Ann Oper Res (2018) 271:831–851
https://doi.org/10.1007/s10479-017-2746-2

ORIGINAL RESEARCH

Minimum tiling of a rectangle by squares

Michele Monaci[1] · André Gustavo dos Santos[2]

Published online: 3 January 2018

Abstract We consider a two-dimensional problem in which one is required to split a given rectangular *bin* into the smallest number of *items*. The resulting items must be squares to be packed, without overlapping, into the bin so as to cover all the given rectangle. We present a mathematical model and a heuristic algorithm that is proved to find the optimal solution in some special cases. Then, we introduce a relaxation of the problem and present different exact approaches based on this relaxation. Finally, we report computational experiments on the performances of the algorithms on a large set of randomly generated instances.

Keywords Two-dimensional packing · Mathematical models · Exact algorithms · Computational experiments

1 Introduction

We consider the problem of splitting a given rectangular *bin* into the smallest number of smaller square *items* having integer sides. These items have to be packed, without overlapping, with their edges parallel to the edges of the bin, so as to cover all the given rectangle. Similar problems have been addressed in the computational geometry literature, where a partitioning of a given region is called a *tiling*, and the smaller items are called *tiles*. Thus, the problem we consider will be denoted as the *Minimum Tiling of a Rectangle by Squares* (MTRS).

Several papers in the literature consider the case in which all the produced tiles must be different from each other. In this case, the tiling is said to be *perfect*. In Brooks et al. (1940) the problem of finding (if any) a perfect tiling of a given rectangle was considered, and a correspondence between feasible solutions of this problem and a certain class of planar electrical networks was introduced. In Beaumont et al. (2002) some different problems con-

✉ Michele Monaci
michele.monaci@unibo.it

1 DEI, Università di Bologna, Viale Risorgimento 2, 40136 Bologna, Italy

2 DPI, Universidade Federal de Viçosa, Av. P. H. Rolfs s/n, Viçosa, MG 36570-900, Brazil

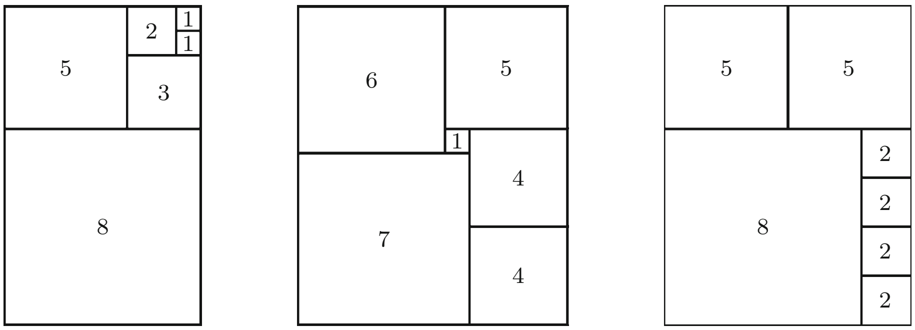

Fig. 1 Example of guillotine pattern, non-guillotine pattern and 3-staged pattern (the numbers denote the sizes of the items)

cerning packing of squares and rectangles were proposed, namely the problems of covering a square using a set of p (say) rectangles that have a given area, so as to minimize either the sum of the perimeters of the rectangles or the maximum among the p perimeters. For these problems, motivated in heterogeneous parallel computing, a proof of NP-completness and some approximation algorithms were given. A tight logarithmic bound on the optimal solution value of MTRS was given in Kenyon (1996), where in Walters (2009) polylogarithmic lower and upper bounds for the generalization of the problem to a higher dimension were provided. Recently, the problem of covering a given $n \times n$ square with the minimum number of squares having side at most $n - 1$ was addressed in Kurz (2012), where an Integer Linear Programming (ILP) model for this problem was proposed.

MTRS is strictly related also to two dimensional packing problems (see, e.g., Lodi et al. 2010). In the *Two Dimensional Bin Packing* problem (2BP), one is required to allocate a given set of rectangular items to a minimum number of larger bins; in the *Two Dimensional Strip Packing* problem (2SP), items must be packed into a unique container having finite width and infinite height, so as to minimize the total height of the packing. However, two main differences arise between MTRS and these two problems: (i) in MTRS we are required to define the dimensions of each item, whereas in classical packing problems they are an input of the problem; (ii) all the used items must be squares.

A natural variant of the problem, and of two-dimensional packing problems in general, arises when the produced pattern must satisfy guillotine constraints. In this case, each item must be obtainable with a sequence of edge-to-edge cuts parallel to the edges of the bin, where each cut removes a so-called *strip* from the bin (see the left packing in Fig. 1). Guillotine two-dimensional packing has received considerable attention in the literature as imposing this requirement has a minor impact on the solution worsening (see, e.g., Lodi et al. 2017) while being a relevant constraint in real-world applications (e.g., when automatic machines are used to cut the items). Some industrial cutting processes also limit the way of producing a guillotine cutting pattern, imposing an upper bound, say k, on the number of cuts needed to produce each item. The rightmost part of Fig. 1 shows a k-stage pattern for $k = 3$; ILP models for k-stage packing problems have been given in Lodi and Monaci (2013) and Silva et al. (2010) for the cases $k = 2$ and $k = 3$, respectively.

MTRS arises as a subproblem of more complex packing problems, and has some practical applications, e.g., in telecommunications. Consider, for example, the IEEE 802.16-2009 standard which is the basis of Mobile WiMAX; in this protocol, data packets have to be transmitted from a base station to mobile users, and transmission is implemented using different time slots and different frequencies. This can be modelled as a two-dimensional packing problem in which a rectangular *data bin* is used to transmit some rectangular *data packets*; in this model, widths and heights represent time slots and frequencies, respectively, see Lodi et al. (2011) for more details. Each data packet that is transmitted requires additional information to be stored (and coded/decoded), that is possibly reduced in case the packet is sent as a square instead of a rectangle. To maximize the throughput of the system, one is thus interested in filling the entire data bin with the smallest number of square items.

MTRS is also interesting from the computational complexity viewpoint; on the one hand, it is a very simple problem which is not known to be polynomially solvable, while on the other hand no proof of NP-hardness has been proposed in the literature.

The paper is organized as follows. In Sect. 2 we introduce an Integer Linear Programming (ILP) model for MTRS, while Sect. 3 gives two heuristic algorithms for the problem. In Sect. 4 we propose a mathematical model for a relaxation of the problem, and strengthen this relaxation by adding some valid inequalities; alternative exact algorithms based on this relaxation are then outlined in Sect. 5. Finally, Sect. 6 reports an extensive computational analysis of the proposed algorithms on a set of randomly generated instances, and Sect. 7 draws some conclusions.

2 Problem formulation

In this section we give a formal description of the problem we consider, and introduce a mathematical formulation for its solution.

We are given a rectangular bin having integer width W and integer height H. The *Minimum Tiling of a Rectangle by Squares* (MTRS) problem requires to define n square items, so that:

- each item j has an integer side a_j;
- items are orthogonally allocated to the bin without overlapping;
- the set of square items entirely covers the given bin; and
- the number n of used items is a minimum.

Throughout the paper we assume that H and W are positive integers. Noting that the cases $W = 1$ (or $H = 1$) and $W = H$ would lead to trivial optimal solutions (with value H and 1, respectively), we will further assume that $1 < W < H$, possibly rotating the rectangle by 90 degrees if necessary.

To provide an Integer Linear Programming (ILP) formulation for MTRS, we make use of a Cartesian system having axes x and y and assume that the bottom left corner of the bin is at coordinate $(0, 0)$.

As we assume $W < H$, the set of possible items' sizes is $\widetilde{V} = \{1, \dots, W\}$. Observing that an item of size $t \in \widetilde{V}$ can be placed with its bottom-left corner at any (i, j)-coordinate such that $i \in W_t = \{1, \dots, W - t + 1\}$ and $j \in H_t = \{1, \dots, H - t + 1\}$, we can define the following set of decision variables:

$$\alpha_{ijt} = \begin{cases} 1 \text{ if an item of size } t \text{ is placed in position} (i, j); \\ 0 \text{ otherwise} \end{cases} \tag{1}$$

for $t \in \widetilde{V}, i \in W_t, j \in H_t$. This yields the following ILP model for MTRS

$$\min \sum_{t \in \widetilde{V}} \sum_{i \in W_t} \sum_{j \in H_t} \alpha_{ijt} \tag{2}$$

$$\sum_{t\in\widetilde{V}} \sum_{u=\max(0,i-t+1)}^{\min(i,W-t+1)} \sum_{v=\max(0,j-t+1)}^{\min(j,H-t+1)} \alpha_{uvt} = 1 \quad i = 1,\ldots,W;\ j = 1,\ldots,H \tag{3}$$

$$\alpha_{ijt} \in \{0, 1\} \qquad t \in \widetilde{V}; i \in W_t;\ j \in H_t. \tag{4}$$

The objective function (2) minimizes the number of items that are used; constraints (3) impose that any unit square of the bin, with bottom left corner, say, at coordinate (i, j), is occupied by exactly one item. Finally, constraints (4) impose decision variables to be binary.

The formulation above was used in Beasley (1985) for 2BP and in Kurz (2012) for the problem of splitting an $n \times n$ square into square items having size at most $n - 1$. The model has W^2H variables and WH constraints; this may prevent the possibility of directly using it in practice for large values of the W and H parameters. On the contrary, the model can be solved in an efficient way for small values of W and H, possibly producing approximate solutions for MTRS in case the optimum cannot be computed.

3 Heuristic solution of MTRS

In this section we present two heuristic approaches for MTRS. The first one, described in Sect. 3.1, is based on a dynamic programming scheme and is particularly suited for those instances for which the optimal solution corresponds to a pattern that is guillotinable, see again Fig. 1. Conversely the second heuristic, described in Sect. 3.2, is more suitable for patterns that are not guillotinable. Since no dominance exists among the algorithms and the required computing times for their execution are usually very small, we run both heuristics and take the best of the two solutions.

3.1 A recursive guillotine heuristic

Our first heuristic is a dynamic programming procedure that iteratively cuts the given rectangle by means of a guillotine cut in a recursive fashion. If the current rectangle is a square, then a single item is produced. Otherwise, the rectangle is divided into two smaller rectangles, to which the procedure is applied recursively. In this case, to determine the edge-to-edge cut that produces the smallest number of items, we try all possible vertical and horizontal guillotine cuts, and select the one that gives a minimum. More in details, let us introduce a $W \times H$ matrix F such that $F(p, q)$ denotes the minimum number of square items associated with a $p \times q$ rectangle ($p = 1,\ldots,W$, $q = 1,\ldots,H$) using guillotine patterns only. Similarly, assume that $X(p, q)$ denotes the associated solution, expressed in terms of multiset of items. The entries of matrices F and X can be computed according to the scheme reported in Fig. 2.

The following observation states that the algorithm provides an optimal solution to MTRS, if an optimal solution satisfying guillotine constraint exists.

Observation 1 *Algorithm* `GUILL` *produces the best MTRS solution among those that satisfy the guillotine constraint.*

Proof We first prove that the solution produced by the algorithm satisfies the guillotine constraint. Indeed, if the given $W \times H$ rectangle is a square, then a single-item solution is produced. Otherwise, the rectangle is split using an edge-to-edge cut into two smaller rectangles. Assume that the first cut is vertical with some value $k \in [1, W - 1]$, i.e., the solution is obtained by the set of items associated with the $k \times H$ rectangle plus those associated with the $(W - k) \times H$ rectangle (the case in which the first cut is horizontal is

```
Algorithm GUILL:

for $p = 1$ to $W$ do
    for $q = 1$ to $H$ do
        if $p = q$ then set $F[p, q] := 1$ and $X[p, q] := \{p\}$;
        else
            let $k_v := \arg\min_{k=1,\dots,p-1} \left\{F[k, q] + F[p-k, q]\right\}$ and $F_v := F[k_v, q] + F[p-k_v, q]$;
            let $k_h := \arg\min_{k=1,\dots,q-1} \left\{F[p, k] + F[p, q-k]\right\}$ and $F_h := F[p, k_h] + F[p, q-k_h]$;
            if $F_v \le F_h$ then set $F[p, q] := F_v$ and $X[p, q] := X[k_v, q] \cup X[p-k_v, q]$;
            else set $F[p, q] := F_h$ and $X[p, q] := X[p, k_h] \cup X[p, q-k_h]$;
        endif
    endfor
endfor
return solution $X[W, H]$ with value $F[W, H]$
```

Fig. 2 Heuristic algorithm that produces the best guillotine solution

analogous). By recursion, we can show in the same way that both these solutions satisfy the guillotine requirement, as they are composed by either a single item or are obtained merging items associated with solutions of two smaller rectangles that, by recursion, satisfy the guillotine constraint.

Now we prove that the produced solution is indeed the optimal among this that satisfy the guillotine constraint. By contradiction, assume the statement is not true. Without loss of generality, consider now the smallest (W, H) instance for which the algorithm fails in producing the best solution among those that are guillotinable, i.e., assume the recursion scheme is exact for all $\overline{W} \times \overline{H}$ rectangles such that $\overline{W} \le W$, $\overline{H} \le H$ and one of the two inequalities is strict. As the solution found is guillotinable, it is produced by a first cut which divides the bin into two smaller parts, and $F(W, H)$ is the sum of the associated optimal values. Thus, either the solution of one of the smaller parts is not optimal (or both), or a better solution exists that uses a different first cut. The first case is ruled out by our assumption, as both parts are smaller than (W, H). The latter cannot occur as well, since the algorithm considers all possible vertical and horizontal positions for applying the first cut, and returns the best solution among them. □

Though it requires the computation of all the $W \times H$ entries of matrices F and X, the algorithm above is very fast in practice, and usually produces solutions of high quality (see, Sect. 6). We note that an upper bound for the number of items produced by this algorithm can be implicitly derived from the results in Kenyon (1996). Indeed, assuming $W > H/2$, the algorithm given in (Kenyon 1996, Sect. 5) produces a guillotinable solution having a number of items not larger than $C \log W \log \log W$, where C is an universal constant. Using Observation 1 we conclude that the same upper bound holds for our algorithm as well. Finally, we mention that recursive algorithms were used in the literature for solving (either heuristically or in an exact way) different two-dimensional packing problems; see, e.g., Cui et al. (2008) for a heuristic algorithm for 2SP, Birgin et al. (2010) for the approximate solution of the problem of packing identical rectangles into a rectangle, and Dolatabadi et al. (2012) for an exact approach to the *Two Dimensional Knapsack* problem.

3.2 A non-guillotine heuristic

According to the results of the previous section, MTRS can efficiently be solved to optimality in case an optimal solution satisfies the guillotine requirement. Unfortunately, there are MTRS

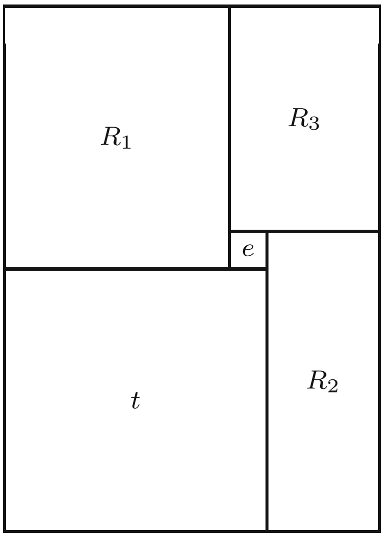

Fig. 3 Non-guillotine patterns exploited by the heuristic algorithm

instances for which the optimal solution corresponds to a non-guillotinable pattern (see, e.g., the central packing in Fig. 1); in these cases, algorithm GUILL produces a heuristic (non necessarily optimal) solution, i.e., an upper bound on the optimal solution value.

Thus, we developed a second heuristic to deal with non-guillotinable patterns that have the structure depicted in Fig. 3, in which one can identify some rectangles $R_j = (w_j, h_j)$, each corresponding to a subset of square items, that encircle a "central" item. This pattern is called *simple blocked ring* in Pietrobuoni (2015), as it corresponds to the non-guillotinable pattern with the smallest number of items. It can be proved that this pattern is the non-guillotinable pattern with the smallest number of items, and that every pattern that is non-guillotinable includes (or may be reconducted to) a simple blocked ring structure (see, Pietrobuoni 2015, Ch. 3 for details).

The non-guillotine heuristic exploits patterns in which one of the rectangles is indeed a square (i.e., a single item) having side t, and the central item has side e. Given the two values $t > e$, the algorithm

1. determines the sizes of the remaining rectangles:
$$R_1 = (t - e, H - t) \qquad R_2 = (W - t, t + e) \qquad R_3 = (W - w_1, H - h_2)\,;$$
2. computes their minimum square guillotine tiling using the dynamic programming algorithm GUILL of Sect. 3.1; and
3. defines a complete solution adding two squares having sides t and e, respectively.

Note that the algorithms runs in constant time if one has executed the first heuristic before, and all entries $F(p, q)$ are known $\forall p = 1, \ldots, W$ and $\forall q = 1, \ldots, H$. Thus, we can execute the algorithm with a large number of tentative (t, e) pairs; in our implementation we used all values from 1 to $W - 1$ for both t and e, provided that $e < t$ and $t + e < H$.

4 Lower bounds for MTRS

In this section we introduce a mathematical model for a simple relaxation of MTRS, allowing to determine a lower bound on the optimal solution value. In Sect. 4.2 we introduce valid inequalities that can be added to the formulation to strengthen the resulting lower bound value.

4.1 A one-dimensional relaxation

Recall that $\widetilde{V} = \{1, \dots, W\}$ denotes the set of possible items' sizes. A simple lower bound on the optimal solution value of an MTRS instance can be obtained by solving the following ILP model

$$\min \sum_{t \in \widetilde{V}} x_t \tag{5}$$

$$\sum_{t \in \widetilde{V}} t^2 x_t = W\,H \tag{6}$$

$$x_t \geq 0 \text{ integer } t \in \widetilde{V} \tag{7}$$

$$\sum_{q=t}^{W} x_q \leq U_t \quad t \in \widetilde{V} \tag{8}$$

$$\sum_{t > W/2} t\,x_t \leq H \tag{9}$$

$$\sum_{t > H/2} t\,x_t \leq W \tag{10}$$

where each variable x_t indicates the number of $t \times t$ items in the solution, and

$$U_t := \min\left(\widetilde{z}, \left\lfloor \frac{W}{t} \right\rfloor \left\lfloor \frac{H}{t} \right\rfloor\right) \tag{11}$$

represents the maximum number of $t \times t$ items to be considered (see below) given a feasible solution with value $\widetilde{z}$.

The model defined by (5)–(7) corresponds to the 1-dimensional relaxation of the problem in which only the area of each candidate item (and of the bin) is taken into account. A similar relaxation was addressed for the two-dimensional knapsack problem in Caprara and Monaci (2004), where the worst-case performance analysis of the resulting bound was established. The relaxed problem turns out to be a Change-Making Problem; although this problem is NP-hard in the general case (see, Lueker 1975), efficient algorithms for its solution have been proposed in the literature (see, e.g., Martello and Toth 1990).

As to constraints (8), note that in any feasible solution the maximum number of $t \times t$ items is bounded by $\lfloor \frac{W}{t} \rfloor \lfloor \frac{H}{t} \rfloor$. As we assume a feasible solution of value $\widetilde{z}$ is available, each variable x_t can be bounded by U_t, defined by (11). This immediately leads to inequalities (8). Finally, inequality (9) bounds the total maximum height of "large" items, i.e., items that are larger than half of the width of the bin and cannot be packed side by side; similarly, (10) gives an upper bound on the total width of "tall" items.

4.2 Strengthening the relaxation

Let us add to the previous model the following additional variables

$$y_{tp} = \begin{cases} 1 & \text{if } x_t = p; \\ 0 & \text{otherwise} \end{cases} \quad \left(t \in \widetilde{V};\, p = 1, \dots, U_t\right) \tag{12}$$

that are linked to the x variables as follows

$$x_t = \sum_{p=1}^{U_t} p\, y_{tp} \quad t \in \widetilde{V} \tag{13}$$

and should satisfy immediate constraints that impose at most one such variable be selected for each possible size t, i.e.,

$$\sum_{p=1}^{U_t} y_{tp} \le 1 \quad t \in \widetilde{V}. \tag{14}$$

Using the y variables, it is possible to add to the formulation a number of constraints, as stated by the following results.

Lemma 1 *The following inequalities*

$$\sum_{p=1}^{U_t} y_{tp} + \sum_{\substack{q=H-t+1 \\ q>t}}^{W} x_q \le 1 \quad t = H - W + 1, \dots, W \tag{15}$$

are valid for any feasible packing.

Proof It is enough to observe that the first term in (15) is 1 only in case some $t \times t$ items have been selected. In this case, no square of side $q \ge H - t + 1$ can be packed. In addition, $q + t > H$ and $q > t$ imply $q > H/2$, which means that $U_q = 1$, i.e., only one $q \times q$ item must be considered; this allows to use variable x_q instead of y_{q1} in (15). □

Lemma 2 *The following inequalities*

$$\sum_{\substack{q=1 \\ q \ne t}}^{W-t} q^2\, x_q \ge t\,(W - t)\, y_{t1} \quad t = 1, \dots, \left\lfloor \frac{W}{2} \right\rfloor \tag{16}$$

and

$$\sum_{q=1}^{W-t} q^2\, x_q \ge t\,(W - t)\, x_t \quad t = \left\lfloor \frac{W}{2} \right\rfloor + 1, \dots, W - 1 \tag{17}$$

are valid for any feasible packing.

Proof Let us prove the validity of (17) for a given value of t. First observe that the inequality is omitted if $t = W$ and is redundant in case $x_t = 0$; thus, assume $x_t > 0$. Since $t > W/2$, the $t \times t$ items cannot be packed side by side, leaving a lateral overall free space of area $(W - t) \times x_t$. This free space must be entirely filled with items whose side is at most $W - t$, which yields (17).

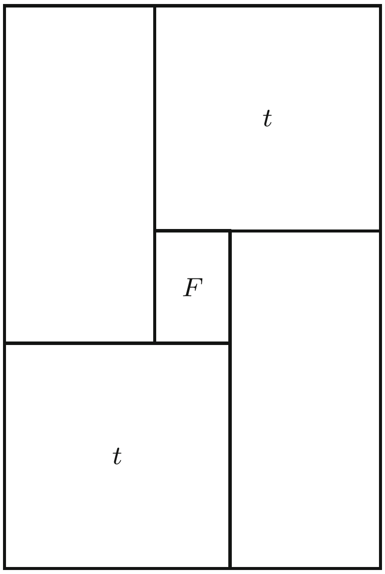

Fig. 4 Free area to be covered in the right-hand side of inequalities (19)

The associated inequality (16) can be proved in a similar way, the only difference being that we cannot rule out the possibility that $t \times t$ items are packed side by side. Thus, we have to weaken the constraint by considering the case in which only one such item is packed, i.e., using variable y_{t1}. □

Lemma 3 *The following inequalities*

$$\sum_{\substack{q=1\\q\neq t}}^{\min\{H-t,W\}} q^2\, x_q \geq t\,(H-t)\, y_{t1} \quad t = 1,\ldots,W \tag{18}$$

and

$$\sum_{\substack{q=1\\q\neq t}}^{\min\{H-kt,W\}} q^2\, x_q \geq (2t-W)\,(H-kt)\, y_{tk} \quad t = \left\lfloor \frac{W}{2} \right\rfloor + 1,\ldots,W; \quad k = 2,\ldots,U_t \tag{19}$$

are valid for any feasible packing.

Proof For a given value of t, inequality (18) is non redundant only in case $y_{t1} = 1$, i.e., only one $t \times t$ item is packed; in this case, the proof of validity is identical to that for inequalities (16), swapping the roles of W and H and recalling that each item has size at most W.

As to inequalities (19), let $t > W/2$ be an item size and $k \geq 2$ the number of $t \times t$ items packed in the solution. Noting that these items cannot be packed side by side, there is a free space (denoted by F in Fig. 4) of height $H - kt$ and width (at least) $2t - W$ that can allocate only items whose size is at most $H - kt$, which concludes the proof. □

The results associated with "large" items in Lemmas 2 and 3 can be generalized as follows:

Lemma 4 *The following inequalities*

$$\sum_{q=1}^{W-t} q^2 x_q \geq \sum_{q=t}^{W-1} q\,(W-q)\,x_q \quad t = \left\lfloor \frac{W}{2} \right\rfloor + 1, \ldots, W-1 \tag{20}$$

and

$$\sum_{q=1}^{\min\{H-t,W\}} q^2 x_q \geq \sum_{q=t}^{W} q\,(H-q)\,x_q \quad t = \left\lfloor \frac{H}{2} \right\rfloor + 1, \ldots, W \tag{21}$$

are valid for any feasible packing.

Proof Consider inequality (20) for a certain $t > W/2$, and all items with side at least t in the solution; as all such items are larger than $W/2$, they cannot be packed side by side. Thus, the lateral free area alongside these items is given by the right-hand side in (20), and must be filled by using items whose size is at most $W - t$. The validity of (21) can be proved in a similar way swapping the roles of W and H. □

Finally, we address the special case in which the solution includes a number of items whose size is equal to the width of the bin.

Lemma 5 *Let $p \in \{1, \ldots, U_W\}$, and define $\overline{W} = \min(W, H - p\,W)$ and $\overline{H} = \max(W, H - p\,W)$. Then, the following inequalities*

$$\sum_{q=1}^{\overline{W}-t} q^2 x_q \geq \sum_{q=t}^{\overline{W}-1} q\,\left(\overline{W}-q\right)\,x_q - M\,\left(1 - y_{Wp}\right) \quad t = \left\lfloor \frac{\overline{W}}{2} \right\rfloor + 1, \ldots, \overline{W}-1 \tag{22}$$

and

$$\sum_{q=1}^{\min\{\overline{H}-t,\overline{W}\}} q^2 x_q \geq \sum_{q=t}^{\overline{W}} q\,\left(\overline{H}-q\right)\,x_q - M\,\left(1 - y_{Wp}\right) \quad t = \left\lfloor \frac{\overline{H}}{2} \right\rfloor + 1, \ldots, \overline{W} \tag{23}$$

where M is a "sufficiently large" value, are valid for any feasible packing.

Proof First, observe that all inequalities associated with a given value of p are deactivated if $y_{Wp} = 0$, i.e., in case the number of $W \times W$ items in the solution is different from p. Otherwise, all such items must be packed one above the other, leaving a residual bin with width $\overline{W}$ and height $\overline{H}$. Thus, (22) and (23) can be derived exactly as (20) and (21) respectively. □

As already observed, each inequality (23) is a conditional constraint that is active only if the associated y_{Wp} variable is set to 1. Otherwise the big-M coefficient makes the constraint redundant, provided its value is "large enough" to deactivate the inequality when $y_{Wp} = 0$. Needless to say, using a too large value may lead to very weak relaxations, which makes hard the definition of suitable values for big-M coefficients. However, we observe that modern ILP solvers include some coefficient strengthenings that typically produce preprocessed models that, according to our computational experience, are not too hard to solve.

We conclude this section with the following result:

Theorem 1 *The optimal solution of ILP model (5)–(10), (13)–(23) provides a lower bound on the optimal MTRS solution value.*

Proof Immediate from Lemmas 1–5. □

5 Exact approaches to MTRS

In this section we examine different exact approaches to the solution of MTRS. The first algorithm applies an ILP solver to the formulation given in Sect. 2, whereas the remaining schemes are based on the iterative solution of the relaxation introduced in Sect. 4.

5.1 Approach 1: direct use of an ILP solver

An immediate way for solving MTRS is to run any ILP solver on the mathematical model (2)–(4) given in Sect. 2. To take full advantage of the internal heuristics that are commonly embedded in commercial ILP solvers, the enumerative algorithm can be initialized with a heuristic solution. For example, one can define a "dummy" solution in which the rectangular bin is split into $W \times H$ unit-square items or, even better, compute a feasible solution by executing the heuristic algorithms described in Sect. 3.

As already anticipated, this simple formulation may be solved in a very efficient way for small values of W and H. For larger instances, the solver may not be able to provide an optimal solution, but can possibly produce an improved heuristic solution. However, large values of W and H produce a huge number of variables and constraints, which makes it impossible to solve the associated model and, in some cases, even to define it due to memory requirement.

5.2 Approach 2: enumerate-and-cut algorithm

The second approach is based on the relaxed formulation introduced in Sect. 4.1, possibly strengthened using the inequalities of Sect. 4.2.

An exact formulation for MTRS can be derived by adding to the relaxation above the following inequalities

$$\sum_{(t,k)\in C} y_{tk} \le |C| - 1 \quad C \in \mathcal{C} \tag{24}$$

where $\mathcal{C}$ denotes the set of subsets of y variables associated with sets of items C that cannot be completely allocated into the rectangular bin.

In practice, it is not possible to explicitly generate all inequalities (24), as they are exponentially many. Thus, we developed an algorithm, denoted as `Enumerate&Cut`, that starts with no constraints (24); at each iteration the current relaxation is solved, and feasibility of the current solution is checked, using as a black box the *feasibility test* procedure described in Sect. 5.2.1. If the current item set can be packed into the bin, the current solution is feasible, hence optimal, for MTRS. Otherwise, a new item set $C \in \mathcal{C}$ is determined, the associated inequality (24) is added on the fly to the formulation, and the procedure is iterated.

This approach can be seen as a Benders' decomposition (Benders 1962) in which the feasibility check for the slave corresponds to the problem of checking whether a given set of items fits into a bin or not. Indeed this feasibility check requires the solution of an NP-hard problem, and turns out to be by far the most time consuming part of the computation for this approach.

Finally observe that, given an item set $C \in \mathcal{C}$, one can derive a number of valid inequalities (24). In particular, one can define another item set $C' \subset C$ and check if all items in C' can be packed into the $W \times H$ rectangle or not; in this latter case, a new inequality (24) associated with item set C' can be defined and added to the formulation. This procedure can be iterated to define further subsets of items and possibly produce new cuts. Since the number of potential

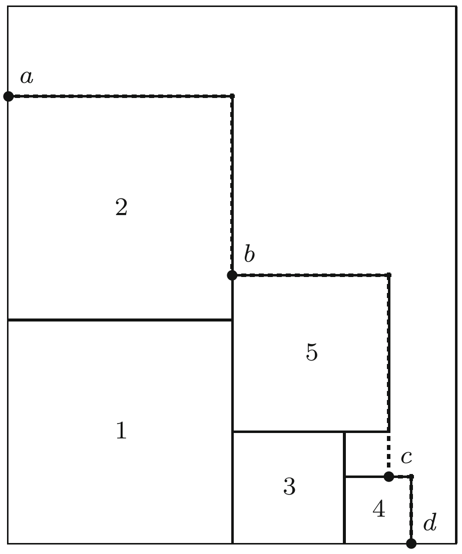

Fig. 5 Envelope and corner points associated with item set $S = \{1, 2, 3, 4, 5\}$

subsets to be considered is exponential in $|C|$ and each feasibility test may require some computing time, in our implementation we prefer to use a non-aggressive policy that keeps the number of generated cuts under control. In particular, we only consider subsets C' defined as $C' = C \backslash \{j\}$, where j denotes the smallest item in C and $C \in \mathcal{C}$, and stop the procedure as soon as we detect an item set C that allows a feasible $W \times H$ packing. In the following, we will denote this option as the *aggressive* cut generation policy.

5.2.1 Checking feasibility

The problem of checking the feasibility of a given set of items can be stated either as a 2BP (checking if the optimal solution value is equal to 1) or as a 2SP (checking if a packing exists with height not larger than H). In our algorithm we used the latter, and adapted the enumerative scheme (proposed in Martello et al. 2003 for 2SP) that packs one item at a time according to the concepts of *envelope* and *corner points* (see Fig. 5). Let S denote the set of items that are actually allocated at a certain node of the enumerative tree; then, we compute the $O(|S|)$ corner points for the current node, and generate a number of descendant nodes, each associated to the placement of each item $j \notin S$ in each corner point (see Martello et al. 2003 for details).

To speed up enumeration, at each node we use only simple fathoming criteria, based on the consideration that any feasible solution cannot leave uncovered areas. Thus, a backtracking occurs if one of the following conditions holds:

- the area below the current envelope is strictly smaller than the area of the items is S;
- a corner point exists in which no item $j \notin S$ can be allocated;
- an item $j \notin S$ exists that cannot be allocated to any corner point.

The algorithm is halted as soon as a feasible solution is found, i.e., when all items have been allocated without overlapping to the given rectangle, or when it can prove infeasibility for the current item set.

5.3 Approach 3: branch-and-cut algorithm

A third exact algorithm, called `Branch&Cut`, can be obtained by integrating the feasibility check described in Sect. 5.2.1 within the ILP solver. In particular, we assume that the solver can be controlled through a callback function invoked each time the incumbent is going to be updated—as it happens in many modern solvers. In our implementation, we run the solver on a relaxed formulation, using the feasibility check every time a candidate set of items is found. If the current item set turns out to be infeasible, a new cut is added on the fly; otherwise, the incumbent solution is updated. In both cases, enumeration continues until the entire branch-and-bound tree is explored.

The resulting algorithm, that resembles the scheme proposed in Miliotis (1976) for the Travelling Salesman Problem, may turn out to be advantageous as it explores a single tree, including the (generally time consuming) root node that involves preprocessing, cut generation, and so on. On the other hand, the use of callback functions may deactivate some properties of the solver, which can turn in a worsening of the algorithm's performances. In addition, the algorithm may have to check feasibility for a large number of item sets. For this reason, the algorithm is improved by stopping the check for the current set of items after a (short) time limit, so as to restore enumeration. If this happens, possible item sets for which the check has not been completed are evaluated, sorted according to their cardinality, after the complete enumeration terminated.

5.4 Approach 4: hybrid strategy

Algorithm `Branch&Cut` of Sect. 5.3 may require the execution of the feasibility check for many item sets S whose size is larger than the optimal solution value. This operation may be time consuming, as the performances of the check strongly depend on the number of items to be allocated, and turns out to be a waste of time whenever an improving solution is found. Thus, we developed a fourth algorithm, denoted as `Hybrid`, that executes the branch-and-cut algorithm of Sect. 5.3 in an iterative fashion: at iteration k, only item sets S with $|S| = k$ are evaluated. The algorithm terminates as soon as a feasible solution is found.

The iterative nature of the algorithm makes it quite similar to the second approach, described in Sect. 5.2, with two main differences. First, for each possible value of the optimal solution, only one branch-and-bound tree is developed. Second, the additional constraint on the objective function value allows the solver to activate many internal propagation procedures, that may be extremely useful in terms of computing time.

6 Computational experiments

All algorithms described in the previous sections were coded in C language and run on an Intel Xeon E3-1220 V2 in single-thread mode with a time limit of 600 s per instance. All the ILP models were solved by using `IBM-ILOG` Cplex 12.6.3 (`CPLEX` in the following), possibly using callback functions.

To test the effectiveness of our algorithms, we randomly generated a large set of instances, as follows. We first generate the width W of the bin as an uniform integer in $[\underline{W}, \overline{W}]$, where $\overline{W} > \underline{W}$ are two integer parameters. Then, the height H of the rectangle is generated as a random integer in the interval $[\alpha W + 1, \beta W]$, where $\beta > \alpha \geq 1$ are two additional parameters. We generated instances with $(\underline{W}, \overline{W}) \in \{(3, 19), (20, 39), (40, 59)\}$ and $(\alpha, \beta) \in \{(1, 2), (1, 5), (1, 10), (2, 5)\}$. For each combination of the parameters (called

Table 1 Initial heuristic and relaxations on MTRS instances

Instances		Heuristic			L_0		L_1		L_2		L_3	
$\underline{W}, \overline{W}$	α, β	Best	Ratio	Time	Ratio	Time	Ratio	Time	Ratio	Time	Ratio	Time
3, 19	1, 2	10	1.000	0.000	0.552	0.11	0.663	0.11	0.789	0.05	0.858	0.04
	1, 5	8	1.017	0.000	0.604	0.05	0.620	0.05	0.866	0.05	0.914	0.05
	1, 10	9	1.017	0.001	0.662	0.06	0.662	0.04	0.862	0.09	0.898	0.04
	2, 5	10	1.000	0.000	0.593	0.02	0.593	0.02	0.857	0.02	0.919	0.02
20, 39	1,2	7	1.035	0.000	0.346	0.03	0.449	0.06	0.593	0.07	0.703	0.08
	1, 5	8	1.018	0.001	0.381	0.04	0.402	0.05	0.583	0.09	0.691	0.08
	1, 10	9	1.007	0.003	0.510	0.04	0.510	0.04	0.693	0.08	0.760	0.10
	2, 5	6	1.045	0.002	0.404	0.05	0.404	0.05	0.612	0.10	0.707	0.09
40, 59	1,2	7	1.032	0.002	0.338	0.08	0.401	0.08	0.519	0.19	0.606	0.14
	1, 5	9	1.010	0.007	0.389	0.06	0.389	0.07	0.569	0.23	0.630	0.19
	1, 10	8	1.019	0.012	0.435	0.09	0.435	0.07	0.573	0.20	0.678	0.23
	2, 5	9	1.008	0.007	0.379	0.05	0.379	0.05	0.569	0.25	0.658	0.30
Overall		100	1.017	0.003	0.466	0.06	0.492	0.06	0.674	0.12	0.752	0.11

classes hereafter), we randomly generated 10 instances, yielding a benchmark of 120 instances. To avoid possible scaling of the instances, we considered only problems for which W and H are coprime.

In our experiments we considered different ILP models for MTRS. A valid formulation, that we will denote by F0, is obtained by considering (5)–(10) and (24). Then, we will consider 3 alternative ways to define stronger formulations, namely:

– formulation F1 obtained adding (13)–(15) to F0;
– formulation F2 obtained adding (16)–(19) to F1;
– formulation F3 obtained adding (20)–(23) to F2. As to inequalities (22)–(23), in our experiments we used $M = 10{,}000$.

In the following, we will denote by L_0, L_1, L_2 and L_3 the lower bounds obtained relaxing inequalities (24) by these four formulations, respectively.

Table 1 reports the results of our experiments for what concerns the heuristic algorithms of Sect. 3 and the four relaxations above. Each line of Table 1 gives results for the 10 instances of the same class. For the heuristic algorithms we consider the best among the two solutions produced by the algorithms in Sects. 3.1 and 3.2, and counted the number of cases in which this solution is equal to the optimal solution for a given instance (column “best”). In case the optimal was not available, the heuristic was compared with the best known solution for that instance. In addition, we report the average ratio between the heuristic and the best solution values (column “ratio”) and the average computing time for producing the heuristic solution (“time”); this is a very tight approximation of the computational effort to execute the dynamic programming algorithm, as computing time for the non-guillotine heuristic is always negligible. As to lower bounds, we report, for each relaxation, the percentage ratio between the associated lower bound and the best solution value (“ratio”) and the required computing time (“time”).

The results in Table 1 show that the heuristic algorithm provides very good solutions, which can be improved only in a few cases by the exact algorithms. Relaxations L_0 and L_1 can be computed very efficiently, but the associated lower bounds are quite poor. Adding inequalities

Table 2 Exact algorithms with different initial formulations

Initial formulation	`Enumerate&Cut`				`Branch&Cut`				`Hybrid`			
	Opt	Time	#c	#n	Opt	Time	#c	#n	Opt	Time	#c	#n
F0	33	34.89	265	166,314	42	62.26	3850	32,135	42	39.33	4676	31,373
F1	33	26.15	215	133,011	43	75.74	3185	45,546	43	41.77	2037	32,927
F2	56	19.95	117	76,496	68	56.01	961	30,046	67	44.87	782	43,116
F3	61	26.64	104	79,922	78	54.26	793	43,892	81	74.95	878	75,155

(16)–(19) has a considerable impact in the strengthening of the relaxation, especially on large instances, though the required computing times increase by an average factor of 2. Finally, the addition of inequalities (20)–(23) to define formulation F3 seems to have a minor impact on the computing time, while producing some additional improvement of the associated lower bound.

Our second set of experiments is aimed at evaluating the performances of the three exact algorithms of Sect. 5. All these methods start from an initial formulation, that is strengthened adding valid cuts (24) on the fly; in this round of experiments, only one valid cut is added at each iteration, i.e., the aggressive cut generation strategy described in Sect. 5.2 is not applied. To better understand the effect of the initial formulation on the performance of the different algorithms, we executed each algorithms several times, using the four different formulations mentioned above. At each run, the solver was initialized with the best heuristic solution obtained applying the algorithms of Sects. 3.1 and 3.2.

Table 2 reports, for each algorithm and formulation, the number of instances (out of 120) solved to proven optimality within the given time limit (equal to 600 s) and the average computing time (in seconds, column "time"), the average number of cuts that were added during enumeration ("#c") and the average number of nodes explored ("#n"). Average values are taken with respect to the instances solved to proven optimality only.

These results confirm that, for all methods, the tighter the initial formulation the better the associated results. Indeed, tightening the formulation typically produces an increase in the number of instances that are solved to optimality, together with a reduction in the average number of cuts and nodes. Actually, adding all valid inequalities (13)–(23) to the initial formulation, i.e., considering formulation F3, seems to be beneficial for all algorithms in terms of capability of producing MTRS optimal solutions. This is mainly due to the fact that the number of cuts that are added to enforce two-dimensional feasibility is reduced, with respect to F1, by a factor ranging from 2.5 to more than 5; as we already observed, indeed, most of the computing time is typically spent within the feasibility check procedure described in Sect. 5.2.1. For these reasons, in what follows, we will always use formulation F3 for our experiments.

Table 3 reports the performances of the three proposed methods without and with the aggressive cut generation described in Sect. 5.2. Remind that, in this case, a number of valid inequalities is generated for each infeasible item set C.

The results of Table 3 show that an aggressive cut policy is beneficial for the first two algorithms, yielding a reduction of the average computing time of algorithm `Enumerate&Cut` by one third, and allowing the optimal solution of two more instances when using algorithm `Branch&Cut`. Conversely, this tuning produces a certain slowdown when applied to algorithm `Hybrid`, that solves less instances than in the default settings. Finally observe that, as expected, the number of feasibility cuts (24) generated is considerably larger than in the

Table 3 Exact algorithms with and without aggressive cut generation

Aggressive cut generation	`Enumerate&Cut`				`Branch&Cut`				`Hybrid`			
	Opt	Time	#c	#n	Opt	Time	#c	#n	Opt	Time	#c	#n
OFF	61	26.64	104	79,922	78	54.26	793	47,470	81	74.95	878	75,155
ON	61	17.67	217	47,424	80	70.15	1929	36,164	78	72.74	1678	47,189

previous case (typically, but a factor of 2), whereas the number of nodes is reduced (by an average factor ranging from 1.3 to 1.6).

According to these computational experiments, the best tuning for algorithms `Enumerate&Cut` and `Branch&Cut` is obtained by applying the aggressive cut policy; the resulting algorithms will be denoted as `Enumerate&Cut`$^+$ and `Branch&Cut`$^+$, respectively. On the contrary, the best tuning for algorithm `Hybrid` is given by the default setting. For the following experiments, we will consider all algorithms in their best tuning.

Table 4 reports more detailed results for what concerns the exact methods of Sect. 5 (executed with their best tuning) and compares their performances with those obtained with the direct application of an ILP solver to model (2)–(4). In this case too, each algorithm is initialized with the best heuristic solution obtained applying the algorithms of Sects. 3.1 and 3.2. For each approach we report the number of instances solved to proven optimality ("opt"), the average computing time in seconds ("time") and the number of branch-and-bound nodes ("#n"). To keep memory requirement under control, we did not try to solve ILP model (2)–(4) if the number of non-zero coefficients in the constraint matrix exceeded 100 millions. For these instances neither the computing time nor the number of nodes are considered in the statistics; to allow a more meaningful comparison, we also report the number of such instances in column "#f". As to the remaining three approaches (`Enumerate&Cut`, `Branch&Cut` and `Hybrid`), the numbers of nodes refer to all the iterations of the algorithms.

Table 5 reports the same information as Table 4 but considering, for each approach, only the instances that were solved to proven optimality within the time limit. For this reason we do not report column "#f" for the first approach.

These results suggest that using a state-of-the-art ILP solver may be a viable option only for instances with small bin sizes, whereas this approach cannot be used for medium and large instances—in many cases, the model could not even start for memory reasons. However, when optimality is proven, a very small number of enumeration nodes are required. Algorithm `Enumerate&Cut`$^+$ is able to solve only half of the instances, mainly those of small size. On the contrary, algorithms `Branch&Cut`$^+$ and `Hybrid` have better performances in terms of number of optimal solutions and computing times. In particular, the latter qualifies as the best algorithms, at least on this testbed, as it is able to solve to optimality more than 2/3 of the instances with an average computing time of 75 s.

6.1 Perfect tiling

Our last experiments concerns the variant of MTRS in which all the produced squares must be different. Such a solution, if any, is called a *perfect tiling* of the given rectangle.

This constraint can be easily incorporated in model (2)–(4) by adding, for each possible size $t \in \widetilde{V}$ a new constraint $\sum_{i \in W_t} \sum_{j \in H_t} \alpha_{ijt} \leq 1$ that bounds the number of items of size t. In a similar way, one can set $x_t \leq 1$ $(t \in \widetilde{V})$ to model perfect tiling in the relaxation introduced in Sect. 4.1.

Table 4 Exact algorithms on the entire benchmark

Instances		CPLEX				Enumerate&Cut$^+$			Branch&Cut$^+$			Hybrid		
$\underline{W}, \overline{W}$	α, β	Opt	Time	#n	#f	Opt	Time	#n	Opt	Time	#n	Opt	Time	#n
3, 19	1, 2	10	0.69	0	0	10	0.09	102	10	0.06	90	10	0.09	140
	1, 5	10	4.55	5	0	10	5.83	4259	10	2.26	607	10	12.03	1756
	1, 10	10	8.58	0	0	10	0.30	795	10	0.88	184	10	0.26	449
	2, 5	10	1.57	0	0	10	0.06	71	10	0.03	45	10	0.07	51
20, 39	1,2	10	54.07	3	0	6	246.04	674,439	10	105.95	57,110	9	126.73	111,478
	1, 5	6	287.72	10	0	4	372.56	589,355	8	227.45	64,939	8	240.45	171,384
	1, 10	6	336.03	4	0	7	242.16	391,693	8	154.13	44,324	8	166.60	105,966
	2, 5	5	416.49	24	0	3	426.77	707,764	8	223.89	50,981	9	196.48	118,926
40, 59	1,2	4	490.16	4	0	0	600.00	1,448,453	2	542.59	192,646	2	549.19	627,883
	1, 5	0	600.00	0	3	1	554.01	997,449	1	557.30	112,380	1	543.49	560,025
	1, 10	0	600.00	0	3	0	600.00	1,081,316	1	582.47	155,831	2	567.30	588,105
	2, 5	0	600.00	0	4	0	600.00	1,025,331	2	564.18	170,853	2	544.43	565,300
Overall		71	254.53	4	10	61	303.98	576,775	80	246.77	70,832	81	245.59	237,622

Table 5 Exact algorithms on instances solved to optimality

Instances		CPLEX			Enumerate&Cut$^+$			Branch&Cut$^+$			Hybrid		
$\underline{W}, \overline{W}$	α, β	Opt	Time	#n	Opt	Time	#n	Opt	Time	#n	Opt	Time	#n
3, 19	1, 2	10	0.69	0	10	0.09	102	10	0.06	90	10	0.09	140
	1, 5	10	4.55	5	10	5.83	4259	10	2.26	607	10	12.03	1756
	1, 10	10	8.58	0	10	0.30	795	10	0.88	184	10	0.26	449
	2, 5	10	1.57	0	10	0.06	71	10	0.03	45	10	0.07	51
20, 39	1, 2	10	54.07	3	6	10.06	45,946	10	105.95	57,110	9	74.15	78,643
	1, 5	6	79.53	16	4	31.41	108,082	8	134.32	71,748	8	150.57	157,753
	1, 10	6	160.05	7	7	88.79	203,729	8	42.66	16,059	8	58.25	42,208
	2, 5	5	232.97	30	3	22.57	88,072	8	129.86	58,484	9	151.64	123,616
40, 59	1, 2	4	325.41	4	0	–	–	2	312.95	169,031	2	345.93	379,680
	1, 5	0	–	–	1	140.14	442,275	1	172.98	129,173	1	34.93	97,143
	1, 10	0	–	–	0	–	–	1	424.74	213,967	2	436.51	483,490
	2, 5	0	–	–	0	–	–	2	420.92	230,635	2	322.13	410,062
Overall		71	64.77	5	61	17.67	47,424	80	70.15	36,164	81	74.95	75,155

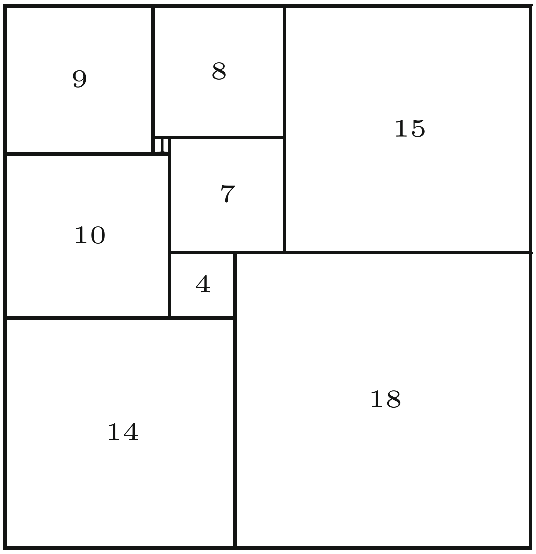

Fig. 6 Perfect tiling of a 32 × 33 rectangle (the numbers denote the sizes of the items)

Table 6 Exact algorithms for perfect tiling of a 32 × 33 rectangle

Seed	`CPLEX`		`Enumerate&Cut`$^+$		`Branch&Cut`$^+$		`Hybrid`	
	Time	#n	Time	#n	Time	#n	Time	#n
Default	98.62	46	6.87	5261	7.35	2312	29.54	5641
1	29.54	0	8.58	6299	37.18	1719	27.93	4727
2	88.54	29	4.92	4225	27.74	2469	29.04	6029
3	100.41	44	6.95	5032	52.18	1545	28.64	6299
4	82.77	24	5.74	4457	68.03	2442	26.93	5499
Average	79.98	29	6.61	5054	38.50	2097	28.42	5639

To test the resulting algorithm in these settings, we ran each algorithm on the smallest rectangle for which a perfect tiling exists, defined by $W = 32$ and $H = 33$. Actually, all the procedures were able to find an optimal solution of value 9; Fig. 6 shows an optimal perfect tiling for the 32 × 33 rectangle. In all cases, the associated computing time was analogous to those reported in Table 4 for instances of similar sizes, though the initial heuristics were not able to produce a feasible perfect tiling, thus they were disabled.

To possibly reduce the erratic behaviour of ILP solvers (see, e.g., Fischetti and Monaci 2014) we ran each algorithm 5 times, using different random seeds for the ILP solver (namely, the default seed and four additional seeds). Table 6 reports computing time and number of nodes for each execution of each algorithm.

The results in Table 6 indicate that, for this specific instance, the best algorithm is `Enumerate&Cut`$^+$, that performs consistently better than the other approaches, with average computing time less than 7 s. As to the other algorithms, the best option is given by algorithm `Hybrid` that has a computing time (and number of nodes) that is almost independent on the random seed (which is not the case, e.g., for algorithm `Branch&Cut`$^+$).

7 Conclusions

In this paper we introduced a new two-dimensional packing problem in which one is required to split a given rectangular bin into a minimum number of non-overlapping square items. For this problem we proposed an ILP model and a heuristic algorithm that is proved to produce the optimal solution in relevant situations. Then, we proposed a mathematical model for a relaxation of the problem, which is embedded into two enumerative schemes. Computational experiments are provided to test the effectiveness of the algorithms on a set of randomly generated instances. The results suggest that the direct use of an ILP solver is a viable option when small instances are concerned, whereas it is not suitable when large instances are addressed, or when additional constraints are required, such as those imposing the produced tiling be perfect. In this case, an alternative approach may be used, in which an easily solvable relaxation is defined and valid constraints are added on the fly, until a proven optimal solution is found.

Acknowledgements This research was supported by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Brazil under Grant PVE 030479/2013-01. Thanks are due to an anonymous referee for helpful comments.

References

Beasley, J. (1985). An exact two-dimensional non-guillotine cutting tree search procedure. *Operations Research*, *33*, 49–64.

Beaumont, O., Boudet, V., Rastello, F., & Robert, Y. (2002). Partitioning a square into rectangles: NP-completeness and approximation algorithms. *Algorithmica*, *34*, 217–239.

Benders, J. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, *4*, 238–252.

Birgin, E., Lobato, R., & Morabito, R. (2010). An effective recursive partitioning approach for the packing of identical rectangles in a rectangle. *Journal of the Operational Research Society*, *61*, 306–320.

Brooks, R., Smith, C., Stone, A., & Tutte, W. (1940). The dissection of rectangles into squares. *Duke Mathematics Journal*, *7*, 312–340.

Caprara, A., & Monaci, M. (2004). On the two-dimensional knapsack problem. *Operations Research Letters*, *32*, 5–14.

Cui, Y., Yang, Y., Cheng, X., & Song, P. (2008). A recursive branch-and-bound algorithm for the rectangular guillotine strip packing problem. *Computers and Operations Research*, *35*, 1281–1291.

Dolatabadi, M., Lodi, A., & Monaci, M. (2012). Exact algorithms for the two-dimensional guillotine knapsack. *Computers and Operations Research*, *39*, 48–53.

Fischetti, M., & Monaci, M. (2014). Exploiting erraticism in search. *Operations Research*, *62*, 114–122.

Kenyon, R. (1996). Tiling a rectangle with the fewest squares. *Journal of Combinatorial Theory*, *76*, 272–291.

Kurz, S. (2012). Squaring the square with integer linear programming. *Journal of Information Processing*, *20*, 680–685.

Lodi, A., Martello, S., Monaci, M., Cicconetti, C., Lenzini, L., Mingozzi, E., et al. (2011). Efficient two-dimensional packing algorithms for mobile WiMAX. *Management Science*, *57*, 2130–2144.

Lodi, A., & Monaci, M. (2013). Integer linear programming models for 2-staged two-dimensional knapsack problems. *Mathematical Programming*, *94*, 257–278.

Lodi, A., Monaci, M., & Pietrobuoni, E. (2017). Partial enumeration algorithms for two-dimensional bin packing problem with guillotine constraints. *Discrete Applied Mathematics*, *217*, 40–47.

Lodi, M., Martello, M., Monaci, M., & Vigo, D. (2010). Two-dimensional bin packing problems. In: *Paradigms of combinatorial optimization* (pp. 107–129). Wiley/ISTE.

Lueker, G. (1975). Two NP-complete problems in nonnegative integer programming. Technical report, Report No. 178, Computer Science Laboratory, Princeton.

Martello, S., Monaci, M., & Vigo, D. (2003). An exact approach to the strip packing problem. *INFORMS Journal on Computing*, *15*, 310–319.

Martello, S., & Toth, P. (1990). *Knapsack problems: Algorithms and computer implementations*. Chichester: Wiley.

Miliotis, P. (1976). Integer programming approaches to the travelling salesman problem. *Mathematical Programming*, *10*, 367–378.

Pietrobuoni, E. (2015). Two-dimensional bin packing problem with guillotine restrictions. Ph.D. thesis, University of Bologna, Bologna, Italy. http://amsdottorato.unibo.it/6810/.

Silva, E., Alvelos, F., & Valrio de Carvalho, J. (2010). An integer programming model for two- and three-stage two-dimensional cutting stock problems. *European Journal of Operational Research*, *205*, 699–708.

Walters, M. (2009). Rectangles as sum of squares. *Discrete Mathematics*, *309*, 2913–2921.