Packing Curved Objects

Ignacio Salas[1] and Gilles Chabert[1]

Mines de Nantes - LINA (UMR 6241)
{ignacio.salas,gilles.chabert}@mines-nantes.com

Abstract. This paper deals with the problem of packing two-dimensional objects of quite arbitrary shapes including in particular curved shapes (like ellipses) and assemblies of them. This problem arises in industry for the packaging and transport of bulky objects which are not individually packed into boxes, like car spare parts. There has been considerable work on packing curved objects but, most of the time, with specific shapes; one famous example being the circle packing problem. A successful approach has been proposed recently in [MVFA13] and the algorithm we propose here is an extension of their work. Martinez et al. use a stochastic optimization algorithm with a fitness function that gives a violation cost and equals zero when objects are all packed. Their main idea is to define this function as a sum of $\binom{n}{2}$ elementary functions that measure the overlapping between each pair of different objects. However, these functions are ad-hoc formulas. The aim of this paper is to generalize the approach by replacing the ad-hoc formulas with a numerical algorithm that automatically measures the overlapping between two objects. Then, we come up with a fully black-box packing algorithm that accept any kind of objects.

1 Introduction

The packing problem consists in placing n items inside a given space, such that no two items overlap. A large number of real-world applications like warehousing, logistics or parallel computing gives a great relevance to this problem. Packing has been well studied in either academic or industrial contexts, but each time considering specific shapes like bins, circles or polytopes. Our goal is to deal with the more general situation where different shapes can be mixed, including non-convex and curved shapes. One generic approach for solving the packing problem has been proposed in [MVFA13]. The approach splits the problem in 3 parts. First, they build an *overlapping function* f_{ij} for each pair (i, j) of objects. Second, all these violations are summed up to form a global violation cost f. Third, the function f is minimized using a generic black-box optimization software called CMA-ES [HO01]. The experimental results they obtain are very encouraging. However, the overlapping functions are ad-hoc formulas for every possible combination of object shapes. Some formulas are easy to obtain but not in all the cases, and mixing shapes considerably complicate the task. The paper is organized as follows. We first formalize the problem in §2. Then, we detail in §3 and §4 how to replace these ad-hoc formulas by an algorithm. Experimental results are reported in §5, followed by concluding remarks.

2 Formalism

Object definition. We consider objects described by nonlinear inequalities. We also accept the logical and/or operators in the shape description, which means

that geometric primitives can be combined to define more complex shapes. Then, the shape of Object i can be seen as a regular constraint $c_i(\boldsymbol{p})$.

The non-overlapping constraint. If Object i is placed at a position $\boldsymbol{o_i}$ and rotated by an angle α_i, we will denote by $\boldsymbol{q}_i$ and call the *parameters* of Object i the vector $\boldsymbol{q}_i := (\boldsymbol{o}_i, \alpha_i) = (x_i, y_i, \alpha_i)$. The size of $\boldsymbol{q}$ is the number of degrees of freedom. If only translation is considered, the angle is dropped and $\boldsymbol{q} \in \mathbb{R}^2$. Otherwise, it belongs to $\mathbb{R}^3$. Given a vector of parameters $\boldsymbol{q}_i$, the points $\boldsymbol{p} \in$ Object i are the ones satisfying

$$c_i(R_{-\alpha_i}(\boldsymbol{p} - \boldsymbol{o_i})), \tag{1}$$

where R is the usual rotation matrix. Therefore, Objects i and j overlap iff

$$\text{overlap}_{ij}(\boldsymbol{q_i}, \boldsymbol{q_j}) \iff \exists \boldsymbol{p} \in \mathbb{R}^2, c_i(R_{-\alpha_i}(\boldsymbol{p} - \boldsymbol{o_i})) \wedge c_j(R_{-\alpha_j}(\boldsymbol{p} - \boldsymbol{o_j})). \tag{2}$$

The non-overlapping constraint between two objects is just the negation of the latter relation.

The packing problem. The packing problem is a set of pairwise non-overlapping constraints between n objects and an additional constraint that all objects must be packed inside some *container*. The object is inside the container iff this object does not overlap the complementary of the container. And the complementary of a shape is simply obtained with the negation of the underlying constraint. Hence, the packing problem consisting in placing n objects of shapes $c_1, \ldots, c_n$ inside a container c is reformulated as a non-overlapping constraint between $n+1$ shapes $c_1, \ldots, c_n, \neg c$. Where the position of the container is fixed.

The overlapping function. The overlapping function $f_{ij}(\boldsymbol{q_i}, \boldsymbol{q}_j)$ gives a measure of "how much" Objects i and j overlap. The output must fulfill 2 properties. First, it has to be 0 iff both objects are disjoint. Second, it has to decrease when both objects gets more distant.

Definition 1 (Overlapping Function).

$$f_{ij}(\boldsymbol{q_i}, \boldsymbol{q_j}) = \min\{\|\boldsymbol{q_j}' - \boldsymbol{q_j}\|,\ \neg \text{overlap}_{ij}(\boldsymbol{q_i}, \boldsymbol{q_j}')\}.$$

This function exactly matches the concept of *penetration depth* [CC86]. Object i is called the *reference* object and Object j the *moving* object. Without rotation, the roles are symmetric in the sense that: $f_{ij}(\boldsymbol{q_i}, \boldsymbol{q_j}) = f_{ji}(\boldsymbol{q_j}, \boldsymbol{q_i})$. However, the previous equality does not hold with rotation.

The overlapping region. The implementation we will propose for the overlapping function is based on the concept of *overlapping region* [SCG14]. The overlapping region simply corresponds to the set of parameters for Object j that make the objects i and j overlap, the parameters of Object i being all set to 0:

Definition 2 (Overlapping Region). *Given Objects i and j, the overlapping region is* $\mathcal{S}_{ij} := \{\boldsymbol{q_j} \in \mathbb{R}^d \mid \text{overlap}_{ij}(\boldsymbol{0}_{\mathbb{R}^d}, \boldsymbol{q_j})\}$.

This concept is depicted in Figure 1. We show now the relation that connects the overlapping function to the overlapping region.

Proposition 1. *Given two vectors of parameters $\boldsymbol{q_i}$ and $\boldsymbol{q_j}$, let*

$$\boldsymbol{r}(\boldsymbol{q_i}, \boldsymbol{q_j}) := (R_{-\alpha_i}(\boldsymbol{o_j} - \boldsymbol{o_i}), \alpha_j - \alpha_i). \tag{3}$$

Then $f(\boldsymbol{q_i}, \boldsymbol{q_j}) = \min\{\|\boldsymbol{q''} - \boldsymbol{r}(\boldsymbol{q_i}, \boldsymbol{q_j})\|,\ \boldsymbol{q''} \notin \mathcal{S}_{ij}\}$.

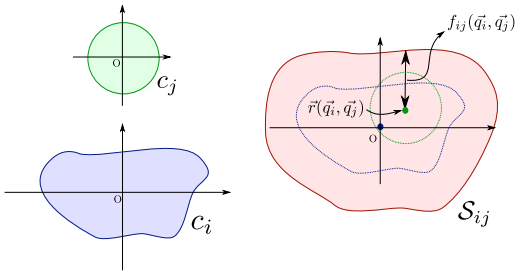


Fig. 1: **Overlapping region.** For visibility, the dimension here is 2 (without rotation). **(left)** Objects i (in blue) and j (in green). **(right)** The overlapping region $\mathcal{S}_{ij}$ (in red). Represents all the positions where Object j overlaps Object i, placed at the origin. The distance between the point $\boldsymbol{r}(\boldsymbol{q_i}, \boldsymbol{q_j})$ and the boundary of $\mathcal{S}_{ij}$ is the value of the overlapping function f_{ij} (Proposition 1).

3 Main Algorithm

The algorithm we propose for the overlapping function is a direct consequence of Proposition 1. We first calculate (off-line) the overlapping region $\mathcal{S}_{ij}$ and then obtain (on-line) the value of $f(\boldsymbol{q}_i, \boldsymbol{q}_j)$ from it.

Paving of the overlapping region. The data structure we use for $\mathcal{S}_{ij}$ is called a *paving* [CJ09]. A paving of a set $\mathcal{S} \subset \mathbb{R}^d$ is a triplet $(\mathcal{I}, \mathcal{B}, \mathcal{O})$ where $\mathcal{I}$ (for "inside"), $\mathcal{O}$ (for "outside") and $\mathcal{B}$ (for "boundary") are three sets of boxes verifying $\cup\mathcal{I} \subset \mathcal{S}, \quad (\cup\mathcal{O}) \cap \mathcal{S} = \emptyset \quad$ and $\quad \cup(\mathcal{B} \cup \mathcal{I} \cup \mathcal{O}) = \mathbb{R}^d$. To calculate this paving, we apply the strategy proposed in [SCG14]. It ensures that the total surface of the boundary is less than a given value ε. The *paver* is a recursive algorithm that starts with an arbitrarily large box $[q_j]$ and alternates three steps:

(outer rejection test). If unsatisfiability of (2) is proven for all $\boldsymbol{q_j} \in [q_j]$ ($\boldsymbol{q_i}$ being fixed), then $\mathcal{O} := \mathcal{O} \cup \{[q_j]\}$. The rejection test resorts to an embedded solver that we shall not describe further, since this part has not changed from the paper cited above.

(inner inflation). Pick randomly a point $\tilde{q}_j \in [q_j]$ and check if $\tilde{q}_j \in \mathcal{S}_{ij}$. If it does, then "inflate" $\tilde{q}_j$ to a subbox $[\tilde{q}_j] \subset \mathcal{S}_{ij}$. This part will be the topic of Section 4. Then, $\mathcal{I} := \mathcal{I} \cup \{[\tilde{q}_j]\}$. Break the remaining part $[q_j] \backslash [\tilde{q}_j]$ into boxes and perform a recursive call with each box.

If $[q_j]$ is small enough, $\mathcal{B} := \mathcal{B} \cup \{[q_j]\}$. Otherwise bisect $[q_j]$ into two sub-boxes and perform a recursive call with each box.

Distance to the boundary set. Once the paving $(\mathcal{I}, \mathcal{B}, \mathcal{O})$ is calculated, we first set $\boldsymbol{r} := \boldsymbol{r}(\boldsymbol{q_i}, \boldsymbol{q_j})$ using (3) and then find the closest point to $\boldsymbol{r} \notin \mathcal{S}_{ij}$. However, because the boundary of the region is uncertain, we can actually only obtain an enclosure of this minimal distance:

$$\min_{\boldsymbol{q''} \in \cup\mathcal{B}} \|\boldsymbol{q''} - \boldsymbol{r}\| \leq f_{ij}(\boldsymbol{q_i}, \boldsymbol{q_j}) \leq \min_{\boldsymbol{q''} \in \cup\mathcal{O}} \|\boldsymbol{q''} - \boldsymbol{r}\|.$$

Clearly, the accuracy of the enclosure is related to the precision ε the paving has been generated with. In practice, to calculate a bound, say the upper one, we consider each box $[q''] \subset \mathcal{O}$ generated by the paver, and minimize the distance over that box. That is, we calculate:

$$\text{mindist}(\boldsymbol{r}, [q'']) := \min_{\boldsymbol{q''} \in [q'']} \|\boldsymbol{q''} - \boldsymbol{r}\|. \tag{4}$$

Then, we have:

$$\min_{\boldsymbol{q''}\in\cup\mathcal{O}} \|\boldsymbol{q''}-\boldsymbol{r}\| = \min_{[q'']\in\mathcal{O}} \text{mindist}(\boldsymbol{r},[q'']). \quad (5)$$

The same holds for the lower bound. Formula (4) is nothing different than the distance between a box and a point. This distance can be easily obtained in constant time, either using interval arithmetics or by a direct geometric argument. Formula (5) can however be very time-consuming if the boxes in $\mathcal{O}$ are stored in a flat unsorted list. One has to systematically scan the whole list to get the minimal distance. This linear complexity can be easily transformed to a logarithmic complexity using a tree-structure representation. The structure is then explored using standard global optimization techniques. In particular, pending nodes are stored according the distance to $\boldsymbol{r}$. Nodes with a distance exceeding the current upper bound for the minimum can then be discarded.

4 A new inflator for the overlapping constraint

The inflation plays a key role in the paving algorithm. We describe in this section an original inflation technique that improves packing performances. Recall that the inflator builds, from a single vector of parameters $\tilde{q_j}$, a box $[q_j] \subset \mathcal{S}_{ij}$. In [SCG14], the inflator proposed inflates in all the dimensions of $\boldsymbol{q}$ simultaneously and turns out to be not efficient when the dimension is 3 (i.e., with rotation handled). In fact, this inflator is very sensitive to the number of times a variable appears in Equation (1), and the angle α has precisely many occurrences yield by the rotation matrix $R_{-\alpha}$. Our new inflator does not suffer from the multiple symbol occurrences introduced by rotation matrices. It splits the inflation in two steps: it first inflates with respect to $\boldsymbol{o}$ and then with respect to α.

Translation. The inflation w.r.t. $\boldsymbol{o}$ is based on the following principle (see Figure 2).[1] Given initial parameters $(\tilde{o_j},\tilde{\alpha_j})$ we first look for a vector $\tilde{p}$ that belongs to both objects, the parameters of Objects i and j being respectively $\mathbf{0}$ and $\tilde{q_j}$, that is, $\tilde{p}$ satisfies $c_i(\tilde{p}) \wedge c_j(R_{-\tilde{\alpha_j}}(\tilde{p}-\tilde{o_j}))$. Then, we inflate $\tilde{p}$ inside Object i using generic inflation technique for inequalities [CB10]. Let $u := \tilde{p}-\tilde{o_j}$. The resulting box $[\boldsymbol{p}]$ can be translated to $\tilde{o_j}$ in the sense that if o_j moves inside $[\boldsymbol{o_j}] := [\boldsymbol{p}] - \boldsymbol{u}$ then $\boldsymbol{o_j}+\boldsymbol{u}$ is both inside $[\boldsymbol{p}]$ and inside the Object j.

Rotation. The inflation for the angle works as follows (see Figure 3). Given initial parameters $(\tilde{o_j},\tilde{\alpha_j})$ we look for the two angles $\underline{\alpha}$ and $\overline{\alpha}$ that force vector $\tilde{p}$ (the same as before) to meet the boundary of Object j. Then, it is clear that for every angle $\alpha_j \in [\underline{\alpha},\overline{\alpha}]$ there is an overlapping. However, some caution is required. First, there are generally more than 2 candidate angles (see Figure 3.d), especially if Object j is non-convex. The angle values that form the narrowest interval around $\tilde{\alpha_j}$ are kept. Second, there may be no angle at all. This means that Object j can make a complete turn while containing $\tilde{p}$. Third, we have to take into account the 2π-periodicity. Indeed, the search space for angles is a bounded domain like $[-\pi,\pi]$ or $[0,2\pi]$. This has the unpleasant consequence to make the inflated domain disconnected.

Shape Boundary. Let us explain now how the angles $\underline{\alpha}$ and $\overline{\alpha}$ are calculated. First, we have to generate a constraint c'_j for the boundary of Object j. In the

[1] This part is similar to the old inflator applied in the 2D case.

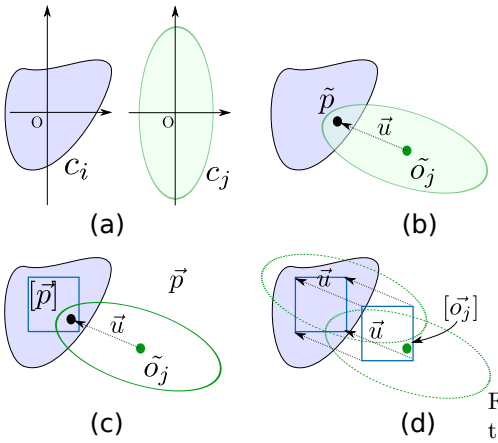


Fig. 2: **Inflation w.r.t. translation. (a)** The shapes of Objects i (in blue) and j (in green). **(b)** A point $\tilde{p}$ at the intersection of the two objects when Object j has parameters $(\boldsymbol{o_j}, \tilde{\alpha_j})$. **(c)** Inflation of $\tilde{p}$ inside Object i. **(d)** Inflation of $\boldsymbol{o_j}$.

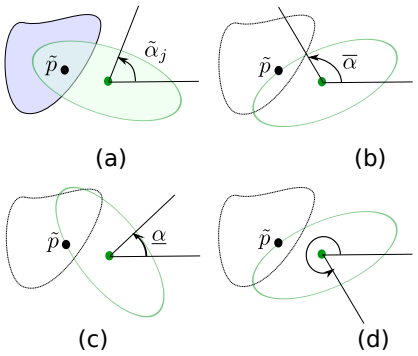


Fig. 3: **Angle inflation. (a)** Point $\tilde{p}$ obtained with initial parameters (translation step). **(b)** The smallest angle greater than $\tilde{\alpha_j}$ that makes $\tilde{p}$ meet the boundary of object j. **(c)** The greatest angle lower than $\tilde{\alpha_j}$ with the same property. **(d)** Another angle with the same property (there are 4) that is discarded.

simplest case of a single inequality, $c_j(\boldsymbol{p}) \iff f(\boldsymbol{p}) \leq 0$, the boundary of the object is described by the equality $f(\boldsymbol{p}) = 0$. In the case of a conjunction of inequalities, $c_j(\boldsymbol{p}) \iff (f_1(\boldsymbol{p}) \leq 0 \wedge \ldots \wedge f_n(\boldsymbol{p}) \leq 0)$. The boundary is described by a disjunction of n systems: $c'_j(\boldsymbol{p}) \iff (f_1(\boldsymbol{p}) = 0 \wedge f_2(\boldsymbol{p}) \leq 0 \ldots \wedge f_n(\boldsymbol{p}) \leq 0) \vee \ldots \vee (f_1(\boldsymbol{p}) \leq 0 \wedge \ldots \wedge f_{n-1}(\boldsymbol{p}) \leq 0 \wedge f_n(\boldsymbol{p}) = 0)$. The angle $\overline{\alpha}$ and $\underline{\alpha}$ are then 2 solutions of the system $c'_j(R_{-\alpha}(\tilde{p} - \tilde{o_j}))$ which is a disjunction of 1-dimensional equations (of variable α). If the shape is a disjunction with 2 subparts that plainly intersect, the boundary is only a subset of the generated system, then the angle inflation may not be optimal.

Global Inflation. We finally show that the cartesian product of $[\boldsymbol{o_j}]$ and $[\alpha_j] = [\underline{\alpha}, \overline{\alpha}]$ is a valid inflation of $\tilde{q_j}$. The inflation w.r.t. translation produces a box $[\boldsymbol{o_j}]$ that satisfies $\forall o_j \in [o_j], \quad c_i(o_j + (\tilde{p} - \tilde{o_j}))$. The inflation w.r.t. rotation produces a box $[\alpha_j]$ that satisfies $\forall \alpha_j \in [\alpha_j], \quad c_j(R_{-\alpha_j}(\tilde{p} - \tilde{o_j}))$. Substituting $\boldsymbol{p}$ to $(o_j + \tilde{p} - \tilde{o_j})$, we obtain: $\forall o_j \in [o_j],\ \forall \alpha_j \in [\alpha_j],\ \exists \boldsymbol{p},\ c_i(\boldsymbol{p}) \wedge c_j(R_{-\alpha_j}(\boldsymbol{p} - \boldsymbol{o_j}))$, that is, $\forall o_j \in [o_j],\ \forall \alpha_j \in [\alpha_j],\ (\boldsymbol{o_j}, \alpha_j) \in \mathcal{S}_{ij}$.

5 Experimental Results

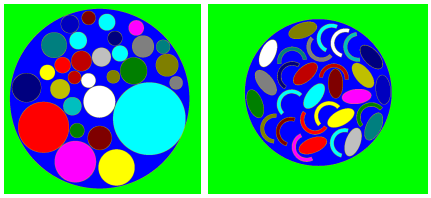
The goal of these preliminary experiments is to validate the theory and to have a first impression of how the algorithm behaves in practice.

Setup. We have created 5 cases of increasing difficulty. In each case, n objects have to be placed with $n = 10$, 20 and 30. The size of the container has been adjusted manually so that the density of the resulting packing looks similar.

Case 1. Circle packing problem. The radius of the i^{th} circle is equal to $\sqrt{i}$, $1 \leq i \leq n$. The radius of the enclosing circle is set to $r_{n=10} = 2.1$, $r_{n=20} = 2.35$ and $r_{n=30} = 2.48$.

Case 2. Ellipse packing problem. The problem is kept as simple as possible:

there is only one shape in addition to the enclosing shape; the shape is described by a single inequality; only translation is permitted. The radii of the ellipse is 1 and 0.5. The enclosing circle have a radius $r_{n=10} = 2.9$, $r_{n=20} = 4$ and $r_{n=30} = 5$.
Case 3. Now allow rotation in the same ellipse packing instance. The radius of the circle is set to $r_{n=10} = 2.7$, $r_{n=20} = 3.9$ and $r_{n=30} = 4.9$.
Case 4. We replace now the ellipse with a non-convex shape which description involves several inequalities (a horseshoe). The enclosing ellipse have a radii $r_{n=10} = 2/4$, $r_{n=20} = 3.9/1.8$ and $r_{n=30} = 5/2.9$.
Case 5. The last case is a mixed packing problem where n/2 ellipses and n/2 horseshoes have to be packed inside a circle of radius $r_{n=10} = 2.9$, $r_{n=20} = 4$ and $r_{n=30} = 6$. Again, both translation and rotation are considered.
Results.
Since some pavings are used in several cases, paving and packing times are reported separately. We also only give the average time for the Case 1 and for the other pavings involving the enclosing shape (since this shape changes with n). The pavings were calculated with a precision $\varepsilon = 0.1$.



Case 1 **Case 5**

Fig. 4: **Packing results.**

d.o.f.	*Object i*	*Object j*	*Time*
translation	circle	circle	0.01 (avg)
	ellipse	ellipse	0.16
	enclosing circle	ellipse	0.81 (avg)
translation and rotation	ellipse	ellipse	300
	enclosing circle	ellipse	2245 (avg)
	ellipse	horseshoe	1740
	horseshoe	horseshoe	3780
	enclosing circle	horseshoe	8311 (avg)
	enclosing ellipse	horseshoe	10909 (avg)

Table 1: **Paving time** (in seconds)

#objects	case 1	case 2	case 3	case 4	case 5
10	<0.1	4.5	66	103	53
20	54	32	235	372	194
30	203	53	559	1005	395

Table 2: **Packing time** (in seconds)

First, we can see that the circle packing problem has been solved in a few seconds only. However, the enclosing circle is not optimal. In [MVFA13], the circles are packed in a circle of radius $r_{n=10} = 1.95$, $r_{n=20} = 2.15$ and $r_{n=30} = 2.27$. This gap is due to the precision ε. Using this precision for paving amounts to consider that objects to pack are "enlarged by" ε. The radius we have set for the enclosing circle is actually the smallest one for which packing is successful, given the precision ε of 0.1.

6 Conclusion

We have proposed a generic algorithm for packing non convex curved objects. The approach is not limited to polyhedral shapes and does not make convex approximation. Cavities introduced by non-convexity are used to place objects. The approach is split in two steps. The *paving* step calculates off-line the set of relative positions and orientations of one object with respect to a second one that make them overlap. The *packing* step minimizes a fitness function which is a sum of distances between points and boundaries of the previously calculated sets.

These distances are fast to calculate thanks to a tree-structured representation of the sets. The other contribution is a new inflator for the overlapping constraint, i.e., the paving step. This new inflator uses separate techniques for inflating with respect to the position and the angle, which leads to a much larger inflation than with previously existing techniques. Preliminary experimental results are encouraging. One possible future experiment could be to see how both paving and packing time increases as ε decreases and to mix this numerical approach with the available ad-hoc formulas.

References

[ATNC14] I. Araya, G. Trombettoni, B. Neveu, and G. Chabert. Upper Bounding in Inner Regions for Global Optimization Under Inequality Constraints. *Journal of Global Optimization*, 60(2):145–164, 2014.

[CB10] G. Chabert and N. Beldiceanu. Sweeping with Continous Domains. In *CP, International Conference on Principles and Practice of Constraint Programming*, volume 6308 of *LNCS*, pages 137–151, 2010.

[CC86] S. Cameron and R. Culley. Determining the Minimum Translational Distance Between Two Convex Polyhedra. In *IEEE International Conference on Robotics and Automation*, volume 3, pages 591–596. IEEE, 1986.

[CJ09] G. Chabert and L. Jaulin. Contractor Programming. *Artificial Intelligence*, 173(11):1079–1100, 2009.

[HO01] N. Hansen and A. Ostermeier. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[MVFA13] T. Martinez, L. Vitorino, F. Fages, and A. Aggoun. On Solving Mixed Shapes Packing Problems by Continuous Optimization with the CMA Evolution Strategy. In *Proceedings of the first BRICS countries congress on Computational Intelligence*, 2013.

[SCG14] I. Salas, G. Chabert, and A. Goldsztejn. The Non-Overlapping Constraint between Objects described by Non-Linear Inequalities. In *CP, International Conference on Principles and Practice of Constraint Programming*, volume 8656 of *LNCS*, pages 672–687, 2014.