J Glob Optim (2014) 59:367–404
DOI 10.1007/s10898-013-0129-z

A fully general, exact algorithm for nesting irregular shapes

Donald R. Jones

Received: 1 April 2013 / Accepted: 3 December 2013 / Published online: 28 December 2013

Abstract This paper introduces a fully general, exact algorithm for nesting irregular shapes. Both the shapes and material resource can be arbitrary nonconvex polygons. Moreover, the shapes can have holes and the material can have defective areas. Finally, the shapes can be arranged using both translations and *arbitrary* rotations (as opposed to a finite set of rotation angles, such as 0°and 180°). The insight that has made all this possible is a novel way to relax the constraint that the shapes not overlap. The key idea is to inscribe a few circles in each irregular shape and then relax the non-overlap constraints for the *shapes* by replacing them with non-overlap constraints for the *inscribed circles*. These relaxed problems have the form of quadratic programming problems (QPs) and can be solved to optimality to provide valid lower bounds. Valid upper bounds can be found via local search with strict non-overlap constraints. If the shapes overlap in the solution to the relaxed problem, new circles are inscribed in the shapes to prevent this overlapping configuration from recurring, and the QP is then resolved to obtain improved lower bounds. Convergence to any fixed tolerance is guaranteed in a finite number of iterations. A specialized branch-and-bound algorithm, together with some heuristics, are introduced to find the initial inscribed circles that approximate the shapes. The new approach, called "QP-Nest," is applied to three problems as proof of concept. The most complicated of these is a problem due to Milenkovic that has four nonconvex polygons with 94, 72, 84, and 74 vertices, respectively. QP-Nest is able prove global optimality when nesting the first *two* or the first *three* of these shapes. When all *four* shapes are considered, QP-Nest finds the best known solution, but cannot prove optimality.

Keywords Fabric nesting · Leather nesting · Nonconvex polygon · Irregular shape · Inscribed circle · Inner approximation · Global optimization · Quadratic programming · Cutting and packing · Translation · Rotation · Guaranteed optimality

D. R. Jones (✉)
General Motors, Warren, MI, USA
e-mail: don.jones@gm.com

1 Introduction

Figure 1 shows a "nest" of patterns to be cut from a roll of cloth and later sewn together to make the seats and headrests for a car. The nest was made by an experienced human nester working with a computer aided design (CAD) system. The CAD system allows the user to easily translate the parts horizontally and vertically, rotate them, butt one part up against another, align a part with the material edge, etc. Experienced nesters try many arrangements, looking for one that fits all the patterns on a roll of given width and occupies the least length. Fabric patterns like those shown in Fig. 1 often must align with a design or nap in the fabric, so the allowed part rotations will be either 0°or 180°; however, experienced nesters will sometimes violate this rotation constraint by a few degrees for little noticed parts if this allows them to get a much tighter nest. In Fig. 1, the desired orientation of a pattern is indicated by an arrow at the center and, if one looks closely, one can see where the nester has "cheated" the orientation a little from true horizontal. The full "real-world" nesting problem involves even more variables and decisions. In the garment industry, for example, one must decide what styles of garments, and what combination of sizes, to include in the nest, as well as the number of layers of fabric to cut at one time, taking into account customer demand and inventory holding costs [12].

Leather nesting problems are equally, if not more, complex. Because there is no nap, the parts can be freely rotated. Unlike a roll of cloth, the material resource—an animal hide—is highly irregular in shape and may have zones of different leather quality (including blemishes that must be avoided). The goal may be to cut a given set of patterns from the fewest hides, or perhaps to nest as many patterns as possible on a single hide.

One of the greatest challenges in the field of cutting and packing is the construction of computer programs that can consistently match or beat experienced nesters. Milenkovic states that, in the apparel industry, "no layout software has yet replaced a human" and estimates that available nesting software falls about 5 % behind humans in cloth utilization [21,22]. Even so, good nesting software can play an important role whenever nests must be constructed quickly or when experienced nesters are not available.

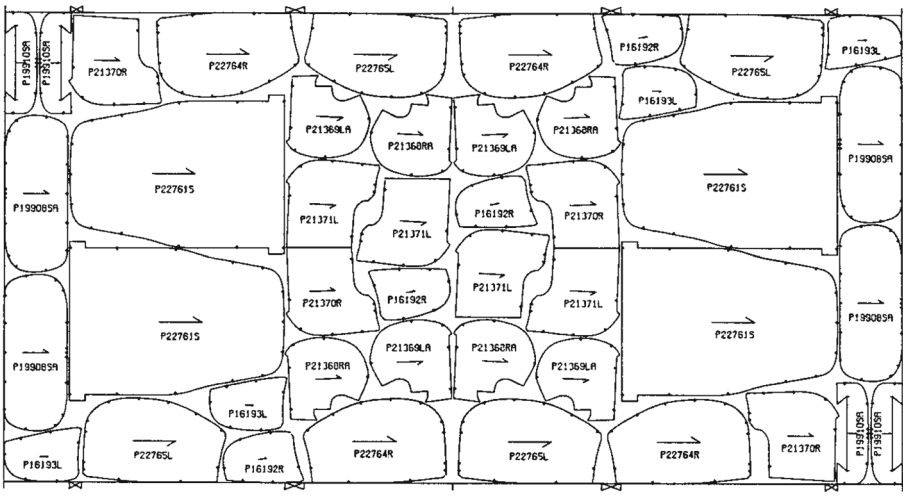


Fig. 1 Fabric nesting

A common element in nesting problems is the requirement that the shapes not overlap. While human nesters can recognize overlap effortlessly, for a computer the calculation of overlap is complicated. Given two nonconvex polygons, each with n sides, computing the overlap region requires order n^2 time in the worst case [10,25].[1] If one only wants to determine *whether* the polygons intersect (without computing the overlap region), this can be done in order $n \log n$ time. Even though the overlap area can be computed *numerically*, there is no simple *closed-form expression* for the overlap area of two nonconvex polygons. One might think that there would be simple expressions for simple shapes, but even for two triangles, the expression for the overlap area is extremely messy.[2] This lack of a simple closed-form expression for the overlap area has made it difficult to develop exact algorithms, because essentially all exact methods in nonlinear global optimization exploit properties of explicit problem structure. In addition, the nesting problem is intrinsically difficult: Li and Milenkovic prove that the nesting problem, *even with rectangles*, is NP hard, meaning that the effort to solve it is expected to go up exponentially with the number of parts being nested [20].

In this paper, we will present a new approach to this problem, called "QP-Nest," that can solve certain small nesting problems to guaranteed optimality. The parts can be arbitrary, nonconvex, simple polygons, and they can be freely translated or rotated.[3]

Before describing this new approach, we will set the stage by reviewing the relevant literature on nesting irregular shapes, with a special focus on exact algorithms; comprehensive reviews can be found in references [3,4,11]. In general, most approaches in the literature are heuristic in nature. Exact algorithms are rare and, as of this writing, are limited to nesting from 2 to 10 parts. However, even though exact methods can only solve small problems, they are still highly relevant for two reasons. First, a few important real-world problems are small. Second, many of the best heuristics for large problems leverage exact methods to solve difficult small subproblems.

One purely heuristic approach to nesting is to mimic human nesters. A good example is the work of Lamousin and Waggenspack [17]. At any point in their algorithm, some parts will have been placed and some will not. Like humans solving a puzzle, they look for good matches between the contours of a part that has not been placed, and the contours of the remaining "open area" on the resource (i.e., the area *not* covered by parts that *have* been placed). Part rotations are determined so as to optimize the matching of a contour of an unplaced part to the contours of the open area. Many potential placements are considered and ranked by a "matching index." The top 20 or so of these potential placements are then evaluated for how much "waste" they produce. The waste function is again heuristic, consisting of terms that compute the unusable area or gaps between parts, plus a look ahead term that estimates the "future waste" that might be generated when the remaining parts are placed. This look-ahead term favors the early placement of large parts with complex profiles, since such early placement makes it more likely that the complex profiles can later be filled by smaller, more regularly shaped parts. While the resulting algorithm is *conceptually* quite complex—full

[1] O'Rourke [24] shows that the computation can also be done in $O(n \log n + k)$ time, where k is the number of intersection points of edges, using a variation on the Bentley-Ottmann sweepline algorithm.

[2] A triangle can be considered the intersection the three half-planes defined by its sides. Hence, the intersection of two triangles can be considered the intersection of the associated *six* half-planes. A region defined by the intersection half-planes is called a convex polyhedron, and Lasserre [18] has derived a general, recursive formula for the volume of such polyhedra. This formula is easy to implement in a computer, but it makes intensive use of the absolute value, minimum, and maximum operators. As a result, even for two triangles, the resulting closed-form expression is quite ugly, nondifferentiable, and not very amenable for use in an optimization algorithm.

[3] A simple polygon is one whose sides do not intersect each other. A simple polygon can, however, have holes.

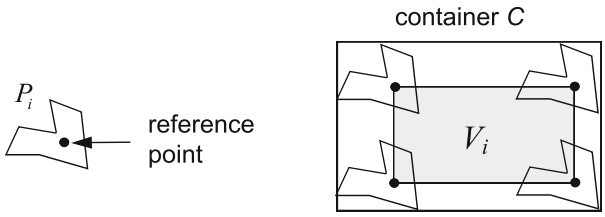


Fig. 2 The set V_i used in containment problems

of algorithms for profile simplification, feature matching, and so on—it is *computationally* relatively fast even for a large number of parts (they have examples up to 295 parts).

Heistermann and Lengauer [14] follow a similar approach, in their case motivated by watching how people nest patterns on animal hides. Their code consists of 40,000 lines and apparently has been in industrial use since 1992. Their paper shows an impressive nest of 51 parts for a leather sofa on a irregularly shaped hide that was computed in just 6 min, a nest that beat human nesters by 5 %.

So far we have discussed methods that are purely heuristic in nature and solve nothing to guaranteed optimality. Let us now turn our attention to the exact opposite: algorithms that can exactly solve small problems. In general, all such methods are characterized by an intense use of advanced computational geometry.

A good example of the exact approach is the work on "translational containment" by Daniels and Milenkovic [8,9]. In translational containment, the goal is to determine whether a given set of polygons can be translated horizontally and vertically to fit within a container without overlap. Containment is intimately connected with nesting, because any containment algorithm can be converted into a nesting algorithm—one simply searches for the smallest container that can hold the polygons. Containment is also important in fabric nesting, where one often wants to determine whether a set of small patterns can be fit into the gap between bigger patterns.

Two geometric sets are central in the algorithms of Daniels and Milenkovic. The first set, called V_i, consists of all of those positions of the reference point of polygon P_i that place it in the interior of the container C. Figure 2 illustrates this V_i set for a nonconvex polygon and a rectangular container. As shown in Fig. 2, the set can be found by moving polygon P_i around the perimeter of the container and tracing the position of the reference point. For a rectangular container, V_i will be a rectangle.

This set V_i can also be expressed using the Minkowski sum $\oplus$ and the complement operator as follows

$$V_i = \overline{\overline{C} \oplus -P_i} \tag{1}$$

(This is not obvious).[4] Here $A \oplus B \equiv \{a + b \,|\, a \in A, b \in B\}$, $-A \equiv \{-a \,|\, a \in A\}$, and $\overline{A} \equiv \{x \,|\, x \notin A\}$. The second set, U_{ij}, is the set of all relative displacements of the reference points of polygons P_i and P_j such that they do not overlap. As shown in Fig. 3, this set can be found by locating polygon P_i so that its reference point is at the origin and then moving polygon P_j around it. The boundary of U_{ij} is then found by tracing the movements of the reference point of polygon P_j.

Note that the U_{ij} region will typically consist of the plane with a piecewise-linear hole; it therefore is not convex. The boundary of U_{ij} often goes by the name of the "no-fit polygon"

[4] If p is a point in the polygon and v is a translation, then the translation is *not* valid if $v + p \in \overline{C}$ or, equivalently, $v \in \overline{C} - p$. Hence, the translation *is* valid if v lies in the complement of the right hand set, $v \in \overline{\overline{C} - p}$.

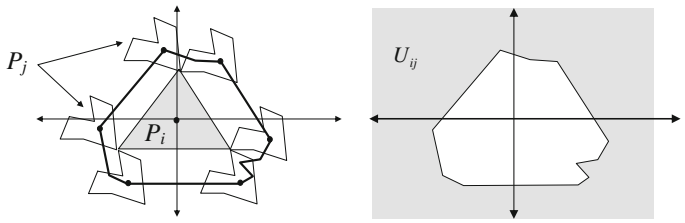


Fig. 3 The set U_{ij} used in containment problems

because any translation that lies within this polygon will cause the polygons to intersect or "not fit." The set U_{ij} can be expressed with the help of the Minkowski sum as follows:

$$U_{ij} = \overline{P_i \oplus -P_j} \tag{2}$$

Again, this is not obvious.[5] If we let t_i denote the location of the reference point of polygon P_i, the containment problem for N polygons reduces to finding locations $t_1, \ldots, t_N$ such that $t_i \in V_i$ and $t_j - t_i \in U_{ij}$. If the sets U_{ij} were convex, this would actually be a linear programming problem but, alas, this is not the case.

The exact algorithm of Daniels and Milenkovic, called kNN, relaxes the problem into a linear programming problem by replacing U_{ij} with its convex hull. Initially, the convex hull is the entire space, so the first iteration is not very interesting. If, in the solution to the linear program, there are some i and j with $t_j - t_i \notin U_{ij}$ (as there will certainly be on the first iteration), then the algorithm branches by dividing U_{ij} into two "more convex" pieces. Convergence is further accelerated by using various "geometric restriction" arguments to reduce the size of the U_{ij} sets before taking the convex hull and solving the linear programming problem.[6] To date, this method has been used to nest up to 10 polygons.

Avnaim and Boissonnat [2] have extended the concept of the no-fit polygon to consider rotations. In this case, no-fit polygon becomes a "no-fit three-dimensional volume" consisting of all relative translations *and* relative rotations of polygons i and j for which the polygons overlap. One would hope that this would allow any exact translational nesting algorithm based on no-fit polygons to be easily extended to rotations. Unfortunately, due to the rotations, this three-dimensional no-fit region has curved surfaces that are not easily adapted for use in an optimization algorithm.

To avoid this complexity, Milenkovic [21] develops a "rotational polygon containment" algorithm that handles rotations in a different way. At any node in the branch-and-bound search tree, each polygon has a range of admissible rotation angles. He then computes the intersection of each polygon with itself in all these admissible rotation angles. The result is a smaller polygon consisting of points that *must* be in the polygon *no matter what rotation*

[5] Let x_i and x_j be points in polygons i and j, respectively, and let t_i and t_j be the locations of the two polygons. Points x_i and x_j will coincide, and hence the polygons will overlap, if $x_i + t_i = x_j + t_j$ or, rearranging, if $t_j - t_i = x_i - x_j$. To prevent overlap, this must never happen for any x_i and x_j, which means we can *not* have $t_j - t_i \in P_i \oplus -P_j$. Hence, we must have $t_j - t_i \in \overline{P_i \oplus -P_j}$. Besides the use of the Minkowski sum for computing the no-fit polygon, a newer and apparently more efficient approach is given in [5].

[6] The geometric restriction is based on considering triples of parts (i, j, h). Any valid, non-overlapping translation of polygon j relative to polygon i should also be computable as a valid translation from i to h plus a valid translation from h to j. This leads to the following tightening rule: $U_{ij} \leftarrow U_{ij} \cap \left(U_{ih} \oplus U_{hj}\right)$. This range reduction formula is repeatedly applied, considering different triples of parts, until no further reduction is obtained.

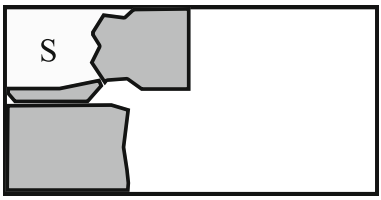


Fig. 4 Sequential nesting methods. The method of Albano and Sapuppo locates builds the nest one part at a time, always locating the part in the leftmost and lowest possible position. The choice of *which* part to add next is based on which one causes the least increase in "waste," defined as the area to the *left* of the shapes (*S*). Reproduced from [1]

is ultimately selected (from the admissible range at that node). Any relative translation that would cause these "range intersection" versions of the polygons to overlap is therefore definitely invalid. This allows Milenkovic to apply the same logic as in the kNN algorithm for translational nesting, replacing the polygons with their range intersections. More branching is required because one must reduce the range of admissible rotations at each node in order to reduce the difference between the polygons and their range intersections. The net result is that, when considering both translations and rotations in this way, Milenkovic can only solve problems with 2–3 parts to optimality and zero overlap—many fewer than the 10 parts he can handle in pure translational nesting.

Kallrath [15] has introduced an exact nesting algorithm for nesting *convex* shapes (circles, rectangles, polygons) based on the idea of the *separating hyperplane*. For each pair of convex polygons, the non-overlap constraint is enforced by searching for a hyperplane that separates the two polygons. This approach could, in principle, be extended to handle nonconvex polygons by first decomposing each nonconvex polygon into convex pieces. However, Kallrath's method already has difficulty finding guaranteed optimal solutions to problems with more than two polygons; if one decomposes the nonconvex polygons into many convex pieces, the size of the problem would soon exceed the algorithm's capability.

Now let us turn our attention to heuristic methods that leverage exact methods to solve certain small auxiliary problems to optimality. Perhaps the most common method of this sort is the "sequential approach" in which the nest is built by adding one shape at a time. For example, Albano and Sapuppo [1] build their nest from left to right. The next part being added is translated and rotated to place it in the leftmost and lowest possible position, considering a finite set of possible rotations (in their case, 0°and 180°). To determine *which* shape to add to next, Albano and Sapuppo select the one that creates the least increase in waste, where "waste" is defined as the "shadow area" that cannot be filled by any remaining shapes. In Fig. 4, region *S* is waste. The subproblem of finding the leftmost and lowest location for the each remaining part is solved *exactly* using a no-fit polygon approach for two shapes, namely, the part being added and the conglomerate of the parts already placed. The exact optimization consists of enumerating over the set of possible positions (vertices in the no-fit polygon) and the set of allowed rotations (separate no-fit polygons must be computed for each allowed rotation).

Verkhoturov and Sergeyeva [27] develop a more elaborate version of this approach called the "sequential value correction" method. The key difference lies in the rule used to decide *which* part to add in each iteration. Their approach not only considers the *areas* of the "waste regions" produced by adding a part, but also the *perimeters* of these waste regions, preferring part placements that share perimeter with waste regions from previous iterations.

Perhaps the most successful sequential method is the recent work of Burke et al. [4]. This approach goes beyond the no-fit polygon approach by considering not only polygonal shapes,

but also shapes with true *arc geometry*. This is important, since approximating arcs with many polygonal segments greatly increases the computation time of no-fit polygons. As parts are sequentially added, each part is first placed in the lower left corner and then adjusted upward, and if necessary rightward, until it does not overlap the parts already placed. It appears that this step is solved exactly, based on their new algorithm for "intersection resolution" that incorporates arc geometry. The sequence in which the parts are added is optimized using both hill climbing and tabu local search methods. The authors test their algorithm on a set of 26 standard problems in the literature, ranging from 12 to 99 parts, and produced 25 new best solutions.

Recall that the human-motivated approach of Lamousin and Waggenspack [17] also was a sequential procedure, the difference being that the location of the next part was based on feature-matching, not "bottom left" placement. Interestingly, Lamousin and Waggenspack have tried both approaches using the same framework. It turned out that the feature-matching approach worked best. Both methods optimize an *objective function for a partial solution* to decide where to place the next part: either to make the placement the most low and left possible, or to maximize the feature matching. The fact that the procedures give different results means that the choice of objective function matters. In all likelihood, the best choice is problem dependent.

All of the sequential nesting approaches discussed above impose the constraint that, once a part is placed, it is fixed. By relaxing this constraint, any sequential approach can be made potentially more powerful—albeit more complicated and/or costly. A good example of relaxing this constraint is the algorithm of Kallrath and Rebennack [16] that appears in this same issue of the *Journal of Global Optimization*. The authors consider the problem of nesting ellipses on an area-minimizing rectangle. For this purpose, they have developed an exact algorithm that converges to full optimality for two ellipses and produces good solutions (with some optimality gap) for up to ten ellipses. They leverage this approach to nest many ellipses using a sequential approach. Given some sequence of the ellipses, they begin by nesting the first n_1 ellipses. Thereafter, they "unfreeze" the rightmost $n_3 < n_1$ ellipses and renest them together with the next n_2 ellipses in the sequence. In this way, some of the previously placed ellipses on the rightmost "front" of the evolving nest can be slightly adjusted to better accommodate the next ellipses in the sequence. They continue in this way until all ellipses are placed, and the whole procedure is repeated for a number of randomly generated sequences. Typical values used are $n_1 = 10$, $n_3 = 5$, and $n_2 = 1$ or 2. Another innovation in their approach is that they do not insist on running the exact algorithm to full optimality, but rather stop it based on a time limit. Of course, in their case they *must* use a time limit, because their algorithm doesn't always converge for more than two parts. Even so, it may be advantageous to impose a time limit, because this shortens the time to build a nest based on a given sequence and, hence, allows more sequences to be considered. The whole procedure appears to work quite well and is a good example of how an exact algorithm for nesting a small number of parts can be leveraged to create an effective heuristic for nesting more parts.

A second approach for leveraging exact algorithms to solve large problems is the "column-based strip packing" approach of Daniels and Milenkovic [9] illustrated in Fig. 5. Using exact containment algorithms, the authors build columns of shapes (around 5–12 for each column) and then put the columns side-by-side to make a column-based layout of the larger apparel patterns (e.g., trouser legs). After the columns are built, gaps are opened and filled with smaller trim pieces, again using exact containment algorithms to fit the trim pieces in the gaps. Compaction algorithms [19,20]) are used to locally perturb the positions of the parts to form a tight nest.

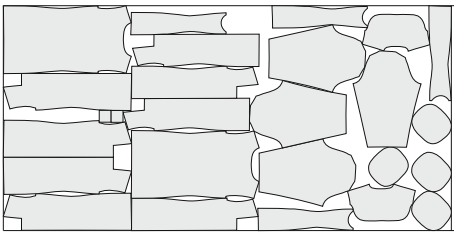
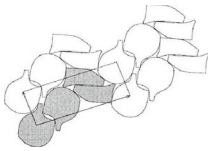


Fig. 5 The column based strip packing algorithm of Daniels and Milenkovic. Reproduced from [9]



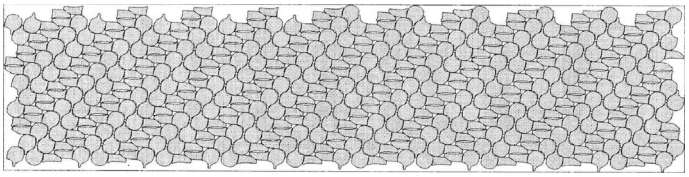


Fig. 6 Lattice-based packing. Reproduced from [22]

A final approach for leveraging exact algorithms to solve larger problems involves lattice packing. In lattice packing, one finds a non-overlapping arrangement of a small number of parts (usually 1–4) that can be replicated without overlap at each point of a lattice $i_0 v_0 + i_1 v_1$, where v_0 and v_1 are vectors generating the lattice and i_0 and i_1 range over the integers. Figure 6 provides an example from the work of Milenkovic [22] in which he nests bra parts (cups and straps). The lattice packing has four parts—the cup and the strap, as well as their 180°rotated versions. The relative positions of these four parts are selected to minimize the area $|v_0 \times v_1|$ of the "fundamental parallelogram." While the approach maximizes the nest efficiency in the entire plane, it is sub-optimal for any clipped rectangle due to edge effects. Nevertheless, Milenkovic's nest in Fig. 6 outperformed a human-created, non-lattice nest for the same parts.

To take into account edge effects in lattice nests, Costa, Gomes, and Oliveira [7] develop a heuristic method that takes into account *both* the area of the fundamental parallelogram *and* the percentage of stock sheet utilization on a *given*, finite rectangular sheet.

As the above review indicates, the literature on nesting irregular shapes is dominated by heuristics. The only exact algorithm for nesting with both translations *and* rotations appears to be that of Milenkovic [21], and this algorithm can only handle up to *three* parts. The current paper attempts to push the state of the art a little further: as we will see, we can now sometimes reach to *four* parts. For those readers who have not spent years trying to solve this difficult problem, the increase from three to four parts may seem a small advance indeed. But for those that know the true difficulty of the problem, the advance is significant.

Like Milenkovic, our approach uses computational geometry but, as we will see, it is of an entirely different and much simpler sort, not relying on the concepts of the Minkowski sum or the no-fit polygon. Instead, the new approach relies heavily on recent advances in global optimization that make it possible to solve certain quadratically constrained, quadratic

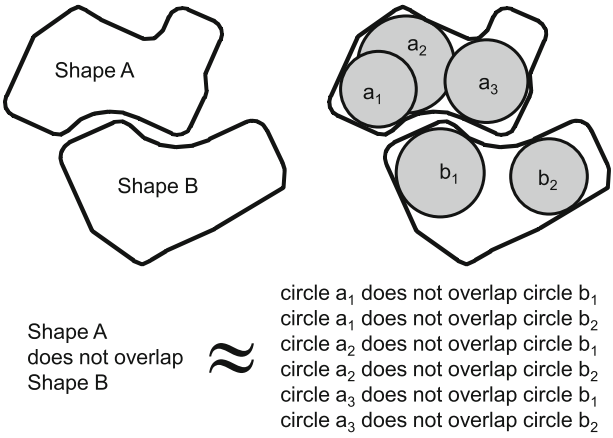


Fig. 7 Approximating the non-overlap constraint using inscribed circles

programming problems to full optimality. At the time Milenkovic did his work, such methods were in their infancy, and the only reliable large-scale global optimization method was linear programming. As a result, he naturally built his exact kNN algorithm around a linear programming core.

The key insight in the new approach is a novel way to handle the non-overlap constraint. In particular, we inscribe several circles within each irregular shape and relax the constraint that the *shapes* do not overlap by replacing it with the constraint that the *inscribed circles* from one shape do not overlap with the *inscribed circles* from another shape (see Fig. 7).

For two circles not to overlap, we need only require that the distance between their centers be greater than the sum of their radii, and this is a quadratic constraint. Furthermore, it is clear that we can approximate the non-overlap constraint for the shapes as closely as we want by inscribing more and more circles (note that circles within the *same* shape *can* overlap with each other, as shown in Fig. 7). With this insight, it becomes possible to pose the nesting problem as a quadratic programming problem. As we will see, it is also not necessary to start with a huge number of circles. Instead, one can start with a few circles per shape, enough to give a reasonable covering, and then add circles sequentially only where they are needed until the desired accuracy is achieved (more on this later). Due to its use of quadratic programming, we call the new algorithm "QP-Nest."

In what follows, I will use a simple numerical example with four parts to illustrate the construction of the QP-Nest algorithm. My focus will be on the structure of the algorithm and its convergence properties, as well as on ways to enhance the problem formulation to improve performance. The simple numerical example is easily solved to full optimality. We then consider harder problems with four and six parts respectively. For these problems, QP-Nest can find good solutions, but has *not* been able to prove optimality. Based on our limited testing, it thus appears that QP-Nest can reliably solve problems with two parts, often solve problems with three parts, and sometimes solve problems with four parts. It is still unclear what problem characteristics make a problem easy or difficult to solve; my best guess is that the difficulty of a problem increases with the number of nonconvex vertices and the number of circles required to get good approximations of the shapes.

While the main focus of QP-Nest is on solving problems with translations and rotations, it can equally well solve pure translational problems in which the rotations are fixed. For pure translational problems, we are able to solve our six-part hard problem to full optimality.

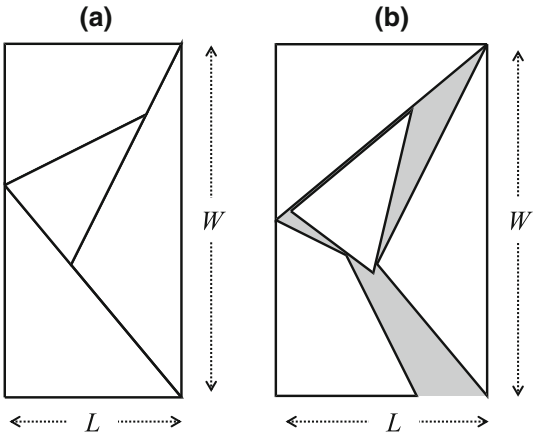


Fig. 8 A simple nesting problem with four polygons on a roll of fixed width W: **a** the globally optimal nest with $L = 5$; **b** a bad solution with length $L = 6$

In any case, the inscribed-circle approach to exact nesting opens up new territory for research and provides challenging test problems for quadratically constrained global optimization.

2 A simple problem

To illustrate the exact algorithm, let us consider the problem of nesting four polygons on a roll of fixed width W to minimize the length L of the nest. For simplicity, we will use the four shapes shown in Fig. 8a which have the convenient property that they fit together to form a rectangle (zero scrap). Thus, for this problem, we know the global optimum.

While the nest is obvious once you see it, the problem has been designed to be a bit confusing. Most people would naturally put the shapes with right angles in the corners, but if the shapes are placed incorrectly, as shown in Fig. 8b, one can end up with a nest with significant scrap (gray region).

3 The QP-nest algorithm

For concreteness, I will illustrate each step of the algorithm using the simple problem of the previous section. This problem is a bit more special than it needs to be, since all the shapes are convex. The algorithm, on the other hand, can handle nonconvex shapes that can even have holes. However, as in the sample problem, we *will* assume that the shapes are simple *polygons* (the holes, if any, must also be polygons). Shapes whose boundaries include arcs can be handled by approximating the arcs with many line segments. Increasing the number of segments will not increase the algorithm's running time very much, because most of the time is spent solving quadratic programming problems whose complexity depends primarily on the number of inscribed circles as opposed to the number of segments.

The algorithm is initialized by inscribing a few circles inside each shape, just enough to provide a reasonably good inner approximation. While there are many ways to do this, we will begin here with what is probably the simplest and most obvious approach, an approach in which none of the inscribed circles overlap. After introducing this simple version, we will add an improvement that allows the circles *within* a polygon to overlap.

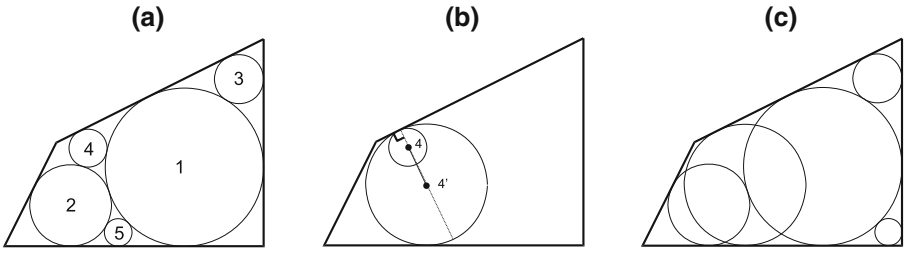


Fig. 9 An example of inscribing 5 circles in a polygon. **a** The simple circle-filling procedure proceeds iteratively, in each step finding the largest circle that lies *inside* the polygon and does not intersect any previously inscribed circle. **b** In the simple procedure, circle 4 only touches one side, so it can be grown into a bigger one by moving its *center* along a *line* perpendicular to the contacted side. **c** The enhanced circle filling procedure proceeds in the same way as the simple one, except that it grows a circle whenever it touches only one side before proceeding to insert the next one

The simplest way to inscribe circles is a purely greedy one. We start by finding the largest circle that can be inscribed in the polygon. We then proceed iteratively, in each step finding the largest circle that lies entirely inside the polygon and does not intersect with any previously inscribed circle. We continue in this way until a pre-specified number of circles have been inscribed, or until the next circle's radius is below some tolerance. Figure 9a illustrates the process, showing the first five circles inscribed in one of the polygons from the example.

Notice that circle 4 in Fig. 9a only touches one side of the polygon. When this happens, one can find an even bigger circle that contains the original one and still lies within the polygon, as shown in Fig. 9b. One simply moves the center of the circle along a line perpendicular to the contacted side, growing the circle, until the growth is stopped by contact with a polygon side or vertex. This is the important improvement alluded to earlier.

Our "enhanced circle-filling procedure" proceeds in exactly the same way as the simple method but, after each next circle is inscribed, we check if it touches just one side and, if so, we "grow" it as just explained. This enhanced procedure results in the approximation shown in Fig. 9c. The first three circles are the same as in Fig. 9a, but the fourth circle has been grown and, as a result, the fifth circle is different. Comparing Fig. 9a, c, you can see that the enhanced procedure better fills the polygon, especially the lower right.

To implement the enhanced circle-filling procedure, I developed a special branch-and-bound algorithm for inscribing the largest circle in a polygon that does not overlap any previously inscribed circles. The algorithm is based on covering the polygon with triangles and then using bounding arguments to discard triangles that cannot possibly contain the center of the largest inscribed circle. The triangles that are not discarded are subdivided into smaller triangles, thereby tightening the bounds, and the process is repeated until the center point is located to the desired accuracy; details are provided in the "Appendix". While Fig. 9 illustrates the approach for a convex polygon, the method is fully general and can handle any nonconvex polygon.

Readers familiar with computational geometry will recognize that the centers of the circles inscribed by the enhanced method lie on the "medial axis" of the polygon. In fact, one common definition of the medial axis is the locus of all points that can be centers of circles that lie within the polygon and touch two or more sides or vertices. As shown in Fig. 10, the medial axis forms a tree-like skeleton of the polygon.

Because the medial axis can be computed efficiently in linear time [6], one might think that it would be an excellent tool for inscribing circles. However, it is not clear how one would

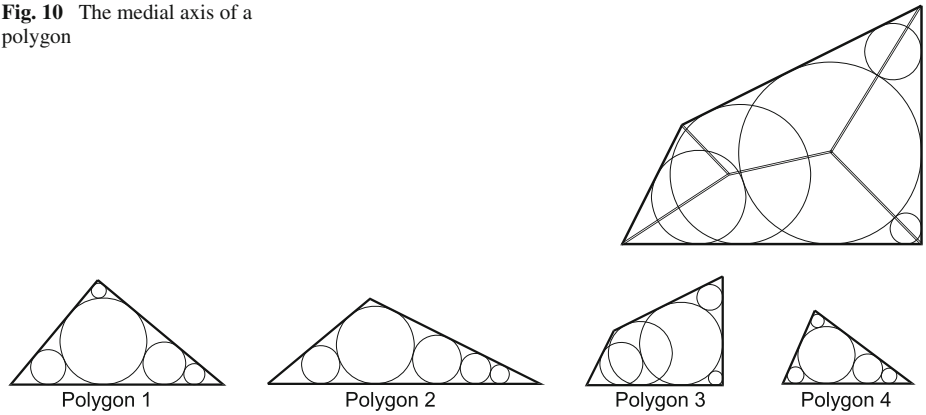


Fig. 10 The medial axis of a polygon

Fig. 11 The four polygons of the simple example, together with their initial inscribed circles

decide *which points* on the medial axis to use as centers of the inscribed circles; these points would somehow have to be selected so as to cover as much of the polygon as possible.[7]

Our enhanced circle-filling procedure also provides a natural criterion for deciding *how many* circles to inscribe; in particular, one can stop adding circles when the radius of the last added circle, *before* growth, falls below some absolute tolerance. For example, the tolerance might be set so that the area of the circle is less than (say) 1 % of the area A of the polygon: $\pi r^2 < 0.01A$ or $r < \sqrt{0.01A/\pi}$.

Figure 11 shows the four polygons, along with their initial inscribed circles, in their so-called "reference position" before nesting. The reference position of a polygon is found by translating the polygon so that the center of the largest inscribed circle is at the origin. In Fig. 11, I have also rotated each polygon so that the longest side is horizontal and at the bottom, but this is of no significance.

The next step is to nest the shapes to minimize the length L of the nest, subject to the constraint that polygons lie within a roll of width W and that circles from *different* polygons do not overlap. This is the part of the algorithm that can be posed as a quadratic program. To describe this quadratic program formally, we need the following notation:

N		number of polygons
n_ℓ		number of vertices in polygon ℓ
$(x_{k\ell}, y_{k\ell})$,	$k = 1, \ldots, n_\ell$	vertices of polygon ℓ
m_ℓ		number of circles inscribed in polygon ℓ
$(a_{k\ell}, b_{k\ell}), r_{k\ell}$,	$k = 1, \ldots, n_\ell$	circle centers and radii for polygon ℓ
L_0		length of a known feasible nest

All the coordinates given above refer to the shapes in their reference positions in which the center of the first (largest) circle is at (0, 0). In the nest, the polygons and their inscribed circles will be rotated and translated relative to these reference positions. Let us denote the angle of rotation of polygon ℓ by θ_ℓ and the horizontal and vertical translations by h_ℓ and v_ℓ, respectively. Instead of working directly with θ_ℓ, we will find it more convenient to work with the sine and cosine of θ_ℓ, denoted s_ℓ and c_ℓ, and then add the constraint $c_\ell^2 + s_\ell^2 = 1$. This trick eliminates trigonometric functions from the problem formulation and leaves us

[7] For nonconvex polygons, the medial axis can have curved segments as well as straight lines, which would further complicate its use in selecting the centers of the initial inscribed circles.

with a quadratic program. Using the standard formulas for rotations, the coordinates of the polygon vertices and circle centers in the nest are as follows:

$$\text{Vertex } i \text{ of polygon } j \qquad \left(c_j x_{ij} - s_j y_{ij} + h_j, s_j x_{ij} + c_j y_{ij} + v_j\right) \tag{3}$$

$$\text{Center circle } i \text{ of polygon } j \qquad \left(c_j a_{ij} - s_j b_{ij} + h_j, s_j a_{ij} + c_j b_{ij} + v_j\right) \tag{4}$$

For circle i of polygon j not to overlap circle k of polygon ℓ, the distance between their centers must be greater than or equal to the sum of their radii. This results in the following quadratic constraint:

$$\left(c_j a_{ij} - s_j b_{ij} + h_j - c_\ell a_{k\ell} + s_\ell b_{k\ell} - h_\ell\right)^2 + \left(s_j a_{ij} + c_j b_{ij} + v_j - s_\ell a_{k\ell} - c_\ell b_{k\ell} - v_\ell\right)^2 \geq (r_{ij} + r_{k\ell})^2$$

Our quadratic programming problem is then to choose L, h_ℓ, v_ℓ, c_ℓ, and s_ℓ, $\ell = 1, \ldots, N$, to solve the following problem.

$$\text{Minimize} \quad L \tag{5}$$

Subject to:

$$s_\ell^2 + c_\ell^2 = 1 \qquad \forall\, \ell = 1, \ldots, N \tag{6}$$

$$\left(c_j a_{ij} - s_j b_{ij} + h_j - c_\ell a_{k\ell} + s_\ell b_{k\ell} - h_\ell\right)^2 + \left(s_j a_{ij} + c_j b_{ij} + v_j - s_\ell a_{k\ell} - c_\ell b_{k\ell} - v_\ell\right)^2 \geq (r_{ij} + r_{k\ell})^2 \qquad \begin{array}{l}\forall\, i, j, k, \ell \text{ with} \\ 1 \leq j < \ell \leq N; \\ 1 \leq i \leq m_j; \\ 1 \leq k \leq m_\ell\end{array} \tag{7}$$

$$0 \leq c_j x_{ij} - s_j y_{ij} + h_j \leq L \qquad \begin{array}{l}\forall\, i, j \text{ with} \\ 1 \leq j \leq N; \\ 1 \leq i \leq n_j\end{array} \tag{8}$$

$$0 \leq s_j x_{ij} + c_j y_{ij} + v_j \leq W \qquad \begin{array}{l}\forall\, i, j \text{ with} \\ 1 \leq j \leq N; \\ 1 \leq i \leq n_j\end{array} \tag{9}$$

$$-1 \leq c_\ell, s_\ell \leq +1 \qquad \forall\, \ell = 1, \ldots, N \tag{10}$$

$$r_{1\ell} \leq h_\ell \leq L_0 - r_{1\ell} \qquad \forall\, \ell = 1, \ldots, N \tag{11}$$

$$r_{1\ell} \leq v_\ell \leq W - r_{1\ell} \qquad \forall\, \ell = 1, \ldots, N \tag{12}$$

$$0 \leq L \leq L_0 \tag{13}$$

$$s_1 \geq 0 \tag{14}$$

Constraint (6) insures that the variables c_ℓ and s_ℓ will actually be the sine and cosine of some angle. Constraint (7) insures that inscribed circles from different polygons do not overlap. Constraints (8)–(9) force the polygons to lie within a roll of width W and length L (implicitly defining L). Constraint (10) provides the obvious bounds on the cosines and sines of the polygon rotation angles. Constraint (11) provides lower and upper bounds on the horizontal translations; the bounds are set so that the *first inscribed circle* of polygon ℓ, which has radius $r_{1\ell}$, does not extend beyond the left edge of the nest (this requires $h_\ell \geq r_{1\ell}$) and does not extend beyond the right edge of the known feasible nest (this requires $h_\ell \leq L_0 - r_{1\ell}$). Constraint (12) provides similar bound on the vertical translations of the polygons. Constraint (13) provides upper and lower bounds on the length of the nest.

Constraint (14) is a redundancy-breaking constraint. Given any solution to the QP given by (5)–(13), one could always obtain an equivalent solution by rotating the entire nest by

180 degrees. To remove this ambiguity, we can insist that one of the angles (say, θ_1) not have the full range of $[0, 2\pi]$, but only the half range of $[0, \pi]$, which is equivalent to requiring $s_1 \geq 0$.

One reviewer suggested that it might be useful, in Eq. (13), to include a stronger lower bound on the length of the nest than the obvious lower bound of zero. In particular, the reviewer suggested letting the lower bound be the solution to the linear programming relaxation of the QP given in (5)–(14). However, most QP solvers already solve this LP relaxation and use it as a lower bound on the objective. Just to be sure, I tried the suggestion and, as expected, found that it did not help. What *did* help, however, was a tighter *upper* bound on the nest length. This suggests that there would be great value, in practice, to first solving the problem with an efficient heuristic, and then using the best solution found as an *upper* bound on the nest length.

In this example problem, all the parts are unique and rotationally non-symmetric. However, whenever this is *not* the case, there will be *visually equivalent* solutions that correspond to different *numerical* values for the variables. For example, if parts 1 and 6 were the same, we could get two equivalent solutions based on how we number those two parts. Similarly, if one of the parts were a square, one could rotate the square 90 degrees and the solution would look the same but be numerically different. The presence of redundant solutions usually causes a branch-and-bound solver to progress slowly, so it is important to add so-called "redundancy-breaking constraints" that eliminate all but one of the redundant solutions.

We say a part is *rotationally symmetric about its reference point* if rotating it by some amount θ leaves the shape unchanged. For example, a rectangle centered at the origin can be rotated by 180°and will appear the same. For such parts, one can limit the rotation being explored to the range $[0, \theta]$ which, in turn, will lead to tighter lower and upper bounds on the corresponding sine and cosine variables (s_ℓ and c_ℓ). If two parts in the nest are the same, one can break the redundancy by requiring higher-numbered versions to be to the right of lower-indexed ones. For example if polygons 1 and 6 were the same, one could require $h_6 > h_1$.

The quadratic program given in (5)–(14) is of the hardest variety, because both the objective and the constraints are quadratic. Nevertheless, a few exact methods have been developed. For details, the interested reader should consult the chapter by Floudas and Visweswaran in [13], especially section 1. To solve the numerical examples in this paper, I have used three solvers: BARON [26]; LINDOGlobal from Lindo Systems, Inc.; and GloMIQO [23]. All solutions shown in this section will be from BARON; in a later section, we will compare the solvers with respect to computational time, quality of lower bound, etc.

The solution to the quadratic program for the sample problem is shown in Fig. 12. None of the circles from different polygons overlap (this was a constraint) and the length of the nest is 4.99987. Now, while *circles* from different polygons do not overlap, the *polygons* themselves *do* overlap. This is to be expected because we have relaxed the non-overlap constraints for the shapes by replacing them with non-overlap constraints for the inscribed circles. Because we have solved a relaxed problem, its solution is a lower bound on the solution to the nesting problem.

Figure 12 also shows the largest circle that can be inscribed in the overlap between any two polygons in the nest; this circle was found via a custom branch-and-bound algorithm, similar to the one used to inscribe circles (see the "Appendix"). The largest circle in the overlap is quite small, so small that I have had to magnify the picture by a factor of five in order to make it visible. To stop the polygons from overlapping in the next iteration, we need to inscribe new circles in such a way that *these new circles would have overlapped in*

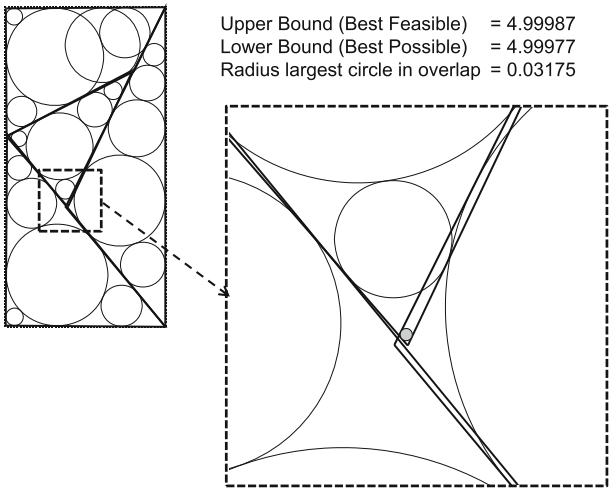


Fig. 12 Solution to the quadratic programming problem for the simple problem in iteration 1. The *lower* and *upper* bound shown refer to the *quadratic programming problem*. Because the quadratic programming problem is a relaxation of the original nesting problem, lower bound is also a valid lower bound on the original nesting problem

the current solution, thereby making the current (overlapping) solution invalid. Perhaps the simplest way to do this is to map the "overlap circle" shown on the right of Fig. 12 back to the reference position of the two polygons, as shown in Fig. 13. The two circles so mapped back would, of course, have overlapped perfectly in the current solution, so they make the current solution infeasible. Figure 13 also illustrates that, after mapping back the circles, it may happen that a circle contacts just one side; in this case, we can grow the circle to a larger circle that contains the original circle, as described earlier (see discussion around Fig. 9b).

From here on we simply iterate, solving the QPs and adding more circles until a stopping criterion is reached. We stop if we meet either of the following two criteria:

1. *We have feasible solution within* $\epsilon > 0$ *of global optimum.*
   To determine if this criterion is satisfied, one performs local search with strict non-overlap constraints, starting from the QP solution. If the local search is successful, and if its objective value is within $\epsilon > 0$ of the lower bound from the solving the QP relaxation, then the criterion is satisfied.
2. *The maximum penetration of one polygon into another is less than* $\tau > 0$.
   Intuitively, this means that any remaining overlap is trivial. The notion of "maximum penetration" of one polygon into another is made precise in the following definition.

Definition 1 Shape A is said to penetrate shape B by an amount τ if there exists a point x in A such that the circle with center x and radius τ lies entirely inside B (see Fig. 14).

In practice, we will most often stop based on the first, traditional criterion. However, it is the second stopping criterion that we will prove, in the next section, must be met in a finite number of iterations. The maximum penetration can be computed using a variation of the same algorithm used for finding the largest inscribed circle (see the "Appendix" for details). Note that the maximum penetration is zero ($\tau = 0$) if and only if there is no overlap. We may now summarize the algorithm as follows:

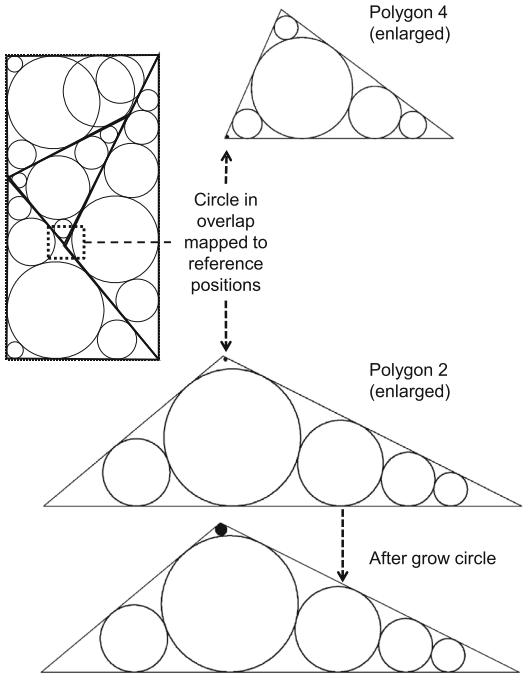


Fig. 13 Mapping the largest circle that can be inscribed in the overlap of two polygons back to the reference position of the polygons. The circle mapped back for polygon 2 only contacted one side, so it was "grown" to a larger circle that contained it, as discussed earlier near Fig. 9b

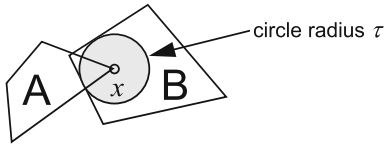


Fig. 14 Shape A is said to penetrate shape B by an amount τ if there exists a point x in A such that the circle with center x and radius τ lies entirely inside B

Algorithm QP-Nest

1. Set convergence criterion $\epsilon > 0$ for closeness to the optimal function value. Set convergence criterion $\tau > 0$ for the allowable amount of polygon penetration. Set the absolute convergence tolerance for the QP solver to a value less than ϵ (e.g., $\epsilon/1000$). Use local search, or a global optimization heuristic, to find a feasible solution with length L_0.
2. Inscribe a few circles in each polygon to get "reasonable coverage," that is, so that no major area of any polygon is left uncovered.
3. Solve the QP relaxation given in Eqs. (5)–(14). If any parts are rotationally symmetric, or the same as other parts, be sure to add the appropriate redundancy-breaking constraints.
4. Perform local search from the QP solution using strict non-overlap constraints for the shapes. If the local search is successful, and if its solution is less than the current value of L_0, then update L_0. If L_0 is within ϵ of the lower bound from solving the QP relaxation, then stop. Otherwise, go to Step 5.
5. Compute the maximum penetration of one shape into another. If this penetration is less than or equal to the convergence tolerance τ, then stop. Otherwise, go to Step 6.
6. Find the single largest circle that can be inscribed in the overlap between two shapes in the QP solution. For each shape that contains this new circle, map the circle back to the reference position. If, when mapped back, the added circle touches just one side of a polygon, grow the circle as explained in Fig. 9b. Go to Step 3.

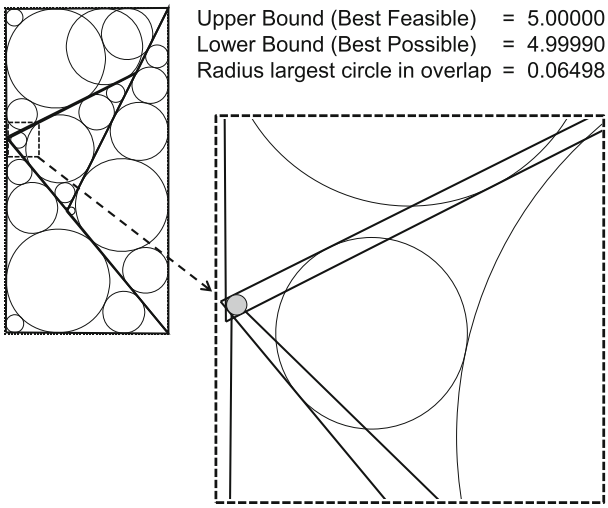


Fig. 15 Solution to the quadratic programming problem for the simple example in iteration 2

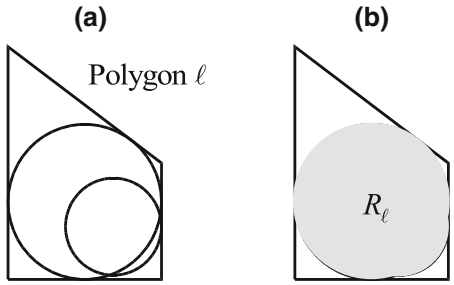


Fig. 16 The region R_ℓ is the subset of polygon ℓ covered by inscribed circles

For our sample problem, local search from the solution in Fig. 12 produces the optimal nest with a length of 5.00000. The best possible value from the QP relaxation is 4.99977, so we would stop if the value of ϵ were less than or equal to 0.00023. In practice, we would most likely stop at this point. To illustrate the method, however, we show one more iteration in Fig. 15. You can see (if you look closely) that the overlap has been removed from where it was on Iteration 1. The lower bound from the QP solver tightens to 4.99990, reflecting the fact that the absolute tolerance for the BARON solver was set to 0.00001.

4 Convergence

The algorithm converged nicely for our simple problem. But will this always happen? Is it possible to iterate indefinitely and never trigger the stopping criterion in Step 5? In short, is the algorithm guaranteed to converge? That is the question we address in this section.

The key to the convergence proof is to consider the subregion R_ℓ of polygon ℓ that is covered by circles—that is, the *union* of all the circles inscribed in polygon ℓ. For the two circles in Fig. 16a, R_ℓ would be the shaded region in Fig. 16b. If we consider all the polygons, then the total area covered by circles is $A = \sum_\ell area(R_\ell)$.

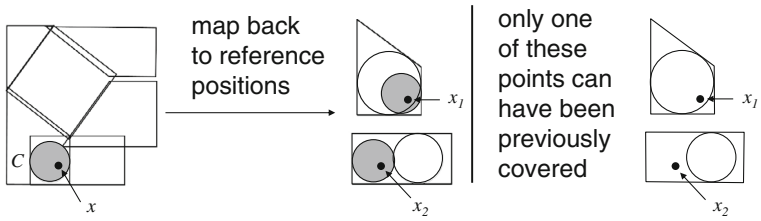


Fig. 17 When a point in the overlap region is mapped back to the reference positions of the shapes, we get *two* points

Clearly, the largest value of A occurs when the circles cover all the shapes completely, in which case A equals the sum of the areas of all the shapes. As we will see, by focusing on A, we can first prove that the area of the largest circle that can be inscribed in the overlap of any two parts (i.e., the circle in Step 6) must go to zero as the iterations proceed. In a second step, we will use this result to prove that the *maximum penetration* of any two parts must also go to zero and, hence, QP-Nest must converge in the limit as the number of iterations goes to infinity. We begin with the following theorem.

Theorem 1 *The area of the largest circle that can be inscribed in the overlap of any two parts (i.e., the circle in Step 6 of QP-Nest) must go to zero as the number of iterations goes to infinity.*

Proof Let C denote the circle that is inscribed in the overlap region in Step 6. Since C is in the overlap region, at least two shapes contain it. For convenience, let us renumber the shapes so that the shapes containing C are shapes 1 and 2. If it happens that there are more than two shapes containing C, pick any two. To illustrate this, Fig. 17 (left) shows a nest of six parts and a shaded circle that is the largest circle that can be inscribed in the overlap. Now when circle C is mapped back to the reference positions of shapes 1 and 2, we will get *two* circles (see Fig. 17, middle). Adding these circles will cover more of the shapes and therefore A will increase. But by *how much* will A increase?

Well, consider any point x in the interior of C. As shown in Fig. 17, when this point is mapped back to the reference positions of shapes 1 and 2, it will map back to two points, which we will call x_1 and x_2.

Now observe that at most one of x_1 and x_2 could previously have been in the interior of an inscribed circle. For suppose this was not true. In this case, there would have existed some circle C_1 in shape 1 that contained x_1 in its interior, and some circle C_2 in shape 2 that contained x_2 in its interior. But since points x_1 and x_2 coincide in the nest, circles C_1 and C_2 would have had a point in common in the nest and would therefore have overlapped. But this is impossible, because circles from different shapes do not overlap in the QP solution. This result is nicely illustrated in Fig. 17, where point x_1 is covered by a previous circle, but point x_2 is not.

Now the two new circles added to the reference positions (the shaded circles in the middle part of Fig. 17) have a total area equal to $2 \times area(C)$. The above argument shows that no more than half of this area could have been previously covered by inscribed circles. Thus, at least half of $2 \times area(C)$—that is, at least $area(C)$—must be newly covered. In short, A must increase by at least $area(C)$.

It immediately follows that the area of the largest circle that can be inscribed in the overlap of two parts—that is, $area(C)$—must go to zero as the iterations go to infinity. For suppose $area(C)$ always stayed above some value $\delta > 0$. Then, in each iteration, A would have to

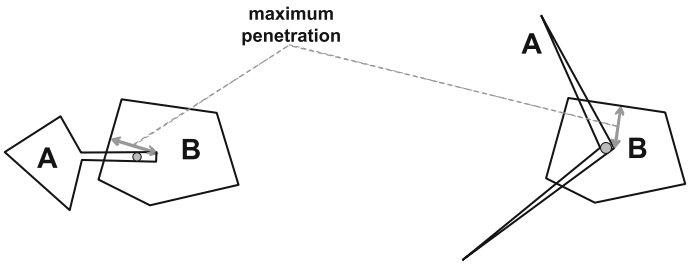


Fig. 18 Two examples where the largest circle that can be inscribed in the overlap region is *small*, yet the maximum penetration of shape A into shape B is *large*

go up by δ. But this is impossible! The value of A can be no larger than the total area of the shapes, and if one kept increasing it by δ on each iteration, it would eventually become infinite. □

Note that the logic of the proof works whether or not the polygons have holes. Note also that in the proof, as well as in Fig. 17, we have ignored any possible "growth" of one of the "mapped back" circles, as discussed earlier in Fig. 9b. Such growth will, if anything, only augment any increase in A, and therefore only strengthen the argument.

Theorem 1 only tells us that, as iterations proceed, the *area of the largest circle that can be inscribed in the overlap region* will become very small. What we must prove, however, is that the *maximum penetration* of one shape into another goes to zero—and the connection between the two ideas is far from obvious. To see why it is not obvious, consider Fig. 18 which shows two examples in which the largest circle in the overlap region is small, and yet the penetration of shape A into shape B is large. These examples tend to make one doubt whether the fact that the area of the largest circle in the overlap region gets small (Theorem 1) is sufficient to prove that the maximum penetration gets small. But this result nevertheless is true. The intuition is the following. The two quantities (maximum penetration and the area of the largest circle) must indeed move together, but the *ratio* depends on the nature of the shapes. The pathological shapes in Fig. 18 simply have the property that the ratio between maximum penetration and the area of the largest circle is large. Nevertheless, as the area of the largest circle goes to zero, so will the maximum penetration.

To actually prove this result, I have had to use the assumption that the shapes are *simple, non-intersecting polygons* (this is all we need; the polygons can also have holes). The assumption that the shapes are *polygons* means that the minimum distance between any two vertices is bounded above zero. The assumption that the polygons are *simple* implies that the distance between any two non-adjacent sides is bounded above zero. The next theorem uses these properties to prove that, as the radius of the largest overlap circle goes to zero, so must the maximum penetration.

Theorem 2 *If the shapes are simple polygons, the maximum penetration between the shapes will go to zero as the number of iterations goes to infinity. Hence, algorithm QP-Nest will converge.*

The proof of Theorem 2 is a bit tedious and is given in the "Appendix". An offshoot of that proof, however, is the following asymptotic bound on the ratio of the maximum penetration of two shapes to the radius of the largest circle in the overlap region. The ratio is a function of the least (most acute) angle $\theta_{\min}$ between any two vertices of the polygon:

$$\frac{\text{maximum penetration of one shape into another}}{\text{radius of largest circle in the overlap of two shapes}} \leq \frac{1+\sin(\theta_{\min}/2)}{\sin(\theta_{\min}/2)} \tag{15}$$

Clearly, this ratio can be large when the shapes have some very acute angles, but it is finite. This does suggest, however, that the QP-Nest algorithm might take longer to converge for shapes that have many acute angles like those in Fig. 18.

5 Computation time

Besides the "basic" problem formulation in equations (5)–(14), I have explored two variations that I call the 'four cases" variation and the "four cases with expansion" variation. These variations help make some of the problem structure more visible to the QP solvers, and so tend to improve performance. In this section, I will describe these modified formulations and compare the performance of three global solvers: LINDOGlobal from Lindo Systems, Inc.; BARON 11.1.0 [26]; and GloMIQO 2.2 [23]. LINDOGlobal was run on the NEOS Server (see http://www.neos-server.org), whereas BARON and GloMIQO were run at Princeton University by gracious permission of C. A. Floudas.

5.1 The "four cases" variation

Constraint (7)—the "non-overlap constraint"—specifies that circle i of polygon j does not overlap circle k of polygon ℓ. Now when either i or k equals one, we are dealing with the *first* circle of a polygon, whose center is located at $(0, 0)$. Because of these zeros, certain terms in the non-overlap constraint vanish when either i or k or both equal one. We can make this fact explicitly visible to the solver by breaking Eq. (7) into four subcases based upon whether $i = 1$ and/or $k = 1$, and then explicitly dropping the terms that vanish due to zero coefficients:

$\begin{aligned}&(h_j - h_\ell)^2 + \\ &(v_j - v_\ell)^2 \\ &\geq (r_{ij} + r_{k\ell})^2\end{aligned}$	$\begin{aligned}&\forall\, i, j, k, \ell \text{ with} \\ &1 \leq j < \ell \leq N; \\ &i = 1; \\ &k = 1\end{aligned}$
$\begin{aligned}&(h_j - c_\ell a_{k\ell} + s_\ell b_{k\ell} - h_\ell)^2 + \\ &(v_j - s_\ell a_{k\ell} - c_\ell b_{k\ell} - v_\ell)^2 \\ &\geq (r_{ij} + r_{k\ell})^2\end{aligned}$	$\begin{aligned}&\forall\, i, j, k, \ell \text{ with} \\ &1 \leq j < \ell \leq N; \\ &i = 1; \\ &k \in \{2, \ldots, m_\ell\}\end{aligned}$
$\begin{aligned}&(c_j a_{ij} - s_j b_{ij} + h_j - h_\ell)^2 + \\ &(s_j a_{ij} + c_j b_{ij} + v_j - v_\ell)^2 \\ &\geq (r_{ij} + r_{k\ell})^2\end{aligned}$	$\begin{aligned}&\forall\, i, j, k, \ell \text{ with} \\ &1 \leq j < \ell \leq N; \\ &i \in \{2, \ldots, m_j\} \\ &k = 1\end{aligned}$
$\begin{aligned}&(c_j a_{ij} - s_j b_{ij} + h_j - c_\ell a_{k\ell} + s_\ell b_{k\ell} - h_\ell)^2 + \\ &(s_j a_{ij} + c_j b_{ij} + v_j - s_\ell a_{k\ell} - c_\ell b_{k\ell} - v_\ell)^2 \\ &\geq (r_{ij} + r_{k\ell})^2\end{aligned}$	$\begin{aligned}&\forall\, i, j, k, \ell \text{ with} \\ &1 \leq j < \ell \leq N; \\ &i \in \{2, \ldots, m_j\} \\ &k \in \{2, \ldots, m_\ell\}\end{aligned}$

The fourth subcase is the same as the original version, but the other three cases are substantially simpler. One might think that a good global solver would notice these possible simplifications in its preprocessing phase, and so there would be no advantage to breaking

out the four subcases explicitly. But the surprising fact is that this is not the case: using the four-cases variation reduces BARON's solution time from 4,940 to 3,344 s, and reduces the number of branch-and-bound iterations from 210,786 to 97,394. (These results, like all those in this section, use an absolute convergence tolerance for the BARON QP solver of 0.01).

5.2 The "four cases with expansion" variation

When one expands Eq. (7), and collects terms, one gets the following equivalent version of the nonoverlap constraint:

$$\begin{aligned}
&a_{ij}^2\left(c_j^2+s_j^2\right)+b_{ij}^2\left(c_j^2+s_j^2\right)+a_{k\ell}^2\left(c_\ell^2+s_\ell^2\right)b_{k\ell}^2\left(c_\ell^2+s_\ell^2\right)\\
&\quad+\left(h_j-h_\ell\right)^2+\left(v_j-v_\ell\right)^2\\
&\quad-2(a_{ij}a_{k\ell}+b_{ij}b_{k\ell})(c_jc_\ell+s_js_\ell)\\
&\quad+2(b_{ij}a_{k\ell}-a_{ij}b_{k\ell})(s_jc_\ell-s_\ell c_j)\\
&\quad-2c_\ell a_{k\ell}h_j+2s_\ell b_{k\ell}h_j+2c_\ell a_{k\ell}h_\ell-2s_\ell b_{k\ell}h_\ell\\
&\quad-2s_\ell a_{k\ell}v_j-2c_\ell b_{k\ell}v_j+2s_\ell a_{k\ell}v_\ell+2c_\ell b_{k\ell}v_\ell\\
&\quad+2c_ja_{ij}h_j-2s_jb_{ij}h_j-2c_ja_{ij}h_\ell+2s_jb_{ij}h_\ell\\
&\quad+2s_ja_{ij}v_j+2c_jb_{ij}v_j-2s_ja_{ij}v_\ell-2c_jb_{ij}v_\ell
\end{aligned}\quad\geq\left(r_{ij}+r_{k\ell}\right)^2 \tag{16}$$

In the above, we see terms like $c_j^2+s_j^2$ and $c_\ell^2+s_\ell^2$, which of course equal 1. So we can simplify (14) to the following:

$$\begin{aligned}
&a_{ij}^2+b_{ij}^2+a_{k\ell}^2+b_{k\ell}^2+\left(h_j-h_\ell\right)^2+\left(v_j-v_\ell\right)^2\\
&\quad-2(a_{ij}a_{k\ell}+b_{ij}b_{k\ell})(c_jc_\ell+s_js_\ell)\\
&\quad+2(b_{ij}a_{k\ell}-a_{ij}b_{k\ell})(s_jc_\ell-s_\ell c_j)\\
&\quad-2c_\ell a_{k\ell}h_j+2s_\ell b_{k\ell}h_j+2c_\ell a_{k\ell}h_\ell-2s_\ell b_{k\ell}h_\ell\\
&\quad-2s_\ell a_{k\ell}v_j-2c_\ell b_{k\ell}v_j+2s_\ell a_{k\ell}v_\ell+2c_\ell b_{k\ell}v_\ell\\
&\quad+2c_ja_{ij}h_j-2s_jb_{ij}h_j-2c_ja_{ij}h_\ell+2s_jb_{ij}h_\ell\\
&\quad+2s_ja_{ij}v_j+2c_jb_{ij}v_j-2s_ja_{ij}v_\ell-2c_jb_{ij}v_\ell
\end{aligned}\geq\left(r_{ij}+r_{k\ell}\right)^2 \tag{17}$$

While we have explicitly given the solver constraint (6), which says that $c_\ell^2+s_\ell^2=1$, some solvers (such as GloMIQO) do not take advantage of such possible cross-equation simplifications when computing bounds. Thus, for such solvers, expanding the terms and enforcing $c_\ell^2+s_\ell^2=1$ may improve performance.

Now equation (17) further simplifies when either i or k equals one. If we break up equation (17) based on our four cases, we get the following version of the non-overlap constraints used in the "four cases with expansion" variation:

$\left(h_j-h_\ell\right)^2+\left(v_j-v_\ell\right)^2$ $\geq\left(r_{ij}+r_{k\ell}\right)^2$	$\forall\, i,j,k,\ell$ with $1\leq j<\ell\leq N$; $i=1$; $k=1$
$+a_{k\ell}^2+b_{k\ell}^2+\left(h_j-h_\ell\right)^2+\left(v_j-v_\ell\right)^2$ $-2c_\ell a_{k\ell}h_j+2s_\ell b_{k\ell}h_j+2c_\ell a_{k\ell}h_\ell-2s_\ell b_{k\ell}h_\ell$ $-2s_\ell a_{k\ell}v_j-2c_\ell b_{k\ell}v_j+2s_\ell a_{k\ell}v_\ell+2c_\ell b_{k\ell}v_\ell$ $\geq\left(r_{ij}+r_{k\ell}\right)^2$	$\forall\, i,j,k,\ell$ with $1\leq j<\ell\leq N$; $i=1$; $k\in\{2,\ldots,m_\ell\}$

$$
\begin{array}{ll}
a_{ij}^2 + b_{ij}^2 + \left(h_j - h_\ell\right)^2 + \left(v_j - v_\ell\right)^2 & \forall\, i, j, k, \ell \text{ with} \\
+ 2c_j a_{ij} h_j - 2s_j b_{ij} h_j - 2c_j a_{ij} h_\ell + 2s_j b_{ij} h_\ell & 1 \le j < \ell \le N; \\
+ 2s_j a_{ij} v_j + 2c_j b_{ij} v_j - 2s_j a_{ij} v_\ell - 2c_j b_{ij} v_\ell & i \in \{2, \ldots, m_j\} \\
\ge \left(r_{ij} + r_{k\ell}\right)^2 & k = 1 \\
\hline
a_{ij}^2 + b_{ij}^2 + a_{k\ell}^2 + b_{k\ell}^2 + \left(h_j - h_\ell\right)^2 + \left(v_j - v_\ell\right)^2 & \\
- 2(a_{ij} a_{k\ell} + b_{ij} b_{k\ell})(c_j c_\ell + s_j s_\ell) & \\
+ 2(b_{ij} a_{k\ell} - a_{ij} b_{k\ell})(s_j c_\ell - s_\ell c_j) & \forall\, i, j, k, \ell \text{ with} \\
- 2c_\ell a_{k\ell} h_j + 2s_\ell b_{k\ell} h_j + 2c_\ell a_{k\ell} h_\ell - 2s_\ell b_{k\ell} h_\ell & 1 \le j < \ell \le N; \\
- 2s_\ell a_{k\ell} v_j - 2c_\ell b_{k\ell} v_j + 2s_\ell a_{k\ell} v_\ell + 2c_\ell b_{k\ell} v_\ell & i \in \{2, \ldots, m_j\} \\
+ 2c_j a_{ij} h_j - 2s_j b_{ij} h_j - 2c_j a_{ij} h_\ell + 2s_j b_{ij} h_\ell & k \in \{2, \ldots, m_\ell\} \\
+ 2s_j a_{ij} v_j + 2c_j b_{ij} v_j - 2s_j a_{ij} v_\ell - 2c_j b_{ij} v_\ell & \\
\ge \left(r_{ij} + r_{k\ell}\right)^2 &
\end{array}
$$

5.3 Pros and cons of the different formulations

The four cases variation is a definite improvement over the basic one. It helps the solver recognize that certain terms with zero coefficients completely vanish, and so avoids the potential wasted time in computing underestimators for such terms. All solvers should do better with the four cases variation than with the basic one.

However, there are pros and cons for expanding each of the four cases. If one does not expand, then the left-hand-side is a sum of squares, and so a good solver will recognize this structure and realize that each squared term is bounded below by zero. By recognizing such structure, the solver may obtain sharper bounds.

On the other hand, if a solver already internally expands the terms before doing any bounding, there may be no advantage to making the sum-of-squares structure clearly visible. In this case, it may be more important to make sure that the solver does not miss the simplification that is possible by enforcing the constraint $c_\ell^2 + s_\ell^2 = 1$.

5.4 Numerical results

Table 1 below compares the QP solution (upper bound and lower bound) and computational time in seconds for the three solvers when applied to the simple four-part problem of Fig. 8a. The results shown are only for the first iteration of QP-Nest, starting with five inscribed circles per part, as in Fig. 11. In all cases, the solvers were given the initial known solution from Fig. 8b, corresponding to a solution value of $L_0 = 6$. The convergence tolerances for the global QP solvers were set to 0.01 (absolute) and 10^{-9} (relative). A time limit of 7 h (25,200 s) was imposed, due to limitations of the NEOS server.

All of these solvers have numerous options that, with different settings, might improve performance. In Table 1, however, we have simply used the default settings. The computational times should be taken with a grain of salt, because I have not controlled for which node was used on the NEOS Server, nor for the load on the servers (number of simultaneously running jobs). Moreover, different computers were used at the NEOS Server (LINDOGlobal) and at Princeton University (for BARON and GloMIQO). Still, the results show that different solvers do best with different problem formulations. For all methods, the "basic" formulation was dominated by one of the other two formulations, in the sense that it performed the same or worse on lower bound and computational time. In fact, the basic formulation actually

Table 1 Results for three global solvers on the four-part test problem using 5 circles per polygon

	Problem formulation		
	Basic	Four cases	Four cases with expansion
BARON			
Upper bound	4.9998	4.9999	4.9998
Lower bound	4.9898	4.9900	4.9898
Solution time in seconds	4,940	3,344	6,642
GloMIQO			
Upper bound	5.6034	5.0000	5.9136
Lower bound	5.0579	4.9900	4.9845
Solution time in seconds	25,200	10,320	25,200
LINDOGlobal			
Upper bound	5.9812	5.6034	4.9999
Lower bound	3.8710	4.2302	4.9990
Solution time in seconds	25,200	25,200	19,003

Table 2 Results for three global solvers on the four-part test problem using 13 circles per polygon

	Problem formulation	
	Four cases	Four cases with expansion
BARON		
Upper bound	5.0000	5.9963
Lower bound	4.9990	4.0372
Solution time in seconds	22,671	25,200
GloMIQO		
Upper bound	5.0000	5.0000
Lower bound	4.9717	4.9990
Solution time in seconds	25,200	20,807
LINDOGlobal		
Upper bound	6.0000	5.8143
Lower bound	3.7207	3.8411
Solution time in seconds	25,200	25,200

confused the GloMIQO solver, which returned a lower bound above the known solution, a clear error.

The ability to use a small number of initial circles is important to making QP-Nest a practical algorithm, because using many circles can substantially increase computational time. To illustrate this, Table 2 shows the results when one uses 13 initial circles per polygon. BARON's solve time for the "four cases" variation increases from 3,344 to 22,672 s. Similarly, whereas LINDOGlobal previously solved the "four cases with expansion" variation to full optimality in 19,033 s, it now ends after 25,200 s with a substantial optimality gap. The effect on GloMIQO was mixed: going from five to thirteen circles led to worse performance (in terms of lower bound) for the "four cases" variation and better performance for the "four cases with expansion" variation.

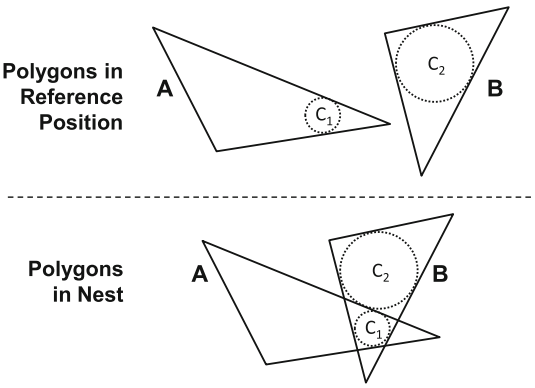


Fig. 19 In this case, the largest circle that can be inscribed in the overlap of the two polygons, when mapped back to the polygon A, will be identical to circle C_1, and hence superfluous

The better performance of GloMIQO in the "four cases with expansion" variation was what we would expect. The reason is that GloMIQO 2.2 internally *always* expands the constraints, so the only difference between the formulations is whether or not one makes the simplification $c_\ell^2 + s_\ell^2 = 1$. In general, making the simplification should be better. Of course, it is always possible for a weaker problem formulation give better results "by luck," because the weaker bounds lead to different branching patterns that just happen to find a good feasible solution earlier in the search. It was probably just this sort of luck that enabled the "plain" four cases variation (i.e., no expansion) to perform better in Table 1 where we used only five circles per polygon. In my experience with the three solvers, the general trend is that BARON does best with the plain "four cases" formulation, whereas GloMIQO and LINDOGlobal perform better with the "four cases with expansion" formulation.

While using a few circles is generally helpful to convergence, I have found that going to the extreme of starting with just *one* circle per polygon is *not* a good idea. In the resulting nest, there will be large overlap between the parts, and one will have to iterate many times before enough circles have been inscribed to prevent substantial overlap. Once this point is reached, the circles so inscribed often overlap quite a bit, and appear to be less efficient at covering of the polygons compared to what one might have achieved with a good initial covering with *several* circles. A practical procedure, as discussed earlier, is to start by inscribing a lot of circles, visually inspect them, and then retain just enough circles so that no major part of the polygon is uncovered.

One reviewer of this paper asked whether it might be possible, as the iterations proceed, to delete some circles that become "superfluous." This is an interesting idea. Figure 19 shows how QP-Nest might actually create such a superfluous circle. In the top of the figure, we show two polygons A and B in their reference positions. Polygon A has one inscribed circle C_1, and polygon B has one inscribed circle C_2. At the bottom of the figure we show the positions of the two polygons in a hypothetical nest. In this case, the largest circle that can be inscribed in the overlap of the two polygons, when mapped back to the polygon A, will be identical to circle C_1, and hence superfluous. Of course, any such completely superfluous circle could indeed be deleted.

In practice, we are unlikely to find such *completely* superfluous circles, but we may very well find that some new circles are *mostly* covered by previously inscribed circles. This leads to the following possible improvement to QP-Nest. Recall that the largest circle inscribed in the overlap of polygons A and B can be mapped back to *two* new circles: one in A and one in B (see Fig. 17). Normally we would keep *both* of these circles. But suppose that, of the two circles, we drop the one that was *most* covered by previous circles whenever the

fraction that was covered by previous circles exceeds some value, say, 80 %. Intuitively, we delete one of the circles when it seems highly superfluous. The convergence proof would still go through: before, the two new circles were guaranteed to cover an amount of virgin territory equal to the area of the overlap circle; if we only keep the circle that covers the *most* virgin territory (or, equivalently, was *least* covered by previous circles), we are guaranteed to cover new territory equal to at least *one-half* of the area of the overlap circle, and this is good enough for the proof. Moreover, the single retained circle will still "cut off" the previous solution: the circle that is *not* retained must have been at least 80% covered by previous circles, and so these previous circles will overlap with 80% of the circle that *is* retained, thereby making the solution invalid. I have not implemented this idea, but it is indeed intriguing.

6 Harder problems

6.1 Milenkovic's rotational containment problem

In his paper on rotational containment, Milenkovic [21] presented a test problem for fabric nesting with four polygons that have 94, 72, 84, and 74 vertices, respectively. These polygons are shown in Fig. 20 along with their initial inscribed circles. To decide *how many* circles to start with, I initially inscribed many circles and visually inspected them to determine how many seemed needed for a good initial approximation; this led to retaining 18, 7, 14, and 18 circles, respectively. The polygons are to be nested on a roll of fixed width $W = 2530$.

Before solving Milenkovic's problem, we will introduce a simple enhancement to the QP-Nest algorithm. Since our container is rectangular, the only vertices of a polygon that can contact the container are those on the polygon's convex hull. Thus, we can replace equations (8) and (9), which constrained the parts to lie on the roll, with the following modified versions:

$$0 \le c_j x_{ij} - s_j y_{ij} + h_j \le L \qquad \forall\, i, j \text{ with } 1 \le j \le N;\ i \in ConvexHull(j) \tag{18}$$

$$0 \le s_j x_{ij} + c_j y_{ij} + v_j \le W \qquad \forall\, i, j \text{ with } 1 \le j \le N;\ i \in ConvexHull(j) \tag{19}$$

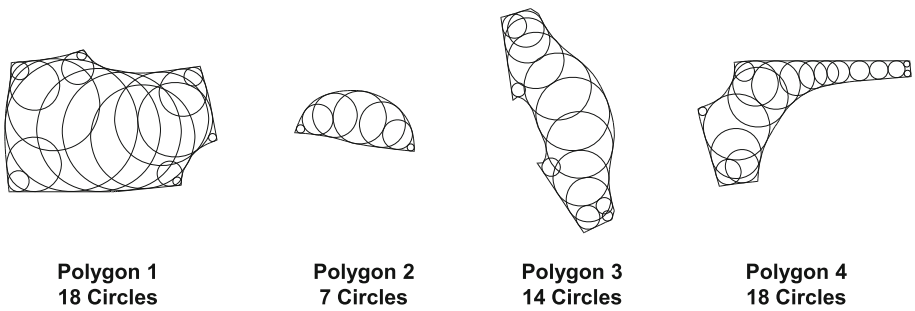


Fig. 20 The four polygons used in Milenkovic's test problem, together with their initial inscribed circles

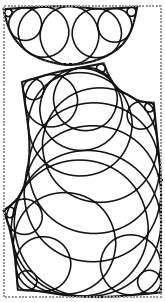


Fig. 21 Solution to Milenkovic's two-part test problem after 3 iterations. There is no overlap. The optimal length is $L = 1364.29$

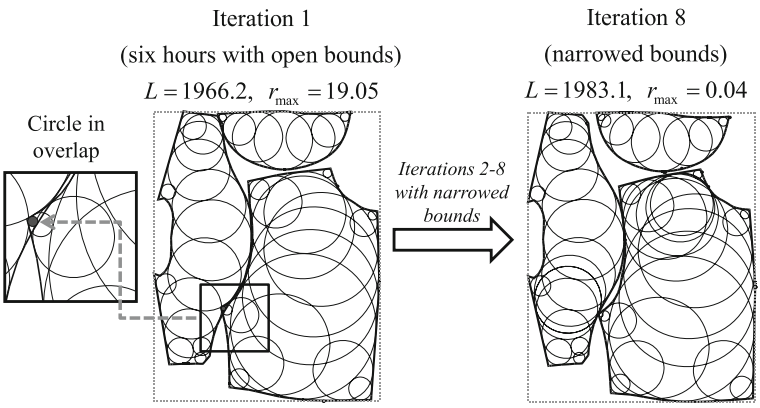


Fig. 22 Solution to Milenkovic's three-part test problem on iterations 1 and 8

The number of vertices on the convex hulls were 23, 52, 23, and 11, respectively, substantially less than the total number of vertices. This enhancement reduces the number of linear constraints and, as a result, modestly speeds up convergence.

Based on my experience with the simple four-part problem, I decided to solve Milenkovic's problems using GloMIQO with the "four cases with expansion" formulation. The quadratic programming problems were solved using an absolute convergence tolerance of 0.1 for GloMIQO.

Milenkovic showed results for nesting just the first two polygons, the first three polygons, and for all four polygons. We will do the same and compare the results of QP-Nest to his solutions.

For the two-part problem, we set the initial feasible nest length to $L_0 = 1403.7$, the length given by Milenkovic for an illustrative *non-optimal* nest. Each iteration solved in about 4 s, and only three iterations were required to remove all overlap and converge. The resulting nest is shown in Fig. 21. The optimal length is $L = 1364.29$, better than the solution of $L = 1385.11$ reported by Milenkovic.

For the three-part problem, we set the initial feasible nest length to $L_0 = 2208.4$, again based on an illustrative non-optimal nest provided by Milenkovic. The first iteration was run for about 6 h (incomplete convergence), producing the nest shown in Fig. 22 (left) with length $L = 1966.2$. We only ran for 6 h because, in the first iteration, there is sure to be some overlap; since more iterations will necessary, so there is no real point in solving to full optimality. The radius of the largest circle in the overlap of two parts is 19.05.

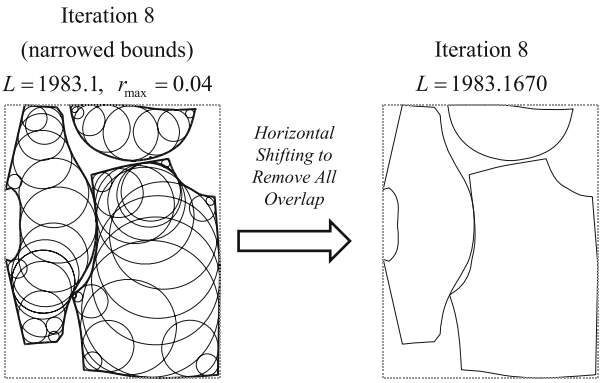


Fig. 23 Using small horizontal shifts, we can remove all remaining overlap from the solution on Iteration 8 of Milenkovic's three-part test problem

At this point, QP-Nest calls for doing "local search with strict non-overlap constraints" to find a feasible solution. However, I had no such local search procedure. Fortunately, it turns out that we can use QP-Nest itself to do this local search by simply continuing to iterate, but *narrowing the bounds on the variables to a small region around the current best solution*. In particular, we set the bounds on c_ℓ and s_ℓ to be the previous solution values ± 0.5 and we set the bounds on h_ℓ and v_ℓ to be the previous solution values ± 50 (about $\pm 2\%$ of the length of the nest). With such a restriction, the quadratic programming problems solve very quickly, often in just a few seconds. It usually took only a few iterations QP-Nest in this "local mode" to drive the amount of overlap to a very small number so that, to the naked eye, and for any practical purpose, we produce a feasible, non-overlapping nest.

In the case at hand, performing iterations 2-8 in this "local mode" resulted in the nest shown in Fig. 22 (right). There is virtually no overlap, as the radius of the largest circle in the overlap region is only 0.04. The nest length is $L = 1983.1$, better than the solution of $L = 2081.8$ reported by Milenkovic. To prove optimality of this solution, of course, we need to go back to using QP-Nest in "global mode," that is, *without* narrowing the bounds. The run converges in 7,631 s, a little over 2 h. The lower bound is 1983.0. To get a valid upper bound, we applied small horizontal shifts to the parts in Fig. 22 (right) in order to eliminate all overlap, resulting in a solution with length 1983.1670 (see Fig. 23). Thus, we have proved optimality to within 0.1670 tolerance. The number of circles in this final QP were 24, 10, and 19, respectively.

For the four-part problem, we ran the first iteration only for about 7.9 h to get the first good solution, shown in Fig. 24 (left). From here we continued using QP-Nest in local mode to get the solution shown in Fig. 24 (right). Surprisingly, the thirteen iterations in local mode changed the solution substantially. In any case, the resulting nest has virtually no overlap, as the radius of the largest circle that can be inscribed in the overlap region is just 0.04. The length is $L = 2613.6$, better than Milenkovic's result of 2699.

To continue, we switch back to "global mode." In this case, in only 7 min we find an improved solution, shown in Fig. 25 (left). Since there is overlap, we again iterate with QP-Nest in "local mode" to drive out overlap and obtain a feasible solution. The local mode iterations finished on iteration 24, shown in Fig. 25 (right). The length 2539.6, substantially better than our previous best of 2613.6.

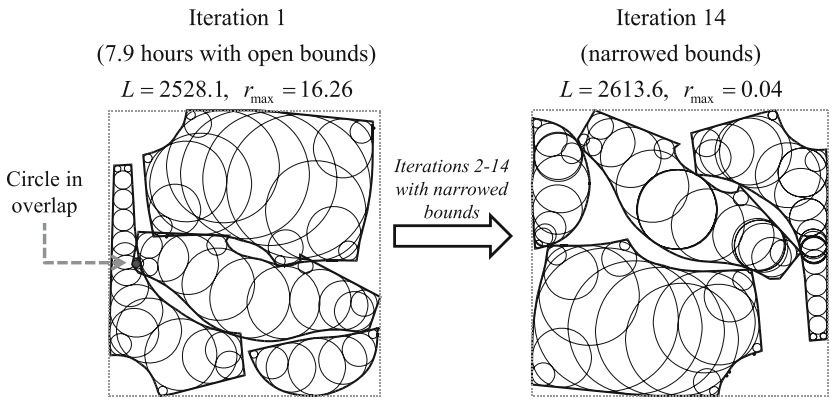


Fig. 24 Best nest found for Milenkovic's four-part problem in iteration 1 (*left*) and in iteration 14 (*right*). Iteration 1 was run just long enough to find a good solution (about 7.9 h). Iterations 2–14 were run just long enough to find a nearly feasible solution (very small overlap) with the bounds on the variables narrowed to a neighborhood around the solution of the previous iteration. These local tuning iterations were fast—about 30 s each

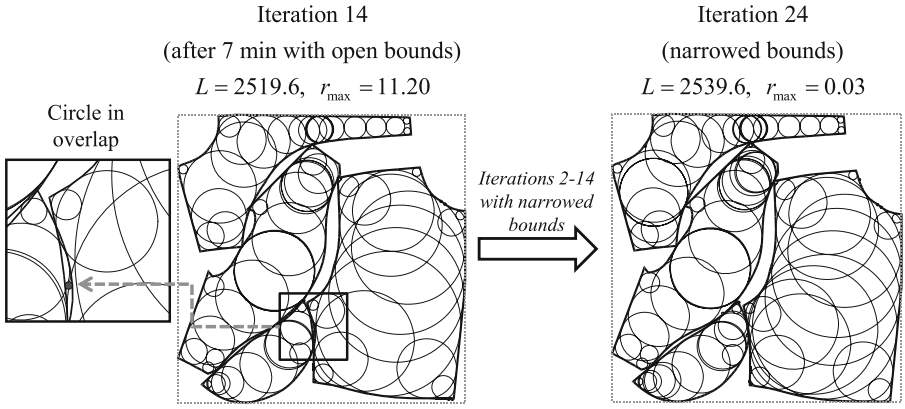


Fig. 25 Improved nest for Milenkovic's four-part problem in iteration 14 (*left*) and in iteration 24 (*right*)

We now return to global mode and, after 2.6 h, we find the better solution shown in Fig. 26 (left). Something interesting now happens: as we iterate with QP-Nest in local mode, the nest becomes longer than our best feasible nest. What this means is that the supposedly "improved" solution on the left of Fig. 26 only achieved its shorter length *by exploiting gaps in the inscribed-circle approximation*. Once these gaps were closed by adding new circles, the solution no longer was better.

Given that the iterations with QP-Nest in local mode failed to find a better feasible solution, we again switch back to global mode. Specifically, this means we solve a new quadratic programming problem using *all* the circles accumulated so far, using the original bounds, and setting the starting point for the variables to the best feasible solution found so far—in this case the solution from Fig. 25 (right). Iterating in this way, we continue until, on iteration 45, we found our best nest shown in Fig. 27 (left) with length 2535.1. This solution still has some overlap, as the radius of the largest circle in the overlap region is 0.04. To eliminate

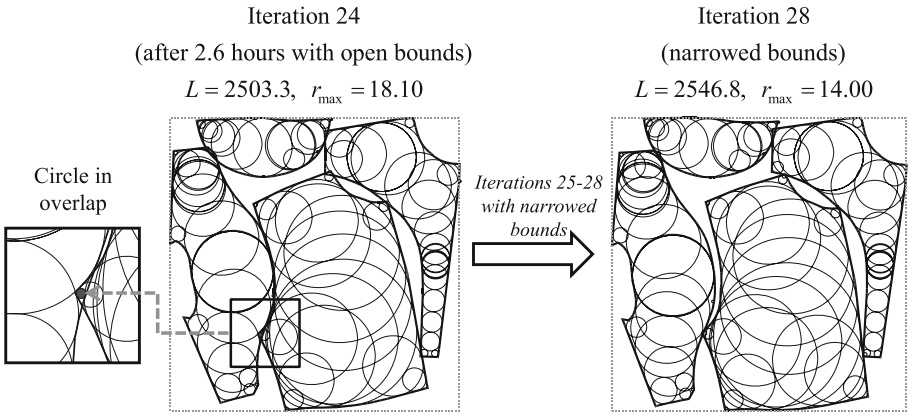


Fig. 26 Results of QP-Nest for Milenkovic's four-part problem in iteration 24 (*left*) and in iteration 28 (*right*)

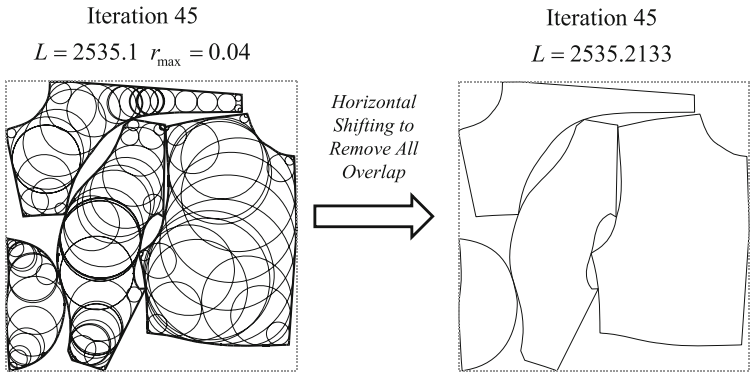


Fig. 27 Best nest for Milenkovic's four-part problem from iteration 45. On the *left*, we show the QP solution. On the *right*, small horizontal shifts have been made to remove all overlap and give a strictly feasible nest

this remaining overlap, we solved for the smallest horizontal shifts that removed this overlap, resulting in the strictly feasible nest in Fig. 27 (right) with length of 2535.2133. At this point, the number of circles per polygon were 40, 21, 48, and 36, respectively.

Having found this solution, we switch back to global mode, but the QP now runs for a very long time without finding a better solution and without converging. Even after 90,000 s, there is still substantial gap between the lower and upper bounds. Thus, for Milenkovic's four-part problem, we have been able to find a very good solution, better than what Milenkovic found, but we have not been able to prove optimality.

6.2 A deceptive six-part problem

We will now turn our attention to a problem that is harder because it has more parts: six instead of just four. The parts are again simple, convex in fact, and the optimal nest is shown in Fig. 28a. Because the parts fit together perfectly to form a 10×10 unit square with zero scrap, we know this is the global optimum. The problem has been designed to be a bit

confusing, because most people would naturally align the square parallel to a side of the nest, but this leads to a nest with significant scrap, as shown in Fig. 28b.

When QP-Nest is applied to this problem (using five initial circles and $L_0 = 12$), the BARON solver, running with the "four cases" variation, stopped after 43 min due to insufficient memory (there was room for 27,795 nodes in 96 MB of RAM). No improvement was found over the starting point.

GloMIQO and LINDOGlobal did better on this problem. LINDOGlobal found the optimal solution using the "four cases with expansion" variation, but could not prove its optimality. GloMIQO provided the best lower bound. The full set of results are shown in Table 3.

As an aside, note that if we assume we know the optimal rotations of the six parts, so that we are solving a pure *translational* nesting problem, QP-Nest easily solves the six-part problem to full optimality. In fact, on the first iteration of QP-Nest with five initial circles, BARON finds a solution with a lower bound of 9.9920—good enough to stop with most any reasonable convergence tolerance—in less than an hour.

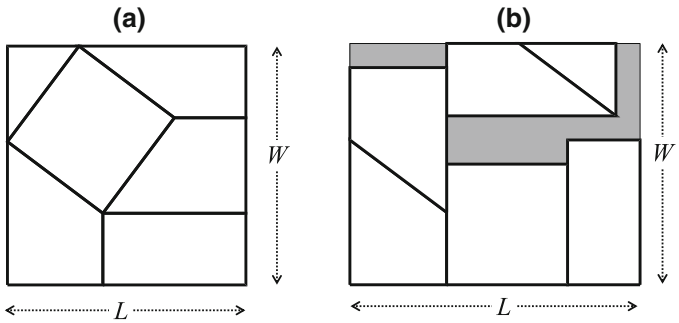


Fig. 28 A deceptive six-part nesting problem on a roll of fixed width $W = 10$: **a** the optimal nest with length $L = 10$; **b** a poor nest with length $L = 12$

Table 3 Results for three global solvers on the six-part test problem (first iteration of QP-Nest with 5 circles per polygon)

	Problem formulation	
	Four cases	Four cases with expansion
BARON		
Upper bound	12.0000	12.0000
Lower bound	5.0000	5.0000
Solution time in seconds	2, 596[a]	25,200
GloMIQO		
Upper bound	11.4369	11.0202
Lower bound	5.4414	5.5225
Solution time in seconds	25,200	25,200
LINDOGlobal		
Upper bound	11.7233	9.9453
Lower bound	5.0000	5.0000
Solution time in seconds	25,200	25,200

[a] Stopped due to insufficient memory; limit was 96 MB

Clearly, finding guaranteed optimal solutions to either Milenkovic's four-part problem, or the six-part problem of this section, will be extremely difficult. However, some progress can be expected due to the continued improvement in global QP solvers. For example, we saw a clear improvement in QP-Nest's performance when we upgraded from from GloMIQO version 2.0 to version 2.2. To facilitate the development of such improvements, I have contributed several test problems to the GloMIQO test suite at http://helios.princeton.edu/GloMIQO/test_suite.html.

7 Discussion

The exact algorithm is made possible by two insights:

- Approximating the shapes with inscribed circles allows the non-overlap constraints to be expressed as closed-form, quadratic constraints.
- One need not initially cover the shapes with many circles. Instead, one can start the iterations with just enough circles so that no major area of the polygon is left uncovered. After that, additional circles can be added sequentially where they are really needed.

In this discussion I will elaborate upon these insights.

Let us start with the idea of inscribing circles. Perhaps the reader has been wondering what is so special about circles. Why not approximate the shapes by triangles, squares, or convex polygons? After all, an approximation by triangles can be exact. Well, all these other approaches *are* possible, but they yield problems with more variables and constraints.

Probably the most promising alternative is to subdivide each shape into convex subpolygons and enforce the constraint that a convex subpolygon from shape A must not overlap with a convex subpolygon from some other shape B. Now two convex polygons are non-overlapping if and only if there exists a line separating them (the separating hyperplane theorem). One can therefore implement the non-overlap constraint by introducing, for every pair of convex subpolygons (k, ℓ), three new variables that define the separating line. For example, if the line is given by $\alpha_{k\ell}x + \beta_{k\ell}y = \gamma_{k\ell}$, then the three new variables would be $\alpha_{k\ell}$, $\beta_{k\ell}$, and $\gamma_{k\ell}$, and we would insist that every vertex of convex subpolygon k satisfy $\alpha_{k\ell}x + \beta_{k\ell}y \leq \gamma_{k\ell}$ and every vertex of convex subpolygon ℓ satisfy $\alpha_{k\ell}x + \beta_{k\ell}y \geq \gamma_{k\ell}$; that is, the two polygons must lie on opposite sides of the line. In fact, this is the approach used by Kallrath [16]. But notice that this way of implementing the non-overlap constraint adds—for each of the *many* pairs of convex subpolygons from different parts—three new variables and as many constraints as there are vertices on the subpolygons. That's a lot of new variables and constraints. In contrast, to enforce the non-overlap of two circles required *no* new variables and only *one* constraint. That's a big difference.

But there's more. The best case for the convex-polygon approach occurs when all the shapes are, in fact, convex. In this case there are $N(N-1)/2$ pairs of convex polygons, so one introduces "only" $3N(N-1)/2$ variables. Moreover, note that the "x" and "y" in the constraint $\alpha_{k\ell}x + \beta_{k\ell}y \leq \gamma_{k\ell}$ are already linear expressions (they are given by equation (3)). So when they are substituted the constraints will have bilinear terms like $\alpha_{k\ell}c_j$. So what will we have achieved? We will have increased the number of variables from order N to order N^2 and increased the number of constraints by a factor related to the number of vertices in each polygon. We indeed get a closed-form problem, but the fact that the number of variables is order N^2 means that the problem becomes intractable for all but the smallest values of N. Hopefully this will convince the reader that circles are indeed special. It seems that no other form of approximation cuts through the complexity of the problem with the same efficiency.

Now consider the second insight, that of introducing circles sequentially and only where they are needed. This leads to an iterative approach and, all else equal, it would be nicer to have a one-shot algorithm. But we gain so much for this small sacrifice. In particular, the resulting subproblems solved during the branch-and-bound algorithm are smaller, and so solve much faster. We saw evidence of this when we compared the solution times for our simple four-part problem using 5 and 13 circles per polygon (Tables 1, 2). When using just 5 circles, all three of our global solvers were able to solve the problem in under 7 h; with 13 circles, no solver was able to do so.

Because the number of circles impacts solution time, we can expect QP-Nest to perform best on "roundish" shapes that require few circles to get a good approximation. Likewise, we can expect worse performance for shapes that are long and skinny or shapes with many protrusions (imagine a starfish), since such shapes will require the most circles. The number of vertices is of less concern, for two reasons. First, when the parts are being nested inside a convex container, only the vertices on convex hull need be considered in the quadratic programming problem. Second, these vertices appear only in the easy-to-handle *linear* constraints (8)–(9). However, the number of *nonconvex* vertices may be significant. The reason is that nonconvexities can lead to parts that interlock like puzzle pieces. In such cases, the translations and rotations must be in a *very narrowly defined region* for the parts to fit together tightly, and so finding this region becomes difficult.

The inscribed circle approach has the further advantage of being easily extended to other problems. For example, the method also applies to three-dimensional bin packing: one merely replaces the circles with spheres. Even problems with irregularly shaped resources that contain defects, such as the cutting of patterns from leather hides, can be handled. One merely adds artificial patterns for the defective areas and for the region outside the resource, and then treats these patterns just as if they were real ones, except that they are not allowed to be translated or rotated.

How many shapes can be nested with QP-Nest? Well, let's review the evidence. Milenkovic's two-part problem was solved in under a minute. Milenkovic's three-part problem was solved in a little over 8 h. The simple four-part example of Fig. 8 was solved in under 3 h of computation time. The six-part problem of Fig. 28 was solvable as a pure *translational* nesting problem (i.e., when the rotations were given) in under an hour. On the other hand, we could only find a good solution, but not prove optimality, for Milenkovic's four-part problem. For the simple six-part problem *with* rotations, LindoGLOBAL was able to find the solution, but no solver came close to proving optimality. Putting it all together, my personal assessment is that QP-Nest will be able to solve most two- or three-part problems to optimality, and will be able to give good solutions (with no proof of optimality) for most four-part problems. For larger problems, good solutions (not proved to be optimal) might be obtained using a sequential approach or metaheuristic that calls QP Nest to solve smaller subproblems.

In the near future, our focus will be primarily on exploring ways to embed QP-Nest into a heuristic for solving larger problems. But who knows, maybe there is still one more insight that will make this problem even easier. We will certainly keep looking for it.

Acknowledgments Lee Bristol of Delphi Interior and Lighting Systems introduced me to the fabric nesting problem, and Bill Crawford of Delphi Interior and Lighting Systems provided Fig. 1 and shared his knowledge on the nature the fabric nesting problem and the relative merits of different nesting approaches. Karen Daniels graciously guided me through the work she has done with Milenkovic and Li. Victor Milenkovic provided me with the data for his four-part test problem. Mary Dalzell of the GM Research library carried out a computerized literature search on the subject of nesting. Paul Sweeney and Bob Haessler of the University of Michigan also helped identify the relevant literature. Special thanks go to Chris Floudas of Princeton University for giving me an account on the Princeton computer system for carrying out the numerical experiments. Thanks also to

Ruth Misener (co-creator of GloMIQO), who provided me with an updated literature search and helped me understand how different problem formulations would affect the performance of GloMIQO. Santosh Tiwari of GM installed LaTeX and BibTeX on my computer and showed me how to use them. Finally, I would like to thank the two anonymous reviewers for their comments and suggestions that pushed me to add clarifications, extend the literature review, and do additional numerical work, all of which substantially improved the paper.

Appendix

Algorithm for inscribing circles and computing maximum penetrations

Figure 29 illustrates the steps in the branch-and-bound procedure for inscribing a circle in a polygon in such a way that: (i) the circle lies entirely inside the polygon and (ii) the circle does not intersect any previously inscribed circles. The figure shows six steps, labeled (a)–(f), that we now explain.

(a) The starting point for the algorithm is a polygon with one or more previously inscribed circles (here just one circle).
(b) We cover the original polygon with triangles. The triangular cover is initialized by finding the bounding box of the polygon and dividing it into two triangles. We have grayed out the original polygon and circle to focus attention on the triangles.
(c) In each iteration, one triangle is selected and subdivided by connecting the midpoint of the longest side to the opposite vertex. Here we show how this is done for the lower right triangle.

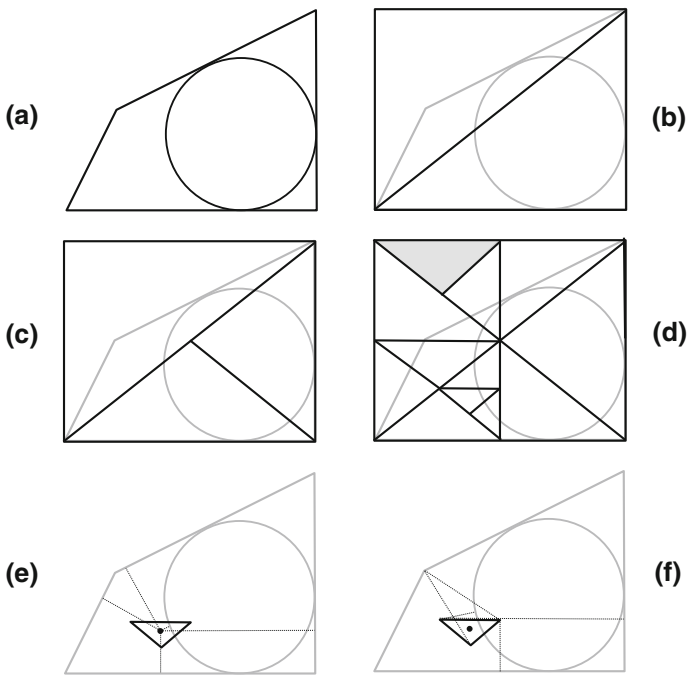


Fig. 29 The steps in the basic circle filling procedure

(d) As the iterations proceed, it is possible that some triangle will have an empty intersection with the polygon; an example is the shaded triangle. Any such triangle is deleted, as it cannot contain the center of any inscribed circle.
(e) For each triangle in the partition, we compute a lower and upper bound on the radius of a circle that has a center in the triangle, lies within the polygon, and does not intersect any previously inscribed circles. Here we illustrate the computation of the lower bound for the thick-lined triangle. We begin by finding the centroid of the triangle, here shown as the black dot. If the centroid lies inside the original polygon (which may not always be the case), we compute the minimum distance from the centroid to each side and to each existing inscribed circle; these distances are shown with dotted lines. The minimum of all these distances is the radius of the largest circle that can be inscribed having the centroid as its center. This is therefore a lower bound on the radius of an inscribed circle with a center in the triangle. We keep track of the largest circle so found, as this is our best feasible solution. If the centroid does not lie in the polygon, we skip the computation of the lower bound.
(f) Here we illustrate the computation of the upper bound for the thick-lined triangle. For each side of the polygon, we determine which vertex of the triangle is furthest from the side (i.e, maximizes the minimum distance between the side and the vertex). In the figure, the dotted lines connect each side to the furthest vertex of the triangle. The furthest distance will be the distance between the triangle vertex and either (i) the closest endpoint of the side or (ii) the projection of the triangle vertex on the side of the polygon. Similarly, for each circle, we compute the distance from the perimeter of the circle to the furthest vertex. Clearly, these are the largest possible distances from a point in the triangle to a polygon side or circle perimeter. The minimum of all these distances is thus a valid upper bound on the radius of an inscribed circle with a center in the triangle that does not intersect any previously inscribed circles.

The entire procedure starts with the initial two triangles in (b), computing the lower and upper bound as discussed. From this point we proceed iteratively, selecting the triangle with the largest upper bound, bisecting it, and computing the bounds for the two child rectangles. Of course, if a child triangle lies outside the polygon, as in (d), we skip the bound calculation and simply delete it. The stopping criterion is the gap between radius of the largest circle successfully drawn from a triangle centroid (our best solution) and the overall upper bound (the largest upper bound of any undeleted triangle). When this gap is sufficiently small, we stop. I normally use a very small tolerance; for example, in the Milenkovic problems the tolerance was 10^{-4}. In general, the tolerance used should be several (say three) orders of magnitude smaller than the accuracy ϵ that one desires for the nest length.

We can compute the *maximum penetration* of polygon A into polygon B using the following modification of the branch-and-bound algorithm:

– The initial bounding box is the bounding box of *both* polygons A and B.
– We delete a triangle if it either has no intersection with polygon A *or* has no intersection with polygon B.
– We only compute a lower bound for a triangle if the centroid lies in both polygons A and B. If this happens, we find the shortest distance from each side of polygon B to the centroid. The minimum of these "shortest distances to the centroid" is the radius of a circle that can be grown from the centroid and still lie within polygon B. It is a valid lower bound on the maximum penetration.

– To compute an upper bound for a triangle, for each side of polygon B, we find the *furthest* vertex of the triangle. The minimium of these "furthest distances" is the upper bound for the triangle.

Finally, we can compute the largest circle that can be inscribed in the overlap of two polygons with the following modifications of the branch-and-bound algorithm:

– The initial bounding box is the bounding box of *both* polygons A and B.
– We delete a triangle if it either has no intersection with polygon A *or* has no intersection with polygon B.
– We only compute a lower bound for a triangle if the centroid lies in both polygons A and B. If this happens, we find shortest distance from each side of polygons A and B to the centroid. The minimum of these "shortest distances to the centroid" is the radius of a circle that can be grown from the centroid and still lie within both polygons A and B. It is a valid lower bound on the radius of a circle inscribed in the overlap of A and B.
– To compute an upper bound for a triangle, for each side of polygons A and B, we find the furthest vertex of the triangle. The minimium of these "furthest distances" is the upper bound for the triangle.

Proof of theorem 2

To set up the proof, for each polygon, assume we have computed the distance between any two vertices and between any vertex and a *non-adjacent* side. Let $s_{\min}$ denote the very smallest of these distances. Because there are only a finite number of vertices and sides, and because the sides do not intersect, the minimum distance $s_{\min}$ must exist and be strictly greater than zero. Also compute the smallest (most acute) angle $\theta_{\min}$ of any vertex. Figure 30 illustrates the computation of $s_{\min}$ and $\theta_{\min}$.

Now suppose Theorem 2 were false. Then there would exist some number $\delta > 0$ with the property that, in every iteration, one shape would penetrate another by at least δ. This means that in every iteration there would exist a point x in some shape A that penetrates some other shape B by δ. By definition, then, the circle with center $x \in A$ and radius δ would lie entirely in B (see Fig. 31a).

Now let $\lambda = \min(\delta, s_{\min}/2)$ and let C denote the circle with center x and radius λ. This circle obviously lies entirely inside B (see Fig. 31b). Moreover, because the diameter of C is less than or equal to $s_{\min}$, at most one vertex of A can lie in C (if there were two vertices inside C, then these vertices would be closer together than $s_{\min}$, which is impossible).

We may now distinguish three cases, as illustrated in Fig. 32. In case 1 there are no vertices of shape A in C. In case 2 there is exactly one vertex and this vertex is *not* x. Finally, in case

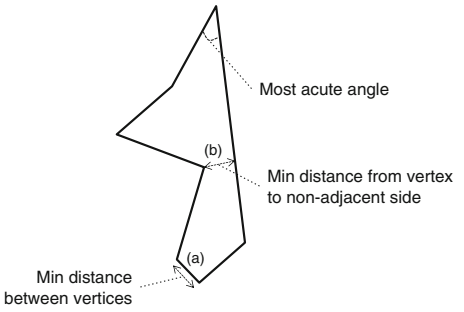


Fig. 30 The value of $s_{\min}$ in the proof of Theorem 2 is the smallest of **a** the minimum distance between any two vertices and **b** the minimum distance between any vertex and a non-adjacent side

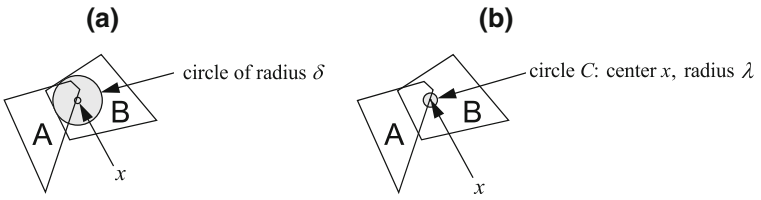


Fig. 31 If Theorem 2 were false, there would always exist some point x in some shape A that penetrates some other shape B by δ

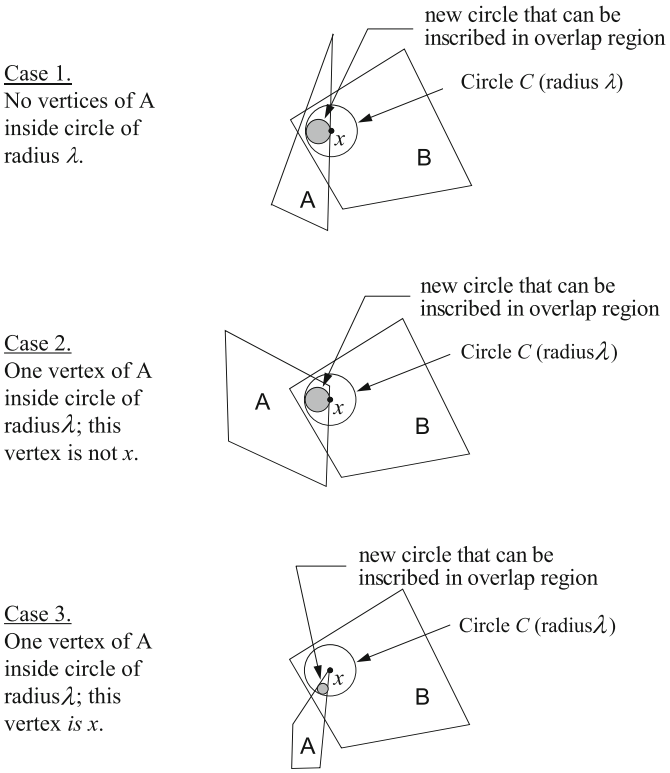


Fig. 32 Three cases considered in the proof of Theorem 2

3 there is exactly one vertex and this vertex *is* x. In all three cases, we can find a circle that lies in the intersection of shapes A and B, as shown in Fig. 32.

Clearly, the smallest overlap of shape A with circle C (and hence with B) occurs in case 3 when the angle at the vertex is the smallest (most acute) angle $\theta_{\min}$. Thus, the smallest possible overlap is a wedge with angle $\theta_{\min}$ from a circle of radius λ. Beside the two sides forming the wedge, no other side from polygon A can be inside circle C, because we have assumed that the radius of circle C is smaller than the smallest distance from a vertex to any non-adjacent side; this is why the entire wedge must be inside shape A and, hence, the entire wedge is in the overlap of polygons A and B. A little trigonometry (see Fig. 33) shows that one can inscribe a circle in this wedge with radius:

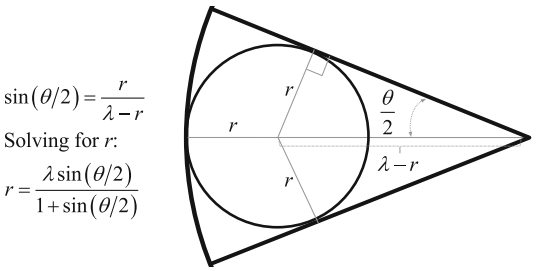


Fig. 33 Computing the radius of the largest circle that can be inscribed in a wedge

$$r = \frac{\lambda \sin(\theta_{\min}/2)}{1 + \sin(\theta_{\min}/2)} > 0. \tag{20}$$

Moreover, this is true no matter what iteration we are on, because we have assumed that in each iteration some polygon penetrates another by at least δ. But if we can always inscribe a circle of radius r inside the overlap region, then the radius of the largest possible inscribed circle would not go to zero, which contradicts Theorem 1!

As an aside, note that "asymptotically," as the iterations proceed and the maximum penetration δ becomes so small that it is less than $s_{\min}/2$, then we will have $\lambda = \min(\delta, s_{\min}/2) = \delta$. Equation (20) then implies that the ratio δ/r_{max}—the ratio of the maximum penetration to the radius of the largest inscribed circle—is indeed bounded, as stated in Eq. (15) in the text. In particular, we have:

$$\frac{\delta}{r_{max}} \leq \frac{\delta}{r} = \frac{\lambda}{r} = \frac{1 + \sin(\theta_{\min}/2)}{\sin(\theta_{\min}/2)} \tag{21}$$

References

1. Albano, A., Sapuppo, G.: Optimal allocation of two-dimensional irregular shapes using heuristic search methods. IEEE Trans. Syst. Cybern. **SMC–10**(5), 242–248 (1980)
2. Avnaim, F., Boissonnat, J.: Polygon placement under translation and rotation. Informatique theorique et applications **23**(1), 5–28 (1989)
3. Bennel, J.A., Oliveira, J.F.: A tutorial in irregular shape packing problems. J. Oper. Res. Soc. **60**(1), 93–105 (2009)
4. Burke, E., Hellier, R., Kendall, G., Whitwell, G.: A new bottom-left-fill heuristic algorithm for the two-dimensional irregular packing problem. Oper. Res. **54**(3), 587–601 (2006)
5. Burke, E., Hellier, R., Kendall, G., Whitwell, G.: Complete and robust no-fit polygon generation for the irregular stock cutting problem. Eur. J. Oper. Res. **279**, 27–49 (2007)
6. Chin, F., Snoeyink, J., Wang, C.A.: Finding the medial axis of a simple polygon in linear time. Discret. Comput. Geomet. **99**(3), 382–391 (1995)
7. Costa, M.T., Gomes, A.M., Oliveira, J.F.: Heuristic approaches to large-scale periodic packing of irregular shapes on a rectangular sheet. Eur. J. Oper. Res. **192**, 29–40 (2009)
8. Daniels, K., Milenkovic, V.: Multiple translational containment: approximate and exact algorithms. In: Clarkson, K. (ed.) Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms (1995)
9. Daniels, K., Milenkovic, V.: Column-based strip packing using ordered and compliant containment. In: Lin, M.C., Manocha, D. (eds.) Proceedings of the First ACM Workshop on Applied Computational Geometry (WACG), pp. 33–38 (1996)
10. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications. Springer, Berlin (2008)
11. Dowsland, K., Dowsland, W.: Solution approaches to irregular nesting problems. Eur. J. Oper. Res. **84**, 506–521 (1995)
12. Farley, A.A.: Mathematical models for cutting-stock problems in the clothing industry. J. Oper. Res. Soc. **39**(1), 41–53 (1988)

13. Floudas, C.A., Visweswaran, V.: Handbook of Global Optimization, chap. Quadratic Optimization, pp. 217–270. Kluwer, Berlin (1995)
14. Heistermann, J., Lengauer, T.: Efficient automatic part nesting on irregular and inhomogeneous surfaces. In: Ramachandran, V. (ed.) Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 251–259 (1993)
15. Kallrath, J.: Cutting circles and polygons from area-minimizing rectangles. J. Glob. Optim. **43**, 299–328 (2009)
16. Kallrath, J., Rebennack, S.: Cutting ellipses from area-minimizing rectangles. J. Glob. Optim. (2013). doi:10.1007/s10898-013-0125-3
17. Lamousin, H., Waggenspack Jr, W.N.: Nesting of two-dimensional irregular parts using a shape reasoning heuristic. Comput. Aided Des. **29**(3), 221–238 (1997)
18. Lasserrre, J.B.: An analytical expression and an algorithm for the volume of a convex polyhedron in R^n. J. Optim. Theory Appl. **39**, 363–377 (1983)
19. Li, Z.: Compaction algorithms for non-convex polygons and their applications. Ph.D. thesis, Harvard University, Department of Computer Science (1994)
20. Li, Z., Milenkovic, V.: Compaction and separation algorithms for non-convex polygons and their applications. Eur. J. Oper. Res. **84**, 539–561 (1995)
21. Milenkovic, V.: Rotational polygon containment and minimum enclosure using only robust 2d constructions. Comput. Geom. **13**, 3–19 (1999)
22. Milenkovic, V.: Densest translational lattice packing of non-convex polygons. Comput. Geom. **22**, 205–222 (2002)
23. Misener, R., Floudas, C.A.: GloMIQO: global mixed-integer quadratic optimizer. J. Glob. Optim. **57**, 3–50 (2013)
24. O'Rourke, J.: Computational Geometry in C, 2nd edn. Cambridge University Press, Cambridge (2001)
25. Preparata, F.P., Shamos, M.I.: Computational Geometry. Springer, Berlin (1985)
26. Sahinidis, N.V.: BARON: a general purpose global optimization software package. J. Glob. Optim. **8**(2), 201–205 (1996)
27. Verkhoturov, M.A., Sergeyeva, O.Y.: The sequential value correction method for the two-dimensional irregular cutting stock problem. Pesquisa Operacional **20**(2), 233–246 (2000)