



Discrete Optimization

The board packing problem

Gyula Abraham^a, Gyorgy Dosa^{a,*}, Lars Magnus Hvattum^b, Tomas Attila Olaj^a, Zsolt Tuza^{a,c}^a University of Pannonia, Veszprém, Hungary^b Faculty of Logistics, Molde University College, Norway^c Alfréd Rényi Institute of Mathematics, Budapest 1053, Hungary

ARTICLE INFO

Article history:

Received 28 March 2022

Accepted 14 January 2023

Available online 18 January 2023

Keywords:

Combinatorial optimization

Genetic algorithm

Binary integer programming

Covering

Location problems

ABSTRACT

We introduce the board packing problem (BoPP). In this problem we are given a rectangular board with a given number of rows and columns. Each position of the board has an integer value representing a gain, or revenue, that is obtained if the position is covered. A set of rectangles is also given, each with a given size and cost. The objective is to purchase some rectangles to place on the board so as to maximize the profit, which is the sum of the gain values of the covered cells minus the total cost of purchased rectangles. This problem subsumes several natural optimization problems that arise in practice. A mixed-integer programming model for the BoPP problem is provided, along with a proof that BoPP is NP-hard by reduction from the satisfiability problem. An evolutionary algorithm is also developed that can solve large instances of BoPP. We introduce benchmark instances and make extensive computer examinations.

© 2023 The Author(s). Published by Elsevier B.V.

This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>)

1. Introduction

We are given a rectangular board with m rows and n columns. Let $M = \{1, \dots, m\}$ be the set of rows and $N = \{1, \dots, n\}$ be the set of columns. Each position $(i, j) \in M \times N$ of the board has an integer value $g_{i,j}$ representing a revenue to be obtained if the position is covered. A set R of rectangles are given, each $r \in R$ with a given height h_r , width w_r , and cost c_r . The objective is to purchase rectangles to place on the board so as to maximize the profit, which is equal to the values of the covered positions minus the cost of purchasing rectangles. The revenues, heights, widths, and costs are expressed as integers. Rectangles must be placed with sides parallel to the sides of the board. They are allowed to overlap, but the revenue from a given board position can only be collected once.

The previous paragraph describes the *board packing problem* (BoPP), which is the object of study in this paper. Figure 1 illustrates an instance of the problem on a board with $m = 6$ rows and $n = 5$ columns. The revenues $g_{i,j}$, which may be negative, are given for each board position. Assuming that two rectangles are available for purchase, with $h_1 = w_1 = 2$, $h_2 = 3$, $w_2 = 2$, and $c_1 = 22$ and $c_2 = 33$, the figure also illustrates an optimal solution produc-

ing a profit of 51. The rectangles are placed with overlap, as other solutions are less profitable.

In a preliminary study of a board packing problem, Dosa, Hvattum, Olaj, & Tuza (2020) looked at the corresponding problem where rectangles are not allowed to overlap. They presented a mathematical model and used a commercial mixed-integer programming solver to find optimal solutions on a small set of test instances. This is closely related to the rectangle blanket problem (Demiröz, Altinel, & Akarun, 2019), which has applications in computer vision problems, including template matching and people tracking. When we allow rectangles to overlap in BoPP, the motivation is to cover applications where rectangular resources can be purchased and applied on a grid to obtain a benefit. For example, the rectangles could represent satellite images (Jang, Choi, Bae, & Choi, 2013) that can be taken to obtain useful or profitable information about geographical regions, with uses ranging from post-disaster damage assessment (Ghaffarian & Kerle, 2019) to highway road maintenance (Guo, Li, & Lou, 2021) or detection of archaeological features (Mondino, Perotti, & Piras, 2012).

Other related optimization problems have been studied in the literature. Demiröz et al. (2019) discussed problem variants within covering and partitioning problems as well as cutting and packing problems, to highlight their relationships to the rectangle blanket problem. In the following, we focus on related facility location problems as well as related geometric covering problems.

Facility location problems constitute a first direction of related research, where the correspondence to BoPP arises when consider-

* Corresponding author.

E-mail addresses: abraham.gyula@virt.uni-pannon.hu (G. Abraham), dosagy@almos.uni-pannon.hu, dosagy@almos.vein.hu (G. Dosa), lars.m.hvattum@himolde.no (L.M. Hvattum), tomas@olaj.no (T.A. Olaj), tuza@dcs.uni-pannon.hu (Z. Tuza).

9	13	11	4	13
18	−10	18	1	19
8	−14	4	13	9
4	5	2	18	13
11	8	13	−18	12
8	1	11	13	15

9	13	11	4	13
18	−10	18	1	19
8	−14	4	13	9
4	5	2	18	13
11	8	13	−18	12
8	1	11	13	15

Fig. 1. Example of a small instance of BoPP to the left, and a solution placing two rectangles to the right.

ing the area covered by a facility as a rectangle, and the board positions as demand to be covered with specified revenues. Church & ReVelle (1974) considered the maximal covering location problem, where a set of users, each with a given demand, is given together with a set of facilities. The goal is to select facilities in a way that the demand covered is maximized. Other facility location problems consider the weighted total distance from the allocated facilities as an objective to minimize, while in applications such as emergency services one instead would make sure that the demand point farthest away from a facility is as close as possible (Drezner, 1987).

Facility location models are divided into continuous and discrete models. Çakir & Wesolowsky (2011) considered a continuous facility location problem where facilities are represented as rectangles. The goal is to determine the location and shape of the rectangles to minimize the cost of expropriated points (Berman, Drezner, & Wesolowsky, 2003). Blanquero, Carrizosa, Tóth, & Nogales-Gómez (2016) considered a similar problem, that aims at locating facilities so as to maximize the market share. While that problem is deeply studied in the field of continuous location, they study the problem on networks, formulated as a mixed integer nonlinear programming problem that can be solved by a branch-and-bound algorithm. Some researchers have considered the case where a partial covering of customer demands is possible (Berman & Krass, 2002), but in BoPP the board positions are either completely covered or not covered.

Dupont, Lauras, & Yugma (2013) investigated the generalized covering location problem (GCL), which is perhaps the facility location problem that most closely resembles BoPP. In GCL, each facility has a constant coverage radius. A cost is taken into account if a facility is built, and the cost depends on where the facility is located. If some demand point is not covered, it provides a penalty. One difference to BoPP is that, in GCL, a demand point provides different contributions if covered multiple times.

A second direction of related research is in the form of geometric covering problems. These are often studied from a more theoretical perspective, without necessarily focusing on concrete practical applications. Within the main branch of this field, a rectilinear polygon should be covered by certain items, and usually the size of the cover (the number of covering items) is minimized. The sides of the objects are parallel (or perpendicular) to the axes in the two-dimensional space. A covering item cannot extend beyond the rectilinear object, and each cell must be covered (by exactly one or several items). This means, in the language of BoPP, that the value of the gain function is $-\infty$ outside of the rectilinear object, and it is uniformly 1 inside.

Possibly the first mention of such a problem is due to an unpublished manuscript of Masek (1978). Garey & Johnson (1979) referred to Masek's problem (page 232, SR25) as rectilinear picture compression and stated that it is NP-complete. The considered problem is the following: Given an $n \times n$ matrix of 0's and 1's and a positive integer indicating the maximum number of rectangles that can be used, the question is whether there is a collection of rectangles that cover precisely those entries in the matrix that are

1's (Garey & Johnson, 1979). The rectangles are allowed to intersect each other, so overlap is allowed.

Culberson & Reckhow (1988) considered whether a polygon can be covered by a set of (smaller) polygons. As a subproblem they considered the covering of an orthogonal polygon (where all edges are orthogonally aligned) by a set of rectangles. Here, the covering items cannot be placed outside of the covered polygon, but are allowed to overlap inside it. A polygon is simple and holeless if the boundary is a simple cycle.

In this framework we can suppose that the coordinates of vertices of a rectilinear polygon are all integers, so the rectilinear object can be assumed to lay in the 2-dimensional grid, in other words it can be divided into small 1×1 cells and any cell corresponds to a certain row and column in a given matrix. If there may be holes in the rectilinear polygon, then already Masek proved the NP-completeness of the problem. If the polygon is holeless but not convex from any direction, Culberson & Reckhow (1988) proved NP-completeness. Aupperle, Conn, Keil, & O'Rourke (1988) distinguished between two cases. If there is no hole within the rectilinear polygon, they showed that covering the object by a minimum number of squares can be solved in polynomial time. If, however, there can be holes within the polygon, the problem is NP-hard. The authors also provided some non-trivial applications. Bereg et al. (2012) considered a problem where there are both good points and bad points in the plane. All the good points must be covered while no bad point is covered, using the minimum number of rectangles. This problem is also NP-hard, even when the rectangles are substituted by squares.

Let us consider the minimal covering of a rectilinear object and its relationship to BoPP. We have the following properties:

- Any gain value outside of the polygon is $-\infty$.
- Any gain value inside the polygon is 1.
- The items (that will be used for covering) are not given in advance. It means that we can create any kind and any number of items, as far as they fit into the polygon; and we can choose arbitrarily from the generated set of items freely.
- The items can overlap.
- The cost of any item is the same. We can choose an appropriate small $\varepsilon > 0$ so that the cost of any item is exactly ε . If the rectilinear object can be encased into an $m \times n$ box, then e.g. $\varepsilon = 1/(mn + 1)$ is a good choice. In the worst case mn items are chosen for the cover, their total cost is less than 1, so covering the whole object by a minimum number of items means a cover by minimum cost.

From the above, we can see the differences between the minimal covering of a rectilinear polygon and the BoPP problem: For the latter, the gain values are not constrained, the items are given in advance, and each item has its own cost. If overlap is not allowed (but the whole board must be covered), the cover is called a partition (or decomposition). Culberson & Reckhow (1988) established that the partitioning of orthogonal polygons into rectangles can be done in polynomial time, even when holes are allowed.

Besides the literature on facility location problems and geometric covering problems, there are several other contributions to somewhat related problems. Given n demand points in the plane, the p -center problem is to find p supply points (anywhere in the plane) so as to minimize the maximum distance from a demand point to its respective nearest supply point, while the p -median problem is to minimize the sum of distances from demand points to their respective nearest supply points. Both problems are NP-hard, no matter whether Euclidean or rectilinear metrics are applied (Megiddo & Supowit, 1984). Mukherjee & Chakraborty (2002) considered a set of given points in the plane that must be covered by a given number of non-intersecting rectangles. The size

Table 1
Overview of existing models and how they relate to the board packing problem.

Authors/model	Reference	Feature										
		a	b	c	d	e	f	g	h	i	j	k
Church and ReVelle	Church & ReVelle (1974)	X			X		X	X	X	X	X	X
Drezner	Drezner (1987)	X			X	X	X	X	X	X	X	X
Çakir and Wesolowsky	Çakir & Weselowsky (2011)	X			X	X	X		X		X	X
Blanquero et al.	Blanquero et al. (2016)		X		X	X	X		X	X	X	
Berman and Krass	Berman & Krass (2002)		X		X	X	X		X	X	X	
Dupont et al.	Dupont et al. (2013)			X			X			X		X
Masek	Masek (1978)			X		X	X	X	X			X
Culberson and Reckhow	Culberson & Reckhow (1988)			X		X	X	X	X			X
Aupperle et al.	Aupperle et al. (1988)			X		X	X	X	X			X
Bereg et al.	Bereg et al. (2012)			X		X	X	X	X			X
Mukherjee and Chakraborty	Mukherjee & Chakraborty (2002)	X			X	X	X	X				X
Leung et al.	Leung et al. (1990)	X			X			X	X		X	X
Mahapatra et al.	Mahapatra et al. (2007)	X			X			X	X	X	X	X
BoPP				X			X					X

of each rectangle can be set freely, and the objective is to minimize the area of the rectangles used to cover the points.

Leung, Tam, Wong, Young, & Chin (1990) considered a problem where, given some small squares and one big square, the question is whether the small squares can be packed (without overlapping) into the big square. They proved that this problem is strongly NP-hard. Mahapatra, Goswami, & Das (2007) dealt with a problem where the goal is to cover several demand points in the plane, the covering items are a fixed number of disjoint unit squares, and the objective is to maximize the number of covered demand points. The problem is polynomially solvable. A general typology for cutting and packing problems without overlap was given by Wäscher, Haußner, & Schumann (2007). In bin packing problems, the goal is typically to minimize the number of surrounding containers used (Martinez-Sykora, Alvarez-Valdes, Bennell, Ruiz, & Tamarit, 2017), rather than to maximize the profit of items placed within a given container.

Since there are many similar models and potential application areas in the literature, we would like to give a better visualization to highlight the similarities and differences between the BoPP and other models, to highlight also the novelty of the new model. We do this in the following way. We can realize that certain combinations of collections of special features define different types of problems. The features include the following:

- allocating in metrical space
- allocating in graph
- allocating in grid
- continuous model (vs. discrete model)
- total coverage (vs. partial coverage)
- allowing overlap (vs. no overlap)
- constant gain/cost across all covered points (vs. varying gains/costs)
- constant cost of items (vs. varying costs of items)
- items/facilities of the same size (vs. different sized items)
- all items/facilities are selected (vs. subset of facilities selected)
- points either completely covered or not covered (vs. partial coverage possible)

Table 1 shows the most relevant models from the literature and how they are related to the features above. If the listed feature holds we indicate this with an X in the appropriate cell, and otherwise, if feature does not hold (meaning the alternative as indicated in parentheses above holds) the corresponding cell is empty. For example, regarding feature d, if the model is continuous we put an X, otherwise if the model is discrete, we do not put an X.

We emphasize that only some of the main features are listed in the table, and there are more differences among the models. As one can see, the most similar to the BoPP is the model of Dupont et al. (2013). However, in their model each covering item has the same size (while in BoPP the items have different sizes), and if a demand point is covered multiple times then this provides different contributions to the objective (while in BoPP we distinguish only between points that are covered and points that are not covered). Moreover, our solution methods are different, and we consider bigger instances. All other models are quite different from the BoPP.

This paper now continues by presenting proofs that the problem is NP-hard in Section 2. Section 3 provides a mathematical programming model for the problem. Then, Section 4 describes an evolutionary algorithm that can provide heuristic solutions for large test instances. A computational study is presented in Section 5 to evaluate whether certain types of instances can be solved to optimality using commercial software. Finally, Section 6 concludes the paper.

2. NP-completeness of Board Packing

In this section we prove that various subproblems of BoPP are NP-hard. The proofs do not require the MILP model introduced in the previous section, they are technical in another way. Formally we consider the following problem, where h and w are positive integers and S is a given set of integers. The latter will put a constraint on the gain values $g_{i,j}$, while the allowed purchaseable rectangles $r \in R$ will be restricted to a single size $h \times w$.

BOARDPACKING($h, w; S$) (BP($h, w; S$))

Input: An $M \times N$ rectangular board (matrix) with all entries from S , and an integer g .

Question: Does there exist a packing with at most $M * N$ rectangles of size $h \times w$, each having a cost of 1, with total profit at least g ?

In fact we shall consider a rather restricted set S as follows:

BINARYBOARDPACKING(h, w) (BBP(h, w)) means BP($h, w; S$) with $S = \{0, 1\}$,

ALMOSTBINARYBOARDPACKING(h, w) (ABP(h, w)) means BP($h, w; S$) with $S = \{-1, 0, +1\}$.

We state our result as a three-part theorem. As one can see, its parts (i) and (ii) are independent, while both of them are implied by (iii). The reason for this way of presentation is that the general ideas of the structural reduction are simpler to describe for (i), from which the proofs for (ii) and (iii) are developed by

local modifications. In addition, (iii) requires a more complicated generic reduction and topological considerations.

Theorem 1. *The following problems are NP-complete, and their optimization versions are NP-hard:*

- (i) $BBP(3,3)$,
- (ii) $ABP(2,3)$ and $ABP(3,2)$,
- (iii) $BBP(2,2)$.

The proofs are based on reductions from variants of the Boolean Satisfiability problem. Generally speaking, when a generic instance of Satisfiability is considered with p clauses over a set of q variables, the derived instance matrix (board) for BPP or ABP has size $O(p) \times O(q)$, and to construct such an instance requires $O(pq)$ time. Hence the reduction is quadratic in both time and space.

Proof of (i). In this part of the proof, “box” means a 3×3 submatrix. We apply a reduction from the Boolean satisfiability problem 3-SAT. Let $\Phi = c_1 \wedge \dots \wedge c_p$ be any instance of 3-SAT over q variables x_1, \dots, x_q ; i.e., each c_i is the disjunction of exactly three distinct literals, each literal being either x_i or $\neg x_i$ for some $1 \leq i \leq q$. It is also assumed that x_i and $\neg x_i$ do not occur together in any clause. We construct a 0–1 matrix $M(\Phi)$ of size $M \times N$, where $M = O(p)$ and $N = O(q)$. Almost all entries of $M(\Phi)$ will be 0, except for a relatively sparse set of entries 1, as specified below and indicated with \bullet (and sometimes with \otimes or \odot) in the illustrations. In text we shall refer to their positions as *1-cells*. Two 1-cells are *related* if they can be covered with a box; i.e., if the indices of their rows—as well as of their columns—differ by at most 2.

By *trajectory* we mean a cyclic sequence of distinct 1-cells which satisfy the following properties:

- it consists of an even number of 1-cells;
- any two consecutive 1-cells in the cyclic order are related;
- if two 1-cells are not consecutive in the cyclic order, then they are not related.

Along a trajectory we also define the *distance* between any two 1-cells in a natural way. Related 1-cells are said to be at distance 1; the two 1-cells related to a 1-cell are at distance 2; and so on.

Each variable x_i is represented with a trajectory t_i . Those trajectories will express the inclusion relation between clauses and variables. A schematic illustration of a possible arrangement of trajectories for a formula with three clauses and five variables is given in Fig. 2. Figure 3 shows the detailed trajectory for the first of the five variables.

By definition, no t_i is allowed to repeat any 1-cell in the cyclic sequence. On the other hand, two trajectories are allowed to cross. We require that such crosses occur in a specific way, as indicated in Fig. 4, hence offering four possibilities for a box to cover exactly two 1-cells simultaneously from both trajectories.

Apart from this very restricted situation, 1-cells of distinct trajectories are not allowed to be related to each other. In particular, trajectories cannot share a single 1-cell like a touching point of two planar curves. If we connect the consecutive pairs of a trajectory with straight line segments, we obtain a simple polygon which splits the plane to an internal (finite) region and an external (infinite) region. It follows that a trajectory either is disjoint from all the other trajectories or it contains an even number of crossing 1-cells.

On the segments of non-crossing 1-cells we impose the following further requirement:

- on each t_i which meets at least one other t_j , between any two crossing 1-cells there is an odd number of intermediate 1-cells; hence the distance between any two crossing 1-cells is even.

Also recall that, by definition, if t_i is disjoint from all the other trajectories, then it consists of an even number of 1-cells.

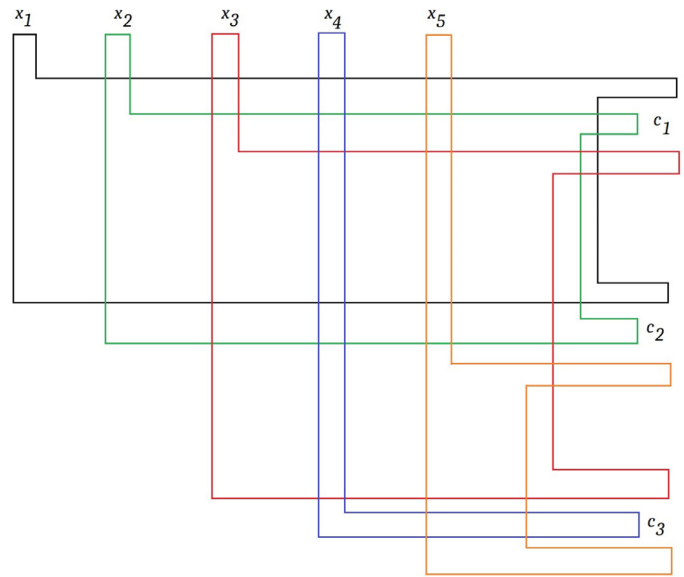


Fig. 2. Schematic arrangement of trajectories for Boolean formula $\Phi = c_1 \wedge c_2 \wedge c_3$ with three clauses $c_1 = x_1 \vee x_2 \vee \neg x_3$, $c_2 = x_1 \vee \neg x_2 \vee x_5$, $c_3 = x_3 \vee x_4 \vee \neg x_5$ over five variables x_1, x_2, x_3, x_4, x_5 .

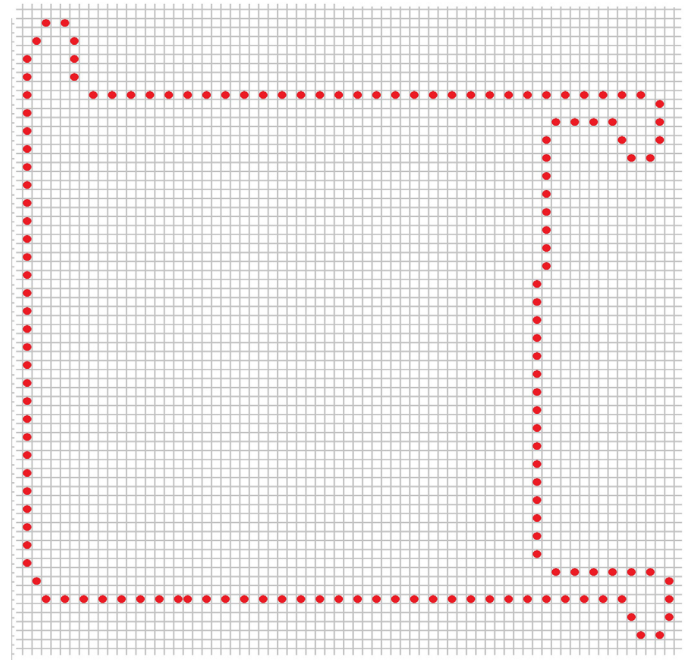


Fig. 3. Detailed trajectory for x_1 in Fig. 2, with red dots corresponding to a 1 in the matrix and other cells corresponding to a 0. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

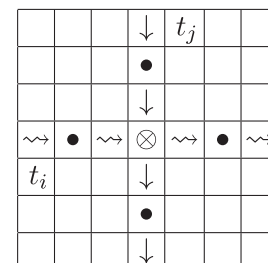


Fig. 4. Crossing 1-cell \otimes of trajectories t_i and t_j .

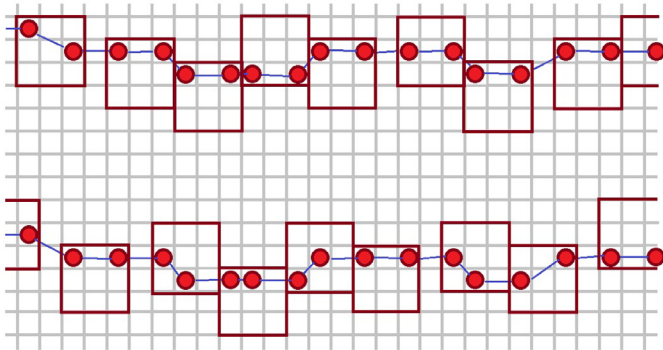


Fig. 5. The two perfect covers are shifted by one 1-cell.

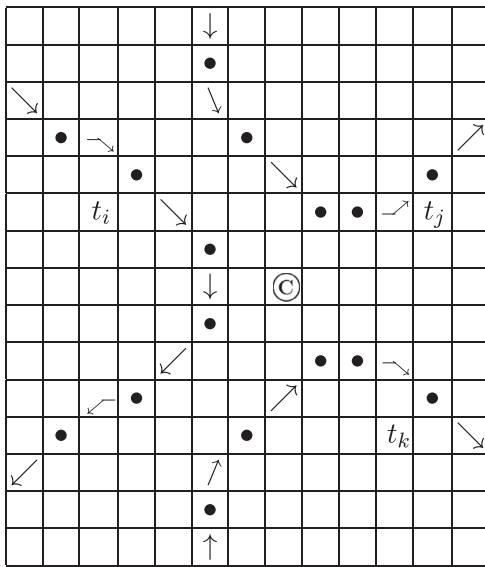


Fig. 6. Example of a clause 1-cell © and its neighborhood, for a clause $c = \ell_i \vee \ell_j \vee \ell_k$ with $\ell_i \in \{x_i, \neg x_i\}$, $\ell_j \in \{x_j, \neg x_j\}$, $\ell_k \in \{x_k, \neg x_k\}$.

As the number of 1-cells in a trajectory is even in either case, each t_i admits precisely two types of “perfect covers” with half that many boxes, as illustrated in Fig. 5. We associate one of them with the truth assignment $x_i = T$ and the other one with $x_i = F$; and call them *true cover* and *false cover*, respectively.

Furthermore, each clause c is represented with a 1-cell, drawn as © in subsequent figures. We arrange exactly three trajectories close to ©. Namely, if an x_i is a literal in c , then there is a consecutive pair of 1-cells in t_i such that a true cover of t_i can cover © together with this pair, but a false cover cannot; and if $\neg x_i$ is a literal in c , then a false cover can do so, but a true cover cannot. All three literals of c are required to satisfy this property. Figure 6 shows that this can be done, indeed, keeping the three trajectories around © unrelated.

Moreover, the set of all crossing 1-cells together with all clause 1-cells must not contain any related pair. This condition implies that every box contains at most three 1-cells.

It is possible to arrange the trajectories within a rectangle proportional to $p \times q$ in a way that respects the requirements on all clause 1-cells. Namely, the condition that defines “related pairs” in the trajectories means that if two cells are at distance d in the cyclic sequence then their row/column indices differ by at most $2d$; but within this bound there is a high degree of flexibility concerning local shape and density, as illustrated in Fig. 7. On the other hand, in order to keep distinct trajectories unrelated except for their crossings, in the schematic structure exhibited in Fig. 2 we

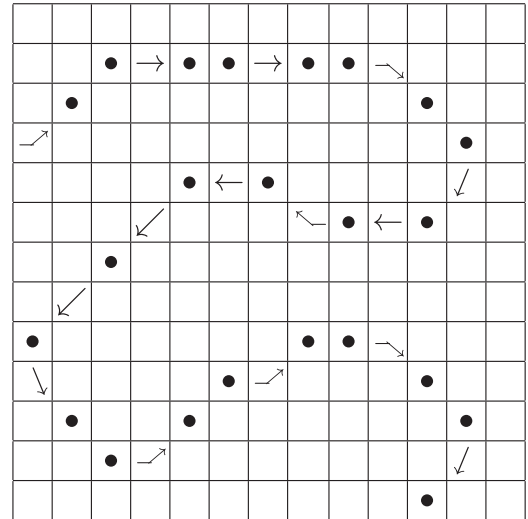


Fig. 7. A possible segment of a trajectory.

do not need large gaps to separate the horizontal/vertical lines, because difference 3 in row/column indices already ensures unrelativeness. These observations imply that for any Φ a matrix of size $O(p) \times O(q)$ will suffice.

Once $M(\Phi)$ has been constructed for Φ , let us introduce the following notation:

- T – the set of all 1-cells in $t_1 \cup \dots \cup t_q$;
- K – the set of all crossing 1-cells, that is the union of the intersections $t_i \cap t_j$ for all $1 \leq i < j \leq q$;
- C – the set of all clause 1-cells.

Moreover, set the input parameters as follows:

- problem input: $M(\Phi)$;
- each box costs 1, hence its profit is the number of 1-cells in it, minus 1;
- ask whether a profit of $|C| + \frac{1}{2}(|T| + |K|)$ is achievable.

We are going to prove that the answer is yes if and only if Φ is satisfiable. Since 3-SAT is an NP-complete problem, the same holds true also for the BOARD PACKING problem.

Suppose first that Φ is satisfiable. We are going to buy $\frac{1}{2}(|T| - |K|)$ boxes and cover all the $|T| + |C|$ 1-cells with them, hence achieving the required profit.

Let $f : \{x_1, \dots, x_q\} \rightarrow \{T, F\}$ (meaning $T = \text{TRUE}$ and $F = \text{FALSE}$) be a satisfying truth assignment of Φ . Each trajectory will be covered with a true or false cover, in accordance with f . Observe that if $t_i \cap t_j \neq \emptyset$, then at each crossing 1-cell for each of the four possibilities of $f(x_i), f(x_j)$ there exists a box usable in the required true/false covers of t_i and t_j that also contains $t_i \cap t_j$. In this way each selected box covers precisely two 1-cells from $T \setminus K$, and none of those 1-cells is covered twice, thus the entire $T \setminus K$ is covered; and as just has been said, properly chosen boxes also cover the entire K without any extra cost. Moreover, since every clause c is satisfied, the corresponding cover of t_i can be taken in the neighborhood of © in such a way that the clause 1-cell representing c is also covered. These facts ensure the profit of $|C| + \frac{1}{2}(|T| + |K|)$.

Suppose now that we have achieved a profit at least $|C| + \frac{1}{2}(|T| + |K|)$, and we have bought k boxes for it. Let R_1, \dots, R_k be those boxes. We can assume without loss of generality that each R_ℓ covers exactly two 1-cells outside of $C \cup K$. It may happen that some 1-cell is contained in more than one R_ℓ ; but its gain $g_{i,j} = 1$ can be obtained only once. To make the counting transparent, one may consider a mapping $h : (i, j) \mapsto \ell$ to express that the gain of

$g_{i,j}$ is counted for R_ℓ (but for no other box covering the corresponding cell). Under this assignment we have $h^{-1}(\ell) \in \{0, 1, 2, 3\}$. In fact if $h^{-1}(\ell) = 0$, then R_ℓ increases the cost and omitting it from the covering we increase the profit.

1. If $h^{-1}(\ell) = 1$, then R_ℓ costs 1 and gets gain 1. In this case we omit R_ℓ from the cover, without changing the profit.
2. If $h^{-1}(\ell) = 2$, then one of the following situations occurs:
 - (a) R_ℓ covers two 1-cells outside of $C \cup K$, both of them are assigned to it, but no cell from $C \cup K$ is assigned to R_ℓ .
 - (b) R_ℓ covers a cell from $C \cup K$, which is assigned to it, but one of the other two 1-cells in R_ℓ is assigned to another $R_{\ell'}$.
3. If $h^{-1}(\ell) = 3$, then R_ℓ covers a cell in $C \cup K$ and two 1-cells outside of $C \cup K$, and all the three are assigned to it.

We can eliminate the situation 2(b) by modifying h . Indeed, if it is $g_{i,j}$ whose 1-cell is covered by R_ℓ but $h : (i, j) \mapsto \ell'$, then we redefine the mapping as $h : (i, j) \mapsto \ell$. After this modification we obtain $h^{-1}(\ell') = 1$ and then $R_{\ell'}$ gets omitted by case 1. At the same time R_ℓ becomes an item of case 3.

After all, we only have the cases 2(a) and 3. Cells in the former and the latter make profit 1 and 2, respectively. The number of the latter is at most $|C| + |K|$, and the total number of boxes is k . Consequently the total profit is at most $k + |C| + |K|$; but at the same time it is at least $|C| + \frac{1}{2}(|T| + |K|)$ by assumption. Hence

$$k \geq \frac{1}{2}(|T| - |K|).$$

On the other hand, the total number of 1-cells is $|C| + |T|$, and we have bought k boxes, hence the profit cannot exceed $|C| + |T| - k$. If this is at least $|C| + \frac{1}{2}(|T| + |K|)$, then

$$k \leq \frac{1}{2}(|T| - |K|)$$

also holds, thus we have equality. Note further that in the cases of 2(a) and 3 each box R_ℓ has its two private 1-cells outside of $C \cup K$. As a consequence, if $k = \frac{1}{2}(|T| - |K|)$ holds and the profit is $|C| + \frac{1}{2}(|T| + |K|)$, then the boxes R_1, \dots, R_k partition $T \setminus K$ and entirely cover K and also C . (If R_ℓ covers a crossing cell, then its other two 1-cells belong to two distinct trajectories, while the boxes disjoint from K contain two 1-cells from the same trajectory.)

We prove that such a cover defines a true or a false cover on each trajectory and in this way corresponds to a truth assignment f on Φ . Once this will be shown, the only possibility to cover C at the same time is that f satisfies Φ .

The claimed property is obvious for any trajectory t_i which is disjoint from all the others, because then there can be only two ways for R_1, \dots, R_k to partition t_i . The situation is different when $t_i \cap t_j \neq \emptyset$, because in principle it might happen that a crossing 1-cell is covered with two antipodal boxes. We show that this is impossible under the restrictions required for the trajectories.

Consider an intermediate segment of t_i with its two neighbor crossing 1-cells, say $w_0, v_1, v_2, \dots, v_{2s-1}, w_{2s}$, where w_0, w_{2s} are crossing 1-cells and $v_1, v_2, \dots, v_{2s-1}$ are non-crossing 1-cells. Assume first that $s \geq 2$. Then, since $T \setminus K$ is partitioned by the covering boxes, either there is a box R_j containing the pair v_1, v_2 or else there is a box R_j containing the pair v_{2s-2}, v_{2s-1} . Assume that the latter holds, the former case being similar. We then consider the continuation $w_{2s}, v_{2s+1}, v_{2s+2}, \dots, v_{2t-1}, w_{2t}$ of t_i , where w_{2t} is the next crossing 1-cell. (It may occur that $w_{2t} = w_0$, but the argument works for this particular situation, too.) Here a box $R_{j'}$ containing w_{2s} cannot contain v_{2s-1} because then this 1-cell of $T \setminus K$ would be covered twice. Thus $R_{j'}$ must contain v_{2s+1} , and then a next box must contain the pair v_{2s+2}, v_{2s+3} , and so on. Also the pair v_{2t-2}, v_{2t-1} is contained in a selected box. Hence, the situation at the end of this second segment is the same as it was at the end of the first segment. In this way the property propagates through the

entire t_i and we have either a true cover or a false cover. The situation is similar if $s = 1$. Then a box covering v_1 has to contain either w_0 or w_2 , causing a transfer to the next segment in the same way.

In conclusion, we have shown that a gain of $|C| + \frac{1}{2}(|T| + |K|)$ is achievable if and only if Φ is satisfiable. This completes the proof of part (i). \square

Proof of (ii). The cases of (2,3) and (3,2) are analogous and can be obtained from each other by reflection on a diagonal (i.e., by row \leftrightarrow column transformation). Thus, it is sufficient to restrict our discussion to boxes that have two rows and three columns. Recall that the entries are now taken from the set $\{-1, 0, +1\}$.

The proof for this case is similar to the previous one; we apply a reduction from the 3-SAT problem. However, in the neighborhood of a clause 1-cell there is smaller space available, which makes it impossible to arrange three entirely unrelated trajectories, unless we relax some restrictions. To overcome this difficulty we use cells with the value -1 , as indicated with N (abbreviating the word “negative”) in Fig. 8, allowing that some particular boxes may contain three non-crossing 1-cells of a trajectory. (Actually this -1 is strictly necessary only for t_k , since t_j might be modified, placing its first two 1-cells in the column next to \odot on its left.)

Note that true covers and false covers can be ensured properly according to the clauses, because 2×3 boxes still offer flexibility for arranging a trajectory. This fact is illustrated in Fig. 9.

Now there exist boxes containing more than two non-crossing 1-cells of the same trajectory, but they necessarily contain a negative cell also. As one can see, the gain of such a box is exactly the same as that of a box not containing the (-1) -cell and the 1-cell above or under it. Hence, concerning profit, a cover using at least one box with more than two non-crossing 1-cells is equivalent to a cover without (-1) -cells such that not all trajectories are completely covered. Due to the computation above, such a cover can never achieve a profit of $|C| + \frac{1}{2}(|T| + |K|)$, thus it cannot be optimal when Φ is satisfiable. We have also seen that for a non-satisfiable formula the profit $|C| + \frac{1}{2}(|T| + |K|)$ cannot be reached either. These facts prove (ii), hence the theorem for 2×3 boxes in boards whose entries are from $\{-1, 0, +1\}$. \square

Proof of (iii). In this last part of the proof “box” means a 2×2 rectangle. Recall that entries from $\{0, 1\}$ will be considered. The case of 2×2 boxes is substantially different from the previous ones, because three mutually unrelated trajectories around a clause 1-cell are not at all possible anymore, no matter what entries the board may contain. For this situation we use a reduction from another variant of satisfiability, namely the problem Max-2-SAT. Its input is a Boolean formula in conjunctive normal form such that every clause has at most two variables, and also an integer k is specified in the input. The question is whether there exists a truth assignment that satisfies at least k clauses.

Garey, Johnson, & Stockmeyer (1976) proved that the Max-2-SAT problem is NP-complete with the value $k = 7p/10$, where p is the number of clauses in the input formula. This fact alone would not be sufficient for the proof of our theorem, we also need to take a closer look at the construction in Garey et al. (1976, Theorem 1.1), which is a reduction from 3-SAT. Furthermore, we shall also need a special geometric arrangement of the trajectories with respect to the clause 1-cells.

Let us start with an instance Φ of 3-SAT, which is now written in the form

$$\Phi = (a_1 \vee b_1 \vee c_1) \wedge \dots \wedge (a_s \vee b_s \vee c_s)$$

where each of a_i, b_i, c_i is a positive or negative literal for $i = 1, \dots, s$. From every clause $C_i = (a_i \vee b_i \vee c_i)$, an expression E_i of ten clauses is created, each clause with at most two literals, using an

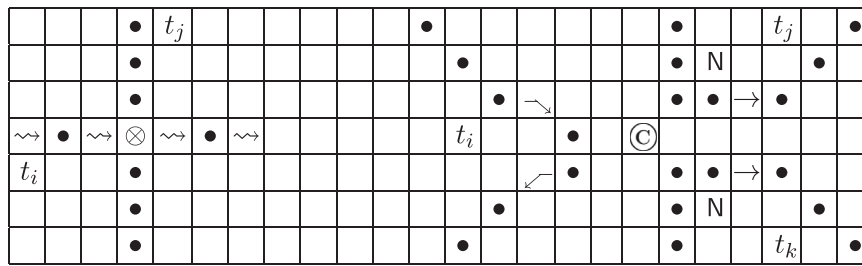


Fig. 8. Crossing 1-cell \otimes and the neighborhood of a clause 1-cell \odot for 2×3 boxes; nonzero cells are $\bullet = 1$ and $N = -1$.

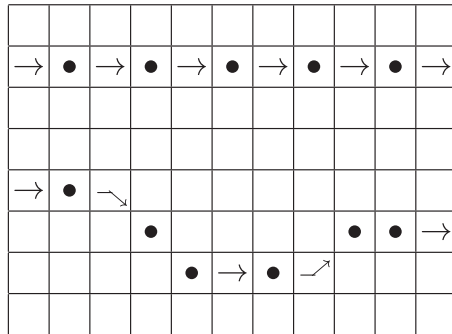


Fig. 9. Horizontal or vertical flexibility of trajectories for 2×3 boxes.

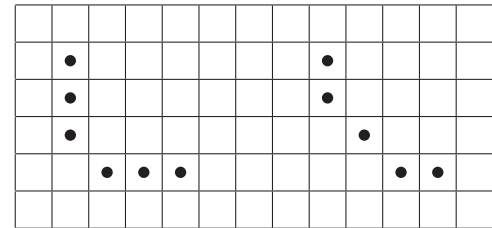


Fig. 10. A parity-switching local transformation.

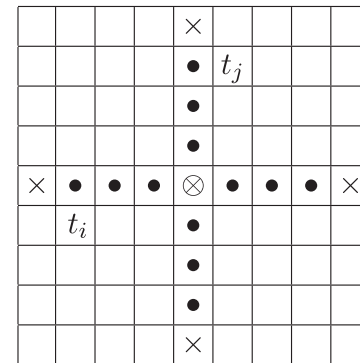


Fig. 11. Crossing 1-cell \otimes of trajectories t_i and t_j , with an indication \times of nearest allowed positions of other crossing 1-cells.

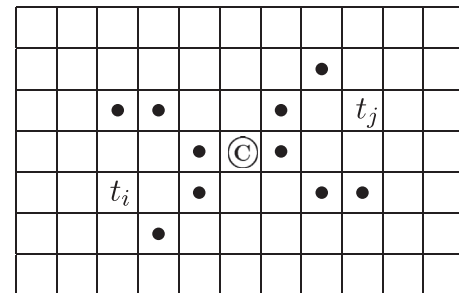


Fig. 12. The neighborhood of a clause 1-cell \odot for 2×2 boxes.

auxiliary variable d_i , too:

$$E_i = (a_i) \wedge (b_i) \wedge (c_i) \wedge (d_i) \wedge (\neg a_i \vee \neg b_i) \\ \wedge (\neg a_i \vee \neg c_i) \wedge (\neg b_i \vee \neg c_i)$$

$$\wedge (a_i \vee \neg d_i) \wedge (b_i \vee \neg d_i) \wedge (c_i \vee \neg d_i).$$

Then the input of Max-2-SAT is defined as the formula

$$\Phi^* = E_1 \wedge \dots \wedge E_s$$

which has $p = 10s$ clauses; and one puts $k = 7s$. A key observation, which is very easy to check, is that in any truth assignment on Φ ,

- if $(a_i \vee b_i \vee c_i)$ is T then there is a proper choice of $d_i \in \{T, F\}$ that makes seven¹ clauses of E_i satisfied simultaneously, but more than seven is impossible independently of the truth values of a_i, b_i, c_i ;
- if $(a_i \vee b_i \vee c_i)$ is F then only six² clauses of E_i can be satisfied simultaneously.

As a consequence, Φ^* admits a truth assignment satisfying $7s$ clauses if and only if the input Φ of 3-SAT is satisfiable.

The basic principles of creating trajectories for variables is the same as in the previous arguments. Each trajectory has to admit a “true cover” and a “false cover”, and it has to approach a clause according to whether setting the variable to T or F satisfies the clause in question. This requirement, as well as the condition that the distance between any two crossing 1-cells is even, requires a way to adjust parities properly. This can be done at any turning part of a trajectory, as shown in Fig. 10. A possible situation of crossing 1-cells is indicated in Fig. 11.

Furthermore, the trajectories in the neighborhood of a clause 1-cell can be arranged as shown in Fig. 12, to ensure that the trajectories are unrelated. Also, if a box covers \odot together with two 1-cells of a trajectory, then shifting it by one unit horizontally (to

the left for t_i or to the right for t_j), we obtain a box which still covers the same two 1-cells of the trajectory but does not contain \odot anymore. As a consequence, without loss of generality we may restrict our attention to packings which cover each clause 1-cell at most once.

We use the notation T, K, C —in the same way as in the proof for the 3×3 case—for the set of 1-cells in the union of all trajectories, crossing 1-cells, and clause 1-cells, respectively.

For technical reasons we impose a lower bound (not too large, say 3 will already do) on the number of 1-cells between any two crossing 1-cells, on every trajectory. Similarly we assume that the two 1-cells on a trajectory that can be covered together with a

¹ Put $d_i = F$ unless $a_i = b_i = c_i = T$.

² Six can always be attained by setting $d_i = F$.

clause 1-cell by a box are not very near to the next crossing 1-cell. These conditions are easily fulfilled.

Contrary to the case of 3-SAT, in the partial satisfaction of Max-2-SAT it does not hold automatically that every optimal solution defines a true cover or a false cover on every trajectory. To ensure that the property remains valid for Max-2-SAT as well, we observe that each of a_i, b_i, c_i occurs in E_i exactly twice as a positive literal and twice as a negative literal. As a consequence, every original variable of Φ occurs in Φ^* an even number of times, in half of them positive and in half of them negative. Based on this fact, we require that each trajectory—except those for the variables d_i , which occur once as positive and three times as negative—alternates between clauses in which its corresponding variable is positive and in which it is negative. Note that before this, we did not put any condition on the order in which a trajectory visits the clause 1-cells of its variable, hence we are free to decide about this order.

The simplest way of ensuring positive-negative alternation is to relax the definition of trajectories, allowing that a trajectory may cross itself. The reader can check that the proof below works under this relaxation, too. On the other hand it can be proved that self-crossing is avoidable by a careful design of trajectories with respect to the order of clause 1-cells. In order to avoid unnecessary technicalities, we demonstrate this fact with an example only—although a rigorous proof can also be given—because crossings can be allowed in the proof of the theorem.

For instance, if variable x_1 occurs in the two clauses $c_1 = (x_1 \vee x_2 \vee x_4)$ and $c_2 = (\neg x_1 \vee x_3 \vee x_5)$, then among the 20 clauses of $E_1 \wedge E_2$ there are four containing x_1 as positive and four as negative:

$$x_1 \quad \neg x_1 \vee \neg x_2 \quad \neg x_1 \vee \neg x_4 \quad x_1 \vee \neg d_1 ; \quad \neg x_1 \quad x_1 \vee \neg x_3 \\ x_1 \vee \neg x_5 \quad \neg x_1 \vee \neg d_2 .$$

The trajectory t_1 is not allowed to approach the corresponding clause vertices in this order, because it is not alternating. Among the $8!$ possible orders only $2 \cdot (4!)^2$ alternate. Once the positions of clause cells are decided, many of the $2 \cdot (4!)^2$ possible orders cannot be realized with a trajectory which does not cross itself, due to topological reasons. Perhaps the simplest way to overcome this difficulty is to relax the definition of trajectories, by allowing that a trajectory crosses itself. Keeping the condition that any two crossing 1-cells are separated by an odd number of non-crossing 1-cells, the argument below works for the proof of (iii). On the other hand, no matter in which order the clause 1-cells are placed, it can be rigorously proved that they admit a non-self-crossing trajectory that approaches those clause 1-cells in an alternating order as regards the sign of the corresponding variable. Instead of a general formal proof here we only describe how this can be done in the current example. Starting from the order above, we select two neighbor clauses in which x_1 occurs with opposite signs, e.g. $x_1 \vee \neg d_1$ and $\neg x_1$, and put them at the end of the order to be constructed. There remain the six clauses

$$x_1 \quad \neg x_1 \vee \neg x_2 \quad \neg x_1 \vee \neg x_4 \quad x_1 \vee \neg x_3 \quad x_1 \vee \neg x_5 \quad \neg x_1 \vee \neg d_2 .$$

This subsequence also contains consecutive opposite occurrences of x_1 , e.g. $\neg x_1 \vee \neg x_4$ and $x_1 \vee \neg x_3$. We put these before the previously found pair, paying attention to keep alternation for x_1 . Analogously, removing the latter two clauses, in the remaining subsequence x_1 occurs with opposite signs in the two consecutive clauses $\neg x_1 \vee \neg x_2$ and $x_1 \vee \neg x_5$. These two clauses will precede the previously found four others in the sequence under construction. Finally x_1 and $\neg x_1 \vee \neg d_2$ remain, which will be placed at the beginning. After all, the following sequence is obtained:

$$x_1, \neg x_1 \vee \neg d_2 ; \quad x_1 \vee \neg x_5 \quad \neg x_1 \vee \neg x_2 ; \quad x_1 \vee \neg x_3 \quad \neg x_1 \vee \neg x_4 ; \\ x_1 \vee \neg d_1 \quad \neg x_1 .$$

Figure 13 shows that there exists a trajectory without self-crossing, that visits the clause 1-cells in this order and also respects the original order of the placement of those cells.

Next we investigate the benefit obtained from alternation. Let us consider any packing \mathcal{P} of boxes as an optimal solution of board packing. We may assume without loss of generality that among all optimal packings, \mathcal{P} has a minimum number of boxes. Then the boxes are mutually disjoint in $T \cup C$, and each of them has its two private 1-cells in $T \setminus K$. (At this point we use the assumption that any two K -cells are at distance 3 or more apart, moreover no C -cell is covered twice, and also that C and K are not too close to each other.) If a box $R \in \mathcal{P}$ meets K , it covers two pairs of consecutive 1-cells along the trajectories. (Those two pairs belong to distinct trajectories, unless the trajectory in question crosses itself at that 1-cell of K .) Hence there is a natural way to associate a set \mathcal{P}_2 of pairs to \mathcal{P} , namely the set of those pairs of consecutive 1-cells in the trajectories which are contained in the members of \mathcal{P} . A box meeting K contains two pairs from \mathcal{P}_2 .

Recall that there is a well-defined partition of any trajectory which we termed true cover, and another called false cover. Each means a partition into pairs. This classifies the members of \mathcal{P}_2 as “true pairs” and “false pairs”. It is immediately observed that a sequence of true pairs is separated from a sequence of false pairs by one (or more) 1-cell which does not belong to any $R \in \mathcal{P}$. We may assume without loss of generality that none of those uncovered cells are crossing 1-cells. Indeed, assume that $k \in K$ is an uncovered cell. Inserting a box R' into \mathcal{P} that covers k , and omitting the members of \mathcal{P} which meet R' , we remove one or two boxes of profit 1 each and insert one box of profit 2. Hence, in the former case of one box we see that \mathcal{P} was not optimal, and in the latter case of two removed boxes $|\mathcal{P}|$ was not minimum, contradicting our assumptions.

Suppose that on each trajectory, \mathcal{P}_2 determines either a true cover or a false cover. Then \mathcal{P} corresponds to a truth assignment, say f , of Φ^* . The number of clause 1-cells covered by \mathcal{P} is at most the number of satisfied clauses under f in Φ^* , and equality can be achieved for any f by properly choosing the members of \mathcal{P} . In particular, if the instance Φ of 3-SAT is satisfiable, then Max-2-SAT admits a truth assignment for the variables d_1, \dots, d_s in such a way that the derived packing \mathcal{P} covers $7|C|/10$ clause 1-cells and the entire T is covered at the cost of $\frac{1}{2}(|T| - |K|)$, so that a total profit of $7|C|/10 + \frac{1}{2}(|T| + |K|)$ is achieved. Also conversely, under the assumption that \mathcal{P}_2 determines either a true cover or a false cover on each trajectory, the cost of the packing is equal to $\frac{1}{2}(|T| - |K|)$, therefore a profit of $7|C|/10 + \frac{1}{2}(|T| + |K|)$ is achievable only if Φ is satisfiable.

It remains to prove that all the other kinds of \mathcal{P} —i.e., if there are trajectories on which \mathcal{P} is neither a true cover nor a false cover—make smaller profit. In proving this, we can assume that no trajectory contains two consecutive uncovered 1-cells, because then a box containing those two cells gains 2 and costs only 1, hence the profit can be increased. So, from now on let \mathcal{P} be an optimal packing in which one or more trajectories consist of sequences of true pairs and sequences of false pairs, separated by uncovered 1-cells, while the other trajectories are completely covered.

As an auxiliary tool, let f_0 be a truth assignment that maximizes the satisfied clauses of Φ^* . We interpret \mathcal{P} as a modification of f_0 . Say, along a certain number u of segments on a trajectory the pairs are switched from “true” to “false” (or from “false” to “true”, depending on the truth value of f_0 on the corresponding variable). This change yields $2u$ uncovered 1-cells, which means a loss with respect to f_0 . Possibly, this loss is partially compensated by clauses which became satisfied when we switched the truth value of the variable in them.

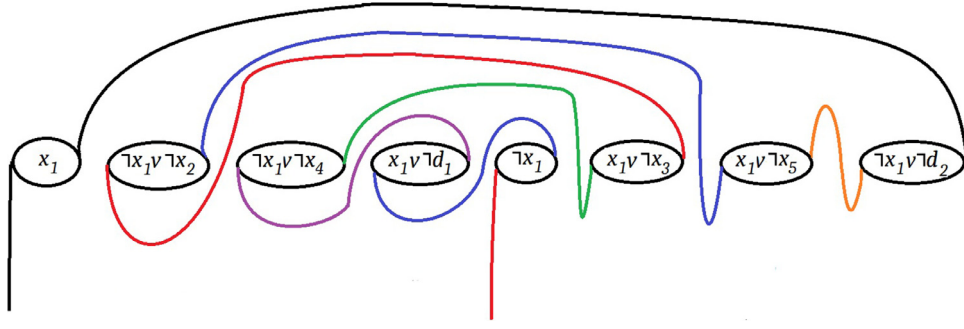


Fig. 13. Trajectory t_1 that respects a pre-defined placement order of clause vertices and does not cross itself, while alternates between positive and negative occurrences of variable x_1 .

Consider first any variable of Φ . By assumption, along its trajectory the positive and negative literals in the corresponding clauses alternate. Hence, if switching on a segment satisfies a number say v of previously unsatisfied clauses Φ^* , then by alternation it also makes $v - 1$ clauses unsatisfied, which were satisfied under f_0 . It implies that with the loss of $2u$ it is possible to make an extra gain no more than u . In this way the total profit decreases by at least u .

Consider now the variables d_i . Each of them occurs only four times, one positive and three negative. Since f_0 maximizes the number of satisfied clauses, at most one of the four clause 1-cells d_i , $(a_i \vee \neg d_i)$, $(b_i \vee \neg d_i)$, $(c_i \vee \neg d_i)$ remains uncovered; e.g., $f_0(d_i) = F$ can be applied. Compared to this, making a switch in the trajectory of d_i would yield at most one extra gain, while it would require two uncovered 1-cells and hence lose gain 2, decreasing the profit.

It follows that the best packings are those originating from truth assignments maximizing the satisfied clauses in Φ^* . This completes the proof of the theorem. \square

3. Mathematical programming model

To formulate a binary integer programming model for the problem, we define A_r as the set of possible coordinates (i, j) for the top left corner of rectangle $r \in R$. Let x_{ijr} be a binary variable taking the value 1 if and only if rectangle r is placed with its top left corner at (i, j) . Let B_{ijr} be the set of positions (u, v) for the top left corner of rectangle r that will result in position (i, j) being covered by that rectangle.

To calculate the revenues from covered board positions, we define an additional binary variable y_{ij} which is equal to 1 if and only if position (i, j) of the board is covered. The model becomes:

$$\max \sum_{i \in M} \sum_{j \in N} g_{ij} y_{ij} - \sum_{r \in R} \sum_{(i, j) \in A_r} c_r x_{ijr} \quad (1)$$

$$\sum_{(i, j) \in A_r} x_{ijr} \leq 1, \quad r \in R \quad (2)$$

$$y_{ij} \leq \sum_{r \in R} \sum_{(u, v) \in B_{ijr}} x_{uvr}, \quad i \in M, j \in N : g_{ij} > 0 \quad (3)$$

$$|R| y_{ij} \geq \sum_{r \in R} \sum_{(u, v) \in B_{ijr}} x_{uvr}, \quad i \in M, j \in N : g_{ij} < 0 \quad (4)$$

$$x_{ijr} \in \{0, 1\}, \quad r \in R, (i, j) \in A_r \quad (5)$$

$$y_{ij} \in \{0, 1\}, \quad i \in M, j \in N \quad (6)$$

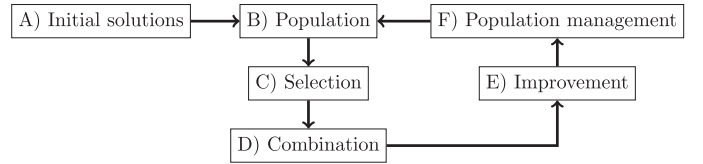


Fig. 14. An overview of the genetic search developed for the BoPP.

The objective function (1) now directly sums up the value of each covered position. Constraints (2) and (5) make sure that each rectangle is only used (at most) once. Constraints (3) and (6) make sure that y_{ij} can only take the value 1 if the position is covered by at least one rectangle when g_{ij} is positive, while constraints (4) and (6) make sure that y_{ij} must take the value 1 if the position is covered by any rectangle and g_{ij} is negative. Constraints (3) and (4) constitute a total of $|M||N|$ restrictions. Although both types of constraints must hold for all board positions (i, j) , the maximization of the objective function assures that half of these automatically hold for any optimal solution to the mathematical program.

4. Evolutionary algorithm

An evolutionary algorithm has been implemented to provide heuristic solutions, being able to handle instances that are too large to be solved to optimality using exact methods. The heuristic keeps a set of solutions in memory, referred to as a population. New solutions are generated by combining pairs of existing solutions. Local improvements are performed on the new solutions before they are added to the population. When the population reaches a certain size, it is trimmed down to a target size by selecting the best solutions from the full population. If the search has trimmed down the population a certain number of times without finding any new best solution in-between, the search is restarted from scratch. Figure 14 illustrates the components of the search, which are described in more detail below.

In the evolutionary algorithm, a solution to the BoPP is represented by storing, for each rectangle $r \in R$, the coordinates (i, j) of the top-left corner of the rectangle's position and a Boolean flag to indicate whether or not the rectangle is purchased. The first step of the heuristic is to generate a set of initial solutions. There are two mechanisms used to achieve this, by having some solutions being randomly generated and some solutions being generated by a simple construction heuristic. To randomly generate a solution, each rectangle is selected as being purchased with a probability of 50%. The coordinates of each rectangle are drawn from uniform distributions over the possible values for the top row and the left-most column of the rectangle. A parameter `pop_init_rand` is used to control the number of random solutions initially generated.

Random solutions are likely to be of poor quality. To increase the likelihood that the initial population contains at least a few decent solutions, a construction heuristic is also used. The idea here is to consider the rectangles in sequence, considering first the rectangles that are cheapest compared to their size. For each rectangle, all possible locations are considered and, if profitable, the rectangle is purchased and inserted into its most profitable position, conditional on the placements of any previously considered rectangles. If it is not profitable to purchase the rectangle, it is left as being unused. By introducing the possibility of changing where in the sorted list of rectangles the insertions should start, we can construct several alternative solutions. The construction heuristic can be summarized as follows:

1. Given: an instance of the BoPP, and a value `con_shift` to allow constructing several different solutions.
2. Sort the rectangles in increasing order of $c_r/|A_r|$.
3. Restructure the sorted list by taking `con_shift` solutions from the beginning of the list and moving them to the end of the list.
4. Follow the sequence of rectangles given in the updated list, and insert each of them in their best possible position of the board, or leave them unused if the insertion into the best position does not lead to an increased objective function value.

The construction heuristic is run for values of `con_shift` from 0 to `pop_init_con - 1`, yielding in total `pop_init_con` constructed solutions and `pop_init` randomly generated solutions. If this total number is larger than the size of the initial population, `pop_min`, the initial population is determined by taking the initial solutions with the highest objective function values.

The main iteration of the evolutionary algorithm consists of steps C, D, and E from Fig. 14. It starts with the selection of two solutions from the current population. This selection is done by performing two binary tournaments. Each binary tournament is executed by drawing two solutions from the population at random and then declaring the best of these solutions as the winner. The two winners, called *P1* and *P2*, are then combined to create two new solutions, called *C1* and *C2*.

The combination method is a form of uniform crossover, where each rectangle is considered independently. Recall that in a solution, each rectangle is described by three values: the coordinates (i, j) of the upper left corner, and a Boolean value representing whether or not the rectangle is purchased. In the combination method, each rectangle is considered in turn. With a 50% probability, the placement information about rectangle *r* in solution *P1* is assigned to the new solution *C1* and the placement information in solution *P2* is assigned to *C2*. Otherwise, the opposite assignment is made with the rectangle placement in solution *P2* being assigned to *C1* and the placement in *P1* being assigned to *C2*.

Each new solution created, including those randomly generated for the initial population, is subjected to local improvements (step E). The local improvement is used instead of a mutation operator, as is common in memetic algorithms (Blum & Roli, 2003). The local improvement starts by considering each rectangle in turn. For a given rectangle, the following modifications are examined, subject to being feasible:

1. If the rectangle is not purchased, try to purchase it and place it at its current position.
2. If the rectangle is currently purchased, try to move it one step up, one step down, one step to the right, and one step to the left.
3. If the rectangle is currently purchased, try to reverse this decision.

When examining the different local improvements for each rectangle, a first-improvement strategy is used: thus if moving a

rectangle one step up improves the solution, no further moves are examined. If no improvements are found for any rectangle after considering each of the above-mentioned changes, one rectangle is selected at random and all possible placements for that rectangle are evaluated. The rectangle is then inserted at its best location, or is not used if the best location is unprofitable.

At the end of each iteration, the population is updated in step F. First, the two new solutions *C1* and *C2* are added to the population, if admissible. We would like to make sure that the population does not contain any duplicated solutions, while not spending too much computational effort to make sure that all solutions are unique. Therefore, we let new solutions be admissible only if they do not have the same objective function value as another solution already included in the population (Prins, 2004).

The population is associated to a minimum size, `pop_min`, and a maximum size, `pop_max`. Whenever the current population size exceeds `pop_max` solutions, or if `pop_max_generated` new solutions have been generated since the population was last reduced, it is trimmed down to `pop_min` solutions. The process of reducing the population down to `pop_min` solutions is done by considering both the quality of the solutions and their diversity, which effectively reduces the risk of a prematurely converging population (Vidal, Crainic, Gendreau, & Prins, 2014).

We first calculate the distance between each pair of solutions in the current population. This distance is measured by considering the placements of each rectangle in the two solutions being compared. For a given rectangle *r*, if both solutions are not using the rectangle, the distance is $d_r = 0$. If one solution is using the rectangle and the other solution is not, the distance is $d_r = 2$. If both solutions use the rectangle, and it is placed with its upper-left coordinate at (i_1, j_1) and (i_2, j_2) , respectively, the distance is defined as $d_r = |i_1 - i_2|/|M| + |j_1 - j_2|/|N|$. The total distance for a pair of solutions is obtained by summing the distances of the individual rectangles.

Next, all solutions in the population are sorted according to their objective function values and assigned a quality-based rank, f^{QB} , with 1 being the best rank and $|P|$ being the worst rank, where *P* is the set of solutions in the current population. All solutions are also sorted according to their contribution to diversity, by considering the average distance to the `pop_closest` most similar solutions in the current population. This leads to a diversity-based rank, f^{DB} , with the same range as f^{QB} .

Following Vidal (2022), we next calculate an overall score for each solution as $f^S = f^{QB} + (1 - \text{pop_elite}/|P|)f^{DB}$, where `pop_elite` is the number of solutions that should survive in the population based on their solution quality alone. At this point, the solution with the highest value of f^S in the population is eliminated, and the process is repeated until the population size has reached `pop_min`.

If at any point the size of the population has been reduced a number of times `pop_max_reductions` without being able to improve the best-found solution, the entire search is restarted from scratch. The heuristic is halted after reaching a total running time of `time_limit` seconds.

In the following, when running the heuristic, we use the following parameter values: `pop_init` = 190, `pop_con` = 10, `pop_min` = 100, `pop_max` = 200, `pop_max_generated` = 500, `pop_max_reductions` = 25, `pop_elite` = 4, `pop_closest` = 5, and `time_limit` = 60.

5. Computational experiments

While the studied problem is NP-hard, we do not know how difficult it is to find optimal solutions in practice. Therefore, we aim to evaluate the ability of a commercial mixed-integer programming solver to identify optimal solutions and obtain a proof

of optimality. Furthermore, we want to study the proposed evolutionary algorithm and its ability to find near-optimal heuristic solutions for instances with known optimal solutions as well as its ability to find good feasible solutions when the commercial solver cannot be applied successfully. Note that our goal is not to show that one of the solution methods is better than the other. Rather, we believe that there are certain characteristics of instances that allow each of the solution methods to exhibit a superior performance, and we wish to gain insights regarding these characteristics and their effects on the overall performance.

In the computational experiments, we use CPLEX (version number: 20.1) as the commercial solver, with a running time limit of 600 seconds. The evolutionary algorithm has a time limit of 60 seconds. Both methods are executed on a desktop computer with Intel Core i3-4150 (Dual-Core) CPU at 3.50 gigahertz and 8.00 gigabyte of RAM, the operating system is Windows 10 Pro 21H1.

In Section 5.1 we consider the effect of increasing the board size, while keeping other aspects of the problem fixed. Section 5.2 analyses the effect of the number of rectangles, while Section 5.3 analyses the effect of the variety of different rectangles. Next, Section 5.4 considers the effect of the topography of the board, that is, how the profits from covering board positions are related to each other. Section 5.5 presents some artificial instances based on covering a given landmass by satellite images. Finally, Section 5.6 investigates the performance of the solution methods on sets of instances that are constructed to be very similar to each other. All the 142 instances used in the computational experiments can be obtained from <https://home.himolde.no/hvattum/benchmarks/>.

5.1. The effect of the board size or resolution

In this section we want to evaluate how the size of the board influences the difficulty of solving the problem, while keeping other aspects of the problem fixed. To this end we start with a board of size $|M| \times |N|$, with $|R|$ rectangles of size $h_r \times w_r$ and cost c_r . Then we introduce a scaling factor $p = 1, 2, \dots$ and generate instances of size $p|M| \times p|N|$, with $|R|$ rectangles of size $ph_r \times pw_r$ and cost p^2c_r . If the original board position (i, j) has a value g_{ij} , then the scaled board has the value g_{ij} for positions $((i-1)p+i', (j-1)p+j')$ with $i' = 1, \dots, p$ and $j' = 1, \dots, p$.

A base instance is created first, with $|M| = 6$, $|N| = 8$, and $|R| = 12$. The gain values are randomly chosen from $\{1, 2, \dots, 20\}$, providing the following board:

$$G = \begin{bmatrix} 9 & 15 & 11 & 4 & 13 & 11 & 12 & 12 \\ 18 & 10 & 18 & 1 & 19 & 13 & 10 & 9 \\ 8 & 14 & 4 & 13 & 9 & 14 & 3 & 10 \\ 4 & 5 & 2 & 18 & 13 & 2 & 2 & 9 \\ 11 & 8 & 13 & 8 & 12 & 19 & 18 & 20 \\ 8 & 1 & 11 & 13 & 15 & 6 & 4 & 2 \end{bmatrix}$$

The heights and widths of the rectangles are also chosen at random. Each height is drawn uniformly at random from $\{1, 2, \dots, |M|/2\}$, and each width from $\{1, 2, \dots, |N|/2\}$. We generated the costs of the rectangles by first calculating the average gain that a rectangle can cover, i.e. $h_r \cdot w_r \cdot g_{\max}/2$, where the maximum possible gain value is $g_{\max} = 20$. Then, this value is multiplied by a random number drawn from $[\frac{1}{2}, 1]$ and the result rounded to the nearest integer. This process resulted in the following values for the 12 rectangles:

r	1	2	3	4	5	6	7	8	9	10	11	12
h_r	1	3	3	1	2	3	2	1	3	2	3	3
w_r	2	2	3	4	3	2	3	4	1	1	4	3
c_r	18	33	72	27	58	43	31	28	18	14	65	54

Table 2

Varying the size of the board with total time (TT) and time to best (TB) reported in seconds.

p	UB	CPLEX		Heuristic	
		Gap [%]	TT	Gap [%]	TB
1	224	0.00	0.0	0.00	0.6
2	896	0.00	0.2	0.00	4.4
3	2016	0.00	0.5	0.00	5.1
4	3584	0.00	1.2	0.00	30.4
5	5600	0.00	3.1	0.00	5.0
6	8064	0.00	5.8	0.00	12.9
7	10,976	0.00	9.3	0.00	7.3
8	14,336	0.00	17.5	0.00	13.5
9	18,144	0.00	26.7	0.00	13.1
10	22,400	0.00	30.4	0.00	1.9
11	27,104	0.00	46.3	0.00	9.5
12	32,256	0.00	81.3	0.00	14.9
13	37,856	0.00	111.5	0.00	32.4
14	43,904	0.00	133.4	0.00	46.0
15	50,400	0.00	244.6	0.00	6.9
16	57,344	Out of memory		0.00	3.1
17	64,736	Out of memory		0.00	17.6
18	72,576	Out of memory		0.00	25.0
19	80,864	Out of memory		0.00	59.0
20	89,600	Out of memory		0.45	4.3
21	98,784	Out of memory		0.45	53.5
22	108,416	Out of memory		0.00	25.0
23	118,496	Out of memory		0.45	29.0
24	129,024	Out of memory		1.34	34.8
25	140,000	Out of memory		0.00	13.9

We now test CPLEX and the evolutionary algorithm on instances with $p = 1, 2, \dots, 25$. The smallest of these have $6 \times 8 = 48$ board positions, whereas the largest has $150 \times 200 = 30,000$ positions. For $p \geq 2$, the instances are constructed in a manner that allows us to calculate upper bounds on the optimal objective function value, given that we know the optimal value for $p = 1$. In particular, we solve the instance to optimality for $p = 1$ and obtain an optimal value of 224. Then, for $p \geq 2$, an upper bound for the optimal objective function value is given by $244p^2$.

Table 2 shows the results for these instances. For each instance we report the upper bound (UB) calculated as $244p^2$. For CPLEX we report the gap to this value and the total time used in seconds (TT). For the heuristic we report the gap and the time used to find the best solution in seconds (TB). To calculate gaps, we start from a given upper bound (UB) and a given lower bound (LB), and use the formula $\text{Gap} = (UB - LB)/LB$ and then round the result to one decimal.

The first observation is that CPLEX solves the smaller instances to optimality very quickly, and CPLEX is able to solve all instances to optimality for p up to and including 15. However, the time required by CPLEX increases gradually with p . When going from $p = 15$ to $p = 16$, which increases the number of positions on the board from 10,800 to 12,288, CPLEX hits a wall and consistently runs out of memory when trying to solve the problem. Nevertheless, we consider a board size of 90×128 , corresponding to $p = 15$, to be a reasonably large board, and could say that the ability of CPLEX to find optimal solutions does not depend too much on just the size of the board.

The evolutionary algorithm was capped at a running time of 60 seconds, but still managed to find optimal solutions for all of the smaller instances, up to $p = 19$. With values of p larger than 19, the gap from the best found solution to upper bound is still very small and occasionally zero.

5.2. The effect of the growing number of rectangles

The size of the board influences the number of locations where a rectangle can be placed, and thus the time required to find the optimal solution. Another factor that influences the number of

Table 3Varying the number of rectangles for an instance with $|M| = 60$ and $|N| = 80$.

$ R $	CPLEX			Heuristic		
	UB	LB	TT	LB	TB	#Rec.
5	5349	5349	14.4	5349	0.2	5
10	9287	9287	53.5	9255	1.5	10
20	17,271	14,984	600.0	16,623	2.5	16
30	50,690	0	600.0	20,780	1.9	13
40	50,690	0	600.0	20,898	12.3	14
50	50,690	0	600.0	23,502	5.4	14
60	50,690	0	600.0	23,441	4.0	15
80	50,690	0	600.0	24,194	7.5	12
100	50,690	0	600.0	24,110	2.5	21
200	Out of memory			24,351	5.0	18
500	Out of memory			24,951	5.6	24
1000	Out of memory			24,662	5.1	15

decisions that must be made is the number of rectangles, $|R|$. In this section we consider a board of size $|M| \times |N|$, with $|R|$ rectangles of random size, with $h_r \in \{1, \dots, |M|/2\}$ and $w_r \in \{1, \dots, |N|/2\}$.

We have chosen $|M| = 60$ and $|N| = 80$, which corresponds to $p = 10$ in the previous test, so the board is too big to visualize here. The gain values are chosen randomly between 1 and 20. The cost of each rectangle is determined in the following way: We consider a random position for the rectangle, and sum up the gain values of the cells that are covered by the rectangle. Then this sum is multiplied by a random number, chosen from $[\frac{1}{2}, 1]$.

We create instances with $|R| = 5, 10, 20, 30, 40, 50, 60, 80, 100, 200, 500$, and 1000 rectangles. For all instances, the board is the same, and the rectangles included in instances with fewer rectangles constitute a subset of the rectangles in instances with more rectangles. In this way we ensure that the optimal objective function value is never smaller when we compare to an instance with more rectangles. This fact is advantageous for estimating the optimum, as the instances in this section are harder to solve for CPLEX.

The results are presented in Table 3. For CPLEX we present the best upper bound found within the time limit (UB), the best feasible solution found (LB), and the total running time (TT) which is limited to at most 600 seconds. For the heuristic we provide the best feasible solution found (LB), the time to find the best solution (TB) and the number of rectangles purchased in the best solution (#Rec.).

In this test, the intuition is that ever more rectangles will be purchased as the number of available rectangles increase. Then, gradually, the number of chosen rectangles will stagnate, although some rectangles will be replaced and the optimum value will grow slowly. Thus, the number of chosen rectangles (the number of rectangles that are used in the optimal solution to cover the board) first grows quickly, then slowly, and then the increment stops.

Only the first two instances could be solved to optimality by CPLEX. For $|R| = 20$, CPLEX cannot determine the optimal solution, but still finds a lower bound (14,984) and an upper bound (17,271) with a gap of 15.54%. For this input the value of the solution provided by the evolutionary algorithm (16,623) is already better, moreover the heuristic uses much less time in total. Then, for the next instances, with $30 \leq |R| \leq 100$, CPLEX only provides the trivial feasible solution where no rectangles are purchased and the trivial upper bound consists of the sum of all gain values. The instances appear too large for CPLEX to handle, and most of the time is spent by preprocessing. For $|R| \geq 200$ the memory runs out during the CPLEX solution process.

On the other hand, the heuristic provides a solid, reliable performance. As we expected based on the construction of the instances, the objective function value is in general growing gradually, but with diminishing marginal improvements by including

Table 4Varying the number of different types of rectangles for an instance with $|R| = 24$.

d	CPLEX			Heuristic	
	UB	LB	TT	LB	TB
1	23,355	23,355	39.9	23,355	14.1
2	23,790	23,790	87.0	23,534	4.1
3	23,616	23,557	600.0	23,543	25.7
4	22,269	22,269	315.7	21,942	2.1
6	18,992	0	600.0	17,932	0.9
8	50,690	0	600.0	18,139	1.4
12	17,144	13,050	600.0	16,329	2.3
24	18,132	0	600.0	17,354	2.3

additional rectangles. For the heuristic, having many rectangles to choose from also makes the instances difficult, and there are some cases where solving an instance with a higher number of rectangles available leads to worse results. Also, the number of rectangles used does not increase steadily as expected, either because the solutions are not optimal, or because some new rectangle may be used to replace several previously used rectangles.

5.3. The effect of rectangle variety

In the previous experiment the rectangles were unique and we varied the number of rectangles included. This section examines whether the problem difficulty changes when the number of different rectangles is varied while keeping the total number of rectangles fixed. To this end, we use the same board as in Section 5.2, but consider new rectangles. Since an instance with $|R| = 30$ rectangles is too hard for CPLEX and an instance with $|R| = 20$ rectangles can almost be solved within 600 seconds, we choose to use $|R| = 24$ in the following. This number allows us to consider same-sized groups of rectangles that are identical, and to denote the number of different rectangles by $d = 1, 2, 3, 4, 6, 8, 12, 24$. If we have d different rectangles, then each group of rectangles consists of $24/d$ identical rectangles (Table 4).

Having several identical rectangles induces a form of symmetry in the mathematical model presented in Section 3. However, CPLEX is nevertheless dealing well with this issue and is able to solve the instance with only one type of rectangle ($d = 1$) to optimality within 40 seconds. In general, it seems harder to solve instances with more types of rectangles: CPLEX solves to optimality three of four instances with $d \leq 4$, and only finds a non-trivial feasible solution to one out of four instances with $d \geq 6$. Although the performance of the heuristic is slightly weaker than CPLEX when $d \leq 4$, the heuristic is superior for instances with many ($d \geq 6$) different types of rectangles, and the performance of the heuristic is solid across the range of instances even when given only 60 seconds of running time.

5.4. The effect of the topography

The size of the board, the number of rectangles, and the number of rectangle types all influence the difficulty of finding optimal solutions in a predictable way. However, the difficulty of an instance may also depend on the structure of the gain values of the board. This is examined in this section, where we generate boards with different topographical features.

5.4.1. Varying the maximum gain for randomly generated boards

For the following experiments we chose a board size of $|M| = 20$ rows and $|N| = 30$ columns. Each instance has $|R| = 15$ rectangles, which are generated as in Section 5.1: the height is a random value from $\{1, \dots, |M|/2\}$, the width is a random value from $\{1, \dots, |N|/2\}$. The costs of the rectangles are generated by assuming that gain values are between 1 and 20, giving an average gain

Table 5
Varying the maximum gain on randomly generated boards.

g^{max}	CPLEX					Heuristic			
	UB	LB	TT	#Rec.	Gap	LB	TB	#Rec.	Gap
15	0	0	0.6	0	0.00	0	0.0	0	0.00
20	282	282	0.3	5	0.00	282	0.6	5	0.00
25	788	788	1.3	9	0.00	783	49.2	9	0.63
30	2087	2087	4.2	11	0.00	2083	0.6	11	0.19
35	3219	3219	6.8	11	0.00	3180	11.9	11	1.21
40	4639	4639	438.0	12	0.00	4579	27.3	11	1.29
45	6316	6171	600.0	12	2.36	6196	47.6	13	1.90
50	7488	7170	600.0	11	4.44	7265	56.7	12	2.98
60	9997	9692	600.0	12	3.15	9712	46.3	12	2.85
70	12,294	11,962	600.0	12	2.78	11,931	15.3	12	2.95

value of 10 and a maximum gain value of 20, and then randomly selecting a cost between 10 times and 20 times the area of the rectangle. The randomly generated rectangles can be summarized as follows:

r	1	2	3	4	5	6	7	8	9
h_r	7	8	4	9	10	9	4	6	2
w_r	3	7	11	1	3	11	12	3	12
c_r	245	870	576	102	530	1504	621	192	479

To generate gain values for the board, we chose a value g^{MAX} , and then the gain values are randomly generated in the range $\{1, 2, \dots, g^{MAX}\}$. We try different values for g^{MAX} to see how this influences the behavior of the solvers. Table 5 shows the results for experiments where g^{MAX} is varied between 15 and 70.

For $g^{MAX} = 15$, the costs of the rectangles are too high compared to the gain values, so it is not advantageous to purchase any of the rectangles. This fact is quickly recognized by both solvers. When $g^{MAX} = 20$ the cost of the rectangles is still relatively high. In the optimal solution, which is found by both CPLEX and the evolutionary algorithm, only 5 rectangles are purchased and used to cover selected parts of the board. Finding the optimal solution appears easy for both methods.

CPLEX is also able to prove optimality within 600 seconds for the instances with g^{MAX} equal to 25, 30, 35, and 40. The case with $g^{MAX} = 35$ is the first where the optimal solution has rectangles that overlap, albeit only slightly. For the instances where g^{MAX} is between 25 and 40, CPLEX finds better solutions than the heuristic.

For $g^{MAX} = 40$, the problem is much harder than for the previous cases. The heuristic finds a solution using 11 rectangles, while CPLEX finds an optimal solution using 12 rectangles. Some rectangles in the optimal solution cover each other, as it is still advantageous to buy a rectangle for some cost, even if it is used only for covering an area that is already partly covered by another rectangle. For the instances with larger gain values, with g^{MAX} from 45 to 70, the heuristic in general finds better solutions than CPLEX, and CPLEX is not able to close the optimality gap. It is clear that the difficulty of solving the problem depends on the structure of the board's gain values and whether or not it is beneficial to let rectangles overlap.

We also performed a sensitivity test where the same instances as above were solved after replacing the set of rectangles with a different set of randomly generated rectangles. Overall, the findings point in the same direction, but the instances were somewhat easier for CPLEX, so that the change in performance appeared for a slightly larger value of g^{MAX} . Also, with the alternative set of rectangles, there was less overlap of rectangles in the optimal solution, even for high gain values, which seems to make the instances somewhat easier to solve for CPLEX.

5.4.2. Randomly generated boards with some large negative gains

For the next experiment, we first randomly generate the gain values from the range $\{1, 2, \dots, g^{MAX}\}$, but then a given number of

cells are modified to $g_{ij} = -g^{LARGE} = -1000$. This latter value can in practice be considered as $-\infty$, as it will dominate any profit obtainable from other board cells in the vicinity. To start with a board

10	11	12	13	14	15
7	9	9	4	6	2
7	13	4	15	11	6
623	1507	371	803	874	132

that is reasonably difficult for CPLEX, we use $g^{MAX} = 40$. Then we put L values of $-g^{LARGE}$ into the board, replacing the original gain values. Values for L vary from 0 to 50 in steps of 5, and the boards are identical except for the cells where $g_{ij} = -g^{LARGE}$. Furthermore, the cells with a value of g^{LARGE} are selected such that when going from L to $L + 5$ cells, the same cells are modified in addition to 5 new cells.

Table 6 summarizes the results. The optimal objective function value decreases slowly as the g_{ij} values are reduced while the costs of the rectangles and all other data remain the same. However, the optimal objective function values decrease only slightly, implying that the rectangles still fit into the board while avoiding the large negative values that are introduced. This can happen because the rectangles are small enough to place in between the large negative values. If we keep adding more negative gain values, eventually no rectangles can be beneficially placed and the optimal value will decrease to zero. However, after setting 50 cells to g^{LARGE} , this is still far from happening. For larger values of L , the problem becomes easier to solve, for both CPLEX and the evolutionary algorithm. Already for $L = 5$ the solution time by CPLEX is radically reduced, from 438 seconds for $L = 0$, to 5.5 seconds.

5.4.3. Gain values generated based on existing benchmark functions

In the tests above, the values of the board have been assigned randomly. This imposes a certain unstructured topography for the boards, which may influence the difficulty of solving the corresponding instances. In this section we consider instances that are based on benchmark functions for unconstrained non-linear optimization. In particular we use the eggholder function and the Matyas function (Abed, Ali, & Kadhim, 2021). First, let a function $f(x, y)$ be defined on $[x^{MIN}, x^{MAX}] \times [y^{MIN}, y^{MAX}]$. For cell (i, j) , let

$$g_{ij} = \left[f(x^{MIN} + (x^{MAX} - x^{MIN}) \frac{i-1}{|M|-1}, y^{MIN} + (y^{MAX} - y^{MIN}) \frac{j-1}{|N|-1}) \right].$$

For a given test function, we vary the focal area $[x^{MIN}, x^{MAX}] \times [y^{MIN}, y^{MAX}]$ to obtain several different instances. We choose regions of the function that provide a mix of negative gain values and positive gain values. These instances then have similarities to geographic areas which have deep valleys or high hills. We again choose the board to have 20 rows and 30 columns, and we keep the same collection of $|R| = 15$ rectangles that we applied in Section 5.4.1.

Table 6
Instances with randomly added large negative gains.

<i>L</i>	CPLEX					Heuristic			
	UB	LB	TT	#Rec.	Gap	LB	TB	#Rec.	Gap
0	4639	4639	438.0	12	0.00	4579	27.3	11	1.29
5	4424	4424	5.5	11	0.00	4402	16.2	11	0.50
10	4062	4062	11.1	11	0.00	4028	28.7	11	0.84
15	3831	3831	6.8	11	0.00	3831	43.7	11	0.00
20	3620	3620	2.9	10	0.00	3620	50.0	10	0.00
25	3473	3473	2.6	10	0.00	3473	22.6	10	0.00
30	3151	3151	4.4	9	0.00	3120	23.8	9	0.98
35	2912	2912	2.3	10	0.00	2912	46.4	10	0.00
40	2773	2773	1.8	9	0.00	2773	21.8	9	0.00
45	2605	2605	2.5	9	0.00	2605	3.7	9	0.00
50	2292	2292	1.9	8	0.00	2292	2.4	8	0.00

Table 7
Description of instances based on benchmark functions.

Instance	Function	<i>x</i>	<i>y</i>	Description
E1	Eggholder	[20, 100]	[30, 70]	Small areas with negative values
E2	Eggholder	[0, 100]	[0, 100]	Two fairly large areas with negative values
E3	Eggholder	[200, 400]	[200, 400]	Quite many negative values, irregular shapes
E4	Eggholder	[300, 500]	[200, 400]	Many valleys with negative values
E5	Eggholder	[300, 400]	[200, 300]	One deep valley, about half of board is negative
M1	Matyas	[−30, 30]	[−30, 30]	Valley in the middle, high hills on the sides
M2	Matyas	[−40, 40]	[−40, 40]	Valley in the middle, very high hills on the sides
M3	Matyas	[0, 50]	[50, 100]	Hill-side, with huge gain values
M4	Matyas	[0, 40]	[10, 50]	Hilly area, some low gain values in one corner
M5	Matyas	[0, 40]	[0, 40]	Hilly area, one part of the board with low gains

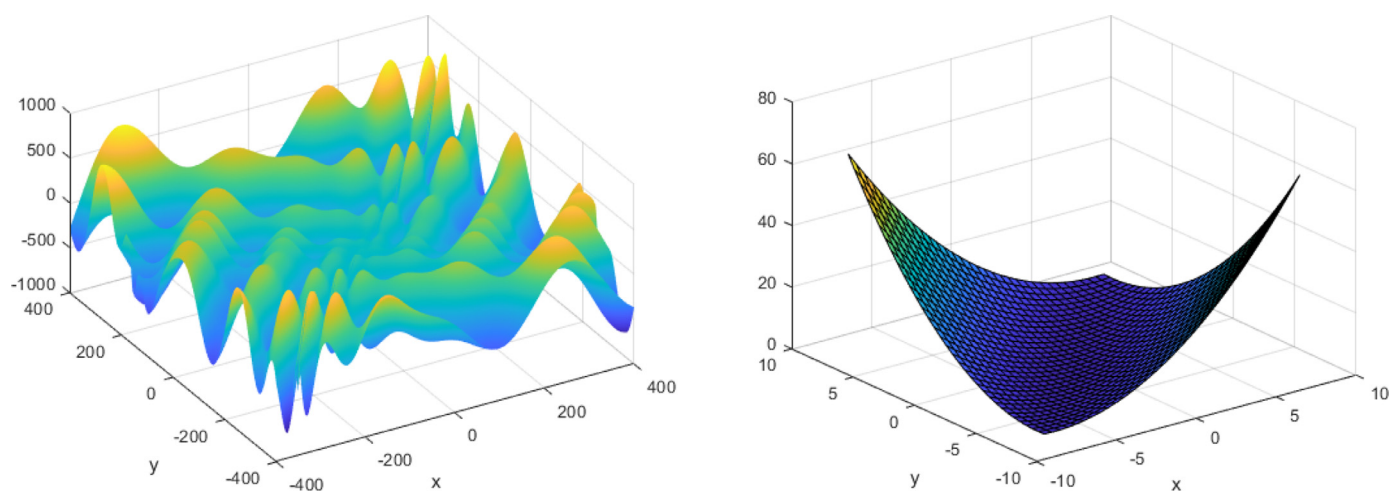


Fig. 15. Two test functions used to generate instances: to the left the Eggholder function, and to the right Matyas function.

Figure 15 illustrates the two functions used as a basis of creating instances. The eggholder function is defined for $-400 \leq x, y \leq 400$, and can be stated as:

$$f(x, y) = -(y + 47) \sin \sqrt{\left| \frac{x}{2} + (y + 47) \right|} - x \sin \sqrt{|x - (y + 47)|},$$

whereas the Matyas function, which is shaped like a big valley, is defined for all real x and y values:

$$f(x, y) = 0.26(x^2 + y^2) - 0.48xy.$$

Table 7 summarizes the instances generated based on the benchmark functions, indicating some of their properties. The instance E3 is illustrated in Fig. 16, which shows both the board values and the superimposed optimal solution, with each rectangle shown in different colors. Figure 17 illustrates two solutions found for the instances based on the Matyas function: an optimal solution for M1, and the best known solution for M3.

Again using a time limit of 600 seconds for CPLEX and 60 seconds for the evolutionary algorithm, Table 8 summarizes the results. The solutions found by CPLEX are proven optimal for three of the eggholder instances and two of the Matyas instances. However, when unable to prove optimality, the solutions obtained by CPLEX are still very good, and the optimality gap is small. The solutions obtained by the heuristic are in general also very good, but not better than CPLEX for any of the instances. Compared to the randomly generated instances, these instances are relatively hard, especially in the cases where many rectangles should be purchased to obtain an optimal solution. As an example, the best solution found for M3 involves covering the entire board using 14 rectangles, yet CPLEX is not able to prove optimality within 600 seconds.

In Sections 5.1 and 5.2 we saw that the performance of CPLEX is worse for larger board sizes and with a larger number of rectangles to be placed. An additional test was executed to see if this still holds when the board is not randomly generated, but rather has

-15	-95	-142	-159	-154	-137	-115	-97	-88	-90	-104	-129	-162	-200	-237	-271	-296	-311	-311	-297	-267	-222	-163	-92	-11	76	166	256	343	422
85	-55	-165	-238	-272	-275	-255	-223	-188	-156	-133	-121	-121	-130	-146	-166	-185	-199	-207	-204	-189	-161	-120	-67	-4	68	144	223	301	375
220	70	-89	-225	-321	-373	-383	-361	-317	-263	-208	-158	-117	-89	-73	-68	-69	-75	-81	-84	-81	-69	-48	-16	25	75	132	193	257	319
237	190	55	-111	-266	-382	-450	-468	-445	-391	-318	-239	-162	-93	-38	3	30	45	51	52	50	50	54	64	81	106	138	176	219	263
-55	115	137	40	-117	-280	-412	-495	-522	-500	-439	-351	-250	-148	-52	29	94	141	171	186	190	186	179	172	167	166	171	182	198	219
-277	-276	-56	57	23	-104	-262	-404	-501	-540	-523	-459	-360	-243	-118	0	105	191	255	297	320	325	317	300	278	255	233	215	202	194
28	-288	-382	-263	-56	-1	-75	-213	-356	-464	-517	-511	-450	-348	-219	-79	58	183	286	365	417	444	448	434	405	367	324	280	238	199
162	61	-206	-477	-466	-203	-42	-37	-135	-268	-385	-454	-463	-315	-183	-35	115	254	371	461	521	551	553	531	490	435	371	304	237	
-1	79	62	-113	-431	-495	-375	-109	-3	-39	-148	-266	-352	-382	-350	-265	-140	9	164	311	437	536	601	633	632	602	549	477	394	305
-255	-139	-21	36	-37	-286	-562	-527	-208	13	62	-4	-116	-216	-270	-266	-203	-93	47	199	348	479	583	652	684	681	644	580	495	394
-442	-382	-267	-125	-8	14	-129	-431	-438	-334	-5	151	147	56	-53	-135	-163	-131	-48	74	216	360	491	597	668	702	697	656	584	487
-496	-523	-482	-375	-221	-60	43	10	-212	-487	-433	-68	205	287	235	125	19	-47	-55	-6	88	211	344	470	575	648	683	679	637	560
-424	-522	-568	-547	-454	-299	-111	53	119	8	-281	-199	-168	-207	394	402	306	181	77	25	30	88	184	299	416	517	591	629	628	588
-272	-406	-513	-574	-573	-499	-353	-155	51	196	197	-8	-296	-245	144	446	537	475	341	204	104	60	74	135	227	329	424	498	541	546
-101	-229	-357	-462	-542	-559	-507	-379	-186	40	242	338	245	-50	15	27	423	619	614	490	326	180	84	49	69	130	213	300	374	423
39	-52	-162	-280	-389	-472	-508	-480	-377	-202	26	261	430	444	235	-88	-67	316	625	704	612	437	252	104	15	-11	14	75	151	224
114	78	14	-74	-177	-281	-370	-423	-421	-349	-201	11	256	473	576	477	154	86	143	539	726	694	529	316	121	-21	-97	-112	-78	-16
111	135	131	95	32	-54	-150	-242	-310	-335	-298	-187	-4	231	470	640	645	410	32	3	358	661	720	591	371	138	-55	-182	-240	-236
38	114	168	194	188	150	83	-3	-97	-179	-230	-229	-161	-18	190	429	639	729	606	239	55	111	498	672	613	410	155	-81	-258	-361
-87	25	126	208	263	286	273	226	150	57	-37	-113	-149	-126	-31	136	356	581	733	715	445	5	-79	239	538	582	426	172	-95	-316

Fig. 16. Instance E3, with the board values shown on the left and the optimal solution, placing nine rectangles, to the right.

42	42	44	40	54	62	73	86	101	118	138	160	184	210	238	269	302	337	375	414	456	500	546	595	646	699	754	811	871	933
39	38	34	35	38	43	51	61	73	87	103	122	143	166	191	219	249	281	315	351	390	431	474	520	567	617	669	723	780	839
41	34	30	28	27	30	34	41	50	61	74	89	107	127	149	174	200	229	260	294	329	367	407	449	494	541	589	641	694	750
48	38	31	25	22	21	22	26	32	39	50	62	77	93	113	134	157	183	211	241	274	308	345	384	426	469	515	563	613	666
60	47	37	28	22	18	16	16	19	24	31	40	51	65	81	99	120	142	167	194	223	255	289	325	363	403	446	491	538	587
78	62	48	36	27	20	15	12	11	13	17	23	31	42	55	70	87	106	128	152	178	206	237	270	305	342	382	423	467	514
101	81	64	49	37	27	18	12	9	7	8	11	16	24	33	45	59	76	94	115	138	163	191	220	252	287	323	361	402	445
129	106	86	68	52	39	27	18	12	7	5	4	6	11	17	26	37	50	66	83	103	125	150	176	205	236	269	305	342	382
162	136	113	92	73	56	42	29	20	12	6	3	2	3	6	12	20	30	42	57	74	92	114	137	163	191	221	253	288	324
200	171	145	120	98	79	61	46	33	22	13	7	3	1	3	8	15	24	35	49	65	83	103	126	150	177	207	238	272	
243	211	182	154	129	106	86	67	51	37	25	16	8	3	0	0	1	5	11	19	30	42	57	75	94	115	139	165	194	224
282	257	224	194	165	139	115	94	75	57	42	19	11	5	1	0	0	3	8	16	25	37	51	67	86	106	129	154	182	
346	308	272	238	207	177	150	126	103	83	65	49	35	24	15	8	3	1	1	3	7	13	22	33	46	61	79	98	120	145
405	363	324	288	253	221	191	163	137	114	92	74	57	42	30	20	12	6	3	2	3	6	12	20	29	42	56	73	92	113
469	424	382	342	305	269	236	205	176	150	125	103	83	66	50	37	26	17	11	6	4	5	7	12	18	27	39	52	68	86
538	491	445	402	361	323	287	252	220	191	163	138	115	94	76	59	45	33	24	16	11	8	7	9	12	18	27	39	52	68
613	562	514	467	423	382	342	305	270	237	206	178	152	128	106	87	70	55	42	31	23	17	13	11	12	15	20	27	36	48
692	639	587	538	491	446	403	363	325	289	255	223	194	167	142	120	99	81	65	51	40	31	24	19	16	16	18	22	28	37
777	720	666	613	563	515	469	426	384	345	308	274	241	211	183	157	134	113	93	77	62	50	39	32	26	22	21	22	25	31
867	807	750	694	641	589	541	494	449	407	367	329	294	260	229	200	174	149	127	107	89	74	61	50	41	34	30	27	28	30

Fig. 17. Instance M1 with its optimal solution is shown to the left, and instance M3 and its best known solution is shown to the right.

Table 8

Results for instances based on the eggholder and Matyas function.

Instance	CPLEX				Heuristic		
	UB	LB	TT	Gap [%]	LB	TB	Gap [%]
E1	33,849	33,719	600.0	0.39	33,349	39.0	1.47
E2	21,800	21,800	50.7	0.00	21,708	59.3	0.42
E3	81,089	81,089	32.4	0.00	81,089	6.6	0.00
E4	108,308	108,059	600.0	0.23	107,829	17.3	0.44
E5	41,082	41,082	127.1	0.00	41,082	43.5	0.00
M1	94,118	94,118	48.2	0.00	93,850	13.8	0.28
M2	173,415	173,266	600.0	0.09	172,977	4.9	0.25
M3	506,058	505,712	600.0	0.07	505,329	1.2	0.14
M4	67,421	67,108	600.0	0.47	67,074	2.7	0.51
M5	45,581	45,581	489.5	0.00	45,435	1.2	0.32

-15	-95	-142	-159	-154	-137	-115	-97	-88	-90	-104	-129	-162	-200	-237	-271	-296	-311	-311	-297	-267	-222	-163	-92	-11	76	166	256	343	422
85	-55	-165	-238	-272	-275	-255	-223	-188	-156	-133	-121	-120	-130	-146	-166	-185	-199	-207	-204	-189	-161	-120	-67	-4	68	144	223	301	375
220	70	-89	-225	-321	-373	-383	-361	-317	-263	-208	-158	-117	-89	-73	-68	-69	-75	-81	-84	-81	-69	-48	-16	25	75	132	193	257	319
237	190	55	-111	-266	-382	-450	-468	-445	-391	-318	-239	-162	-93	-38	3	30	45	51	52	50	50	54	64	81	106	138	176	219	263
-55	115	137	40	-117	-280	-412	-495	-522	-500	-439	-351	-250	-148	-52	29	94	141	171	186	190	186	179	172	167	166	171	182	198	219
-277	-276	-56	57	23	-104	-262	-404	-501	-540	-523	-459	-360	-243	-118	0	105	191	255	297	320	325	317	300	278	255	233	215	202	194
28	-288	-382	-263	-56	-1	-75	-213	-356	-464	-517	-511	-450	-348	-219	-79	58	183	286	365	417	444	448	434	405	367	324	280	238	199
162	61	-206	-477	-466	-203	-42	-37	-135	-268	-385	-454	-463	-413	-315	-183	-35	115	254	371	461	521	551	553	591	409	435	371	304	233
-1	79	62	-113	-431	-493	-575	-109	-3	-69	-148	-266	-352	-352	-350	-265	-90	164	313	437	536	601	633	632	602	549	477	394	304	205
-255	-139	6	36	-37	-286	-562	-527	-208	13	-42	-116	-216	-206	-263	-193	9	199	349	479	583	652	684	681	644	594	495	394	304	205
-442	-382	-267	-125	-8	14	-129	-431	-438	-334	-5	151	147	56	-53	-135	-163	-131	-40	74	216	360	497	569	602	697	656	584	457	337
-496	-523	-482	-375	-221	-60	13	-122	-487	-433	-68	205	287	235	125	19	-47	-55	-8	88	211	344	470	575	648	685	679	637	560	427
-442	-522	-568	-547	-454	-299	111	53	119	8	-281	-199	-168	207	394	402	306	181	77	25	30	88	184	299	416	517	591	629	628	588
-272	-406	-513	-574	-573	-499	-353	-155	51	196	197	-8	-296	245	144	446	537	475	341	204	104	60	74	135	227	329	424	498	541	546
101	-229	-357	-468	-542	-559	-507	-379	-186	40	242	238	245	-50	15	27	423	619	614	690	326	180	49	69	130	230	300	374	423	423
39	-52	-162	-280	-407	-472	-508	-408	-240	262	261	430	244	235	-88	136	615	625	704	612	437	252	104	15	-11	14	75	151	224	261
114	78	14	-74	-177	-281	-370	-423	-421	-349	-201	11	256	473	576	477	154	88	143	539	726	694	529	316	121	-21	-97	-112	-78	-16
118	135	131	95	32	-54	-150	-240	-335	-298	-187	-4	231	470	640	645	610	32	3	358	661	720	591	371	138	-5	-182	-240	-258	-361
38	114	194	186	150	83	-3	-97	-179	-230	-229	-161	-18	129	439	639	729	606	239	55	111	498	672	613	410	155	-81	-258	-361	261
-87	25	126	208	263	267	273	226	150	57	-37	-113	-149	-126	-31	136	356	581	733	715	445	5	-79	239	538	582	426	172	-95	-31

Table 9
Results for additional instances based on the eggholder and Matyas functions.

Instance	CPLEX				Heuristic		
	UB	LB	TT	Gap [%]	LB	TB	Gap [%]
E1b	138,921	127,308	600.0	9.1 %	135,858	1.6	2.3 %
E2b	92,849	91,761	600.0	1.2 %	89,634	2.6	3.6 %
E3b	364,288	359,452	600.0	1.3 %	356,970	1.2	2.1 %
E4b	472,878	445,730	600.0	6.1 %	460,735	2.7	2.6 %
E5b	202,613	199,712	600.0	1.5 %	198,903	2.6	1.9 %
M1b	361,246	348,695	600.0	3.6 %	358,724	2.4	0.7 %
M2b	693,316	0	600.0	N/A	659,870	2.3	5.1 %
M3b	202,5950	0	600.0	N/A	199,0160	3.1	1.8 %
M4b	292,514	0	600.0	N/A	260,790	3.1	12.2 %
M5b	207,555	0	600.0	N/A	177,549	1.5	16.9 %

Table 10
Results for instances based on covering a landmass using satellite images.

Instance	CPLEX				Heuristic		
	UB	LB	TT	Gap [%]	LB	TB	Gap [%]
S1	18,094	18,094	6.8	0.00	18,086	26.2	0.04
S2	25,036	25,036	10.2	0.00	25,018	19.4	0.07
S3	46,106	46,106	40.2	0.00	46,028	3.5	0.17
S4	70,084	70,084	138.9	0.00	69,982	7.0	0.15
S5	96,386	96,386	448.2	0.00	96,206	10.2	0.19
S6	105,675	105,384	600.0	0.28	105,500	6.1	0.17
S7	123,818	123,818	444.7	0.00	123,520	10.2	0.24

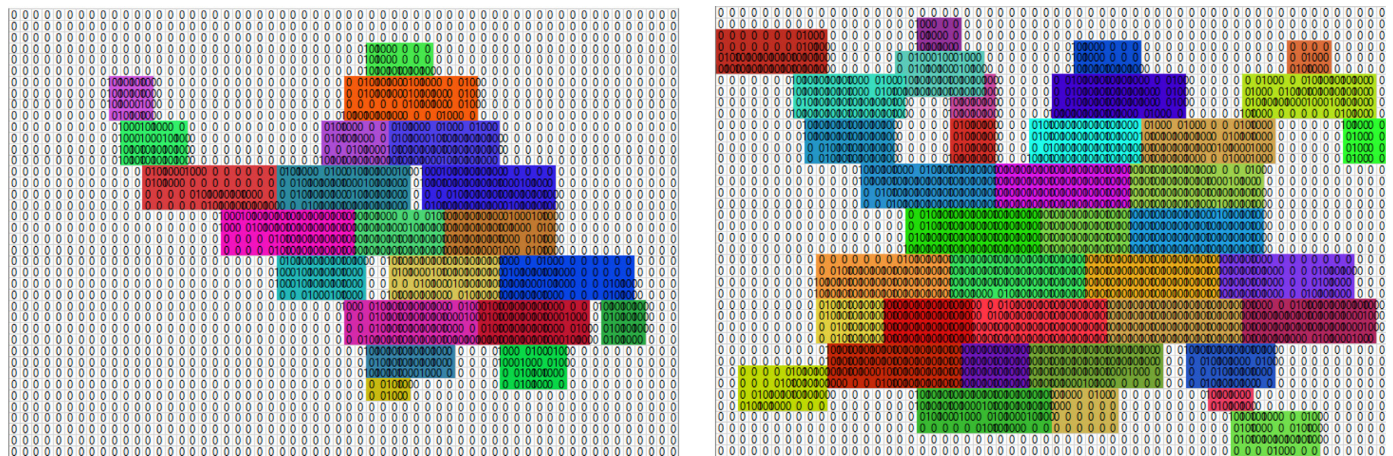


Fig. 18. Solutions to instances S3 (left) and S6 (right).

0.24% of the optimal solutions for all instances, and finds a better solution than CPLEX on instance S6. Figure 18 shows the solutions found by CPLEX for instances S3 (optimal) and S6 (not optimal).

5.6. Robustness of performance

In the previous tests we have evaluated the behavior of the two solution methods when systematically varying certain aspects of the instances solved, such as the size of the board, the number of rectangles, or the structure of the gain values. In the following we examine whether the behavior of the solution methods varies substantially when solving instances that are similar in structure. That is, we keep most of the attributes of the instances constant, and only randomly perturb some aspect thought to be insignificant. Five groups of instances are created, with each group consisting of ten perturbed instances.

In the first group of instances, we fix $m = n = 50$ and $|R| = 20$. However, we randomly select each gain value from $\{1, \dots, 100\}$, and the height and width of each rectangle from $\{1, \dots, 25\}$. The cost of each rectangle is selected randomly between 0 and the size

of the rectangle times the average gain. For the second group of instances, the exact same procedure is followed, except that each gain value is either 0 or 1. Instances in group 3 are generated as in Section 5.1, with $p = 10$, so again the gains and rectangles vary between instances within the group. In group 4, instances are based on the Eggholder function and E3, but the 15 rectangles are chosen randomly for each new instance. Group 5 follows the same setting as group 4, but based on Matyas and M3.

Table 11 shows the results for each group of instances (G). For both CPLEX and the heuristic we report the minimum, average, and maximum gap to the upper bound reported by CPLEX. For CPLEX we also report the number of instances where optimality was proven (Opt). Furthermore, we report the minimum, average, and maximum total time (TT) for CPLEX and time to best (TB) for the heuristic.

For each of the groups, the performances of the solution methods are relatively stable. CPLEX and the heuristic both find the optimal solution for all instances in group 3 and group 4, although the time it takes to find the best (for the heuristic) and to prove optimality (for CPLEX) does vary a bit from instance to instance.

Table 11

Results for five groups of instances, each group containing ten instances with similar characteristics.

G	Opt	CPLEX						Heuristic					
		Gap [%]			TT			Gap [%]			TB		
		Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
1	1/10	0.0	2.2	3.4	137.0	553.7	600.0	2.3	3.3	4.5	4.0	22.3	47.2
2	7/10	0.0	0.8	4.0	19.3	217.0	600.0	0.7	4.1	6.1	2.2	21.9	49.8
3	10/10	0.0	0.0	0.0	14.6	44.0	81.3	0.0	0.0	0.0	2.5	12.3	50.7
4	10/10	0.0	0.0	0.0	8.8	28.6	55.3	0.0	0.0	0.0	0.4	15.3	45.6
5	1/10	0.0	0.0	0.1	538.6	593.9	600.0	0.1	0.1	0.2	0.4	16.1	44.0

Also for group 5 the performances are remarkably stable, and although CPLEX is only able to prove optimality for one of ten instances, the gap to the best known is never above 0.09% for CPLEX and 0.17% for the heuristic.

For groups 1 and 2, the variance in performance is larger: the gaps vary from 0% to 4.0% for CPLEX and from 0.7% to 6.1% for the evolutionary algorithm. However, for these groups we perturb both the rectangles and also the gain values. This could suggest that the gain values are more important for the stability of the performance than the rectangle sizes and costs. The gain values are also varied in group 3, but here the instances seem to be easier, so the performances in terms of gaps are unaffected.

6. Conclusions

This paper has introduced a board packing problem, where we are given a rectangular board with m rows and n columns. Each cell of the board is associated to a gain value that represents a revenue to be obtained if the cell is covered at least once. To cover cells, a finite set of rectangles is given: each of these rectangles can be purchased for a cost and placed on the board to cover cells. The rectangles to be purchased have their own height, width, and cost, and when placed on the board the rectangles are allowed to overlap. The goal is to obtain the highest possible profit.

The problem is shown to be NP-hard, even when restricted to very specific gain values and rectangle sizes. For instance, the problem is NP-hard even if we only consider gain values from $\{0, 1\}$ and all rectangles are of size 2×2 .

We examine two methods to solve the problem. First, a binary programming model is formulated and solved using a commercial mixed-integer programming solver. Second, an evolutionary algorithm is proposed to generate heuristic solutions. In the computational experiments, we find that the difficulty of solving instances depends on the size of the board, the number of rectangles, the variety of rectangles, and the topography of the board. When important structural aspects are fixed, the performances of the solution methods are quite stable. For certain instances, optimal solutions can be obtained using the mathematical programming solver, whereas for other instances the heuristic algorithm performs better. Both methods can deal well with what we consider to be relatively large instances, but the mathematical programming solver does fail for very large instances.

Acknowledgments

The authors would like to thank three anonymous reviewers, whose useful ideas and comments helped to improve this paper significantly. The research of Gyorgy Dosa and Zolt Tuza is supported in part by the [National Research, Development and Innovation Office](#) – NKFIH under the grant [SNN 129364](#). Gyula Abraham and Gyorgy Dosa acknowledge financial support also from the Slovenian – Hungarian bilateral project, “Optimization and fault forecasting in port logistics processes using artificial intelligence,

process mining and operations research”, grant 2019-2.1.11-TÉT-2020-00113.

References

- Abed, I. A., Ali, M. M., & Kadhim, A. A. A. (2021). Using particle swarm optimization to solve test functions problems. *Bulletin of Electrical Engineering and Informatics*, 10(6), 3422–3431.
- Aupperle, L. J., Conn, H. E., Keil, J. M., & O'Rourke, J. (1988). Covering orthogonal polygons with squares. In *26th annual conference on communication, control and computing* (pp. 97–106).
- Bereg, S., Cabello, S., Diaz-Banez, J. M., Pérez-Lantero, P., Seara, C., & Ventura, I. (2012). The class cover problem with boxes. *Computational Geometry*, 45(7), 294–304.
- Berman, O., Drezner, Z., & Wesolowsky, G. O. (2003). The expropriation location problem. *Journal of the Operational Research Society*, 54(7), 769–776.
- Berman, O., & Krass, D. (2002). The generalized maximal covering location problem. *Computers and Operations Research*, 29, 563–581.
- Blanquero, R., Carrizosa, E., Tóth, G.-B., & Nogales-Gómez, A. (2016). p-facility Huff location problem on networks. *European Journal of Operational Research*, 255(1), 34–42.
- Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3), 268–308.
- Çakir, O., & Wesolowsky, G. O. (2011). Planar expropriation problem with non-rigid rectangular facilities. *Computers and Operations Research*, 38, 75–89.
- Church, R., & ReVelle, C. (1974). The maximal covering location problem. *Papers of Regional Science Association*, 32, 101–118.
- Culberson, J. C., & Reckhow, R. A. (1988). Covering polygons is hard. In *SFCS '88 proceedings of the 29th annual symposium on foundations of computer science, October 24–26* (pp. 601–611).
- Demiröz, B. E., Altınel, K., & Akarun, L. (2019). Rectangle blanket problem: Binary integer linear programming formulation and solution algorithms. *European Journal of Operational Research*, 277, 62–83.
- Dosa, G., Hvattum, L. M., Olaj, T. A., & Tuza, Zs. (2020). The board packing problem: Packing rectangles into a board to maximize profit. In I. Vassányi (Ed.), *Proceedings of the Pannonian conference on advances in information technology (PCIT 2020)* (pp. 10–16). Veszprém, Hungary: University of Pannonia.
- Drezner, Z. (1987). On the rectangular p-center problem. *Naval Research Logistics*, 34, 229–234.
- Dupont, L., Luras, M., & Yugma, C. (2013). Generalized covering location problem with multiple-coverage: Exact and heuristic method. In *7th IFAC conference on manufacturing modelling, management and control, June 19–21* (pp. 442–447). Saint Petersburg, Russia: International Federation of Automatic Control.
- Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. W. H. Freeman and Company.
- Garey, M. R., Johnson, D. S., & Stockmeyer, L. (1976). Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3), 237–267.
- Ghaffarian, S., & Kerle, N. (2019). Towards post-disaster debris identification for precise damage and recovery assessments from UAV and satellite images. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2/w13, 297–302.
- Guo, D., Li, X., & Lou, W. (2021). Detection and analysis of road information based on optical satellite image. In *2021 international conference on intelligent transportation, big data & smart city (ICITBS)* (pp. 94–96).
- Jang, J., Choi, J., Bae, H.-J., & Choi, I. C. (2013). Image collection planning for Korea multi-purpose SATellite-2. *European Journal of Operational Research*, 230, 190–199.
- Leung, J. Y.-T., Tam, T. W., Wong, C. S., Young, G. H., & Chin, F. Y. L. (1990). Packing squares into a square. *Journal of Parallel and Distributed Computing*, 10(3), 271–275.
- Mahapatra, P. R. S., Goswami, P. P., & Das, S. (2007). Covering points by isothetic unit squares. In *Proceedings of the 19th annual canadian conference on computational geometry*, CCCG 2007, August 20–22, 2007 (p. 4). Ottawa, Canada: Carleton University.
- Martinez-Sykora, A., Alvarez-Valdes, R., Bennell, J. A., Ruiz, R., & Tamarit, J. M. (2017). Mathheuristics for the irregular bin packing problem with free rotations. *European Journal of Operational Research*, 258(2), 440–455.
- Masek, W. J. (1978). Some NP-complete set covering problems. Unpublished manuscript.

- Megiddo, N., & Supowit, K. J. (1984). On the complexity of some common geometric location problems. *SIAM Journal on Computing*, 13(1), 182–196.
- Mondino, E. B., Perotti, L., & Piras, M. (2012). High resolution satellite images for archeological applications: The Karima case study (Nubia region, Sudan). *European Journal of Remote Sensing*, 45(1), 243–259.
- Mukherjee, M., & Chakraborty, K. (2002). A polynomial-time optimization algorithm for a rectilinear partitioning problem with application in VLSI design automation. *Information Processing Letters*, 83, 41–48.
- Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers and Operations Research*, 31(12), 1985–2002.
- Vidal, T. (2022). Hybrid genetic search for the CVRP: Open-source implementation and SWAP* neighborhood. *Computers and Operations Research*, 140, 105643.
- Vidal, T., Crainic, T. G., Gendreau, M., & Prins, C. (2014). A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, 234, 658–673.
- Wäscher, G., Haußner, H., & Schumann, H. (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3), 1109–1130.