58	675.35	4	0.00	444.80	5
10	150	0.75	1.30	497.13	4	23.23	1000.00	0	2.76	999.66	0	0.06	700.55	3	0.19	847.39	1	0.81	827.27	3
10	150	1.00	4.92	856.56	2	25.88	1000.00	0	0.76	840.67	2	0.12	480.32	4	0.24	814.88	2	0.20	617.29	3
Avg/Sum			1.56	421.83	16	12.28	578.89	10	33.84	959.92	2	0.56	706.96	12	0.55	724.08	10	0.25	502.26	16
10	200	0.25	0.00	145.01	5	0.00	63.85	5	215.58	999.30	0	67.22	1000.00	0	1.75	811.73	2	0.00	253.53	5
10	200	0.50	2.98	712.85	2	27.72	1000.00	0	150.60	999.37	0	36.14	1000.00	0	5.27	871.92	1	4.47	833.35	1
10	200	0.75	5.27	999.86	0	27.97	1000.00	0	0.61	999.33	0	1.04	1000.00	0	0.76	991.38	1	1.59	985.27	1
10	200	1.00	12.34	999.80	0	29.00	1000.00	0	0.13	769.10	3	0.00	622.58	5	0.18	872.20	2	0.32	877.67	2
Avg/Sum			5.15	714.38	7	21.17	765.96	5	91.73	941.78	3	26.10	905.65	5	1.99	886.80	6	1.60	737.45	9
r-Avg/Sum			1.68	335.29	63	9.62	412.21	50	31.39	571.99	45	6.66	464.94	57	0.64	427.62	56	0.46	323.84	65
Over-Avg/Sum			3.45	401.94	165	11.04	418.21	148	15.15	499.12	149	4.17	386.31	185	0.48	302.55	197	0.36	267.02	206

magnitude up to thousands) are way too big for quadratic problems. We thus considered values of n ranging in $\{50, 100, 150, 200, 300, 400, 500\}$.

As for the classes, our benchmark instances were inspired by those adopted by Kataoka and Yamada (2014) for a "similar" problem, the *multiple knapsack assignment problem*, in which a number of knapsacks is available, and each of them can only accommodate items of the same class. We considered three values for the number of classes: $r \in \{2, 5, 10\}$. The number of items per class was set to $q = n/r$, and the items were consecutively assigned to the classes (items $1, \dots, q$ to class 1, items $q+1, \dots, 2q$ to class 2, and so on).

For $h = 1, \dots, r$, the setup costs and reductions of class h were correlated to the overall profits and weights, respectively, of the items in that class:

$$f_h = \left\lfloor e \sum_{i \in R_h} p_i + \alpha \sum_{i \in R_h} \sum_{j \in R_h \,:\, j > i} p_{ij} \right\rfloor \tag{25}$$

$$d_h = \left\lfloor e \sum_{i \in R_h} w_i \right\rfloor, \tag{26}$$

with e uniformly random in $[0.15, 0.25]$ (rounded to two decimal digits) and $\alpha = 0.1$.

For each triplet (r, n, d), 5 random instances were generated, resulting in a total of 420 instances. The instances are available upon request from the authors.

Observe the particular behavior we can expect from all matheuristics for the special case $r = 2$. As already observed in Section 4, Procedure `Greedy` produces the optimal solution whenever the time limit assigned to `Quadknap` allows complete execution. When this happens, procedure `Refined Greedy` produces the same solution as `Greedy`, while procedures `Remove` and `Exchange` never invoke `Quadknap`, as $S \notin \Sigma$ never occurs in their **foreach** loops (see Algorithms 5 and 6 in Sections 5.1 and 5.2), and hence never improve the current solution.

6.2. Results for small and medium size instances

Table 1 compares, on instances with $n \leq 200$, the quadratic model of Section 1 and the RLT models of Section 2. For each instance,

Table 2
Models and Matheuristics: small/medium size instances. 5 instances per entry: average Best value gap, CPU time and solution value; number of Best solutions.

Instance			RLT1(Gurobi)				RLT2(CPLEX)				Greedy				Refined Greedy				Greedy + Local				Refined Greedy + Local			
r	*n*	*d*	%Bgap	Time	Value	#B	%Bgap	Time	Value	#B	%Bgap	Time	Value	#B	%Bgap	Time	Value	#B	%Bgap	Time	Value	#B	%Bgap	Time	Value	#B
2	50	0.25	0.00	0.17	4 024	5	0.00	0.08	4 024	5	0.00	0.03	4 024	5	0.00	0.03	4 024	5	0.00	0.03	4 024	5	0.00	0.03	4 024	5
2	50	0.50	0.00	0.38	7 400	5	0.00	0.14	7 400	5	0.00	0.03	7 400	5	0.00	0.02	7 400	5	0.00	0.02	7 400	5	0.00	0.02	7 400	5
2	50	0.75	0.00	0.86	11 535	5	0.00	0.42	11 535	5	0.00	0.02	11 535	5	0.00	0.02	11 535	5	0.00	0.02	11 535	5	0.00	0.02	11 535	5
2	50	1.00	0.00	1.98	13 950	5	0.00	1.82	13 950	5	0.00	0.02	13 950	5	0.00	0.02	13 950	5	0.00	0.02	13 950	5	0.00	0.02	13 950	5
Avg/Sum			0.00	0.85	9 228	20	0.00	0.61	9 228	20	0.00	0.02	9 228	20	0.00	0.02	9 228	20	0.00	0.02	9 228	20	0.00	0.02	9 228	20
2	100	0.25	0.00	4.23	15 680	5	0.00	1.31	15 680	5	0.00	2.72	15 680	5	0.00	2.72	15 680	5	0.00	2.73	15 680	5	0.00	2.73	15 680	5
2	100	0.50	0.00	9.93	30 622	5	0.00	6.83	30 622	5	0.00	0.17	30 622	5	0.00	0.16	30 622	5	0.00	0.16	30 622	5	0.00	0.16	30 622	5
2	100	0.75	0.00	46.30	40 148	5	0.00	43.37	40 148	5	0.00	0.14	40 148	5	0.00	0.13	40 148	5	0.00	0.13	40 148	5	0.00	0.13	40 148	5
2	100	1.00	0.00	27.72	53 112	5	0.00	35.32	53 112	5	0.00	0.14	53 112	5	0.00	0.14	53 112	5	0.00	0.14	53 112	5	0.00	0.14	53 112	5
Avg/Sum			0.00	22.04	34 891	20	0.00	21.71	34 891	20	0.00	0.79	34 891	20	0.00	0.79	34 891	20	0.00	0.79	34 891	20	0.00	0.79	34 891	20
2	150	0.25	0.00	243.82	33 956	5	0.00	168.53	33 956	5	0.00	5.09	33 956	5	0.00	5.09	33 956	5	0.00	5.09	33 956	5	0.00	5.09	33 956	5
2	150	0.50	0.00	133.04	59 232	5	0.00	125.41	59 232	5	0.00	3.61	59 232	5	0.00	3.61	59 232	5	0.00	3.61	59 232	5	0.00	3.61	59 232	5
2	150	0.75	0.00	189.32	88 956	5	0.00	188.24	88 956	5	0.00	0.55	88 956	5	0.00	0.55	88 956	5	0.00	0.55	88 956	5	0.00	0.55	88 956	5
2	150	1.00	0.00	359.76	118 809	5	0.00	510.25	118 809	5	0.00	0.38	118 809	5	0.00	0.38	118 809	5	0.00	0.38	118 809	5	0.00	0.38	118 809	5
Avg/Sum			0.00	231.48	75 238	20	0.00	248.11	75 238	20	0.00	2.41	75 238	20	0.00	2.41	75 238	20	0.00	2.41	75 238	20	0.00	2.41	75 238	20
2	200	0.25	0.00	336.67	53 524	5	0.00	401.78	53 524	5	0.07	5.19	53 487	3	0.07	5.19	53 487	3	0.07	5.20	53 487	3	0.07	5.20	53 487	3
2	200	0.50	0.01	626.40	100 289	4	0.01	549.02	100 290	4	0.00	5.16	100 300	5	0.00	5.16	100 300	5	0.00	5.15	100 300	5	0.00	5.15	100 300	5
2	200	0.75	0.00	515.23	160 887	4	0.04	646.46	160 815	4	0.00	1.11	160 890	5	0.00	1.10	160 890	5	0.00	1.10	160 890	5	0.00	1.10	160 890	5
2	200	1.00	0.00	463.19	226 464	5	0.02	738.73	226 430	3	0.00	0.75	226 464	5	0.00	0.75	226 464	5	0.00	0.75	226 464	5	0.00	0.75	226 464	5
Avg/Sum			0.00	485.37	135 291	18	0.02	584.00	135 265	16	0.02	3.05	135 286	18	0.02	3.05	135 286	18	0.02	3.05	135 286	18	0.02	3.05	135 286	18
r-Avg/Sum			0.00	184.94	63 662	78	0.00	213.61	63 655	76	0.00	1.57	63 661	78	0.00	1.57	63 661	78	0.00	1.57	63 661	78	0.00	1.57	63 661	78
5	50	0.25	0.00	0.41	4 774	5	0.00	0.14	4 774	5	2.28	0.04	4 677	2	0.85	0.05	4 740	4	0.00	0.08	4 774	5	0.00	0.08	4 774	5
5	50	0.50	0.00	0.93	8 872	5	0.00	0.31	8 872	5	2.66	0.03	8 662	3	0.00	0.05	8 872	5	0.00	0.06	8 872	5	0.00	0.07	8 872	5
5	50	0.75	0.00	3.55	13 303	5	0.00	0.83	13 303	5	0.11	0.03	13 286	4	0.00	0.06	13 303	5	0.00	0.07	13 303	5	0.00	0.08	13 303	5
5	50	1.00	0.00	6.82	16 932	5	0.00	2.48	16 932	5	0.00	0.03	16 932	5	0.00	0.04	16 932	5	0.00	0.06	16 932	5	0.00	0.07	16 932	5
Avg/Sum			0.00	2.93	10 970	20	0.00	0.94	10 970	20	1.27	0.03	10 889	14	0.21	0.05	10 962	19	0.00	0.07	10 970	20	0.00	0.08	10 970	20
5	100	0.25	0.00	6.42	17 898	5	0.00	1.58	17 898	5	0.72	2.93	17 775	3	0.00	3.03	17 898	5	0.00	3.19	17 898	5	0.00	3.24	17 898	5
5	100	0.50	0.00	18.95	35 339	5	0.00	5.74	35 339	5	1.18	0.28	34 984	3	0.00	0.36	35 339	5	0.00	0.51	35 339	5	0.00	0.53	35 339	5
5	100	0.75	0.00	47.85	48 118	5	0.00	49.02	48 118	5	0.17	0.18	48 054	4	0.00	0.27	48 118	5	0.00	0.42	48 118	5	0.00	0.43	48 118	5
5	100	1.00	0.00	86.90	63 571	5	0.00	49.29	63 571	5	0.00	0.20	63 571	5	0.00	0.34	63 571	5	0.00	0.39	63 571	5	0.00	0.47	63 571	5
Avg/Sum			0.00	40.03	41 231	20	0.00	26.41	41 231	20	0.52	0.90	41 096	15	0.00	1.00	41 231	20	0.00	1.13	41 231	20	0.00	1.17	41 231	20
5	150	0.25	0.00	445.27	38 447	5	0.00	195.66	38 447	5	0.00	7.44	38 447	5	0.00	10.46	38 447	5	0.00	18.93	38 447	5	0.00	19.07	38 447	5
5	150	0.50	0.00	422.52	68 801	5	0.00	251.87	68 801	5	0.38	4.67	68 557	3	0.00	7.65	68 801	5	0.00	13.78	68 801	5	0.00	13.82	68 801	5
5	150	0.75	0.00	338.21	105 285	5	0.00	362.78	105 285	5	0.25	0.55	105 011	4	0.00	0.91	105 285	5	0.00	1.47	105 285	5	0.00	1.59	105 285	5
5	150	1.00	0.00	534.31	138 956	5	0.00	603.01	138 952	4	0.57	0.49	138 309	4	0.57	0.87	138 309	4	0.57	1.00	138 309	4	0.57	1.21	138 309	4
Avg/Sum			0.00	435.08	87 872	20	0.00	353.33	87 871	19	0.30	3.29	87 581	16	0.14	4.97	87 711	19	0.14	8.79	87 711	19	0.14	8.92	87 711	19
5	200	0.25	0.00	496.32	63 078	5	0.00	255.15	63 078	5	1.34	9.49	62 302	2	0.01	16.22	63 073	4	0.01	26.10	63 073	4	0.01	28.66	63 073	4
5	200	0.50	0.27	871.98	118 185	2	0.42	838.92	117 999	3	0.61	8.58	117 755	3	0.32	15.02	118 115	4	0.00	21.33	118 485	5	0.00	25.39	118 485	5
5	200	0.75	0.10	626.97	188 533	3	0.08	752.85	188 577	3	0.06	1.52	188 637	4	0.00	2.59	188 740	5	0.00	3.60	188 740	5	0.00	4.01	188 740	5
5	200	1.00	0.21	814.13	263 057	2	0.01	848.00	263 529	4	0.08	1.06	263 355	4	0.00	2.07	263 559	5	0.00	2.61	263 559	5	0.00	3.02	263 559	5
Avg/Sum			0.15	702.35	158 213	12	0.13	673.73	158 296	15	0.52	5.16	158 012	13	0.08	8.98	158 372	18	0.00	13.41	158 465	19	0.00	15.27	158 465	19
r-Avg/Sum			0.04	295.10	74 572	72	0.03	263.60	74 592	74	0.65	2.35	74 395	58	0.11	3.75	74 569	76	0.04	5.85	74 594	78	0.04	6.36	74 594	78
10	50	0.25	0.00	1.19	5 264	5	0.00	0.70	5 264	5	3.25	0.05	5 093	0	0.35	0.10	5 247	4	0.10	0.23	5 259	4	0.10	0.20	5 259	4
10	50	0.50	0.00	2.31	9 647	5	0.00	2.94	9 647	5	5.72	0.05	9 122	0	0.84	0.09	9 574	2	0.00	0.20	9 647	5	0.14	0.18	9 632	3
10	50	0.75	0.00	4.42	14 563	5	0.00	5.66	14 563	5	0.85	0.05	14 438	3	0.27	0.11	14 525	4	0.00	0.18	14 563	5	0.00	0.24	14 563	5
10	50	1.00	0.00	9.43	18 429	5	0.00	3.58	18 429	5	0.55	0.04	18 337	2	0.00	0.09	18 429	5	0.00	0.15	18 429	5	0.00	0.16	18 429	5
Avg/Sum			0.00	4.34	11 976	20	0.00	3.22	11 976	20	2.59	0.05	11 748	5	0.37	0.09	11 944	15	0.03	0.19	11 975	19	0.06	0.20	11 971	17
10	100	0.25	0.00	20.79	19 127	5	0.00	7.86	19 127	5	2.66	4.08	18 652	1	0.00	2.36	19 127	5	0.22	5.65	19 092	4	0.00	3.54	19 127	5
10	100	0.50	0.00	76.04	37 246	5	0.00	19.04	37 246	5	0.87	0.40	36 954	3	0.00	0.89	37 246	5	0.15	1.64	37 194	4	0.00	1.93	37 246	5
10	100	0.75	0.00	95.76	52 193	5	0.00	98.01	52 193	5	0.68	0.25	51 825	2	0.00	0.58	52 193	5	0.08	1.22	52 152	4	0.00	1.34	52 193	5
10	100	1.00	0.00	188.41	67 688	5	0.00	84.78	67 688	5	0.27	0.32	67 513	3	0.00	0.70	67 688	5	0.00	1.30	67 688	5	0.00	1.45	67 688	5
Avg/Sum			0.00	95.25	44 064	20	0.00	52.43	44 064	20	1.12	1.26	43 736	9	0.00	1.13	44 064	20	0.11	2.45	44 032	17	0.00	2.07	44 064	20
10	150	0.25	0.00	558.71	41 165	5	0.00	119.68	41 165	5	1.10	12.98	40 719	2	0.00	23.86	41 165	5	0.00	52.73	41 165	5	0.00	56.68	41 165	5
10	150	0.50	0.00	675.35	73 482	5	0.00	444.80	73 482	5	1.36	6.57	72 572	2	0.00	11.38	73 482	5	0.46	17.24	73 181	4	0.00	23.29	73 482	5
10	150	0.75	0.00	847.39	114 365	5	0.14	827.27	114 205	3	0.59	0.88	113 700	0	0.03	1.94	114 327	4	0.00	4.32	114 365	5	0.00	4.55	114 365	5
10	150	1.00	0.01	814.88	149 175	4	0.01	617.29	149 175	4	0.20	0.82	148 895	4	0.00	1.97	149 191	5	0.00	3.46	149 191	5	0.00	4.19	149 191	5
Avg/Sum			0.00	724.08	94 547	19	0.04	502.26	94 507	17	0.81	5.32	93 971	8	0.01	9.79	94 541	19	0.11	19.44	94 476	19	0.00	22.18	94 551	20
10	200	0.25	0.01	811.73	66 827	3	0.00	253.53	66 834	5	2.29	17.74	65 340	1	0.00	30.40	66 834	5	0.25	81.93	66 681	4	0.00	73.95	66 834	5
10	200	0.50	0.50	871.92	127 111	1	0.28	833.35	127 392	2	0.75	15.22	126 812	2	0.00	26.30	127 739	5	0.00	74.54	127 739	5	0.00	60.20	127 739	5
10	200	0.75	0.23	991.38	198 799	1	0.13	985.27	198 991	1	0.28	2.23	198 705	2	0.00	5.26	199 242	5	0.00	10.12	199 242	5	0.00	12.15	199 242	5
10	200	1.00	0.07	872.20	274 279	2	0.04	877.67	274 366	3	0.89	2.17	272 091	1	0.12	5.28	274 200	4	0.00	8.63	274 486	5	0.00	10.07	274 486	5
Avg/Sum			0.20	886.80	166 754	7	0.11	737.45	166 896	11	1.05	9.34	165 737	6	0.03	16.81	167 004	19	0.06	43.81	167 037	19	0.00	39.09	167 076	20
r-Avg/Sum			0.05	427.62	79 335	66	0.04	323.84	79 361	68	1.39	3.99	78 798	28	0.10	6.96	79 388	73	0.08	16.47	79 380	74	0.02	15.88	79 415	77
Over-Avg/Sum			0.03	302.55	72 523	216	0.02	267.02	72 536	218	0.68	2.64	72 284	164	0.07	4.09	72 539	227	0.04	7.96	72 545	230	0.02	7.94	72 557	233

the models were solved using solvers CPLEX 22.11 and Gurobi 10.02, with a time limit of 1 000 s per solver. Each line refers to a triplet (r, n, d), reported in the first three columns. For each method (fields 01QKP(CPLEX), 01QKP(Gurobi), RLT(CPLEX), RLT(Gurobi), RLT1(Gurobi), and RLT2(CPLEX)), the three entries in the table report (over the corresponding 5 instances)

- %Ogap = average percentage optimality gap $100\,(u - z)/z$ of the solution value z returned by the solver with respect to the best upper bound value u returned by the same solver for the considered instance;
- Time = average CPU time expressed in seconds;
- #opt = number of instances solved to proven optimality.

For each group of 20 instances corresponding to a pair $\{r, n\}$, the average values of %Ogap and Time, and the total value of #opt are reported in lines "Avg/Sum". Lines "r-Avg/Sum" provide the same three values over the 80 instances generated for each value of r, while the last line, "Over-Avg/Sum", gives the overall statistics over the 240 small/medium size instances.

The first two fields show that CPLEX performs globally better than Gurobi, for all scores, when adopted for the classical quadratic model. Opposite behavior is shown by the next two fields (model RLT of Section 2), especially for what concerns %Ogap and #opt. Worth is mentioning that the global performance of CPLEX for RLT is even worse that that provided by the same solver for the quadratic model.

The execution of CPLEX and Gurobi for models RLT1 (Section 2.1) and RLT2 (Section 2.2) showed a very clear difference between the two solvers: while Gurobi was the best method for RLT1, the opposite was observed for RLT2: thus, the last two fields in the table only provide the results for the best solver. They show an evident improvement provided by the revised relaxation linearization techniques: the optimality gap is one order of magnitude smaller, the CPU times are considerably lower, and the number of optimally solved instances is about 85%. Globally, the winner is RLT2 solved with CPLEX, although RLT1 solved with Gurobi performs better for $r = 2$. Worth is mentioning that the "worst" method, 01QKP(Gurobi), is the only method capable of solving to proven optimality *all* instances with $d = 0.25$.

Table 3
Models and Matheuristics: large size instances. 5 instances per entry: average Best value gap, CPU time and solution value; number of Best solutions; average Optimality gap; number of optima.

Instance			RLT1(Gurobi)						RLT2(CPLEX)						Greedy + Local				Refined Greedy + Local			
r	*n*	*d*	%Bgap	Time	Value	#B	%Ogap	#opt	%Bgap	Time	Value	#B	%Ogap	#opt	%Bgap	Time	Value	#B	%Bgap	Time	Value	#B
2	300	0.25	0.00	7197.7	116 572	5	9.66	0	1.41	7200.0	115 194	3	11.84	0	0.02	50.7	116 544	3	0.02	50.7	116 544	3
2	300	0.50	0.04	7199.1	241 791	2	5.69	0	0.01	7200.0	241 867	4	6.03	0	0.00	51.0	241 879	5	0.00	51.0	241 879	5
2	300	0.75	0.12	7199.2	362 043	2	3.15	0	0.00	4460.6	362 513	4	0.29	3	0.00	29.8	362 526	5	0.00	29.8	362 526	5
2	300	1.00	0.05	7199.0	457 238	2	0.79	0	0.05	6143.3	457 210	1	0.47	1	0.07	23.7	457 087	4	0.07	23.7	457 087	4
Avg/Sum			0.05	7198.8	294 411	11	4.82	0	0.37	6251.0	294 196	12	4.66	4	0.02	38.8	294 509	17	0.02	38.8	294 509	17
2	400	0.25	1.34	7198.1	203 594	3	16.45	0	0.00	7200.0	206 060	5	14.80	0	0.03	67.2	205 993	1	0.03	67.2	205 993	1
2	400	0.50	1.38	7199.4	396 258	0	13.13	0	0.20	7200.0	401 048	1	11.62	0	0.00	72.6	401 873	5	0.00	72.6	401 873	5
2	400	0.75	0.11	7199.2	616 407	0	6.45	0	0.19	7200.0	615 861	1	2.35	0	0.00	57.8	617 070	5	0.00	57.8	617 070	5
2	400	1.00	0.09	7199.2	920 049	0	0.79	0	0.11	7012.0	919 950	1	0.35	1	0.00	24.1	920 928	5	0.00	24.1	920 928	5
Avg/Sum			0.73	7199.0	534 077	3	9.20	0	0.12	7153.0	535 730	8	7.28	1	0.01	55.4	536 466	16	0.01	55.4	536 466	16
2	500	0.25	0.01	7198.5	297 595	2	18.04	0	0.00	7200.0	297 620	5	18.39	0	0.01	93.6	297 604	2	0.01	93.6	297 604	2
2	500	0.50	3.07	7199.7	629 730	0	16.28	0	5.06	7200.0	615 847	0	18.92	0	0.00	128.7	649 528	5	0.00	128.7	649 528	5
2	500	0.75	0.28	7199.6	972 400	0	4.56	0	14.57	7200.0	831 725	0	24.44	0	0.00	94.3	975 101	5	0.00	94.3	975 101	5
2	500	1.00	0.23	7199.3	1 351 870	0	1.31	0	6.85	7200.1	1 261 836	0	11.86	0	0.00	36.3	1 355 154	5	0.00	36.3	1 355 154	5
Avg/Sum			0.90	7199.3	812 899	2	10.05	0	6.62	7200.0	751 757	5	18.40	0	0.00	88.2	819 347	17	0.00	88.2	819 347	17
r-Avg/Sum			0.56	7199.0	547 129	16	8.03	0	2.37	6868.0	527 228	25	10.11	5	0.01	60.8	550 107	50	0.01	60.8	550 107	50
5	300	0.25	0.00	5506.8	139 141	5	3.65	2	0.00	7075.0	139 141	5	5.00	1	0.00	197.1	139 141	5	0.00	261.3	139 141	5
5	300	0.50	0.01	6348.6	280 000	4	4.09	1	0.07	7200.0	279 832	2	4.91	0	0.00	202.8	280 028	5	0.00	291.9	280 028	5
5	300	0.75	0.00	6530.7	423 611	4	1.98	1	0.04	6014.2	423 468	4	0.54	1	0.00	54.9	423 615	5	0.00	63.1	423 615	5
5	300	1.00	0.25	7199.4	540 603	1	0.87	0	0.03	5727.2	541 797	3	0.35	2	0.00	26.9	541 964	5	0.00	25.9	541 964	5
Avg/Sum			0.06	6396.4	345 839	14	2.65	4	0.03	6504.1	346 060	14	2.70	4	0.00	120.4	346 187	20	0.00	160.6	346 187	20
5	400	0.25	0.17	7195.5	242 255	4	8.64	0	0.17	7200.0	242 255	4	9.95	0	0.02	217.0	242 624	1	0.02	329.5	242 624	1
5	400	0.50	0.66	7199.4	477 131	0	8.36	0	1.41	7200.0	473 610	0	9.32	0	0.00	220.9	480 240	5	0.00	342.7	480 240	5
5	400	0.75	0.22	7199.3	719 383	1	4.62	0	1.09	7200.0	713 181	0	4.89	0	0.00	158.4	721 023	5	0.00	208.2	721 023	5
5	400	1.00	0.04	7199.6	1 040 600	1	0.64	0	0.72	7200.1	1 033 531	0	1.37	0	0.00	60.1	1 040 960	5	0.00	65.0	1 040 923	4
Avg/Sum			0.27	7198.4	619 842	6	5.56	0	0.85	7200.0	615 644	4	6.38	0	0.00	164.1	621 212	16	0.01	236.4	621 203	15
5	500	0.25	0.53	7197.8	352 513	1	12.32	0	0.71	7200.0	351 973	2	14.07	0	0.00	250.0	354 338	2	0.00	357.0	354 338	2
5	500	0.50	1.52	7199.7	747 108	0	11.57	0	37.30	7200.1	483 613	0	84.90	0	0.00	244.9	759 286	5	0.00	366.8	759 286	5
5	500	0.75	3.78	7199.7	1 101 975	0	27.79	0	27.69	7200.0	829 895	0	67.65	0	0.00	191.2	1 144 130	4	0.00	278.0	1 144 138	5
5	500	1.00	10.02	7199.5	1 402 652	0	31.12	0	27.22	7200.0	1 136 094	0	53.35	0	0.05	109.0	1 557 146	4	0.00	135.7	1 557 876	5
Avg/Sum			3.96	7199.2	901 062	1	20.70	0	23.23	7200.0	700 394	2	54.99	0	0.01	198.8	953 725	15	0.00	284.4	953 909	17
r-Avg/Sum			1.43	6931.3	622 248	21	9.64	4	8.04	6968.1	554 032	20	21.36	4	0.01	161.1	640 375	51	0.00	227.1	640 433	52
10	300	0.25	0.00	4714.8	146 894	5	3.31	2	0.00	6740.4	146 894	5	5.00	1	0.11	472.5	146 738	3	0.01	622.9	146 884	4
10	300	0.50	0.01	7199.4	294 894	2	5.17	0	0.45	7200.0	293 680	0	6.20	0	0.00	501.1	294 928	5	0.00	801.1	294 928	5
10	300	0.75	0.38	7199.5	444 929	0	2.56	0	0.26	7200.0	445 384	0	1.67	0	0.00	89.4	446 616	5	0.00	93.2	446 616	5
10	300	1.00	0.11	7199.4	570 504	1	0.53	0	0.08	6498.8	570 645	1	0.40	1	0.00	50.5	571 119	5	0.00	67.6	571 119	5
Avg/Sum			0.13	6578.3	364 305	8	2.89	2	0.20	6909.8	364 151	6	3.32	2	0.03	278.4	364 850	18	0.00	396.2	364 887	19
10	400	0.25	0.02	7198.0	253 614	4	7.86	0	1.07	7200.0	251 055	1	11.83	0	0.03	622.5	253 585	3	0.02	1003.7	253 612	3
10	400	0.50	1.73	7199.4	494 732	0	29.88	0	2.90	7200.0	489 308	0	11.11	0	0.00	639.6	503 531	5	0.00	973.8	503 531	5
10	400	0.75	0.99	7199.4	749 874	0	27.17	0	19.29	7200.1	614 852	0	33.70	0	0.00	424.0	757 360	5	0.00	613.6	757 360	5
10	400	1.00	1.23	7199.5	1 079 460	0	11.15	0	22.10	7200.0	857 178	0	33.74	0	0.00	127.4	1 092 133	5	0.00	195.5	1 092 133	5
Avg/Sum			0.99	7199.1	644 420	4	19.02	0	11.34	7200.0	553 098	1	22.59	0	0.01	453.4	651 652	18	0.00	696.7	651 659	18
10	500	0.25	0.00	7199.7	373 687	5	11.12	0	5.18	7200.0	355 265	0	18.72	0	0.00	747.3	373 673	3	0.00	1110.3	373 673	3
10	500	0.50	3.15	7199.7	771 953	0	40.84	0	35.10	7200.0	518 752	0	80.04	0	0.00	604.2	797 173	5	0.00	1028.4	797 173	5
10	500	0.75	8.49	7199.7	1 097 074	0	47.90	0	28.24	7200.0	863 345	0	72.94	0	0.00	457.0	1 200 494	5	0.00	722.1	1 200 494	5
10	500	1.00	7.15	7199.6	1 516 073	0	29.97	0	21.83	7200.0	1 286 519	0	44.17	0	0.00	206.3	1 636 452	5	0.00	335.6	1 636 452	5
Avg/Sum			4.70	7199.7	939 697	5	32.46	0	22.59	7200.0	755 970	0	53.97	0	0.00	503.7	1 001 948	18	0.00	799.1	1 001 948	18
r-Avg/Sum			1.94	6992.3	649 474	17	18.12	2	11.38	7103.3	557 740	7	26.63	2	0.01	411.8	672 817	54	0.00	630.6	672 831	55
Over-Avg/Sum			1.31	7040.9	606 284	54	11.93	6	7.26	6979.8	546 333	52	19.37	11	0.01	211.2	621 100	155	0.01	306.2	621 124	157

Table 2 compares, on the same instances, the winners of Table 1 (RLT1(Gurobi) and RLT2(CPLEX)) with the matheuristics of Sections 4 (Greedy and Refined Greedy), and 5 (Greedy and Refined Greedy, both followed by local improvements Remove and Exchange). Each solver had again a time limit of 1 000 s per instance, while each matheuristic had a time limit of 5 s for each execution of the QKP code Quadknap (see Sections 3, 4, and 5).

Each line refers to a triplet (r, n, d), reported in the first three columns. For each method (fields RLT1(Gurobi), RLT2(CPLEX), Greedy, Refined Greedy, Greedy + Local, Refined Greedy + Local), the four entries in the table report (over the corresponding 5 instances)

- %Bgap = average percentage gap $100\,(z - z^h)/z$ of the solution value z^h returned by the method with respect to the *best* solution value z produced by the six methods for the considered instance;
- Time = average CPU time expressed in seconds;
- Value = average solution value (rounded to the nearest integer);
- #B = number of instances for which the method produced the best solution value.

For each group of 20 instances corresponding to a pair $\{r, n\}$, the average values of %Bgap, Time and Value, and the total value of #B are reported in lines "Avg/Sum". Lines "r-Avg/Sum" provide the same four values over the 80 instances generated for each value of r, while the last line, "Over-Avg/Sum", gives the overall statistics over the 240 small/medium size instances.

Model RLT2(Cplex) exhibits a slightly better performance than RLT1(Gurobi), but both models are clearly worse than the matheuristics (among which Refined Greedy + Local appears to be the best method). However, for density $d = 0.25$, RLT2(Cplex) was the only method capable of finding the best solution for all instances, although at the expense of huge CPU times.

For $r = 2$ (remind the previously observed behavior of the heuristics in this case), all methods produced the same solutions for all instances with $n \leq 150$, with the matheuristics requiring one or two orders of magnitude smaller CPU times. For the instances with $n = 200$, the matheuristics and RLT1(Gurobi) exhibited similar behavior for what concerns the solution quality, but again with a great difference in CPU times. Also observe that, for $n = 200$ and $d = 0.25$, the matheuristics were not able to find the best solution for 2 of the considered 5 instances, since the time limit assigned to the QKP code quadknap (5 s) did not allow the code to complete its executions.

For $r = 5$, Greedy + Local and Refined Greedy + Local produced the same solutions, and are clearly the best methods, both concerning solution quality (78 best solutions out of 80 instances) and CPU times. Among the other methods, Refined Greedy considerably improved over Greedy, with negligible CPU time increase. In this case too, the CPU times required by the mathematical models were two orders of magnitude larger than those required by the matheuristics.

For $r = 10$, the above trend is confirmed, with Refined Greedy + Local performing clearly better than Greedy + Local.

We finally observe that the local search procedures (Remove and Exchange) consistently improve on the solutions produced by Greedy and Refined Greedy.

6.3. Results for large size instances

Table 3 evaluates the winners of Table 1 (RLT1(Gurobi) and RLT2(CPLEX)) and Table 2 (Greedy + Local and Refined Greedy + Local) on instances with $n \in \{300, 400, 500\}$. In this case, both solvers CPLEX and Gurobi had a time limit of 2 h per instance, while each matheuristic had a time limit of (i) 20 s for each execution of the QKP code Quadknap within the greedy algorithms, and (ii) 10 s for each execution of the same code within the local search procedures.

Each line refers to a triplet (r, n, d), reported in the first three columns. For all methods, the first four entries provide (over the corresponding 5 instances)

- %Bgap = average percentage gap (see above);
- Time = average CPU time expressed in seconds;
- Value = average solution value (rounded to the nearest integer);
- #B = number of instances for which the method produced the best solution value.

In addition, for the two models, the last two entries give

- %Ogap = average percentage optimality gap (see above);
- #opt = number of instances solved to proven optimality.

For each group of 20 instances corresponding to a pair $\{r, n\}$, the average values of %Bgap, Time, Value and %Ogap, and the total values of #B and #opt, are reported in lines "Avg/Sum". Lines "r-Avg/Sum" provide the same values over the 60 instances generated for each value of r, while the last line, "Over-Avg/Sum", gives the overall statistics over all the 180 large instances.

Contrary to what happens for smaller instances, the performance of RLT1(Gurobi) is globally better than that of RLT2(Cplex). Both models are however clearly worse than the matheuristics, as they produce globally much worse results with CPU times one or two orders of magnitude larger. (The only partial exception occurs for $d = 0.25$, for which RLT1(Gurobi) finds a larger number of best solutions, although almost always hitting the time limit.)

Concerning the matheuristic methods, the results obtained by Refined Greedy + Local are slightly better than those obtained by Greedy + Local, at the expense of a moderate CPU time increase. The overall winner appears to be Refined Greedy + Local, which finds about 90% of best solution values, with an overall average CPU time around 5 min.

7. Conclusions

We have studied a relevant variant of the quadratic knapsack problem, never investigated before in the literature. We have provided a traditional quadratic model and different linear reformulations of the problem. We have developed deterministic matheuristic methods and local search procedures to improve the solutions they obtain. Extensive computational experiments indicate that these methods are able to produce good quality solutions within reasonable computing times.

CRediT authorship contribution statement

Laura Galli: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Silvano Martello:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Carlos Rey:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Paolo Toth:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization.

Acknowledgments

This research was supported by the Air Force Office of Scientific Research, United States under Grants no. FA8655-20-1-7019 and FA8655-20-1-7012, by the National Agency for Research and Development (ANID)/ Subvención a la Instalación en la Academia/Folio: 85220108, and by the Vicerrectoria de Investigación y Postgrado (VRIP)/Universidad del Bío-Bío/RE2360219. Thanks are due to an anonymous reviewer for helpful comments.

Data availability

Data will be made available on request.

References

Adams, W.P., Sherali, H.D., 1986. A tight linearization and an algorithm for zero-one quadratic programming problems. Manage. Sci. 32 (10), 1274–1290.
Amiri, A., 2020. A Lagrangean based solution algorithm for the knapsack problem with setups. Expert Syst. Appl. 143, 113077.
Billionnet, A., Soutif, É., 2004a. An exact method based on Lagrangian decomposition for the 0–1 quadratic knapsack problem. European J. Oper. Res. 157 (3), 565–575.
Billionnet, A., Soutif, É., 2004b. Using a mixed integer programming tool for solving the 0–1 quadratic knapsack problem. INFORMS J. Comput. 16 (2), 188–197.
Burkard, R., Dell'Amico, M., Martello, S., 2012. Assignment Problems: Revised Reprint. SIAM.
Caprara, A., 2008. Constrained 0–1 quadratic programming: Basic approaches and extensions. European J. Oper. Res. 187 (3), 1494–1503.
Caprara, A., Pisinger, D., Toth, P., 1999. Exact solution of the quadratic knapsack problem. INFORMS J. Comput. 11 (2), 125–137.
Chajakis, E.D., Guignard, M., 1994. Exact algorithms for the setup knapsack problem. INFOR Inf. Syst. Oper. Res. 32, 124–142.
Chebil, K., Khemakhem, M., 2015. A dynamic programming algorithm for the knapsack problem with setup. Comput. Oper. Res. 64, 40–50.
Chen, Y., Hao, J.-K., 2017. An iterated " hyperplane exploration" approach for the quadratic knapsack problem. Comput. Oper. Res. 77, 226–239.
della Croce, F., Salassa, F., Scatamacchia, R., 2017. An exact approach for the 0–1 knapsack problem with setups. Comput. Oper. Res. 80, 61–67.
Fomeni, F.D., Kaparis, K., Letchford, A.N., 2022. A cut-and-branch algorithm for the quadratic knapsack problem. Discrete Optim. 44, 100579.
Furini, F., Monaci, M., Traversi, E., 2018. Exact approaches for the knapsack problem with setups. Comput. Oper. Res. 90, 208–220.
Gallo, G., Hammer, P.L., Simeone, B., 1980. Quadratic knapsack problems. Math. Program. 132–149.
Glover, F., Kochenberger, G., Alidaee, B., Amini, M., 2002. Solving quadratic knapsack problems by reformulation and tabu search: Single constraint case. In: Combinatorial and Global Optimization. World Scientific, pp. 111–121.
Guignard, M., 2020. Strong RLT1 bounds from decomposable Lagrangean relaxation for some quadratic 0–1 optimization problems with linear constraints. Ann. Oper. Res. 286, 173–200.
Kataoka, S., Yamada, T., 2014. Upper and lower bounding procedures for the multiple knapsack assignment problem. European J. Oper. Res. 237, 440–447.
Khemakhem, M., Chebil, K., 2016. A tree search based combination heuristic for the knapsack problem with setup. Comput. Ind. Eng. 99, 280–286.
Page, S.E., 2018. The Model Thinker: What You Need to Know to Make Data Work for You. Basic Books.
Pisinger, W.D., Rasmussen, A.B., Sandvik, R., 2007. Solution of large quadratic knapsack problems through aggressive reduction. INFORMS J. Comput. 19 (2), 280–290.
Rhys, J., 1970. A selection problem of shared fixed costs and network flows. Manage. Sci. 17, 200–207.
Sherali, H.D., Adams, W.P., 1990. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. SIAM J. Discrete Math. 3 (3), 411–430.
Witzgall, C., 1975. Mathematical methods of site selection for electronic message systems (EMS). NASA STI/Recon Tech. Rep. N 76.
Yang, Z., Wang, G., Chu, F., 2013. An effective GRASP and tabu search for the 0-1 quadratic knapsack problem. Comput. Oper. Res. 40 (5), 1176–1185.