A heuristic approach for packing rectangles in convex regions

Andrea Cassioli

Dipartimento di Sistemi e Informatica, Università di Firenze
Via di Santa Marta, 3, 50139 Firenze, Italy
e-mail: cassioli@dsi.unifi.it

Marco Locatelli

Dipartimento di Ingegneria Informatica, Università di Parma
Via G.P. Usberti, 181/A, 43124 Parma, Italy
e-mail: locatell@ce.unipr.it

Abstract

In this paper we propose a heuristic approach for the problem of packing equal rectangles within a convex region. The approach is based on an Iterated Local Search scheme (or, using a terminology employed for continuous problems, a Monotonic Basin Hopping), in which the key step is the perturbation move. Different perturbation moves, both combinatorial and continuous ones, are proposed and compared through extensive computational experiments on a set of test instances. The overall results are quite encouraging.

Keywords: Packing Problems, Mixed Integer Global Optimization, Iterated Local Search, Monotonic Basin Hopping

1. Introduction

In packing problems some objects have to be placed into some containers in such a way that the overall unused space within the container (the so called *waste*) is minimized. The objects and the containers have a fixed (usually two- or three-dimensional) shape, while their dimensions and positions can vary. More formally, let us consider N objects. For each object i we introduce the parameter vectors $\mathbf{x}_i, \alpha_i$, respectively a position and a size vector, which allow to uniquely

identify the portion of the space where the object lies, denoted by

$$D_i = D_i(\mathbf{x}_i, \alpha_i), \quad i = 1, \ldots, N.$$

For instance, if the objects are circles, we have a two-dimensional position vector corresponding to the coordinates of the center of the circle, while we have a single size parameter, corresponding to the radius of the circle. Although the shapes of the objects might be different from each other, in most cases the shape is the same for all of them. Possible shapes are convex objects such as circles, squares, rectangles, but also nonconvex ones are sometimes considered.
Next, we consider a container

$$C = C(\mathbf{x}_0, \alpha_0)$$

with some fixed shape (not necessarily equal to those of the objects), and also depending on a position and a size vector. In some applications the vectors, identifying the portion of the space occupied by the container, are replaced by a description of such portion through proper inequalities and/or equalities.
A packing problem can be formulated as an optimization problem. The *constraints* of the problem are the following

- the objects may touch each other but can not overlap, i.e.,

$$D_i^0(\mathbf{x}_i, \alpha_i) \cap D_j^0(\mathbf{x}_j, \alpha_j) = \emptyset \quad \forall\ i \neq j, \tag{1}$$

  where D_i^0, D_j^0 denote the interior of D_i, D_j;

- the objects must lie within the container, i.e.,

$$D_i(\mathbf{x}_i, \alpha_i) \subseteq C(\mathbf{x}_0, \alpha_0) \quad \forall\ i. \tag{2}$$

The *decision variables* depend on the problem at hand. Usually, the position vectors $\mathbf{x}_i$ are variables, while the size vectors α_i can either be variables or fixed values. The position vector $\mathbf{x}_0$ for the container is usually fixed, while its size vector may either be variable or fixed. The *objective* of the problem should state the fact that we aim at minimizing the waste. This can be accomplished in different ways depending on the nature of the position and size vectors:

- *objects: variable* $\mathbf{x}_i$*, fixed* α_i*; container: fixed* $\mathbf{x}_0$*, variable* α_0 - in this case we aim at minimizing the area (or volume) of the container, which is usually obtained as some function of the variable parameter α_0;

- *objects: variable* $\mathbf{x}_i$*, variable* α_i*; container: fixed* $\mathbf{x}_0$*, fixed* α_0 - in this case we aim at maximizing the sum of the areas (or volumes) of the objects, where the area of each object i is usually obtained as some function of the variable parameter α_i;

- *objects: variable* $\mathbf{x}_i$*, fixed* α_i*; container: fixed* $\mathbf{x}_0$*, fixed* α_0 - in this case we aim at maximizing the number N of objects which can be placed within the container without violating the constraints.

Many packing problems have been tackled in the literature. Most papers present heuristic approaches, although some exact approaches have also been proposed. A detailed survey about methods and applications of packing problems can be found in [1], while a problem typology is extensively presented in [2].
Probably, the most widely studied cases are those involving circular objects within containers having some regular form such as a circle or a square. In the field of circular objects/square container we recall some heuristic (see, e.g., [3, 4, 5, 6, 7]) and exact approaches (see, e.g., [8, 9, 10, 11]). We also refer to survey [12] and a book [13]. Heuristic approaches for the case of circular objects/circular container include those discussed in [14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]. In [25] the problem of placing circles with different sizes into a rectangular container with fixed width and minimum height is considered. An even more general problem is tackled in [26], aiming to cut several different shapes (circles and convex polygons) from rectangular plates of raw material. Benchmark results for the problem of packing equal circles in a container whose shape is a square, a circle or an equilateral triangle are reported and continuously updated in E. Specht's web site[1]. Test instances for the problem of packing unequal circles in a circle include those in [17] and the quite challenging instances

[1] `http://www.packomania.com`

from the Circle Packing Contest[2]. The problem of packing irregular polygons has also been well studied (see [27] for an overview), considering either fixed size containers or infinite length strips (the so–called Irregular Strip Packing Problem). The latter has been tackled, for instance, in [28] using a local search–based method, while in [29] linear programming relaxations are solved in a Simulated–Annealing framework.

In this paper we deal with the case of rectangular objects/ convex container. In a series of paper [30, 31, 32] different variants of this problem have been considered. In all variants the size parameters of the rectangles, a and b ($a \geq b$) denoting the length of the two edges of the rectangles, are fixed and equal for all the rectangles. Moreover, the portion of the space occupied by the convex container is fixed and described by (convex) inequalities, i.e.,

$$C = \{(y_1, y_2) \in \mathbb{R}^2 \ : \ g_j(y_1, y_2) \leq 0, \ j = 1, \ldots, m\}, \tag{3}$$

where functions g_j, $j = 1, \ldots, m$, are convex ones. In the first variant [30] the position vector $\mathbf{x}_i$ is made up by three components c_x^i, c_y^i, θ_i, where the first two identify the center of the rectangle, while θ_i denotes the rotation of the rectangle with respect to its horizontal position (the one where the basis of the rectangle is its longest edge). In the second variant [31] the position vector $\mathbf{x}_i$ still has three components c_x^i, c_y^i, p_i, where the first two are as before, while the third one is a binary variable equal to 0 if the rectangle is in horizontal position ($\theta_i = 0$), and to 1 if the rectangle is in vertical position ($\theta_i = 90°$). Finally, the third variant [32] is similar to the first one, but the parameter θ_i is imposed to be equal for all the rectangles, (i.e., all rectangles can be rotated by a common rotation angle).

In this paper we will explore the second variant, although the proposed technique can be extended also to the other two variants. The paper is structured as follows. In Section 2 we will report the model for the problem proposed in [31]. In Section 3 the main components of the proposed heuristic approach are

[2] `http://www.recmath.org/contest/CirclePacking/index.php`

presented and discussed. Extensive computational experiments and their results are presented in Section 4.

2. Mathematical model of the problem

In this section we report the mathematical model proposed in [31] for the packing of equal rectangles into a convex region, when rectangles are allowed to be either in horizontal or vertical position. Therefore, the container is identified by a finite number of convex inequalities as in (3), while each object i has fixed size parameters a, b, and variable position parameters c_x^i, c_y^i, corresponding to the center of the rectangles, and p_i, a binary variable giving the orientation (horizontal, if equal to 0, or vertical, if equal to 1) of the rectangle. As commented in Section 1, position and size parameters allow to identify the portion of the space ($\mathbb{R}^2$ in this case) occupied by the object. In particular, for object i such portion is the rectangle with the four vertices

$$\begin{aligned} V_i^{sw} &= \left(c_x^i - \tfrac{a}{2}(1-p_i) - \tfrac{b}{2}p_i, c_y^i - \tfrac{b}{2}(1-p_i) - \tfrac{a}{2}p_i\right) \\ V_i^{se} &= \left(c_x^i + \tfrac{a}{2}(1-p_i) + \tfrac{b}{2}p_i, c_y^i - \tfrac{b}{2}(1-p_i) - \tfrac{a}{2}p_i\right) \\ V_i^{nw} &= \left(c_x^i - \tfrac{a}{2}(1-p_i) - \tfrac{b}{2}p_i, c_y^i + \tfrac{b}{2}(1-p_i) + \tfrac{a}{2}p_i\right) \\ V_i^{ne} &= \left(c_x^i + \tfrac{a}{2}(1-p_i) + \tfrac{b}{2}p_i, c_y^i + \tfrac{b}{2}(1-p_i) + \tfrac{a}{2}p_i\right) \end{aligned}$$

Constraints (1) are equivalent to

$$\begin{aligned} |\, c_x^i - c_x^j \,| &\geq \tfrac{a}{2}(1-p_i) + \tfrac{b}{2}p_i + \tfrac{a}{2}(1-p_j) + \tfrac{b}{2}p_j \\ &\text{or} \qquad\qquad \forall\ i \neq j \qquad (4) \\ |\, c_y^i - c_y^j \,| &\geq \tfrac{b}{2}(1-p_i) + \tfrac{a}{2}p_i + \tfrac{b}{2}(1-p_j) + \tfrac{a}{2}p_j. \end{aligned}$$

Constraints (2) are equivalent to

$$g_k(V_i^t) \leq 0 \quad i = 1, \ldots, N, \quad k = 1, \ldots, m, \quad t \in \{sw, se, nw, ne\}. \qquad (5)$$

We denote by $\mathcal{F}_N$ the region identified by constraints (4)-(5) with respect to the variables c_x^i, c_y^i, p_i, $i = 1, \ldots, N$. We aim at maximizing N, i.e., we would

like to detect

$$N^* = \max\{N \ : \ \mathcal{F}_N \neq \emptyset\}.$$

Therefore, we need to solve some nonlinear feasibility subproblems. Following [31], we transform such feasibility problems into an unconstrained mixed integer global optimization ones. Indeed, we can define for each i, j, $i \neq j$,

$$f_{ij}^x(c_x^i, p_i, c_x^j, p_j) = \max\left\{0, (c_x^i - c_x^j)^2 - \left[a - \frac{a-b}{2}(p_i + p_j)\right]^2\right\}^2 \geq 0,$$

$$f_{ij}^y(c_y^i, p_i, c_y^j, p_j) = \max\left\{0, (c_y^i - c_y^j)^2 - \left[b + \frac{a-b}{2}(p_i + p_j)\right]^2\right\}^2 \geq 0,$$

so that

$$\text{(4) are satisfied} \quad \Leftrightarrow \quad \sum_{i<j} f_{ij}^x \times f_{ij}^y = 0.$$

Moreover, we can define functions

$$h_i(c_x^i, c_y^i, p_i) = \sum_{k=1}^{m} \sum_{t \in \{sw,se,nw,ne\}} \max\{0, g_k(V_i^t)\}^2 \geq 0,$$

so that

$$\text{(5) are satisfied} \quad \Leftrightarrow \quad \sum_{i=1}^{N} h_i = 0.$$

If we finally define function

$$f_N(c_x^1, c_y^1, \ldots, c_x^N, c_y^N, p_1, \ldots, p_N) = \sum_{i<j} f_{ij}^x \times f_{ij}^y + \sum_{i=1}^{N} h_i = 0,$$

we will have that

$$\mathcal{F}_N \neq \emptyset \quad \Leftrightarrow \quad \min \ f_N = 0,$$

i.e., N^* is the largest value N such that the unconstrained global minimum of function f_N , which is certainly ≥ 0, is exactly equal to 0. Therefore, the solution of the packing problem goes through the solution of a series of unconstrained mixed integer global minimization problems. It is worthwhile to make a couple of remarks. Due to the convexity of the container, it is straightforward to derive trivial lower and upper limits on the c_x^i, c_y^i variables: such limits do not reduce the feasible region, so we do not include them in the mathematical formulation,

keeping the problem an unconstrained one, but they can be used to generate initial configurations as explained in Section 3.1. Moreover, once the container area is known, an upper bound to $N^\star$ can be easily computed.

3. The proposed heuristic approach

The heuristic approach proposed in this paper belongs to the class of Iterated Local Search (ILS) approaches (see, e.g., [33]), if we employ the terminology of combinatorial optimization problems, or Monotonic Basin Hopping (MBH) approaches (see, e.g., [34, 35]) if we employ the terminology of continuous optimization problems. Due to the mixed nature (discrete and continuous) of the problem at hand, we report both terminologies. A general scheme of the approach is depicted in Algorithm 1.

Algorithm 1: The proposed Monotonic Basin Hopping scheme.

1 $Y_0 = RanGen()$;
2 $X_0 = Refine(Y_0)$;
3 $k = 1$, $noimp = 0$;
4 **repeat**
5 $Y_k = Perturb(X_{k-1})$;
6 $Z_k = Refine(Y_k)$;
7 **if** $f(Z_k) < f(X_{k-1})$ **then**
8 $X_k = Z_k$;
9 $noimp = 0$;
10 **end**
11 **else**
12 $X_k = X_{k-1}$;
13 $noimp = noimp + 1$;
14 **end**
15 $k = k + 1$;
16 **until** $noimp \leq MaxNoImp$;
17 **return** X_k;

Steps 1-3 perform the initialization using procedure *RanGen* to generate an initial random configuration which is then refined by procedure $Refine$. Steps 4-16 define the main loop: the current iterate X_{k-1} (which is also the best observed configuration during the current run of the algorithm) is perturbed through procedure $Perturb$ (Step 5); the result of the perturbation Y_k is refined into Z_k through procedure $Refine$ (Step 6); if the function value at Z_k improves that at X_{k-1}, then the next iterate is set equal to Z_k and the counter *noimp* (which gives the number of iterations with no improvement) is reset to 0 (Steps 7-10); otherwise (Steps 11-14) the next iterate is left unchanged and *noimp* is

increased by 1; finally, if no improvement has been observed for $MaxNoImp$ iterations, then the algorithm exits the cycle (Step 16) and the current iterate X_k is returned (Step 17).

A single run of Algorithm 1 may not be enough to reach a solution. Therefore, multiple runs are started in a Multistart fashion.

Different definitions are possible for the three procedures *RanGen*, $Refine$, $Perturb$ and their choice might have a considerable impact on the performance of the approach. In what follows we will discuss our choices for such procedures.

3.1. RanGen procedure

Procedure *RanGen* generates an initial random configuration both for the binary variables p_i and for the continuous ones c_x^i, c_y^i, $i = 1, \ldots, N$. For what concerns the continuous variables, we randomly place the center of each rectangle within the box containing the convex region. A little bit more critical is the random generation of the binary variables. One straightforward possibility is a uniform random generation for the value of each binary variable p_i: the value of p_i is set equal to 0 with probability 0.5, otherwise it is set equal to 1. However, it turns out that this simple generation has a relevant drawback. An alternative (and, according to our experiments, more robust) random generation is based on the uniform random generation between 0 and N of the *number* of binary variables whose value is equal to 0. Note that the first proposed generation gives rise to a *binomial* distribution with parameter 0.5 for the number of variables whose value is equal to 0. Such distribution tends to be more concentrated around the value $0.5N$ and, as we will see in Section 4, this might have some negative implications.

3.2. Refine procedure

For the refinement procedure we exploited the continuous component of the problem. Given some configuration Y, we fixed the value of the binary variables and then started a local search with respect to the continuous ones. In other words, recalling the meaning of the variables, we fixed the orientation of the

rectangles, allowing to move their centers. In particular, given the fact that the problem is an unconstrained one with respect to the continuous variables, we used L-BFGS (Limited Memory BFGS, see [36]) as a local search procedure.
We also point out that at the end of each run of the approach we also used the returned solution as the starting point of a sort of local search with respect to the binary variables: the orientation of each rectangle in turn is changed and a continuous local search from the resulting configuration is started; if the new solution is better than the current one, we move to the new one, otherwise we do not move from the current point; this is repeated until no improvement is obtained after having changed the orientation of all the rectangles. Note that this final search can be viewed as a further run of the proposed approach where at each iteration the perturbation changes the orientation of a single rectangle.

3.3. Perturb procedure

As usual in ILS (or MBH), the perturbation procedure has a great impact on the performance of the heuristic. The perturbation should be large enough to escape from the region of attraction of the current iterate (in order to avoid that the refinement procedure takes the search back to the current iterate itself), but at the same time not too large, in order to avoid that the perturbation is close to a completely random generation. In other words, the perturbation should be small enough in order to retain, at least partially, the structure of the current configuration, in the hope that what is retained is the "good" part of the structure, while the "bad" part is somehow "adjusted", so that a "better" configuration is obtained. Of course, it is quite relevant to have some criterion in order to decide when the newly generated configuration is indeed a "better" one. Here we employed a simple monotonic criterion, i.e., Z_k is "better" then X_{k-1} when it has a better function value. However, other, possibly non monotonic, criteria could be employed.
In defining perturbations for the problem at hand, we considered two basic moves, one related to the binary variables p_i, the other related to the continuous variables c_x^i, c_y^i. The former simply switches the value of one binary variable

p_i, i.e., it changes the orientation of rectangle i. The latter moves the center (c_x^i, c_y^i) of rectangle i into some new position. These basic moves can be combined in different ways to define perturbations. In this work we considered the following combinatorial (only involving the binary variables) and continuous (only involving the continuous variables) perturbations.

`Comb1` this is a combinatorial perturbation defined as follows: let

$$\mathcal{N}_0 = \{i \ : \ p_i^{curr} = 0\}, \quad \mathcal{N}_1 = \{i \ : \ p_i^{curr} = 1\},$$

where p_i^{curr} denotes the value of p_i in the current iterate; randomly select $i_0 \in \mathcal{N}_0$ and $i_1 \in \mathcal{N}_1$; perform the following basic moves

$$\begin{cases} p_{i_0}^{new} = 1, \ p_{i_1}^{new} = 0 & \text{with prob. } 1/3 \\ p_{i_0}^{new} = 1 & \text{with prob. } 1/3 \\ p_{i_1}^{new} = 0 & \text{with prob. } 1/3 \end{cases}$$

if $\mathcal{N}_0 = \emptyset$ ($\mathcal{N}_1 = \emptyset$), then we simply set $p_{i_1}^{new} = 0$ ($p_{i_0}^{new} = 1$). Therefore, this perturbation swaps two elements in $\mathcal{N}_0$ and $\mathcal{N}_1$ with probability $1/3$, moves an element in $\mathcal{N}_0$ into $\mathcal{N}_1$ with probability $1/3$, and moves an element in $\mathcal{N}_1$ into $\mathcal{N}_0$ with probability $1/3$.

`Comb2`(η) $\eta \in [0, 1]$: this is another combinatorial move defined as follows: for each $i = 1, \ldots, N$, set $p_i^{new} = 1 - p_i^{curr}$ with probability η, and $p_i^{new} = p_i^{curr}$ with probability $1 - \eta$. Probability η is usually chosen small in order to avoid a too large perturbation of the current configuration.

`Cont1` This is a continuous perturbation where we randomly select $i \in \{1, \ldots, N\}$ and $\alpha \in [\ell_x, u_x]$ and set

$$c_x^i = \alpha$$

(ℓ_x, u_x are, respectively, a lower and upper limit for the value of the container's first coordinate). Thus, the perturbation selects a rectangle and moves it in a new random position (completely unrelated with the current one) along the first coordinate;

`Cont2`(d), $d \geq 0$ this is another continuous perturbation defined as follows

$$\begin{aligned} c_x^{i,new} &= c_x^{i,curr} + \beta_i \\ c_y^{i,new} &= c_y^{i,curr} + \gamma_i \end{aligned}$$

where β_i, γ_i are randomly sampled within the interval $[-d, d]$. While perturbation `Cont1` involves a single rectangle which might be moved far away from its current position, perturbation `Cont2`(d) involves all rectangles but, at least for small d values (which is the recommended choice), only slightly moves them.

We remark that all the suggested perturbations follow the previously stated principle: the perturbation should not be "too large", i.e., it should not completely disrupt the structure of the current configuration. This is accomplished either by moving few rectangles (`Comb1`, `Comb2`, `Cont1`) or by moving all of them (`Cont2`) but all by a small displacement. Of course, it is possible to define further perturbations but, as we will see in Section 4, the four defined here already allow to get good results.

4. Computational experiments

For the computational experiments we considered the set of problem instances employed in [31] and [32]. The details of the problem instances are reported in Table 1. These include the convex functions describing the container (column `Container Data`, subcolumn `Constraints`), the area of the container (column `Container Data`, subcolumn `Area`), the width a and height b of the rectangles (column `Piece Data`, subcolumn $a \times b$), the area of the rectangles (column `Piece Data`, subcolumn `Area`), the best value N_{best} observed in [31] for the instance (column N_{best}).

Following [31], a variant of problem instances # 12-# 16 has also been considered, where the value of b is halved ($b = 0.5$). The resulting new five instances have been numbered from # 17 to # 21 (the N_{best} values for these instances are respectively 55, 57, 62, 64, 64).

The code used for the experiments is written in `C++` programming language, using `liblbfgs`[3] for local searches. Tests have been performed on standard desktop machines running `Linux`, equipped with `Intel(R) Pentium(R) 4 CPU 3.40GHz, 1Gbyte RAM`. The code has been compiled with `gcc 4.3.3` using standard optimization flags.

For each test problem we performed 10 random runs. In each run we went through all the N values from 1 up to N_{best}, where N_{best} denotes the best value observed in [31]. Following [32], we increased the value of N (or stopped the run if $N = N_{best}$) as soon as a time limit of 22000 seconds was reached (which, however, almost never occurred), or when the objective function value fell below 10^{-8}. In fact, as already observed in [31], most of the computational cost is due to the solution for the case $N = N_{best}$, while all problems with $N < N_{best}$ are solved quite quickly.

Local searches have been first executed with low accuracy (10^{-6}), and if the objective function value was smaller than 10^{-4} a further refinement with higher accuracy (10^{-9}) has been performed.

We performed tests with different strategies, corresponding to different combinations of the perturbation operators defined in Section 3, and different choices of the $MaxNoImp$ parameter. In the following table we report the details of the strategies for which we report complete results over all the test problems (for the continuous move `Cont2`(d) we have always set $d = 0.2 * \sqrt{a^2 + b^2}$).

Strategy	$MaxNoImp$	Perturbation
1	100	`Comb1`
2	25	`Comb1` + `Cont1`
3	100	`Comb1` + `Cont1`
4	500	`Comb1` + `Cont1`
5	500	`Comb2`(0.05) + `Cont2`(d)
6	500	`Comb2`(0.1) + `Cont2`(d)

[3] freely available at `www.chokkan.org/software/liblbfgs/`

For the remaining tested strategies we just give some comments.
The results for Strategies 1-6 are reported in Table 2 together with those reported in [31]. Being a comparison in terms of running times, we remark that the machine used in [31] (in that paper a 1.8 `GHz AMD Opteron 244` processor, with 2 Gb of RAM was used, which is a dual-core machine) as well as the implementation (in `FORTRAN77`) is comparable with those employed here. For each test instance and each strategy we report the average computation time over the 10 random runs. In Figure 1 we display the performance profile for the six strategies and for the approach presented in [31]. In the performance profile a curve is drawn for each of the tested strategies and for the approach in [31]. The curve represents the fraction of instances (over the 21 considered) for which the average computation time is within x times from the lowest average computation time among all tested strategies (i.e., at $x = 1$ we have the number of distinct instances for which a single strategy has the lowest computation time, at $x = 2$ we have the number of instances for which the average computation time is within twice the lowest average computation time, and so on).
The performance profile clearly indicates that Strategies 2-4 outperform the other ones and the approach proposed in [31] (in particular, these strategies turn out to be better, and often much better, than the approach proposed in [31] over each test instance). Strategies 2-4 all perform the same perturbation and just differ for the $MaxNoImp$ value. If we choose a small $MaxNoImp$ value we tend to perform many short runs where we often randomly restart the search, while by increasing the $MaxNoImp$ value we increase the time we are ready to spend in exploring the neighborhood of the incumbent solution, thus increasing the length of a run but decreasing the number of runs. The results in Table 2 show that there is no optimal $MaxNoImp$ value, the best choice is related to the problem to be solved. On the other hand, it also appears that the results are not strongly affected by the choice of this parameter, i.e., at least for this kind of perturbation, 'impatient' strategies, where a small time is dedicated to the exploration of the neighborhood of the incumbent solution, and 'patient' ones have comparable performance.

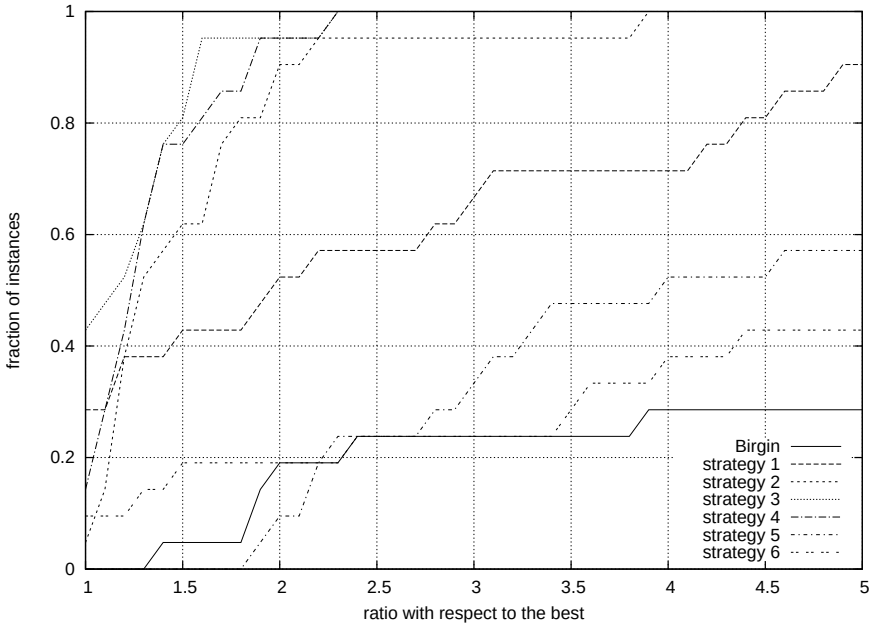


Figure 1: Performance profiles of the proposed strategies and the results in [31].

Strategy 1 performs reasonably well. It is similar to Strategy 3 but without the continuous move `Cont1` in the perturbation. On most problems this turns out to be an effective strategy (sometimes even the best one), but it has a poor performance on the most challenging (with respect to computation times) instances (instances # 3 and # 11). It seems that for such problems the combinatorial move alone is not sufficient and coupling it with the continuous move `Cont1` can really make a difference. We remark here that we also performed some tests with the continuous move `Cont1` alone but in this case the results are much worse, probably because this move alone does not guarantee a large enough perturbation.

Strategies 5-6 display some good results but overall they perform clearly worse than Strategies 1-4. Although we do not display complete results, more tests have been performed with $MaxNoImp = 100$ and others with no continuous move. Mixed results (sometimes better, sometimes worse than those for Strategies 5-6) have been obtained. Extremely bad results (also with some failures

to reach the solution) have been obtained when we tested the continuous move alone. The overall impression is that the perturbation employed in these strategies are not as effective as those employed for Strategies 1-4. However, we report also these strategies because they offer good results on instances # 11, which is one of the two most challenging ones. These results get even more significative if we complete them with a series of tests in which instance # 11 has been tackled with $MaxNoImp = 500$ and perturbation where the continuous move `Cont2`(d) is combined with the combinatorial move `Comb2`(η) for different values of η (the average cumulative computational times are reported in Table 3).

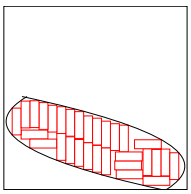
It is interesting to note that the results improve for η up to 0.15 (for which the results are also considerably better than those for Strategies 2-4), and then get worse for larger η values. Increasing η corresponds to an enlargement of the neighborhood explored at each iteration. Therefore, these results offer a significative example of the impact on the performance of the 'size' of the perturbation.

Another experiment that we performed is related to the generation of the initial solution. In all the above experiments the initial solution has been generated by randomly sampling the *number* of rectangles in horizontal position (i.e., the number of p_i variables whose value is equal to 0). As already commented in Section 3.1, an alternative is that of setting to 0 each binary variable p_i with probability 0.5, and to 1 otherwise. On some test problems such alternative performs quite badly. In particular for instance # 6 we ran Strategy 3 with this alternative initial generation and we got clearly worse computation times (312.779 seconds on average). For these instances it seems that the initial number of horizontal rectangles is crucial for the detection of the solution. To confirm this, we performed an experiment where the initial solution has been generated with a fixed number num_horiz of horizontal rectangles. The results are reported in the following table (in the last row we report the result for the alternative initial generation mentioned above).

num_horiz	Avg.Comp.Time
0	9.499
3	9.6
6	39.321
9	81.254
12	446.535
15	1320.626
18	3704.692
21	4117.019
24	1565.815
27	1255.423
30	2198.947
50%	312.779

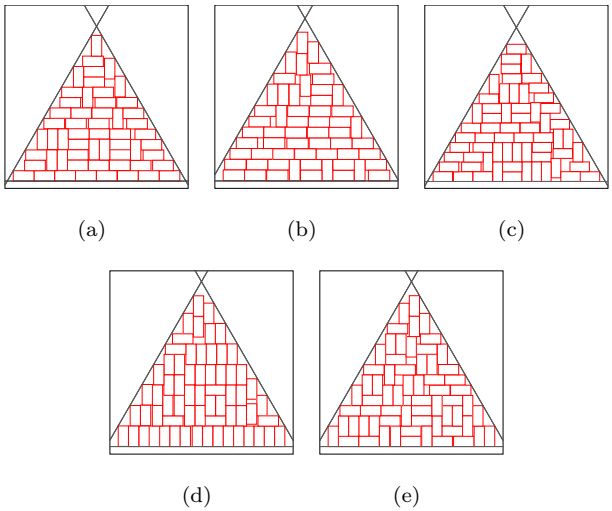
It seems that there is a small range of num_horiz values for which the algorithm performs quite well (the values close to the number of horizontal rectangles in an optimal solution, which is low as it can be seen from Figure 2), but for the other values the performance is quite bad (including values around $N_{best}/2$, which are the most likely ones for the alternative initial generation). Probably, when starting with a 'bad' initial value for num_horiz, the perturbations are not large enough to drive the number of horizontal rectangles towards that in an optimal solution of the problem.

For some other instances this is a less serious issue. E.g., in optimal solutions for instances # 17-# 21, there are many pairs of rectangles forming a square with edge length 1 (see Figure 3). If we change the orientation of both these rectangles, an alternative optimal solution is obtained, so that the range of num_horiz values for which an optimal solution with that number of horizontal rectangles exists, is quite large.



(a)

Figure 2: Optimal configuration for instance # 6.



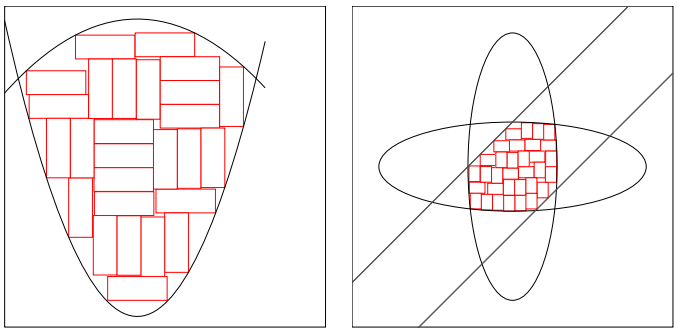
(a) (b) (c)

(d) (e)

Figure 3: Optimal configurations for instances #17-#21.

4.1. Improved solutions

For all the test instances we also checked whether the best values N_{best} reported in [31] could be improved. We ran Strategy 3 through all the instances with $N = N_{best}+1$. By this experiment we were able to detect quite rapidly (average computation time : 230.106 sec.) a new best solution with $N = 33$ for instance # 8, and with much more difficulties (average computation time: 31023 sec. per success and 5 failures, i.e., time limit reached, over 10 runs) a new best solution with $N = 27$ for instance # 4. In both cases we reached a global optimal solution for $N = N_{best} + 1$ with minimum value exactly equal to 0. The new solutions are displayed in Figure 4.



(a) Solution for instance # 4 with 27 items.

(b) Solution for instance # 8 with 33 items.

Figure 4: New optimal configurations found.

5. Conclusion

In this paper we tackled the problem of packing equal rectangles within a convex region. Following [31] the problem can be reduced to the solution of mixed integer global optimization problems. We proposed a heuristic approach to solve such problems. The approach is an Iterated Local Search (or Monotonic Basin Hopping) one. The components of the heuristic have been defined and for the main one, the perturbation operation, different options, based on continuous and combinatorial moves, have been proposed. Many variants of the approach have been compared through extensive computational experiments on a set of test instances. We point out here that improvements over the obtained results are certainly possible, e.g., by defining new perturbation moves, which could be an interesting subject for future research. All the same the current results are quite encouraging and, in our opinion, show that the approach is particularly suitable for these packing problems.

References

[1] I. Castillo, F. J. Kampas, J. D. Pinter, Solving circle packing problems by global optimization: Numerical results and industrial applications, European Journal of Operational Research 191 (2008) 786–802.

[2] G. Wäscher, H. Haußner, H. Schumann, An improved typology of cutting and packing problems, European Journal of Operational Research 183 (3) (2007) 1109–1130.

[3] B. Addis, M. Locatelli, F. Schoen, Disk packing in a square: A new global optimization approach, INFORMS J. on Computing 20 (4) (2008) 516–524.

[4] D. V. Boll, J. Donovan, R. L. Graham, B. D. Lubachevsky, Improving dense packings of equal disks in a square, The Electronic Journal of Combinatorics 7 (R46) (2000) 1–9.

[5] L. G. Casado, I. Garcia, P. Szabö, T. Csendes, Equal circles packing in square II: new results for up to 100 circles using the TAMSASS-PECS algorithm, in: F. Giannessi, P. M. Pardalos, T. Rapcsak (Eds.), Optimization Theory: Recent developements from Mátraháza, Kluwer Academic Publishers, 1998, pp. 207–224.

[6] R. L. Graham, B. D. Lubachevsky, Repeated patterns of dense packings of equal disks in a square, The Electronic Journal of Combinatorics 3 (1996) 1–16.

[7] K. J. Nurmela, P. R. J. Oestergard, Packing up to 50 equal circles in a square, Discrete Computational Geometry 18 (1997) 111–120.

[8] C. de Groot, R. Peikert, D. Würtz, M. Monagan, Packing circles in a square: a review and new results, in: System Modelling and Optimization, Proc. 15th IFIP Conf., Zürich, 1991, pp. 45–54.

[9] M. Locatelli, U. Raber, Packing equal circles in a square: a deterministic global optimization approach, Discrete Applied Mathematics 122 (2002) 139–166.

[10] M. C. Markót, T. Csendes, A new verified optimization technique for the "packing circles in a unit square" problem, SIAM J. on Optimization 16 (2005) 193–219.

[11] K. J. Nurmela, P. R. J. Oestergard, More Optimal Packings of Equal Circles in a Square, Discrete Computational Geometry 22 (1999) 439–457.

[12] P. G. Szabó, M. C. Markót, T. Csendes, Global optimization in geometry - circle packing into the square, in: C. Audet, P. Hansen, G. Savard (Eds.), Essays and Surveys in Global Optimization, Kluwer, 2005, pp. 233–266.

[13] P. G. Szabó, M. C. Markót, T. Csendes, E. Specht, L. G. Casado, I. Garcia, New Approaches to Circle Packing in a Square With Program Codes, Springer, 2007.

[14] B. Addis, M. Locatelli, F. Schoen, Efficiently packing unequal disks in a circle, Operations Research Letters 36 (1) (2008) 37–42.

[15] E. Birgin, F. Sobral, Minimizing the object dimensions in circle and sphere packing problems, Computers and Operations Research 35 (7) (2008) 2357–2375.

[16] A. Grosso, A. Jamali, M. Locatelli, F. Schoen, Solving the problem of packing equal and unequal circles in a circular container, Journal of Global Optimization 47 (1) (2010) 63–81.

[17] M. Hifi, R. M'Hallah, Adaptive and restarting techniques-based algorithms for circular packing problems, Computational Optimization and Applications 39 (1) (2008) 17–35.

[18] H. Huang, W. Huang, Q. Zhang, D. Xu, An improved algorithm for the packing of unequal circles within a larger containing circle, European Journal of Operational Research 141 (2002) 440–453.

[19] W. Huang, Y. Li, B. Jurkowiak, C. M. Li, R. C. Xu, A two-level search strategy for packing unequal circles into a circular container, in: Proceedings of the International Conference on Principles and Practice of Constraint Programming, Springer-Verlag, 2003, pp. 868–872.

[20] A. Müller, J. Schnedier, E. Schömer, Packing a multidisperse system of hard discs in a circular environment, Physical Review E 79 (2009) 021102.

[21] J. D. Pinter, F. J. Kampas, Nonlinear optimization in Mathematica using MathOptimizer Professional, Mathematica in Education and Research 10 (2005) 1–18.

[22] Y. Stoyan, G. Yaskov, A mathematical model and a solution method for the problem of placing various-sized circles into a strip, European Journal of Operational Research 156 (3) (2004) 590–600.

[23] Wang, H. and Huang, W. and Zhang, Q. and Xu, D., An improved algorithm for the packing of unequal circles within a larger containing circle, EJOR 141 (2002) 440–453.

[24] D. Zhang, A. Deng, An effective hybrid algorithm for the problem of packing circles into a larger containing circle, Computers & Operations Research 32 (2005) 1941–1951.

[25] Y. G. Stoyan, G. N. Yaskow, Mathematical model and solution method of optimization problem of placement of rectangles and circles taking into account special constraints, International Transactions in Operational Research 5 (1998) 45–57.

[26] J. Kallrath, Cutting circles and polygons from area-minimizing rectangles, Journal of Global Optimization 43 (2-3) (2009) 1–30.

[27] J. A. Bennell, J. F. Oliveira, The geometry of nesting problems: A tutorial, European Journal of Operational Research 184 (2) (2008) 397 – 415.

[28] T. Imamichi, M. Yagiura, H. Nagamochi, An iterated local search algorithm based on nonlinear programming for the irregular strip packing problem, Discrete Optimization 6 (4) (2009) 345–361.

[29] A. M. Gomes, J. F. Oliveira, Solving irregular strip packing problems by hybridising simulated annealing and linear programming, European Journal of Operational Research 171 (2006) 811–829.

[30] E. Birgin, J. Martínez, W. Mascarenhas, D. Ronconi, Method of sentinels for packing items within arbitrary convex regions, Journal of the Operational Research Society 57 (6) (2006) 735–746.

[31] E. Birgin, J. Martínez, F. Nishihara, D. Ronconi, Orthogonal packing of rectangular items within arbitrary convex regions by nonlinear optimization, Computers & Operations Research 33 (12) (2006) 3535–3548.

[32] E. Birgin, R. Lobato, Orthogonal packing of rectangles within isotropic convex regions.
URL {http://www.ime.usp.br/~egbirgin/publications/bl.pdf}

[33] H. R. Lourenço, O. C. Martin, T. Stülze, Iterated local search, in: F. W. Glover, G. A. Kochenberger (Eds.), Handbook of Metaheuristics, Kluwer Academic Publishers, Boston, Dordrecht, London, 2003, pp. 321–353.

[34] R. H. Leary, Global optimization on funneling landscapes, Journal of Global Optimization 18 (2000) 367–383.

[35] D. J. Wales, J. P. K. Doye, Global optimization by basin-hopping and the lowest energy structures of Lennard-Jones clusters containing up to 110 atoms, Journal of Physical Chemistry A 101 (1997) 5111–5116.

[36] D. Liu, J. Nocedal, On the limited memory bfgs method for large scale optimization, Mathematical Programming B 45 (1989) 503–528.

	Container Data		Piece Data		
#	Constraints	Area	$a \times b$	Area	N_{best}
1	$g_1(x_1, x_2) = -x_1$ $g_2(x_1, x_2) = -x_2$ $g_3(x_1, x_2) = -x_1 - x_2 + 3$ $g_4(x_1, x_2) = x_1^2 + x_2^2 - 100$	74.1	2×1	2	32
2	$g_1(x_1, x_2) = -7x_1 + 6x_2 - 24$ $g_2(x_1, x_2) = 7x_1 + 6x_2 - 108$ $g_3(x_1, x_2) = (x_1 - 6)^2 + (x_2 - 8)^2 - 9$	21.7	1.1×0.65	0.61	28
3	$g_1(x_1, x_2) = -x_1$ $g_2(x_1, x_2) = x_1 - 8$ $g_3(x_1, x_2) = (x_1 - 6)^2 - x_2^2 - 81$ $g_4(x_1, x_2) = (x_1 - 1.7)^2 + (x_2 - 10)^2 - 81$	54.4	2×0.6	1.2	40
4	$g_1(x_1, x_2) = x_1^2 - x_2$ $g_2(x_1, x_2) = x_1^2/4 + x_2 - 5$	13.3	1×0.4	0.4	26
5	$g_1(x_1, x_2) = x_1^2 - x_2$ $g_2(x_1, x_2) = -x_1 + x_2^2 - 6x_2 + 6$ $g_3(x_1, x_2) = x_1 + x_2 - 6$	10.9	0.9×0.3	0.27	33
6	$g_1(x_1, x_2) = -x_1 + x_2^2 - 6x_2 + 6$ $g_2(x_1, x_2) = x_1 + x_2^2 - 3x_2 - 3/4$	10.2	0.9×0.3	0.27	30
7	$g_1(x_1, x_2) = (x_1 - 2)^2/4 + (x_2 - 4)^2/16 - 1$	25.1	2×0.5	1.00	19
8	$g_1(x_1, x_2) = (x_1 - 6)^2/4 + (x_2 - 6)^2/36 - 1$ $g_2(x_1, x_2) = (x_1 - 6)^2/36 + (x_2 - 6)^2/4 - 1$ $g_3(x_1, x_2) = x_1 - x_2 - 3$ $g_4(x_1, x_2) = -x_1 + x_2 - 2$	13.2	0.7×0.5	0.35	32
9	$g_1(x_1, x_2) = (x_1 - 3)^2/4 + (x_2 - 4)^2/16 - 1$ $g_2(x_1, x_2) = (x_1 - 2.65)^2/4 + (x_2 - 4)^2/16 - 1$ $g_3(x_1, x_2) = -x_1 + 1$ $g_4(x_1, x_2) = x_1 - x_2 - 1$ $g_5(x_1, x_2) = x_1 + x_2 - 9$	13.7	0.8×0.6	0.48	22
10	$g_1(x_1, x_2) = (x_1 - 6)^2/36 + (x_2 - 6)^2/4 - 1$ $g_2(x_1, x_2) = (x_1 - 6)^2/9 + (x_2 - 8)^2/9 - 1$	13.6	0.95×0.35	0.33	34
11	$g_1(x_1, x_2) = (x_1/6)^4 + (x_2/2)^4 - 1$ $g_2(x_1, x_2) = 8x_1 - 11x_2 - 26$	34.7	1.9×0.5	0.95	31
12	$g_1(x_1, x_2) = \sqrt{3}x_1 + x_2 - \sqrt{3}(3/2 + \sqrt{3})$ $g_2(x_1, x_2) = -\sqrt{3}x_1 + x_2$ $g_3(x_1, x_2) = -x_2$	32.3	1.0×1.0	1.0	25
13	$g_1(x_1, x_2) = \sqrt{3}x_1 + x_2 - \sqrt{3}(2 + 4/\sqrt{3})$ $g_2(x_1, x_2) = -\sqrt{3}x_1 + x_2$ $g_3(x_1, x_2) = -x_2$	33.3	1.0×1.0	1.0	26
14	$g_1(x_1, x_2) = \sqrt{3}x_1 + x_2 - \sqrt{3}(3 + 4/\sqrt{3})$ $g_2(x_1, x_2) = -\sqrt{3}x_1 + x_2$ $g_3(x_1, x_2) = -x_2$	36.3	1.0×1.0	1.0	29
15	$g_1(x_1, x_2) = \sqrt{3}x_1 + x_2 - \sqrt{3}(2 + 2\sqrt{3})$ $g_2(x_1, x_2) = -\sqrt{3}x_1 + x_2$ $g_3(x_1, x_2) = -x_2$	37.5	1.0×1.0	1.0	29
16	$g_1(x_1, x_2) = \sqrt{3}x_1 + x_2 - \sqrt{3}(4 + 4\sqrt{3})$ $g_2(x_1, x_2) = -\sqrt{3}x_1 + x_2$ $g_3(x_1, x_2) = -x_2$	37.5	1.0×1.0	1.0	30

Table 1: Test problems

Test	[31]	Str.1	Str.2	Str.3	Str.4	Str. 5	Str.6
1	6453.80	366.206	450.518	392.754	679.709	1435.169	535.969
2	31.94	20.976	18.245	17.642	23.34	38.254	102.628
3	19508.71	6515.556	3828.85	2676.894	1939.775	4171.407	7584.423
4	485.29	8.921	7.041	2.931	1.847	16.432	12.227
5	329.65	32.99	20.139	17.688	12.105	40.3	53.129
6	780.77	103.719	93.508	52.722	95.259	543.054	652.275
7	157.71	3.24	3.823	3.168	3.327	5.323	2.705
8	880.40	4.894	7.886	6.183	6.276	15.133	16.972
9	108.11	5.414	2.934	5.715	3.345	8.094	2.495
10	5554.20	48.969	58.381	26.601	60.766	50.415	61.889
11	4199.44	10965.528	2131.46	2879.064	2398.138	6259.294	2666.744
12	1.13	1.329	0.347	0.293	0.345	8.467	8.464
13	1.21	1.912	0.724	0.657	0.757	5.699	5.651
14	0.35	0.627	0.184	0.152	0.186	12.333	12.445
15	0.34	1.429	0.268	0.25	0.275	3.063	3.084
16	1.19	0.872	0.232	0.2	0.257	20.318	20.477
17	2812.07	171.818	201.619	212.127	270.67	2044.37	6092.715
18	13575.78	144.806	192.872	224.046	233.947	873.257	1702.235
19	640.35	52.384	66.599	72.15	55.392	236.278	389.718
20	14811.55	139.276	187.932	94.423	101.573	260.315	713.423
21	1126.21	60.585	101.246	93.277	78.946	138.99	212.7

Table 2: Complete results over the 21 test instances for the approach presented in [31] and Strategies 1-6.

η	0.05	0.1	0.15	0.2	0.25
Avg.Comp.Time	6259.294	2666.744	996.425	5617.58	5567.151

Table 3: Average computation times over instance # 11 for strategies with $MaxNoImp = 500$ and perturbation where the continuous move `Cont2`(d) is combined with the combinatorial move `Comb2`(η).