OPERATIONS RESEARCH
Vol. 54, No. 3, May–June 2006, pp. 587–601
ISSN 0030-364X | EISSN 1526-5463 | 06 | 5403 | 0587

DOI 10.1287/opre.1060.0293

A New Bottom-Left-Fill Heuristic Algorithm for the Two-Dimensional Irregular Packing Problem

Edmund Burke, Robert Hellier, Graham Kendall, Glenn Whitwell
School of Computer Science and Information Technology, University of Nottingham,
Jubilee Campus, Nottingham NG8 1BB, United Kingdom
{ekb@cs.nott.ac.uk, rsh@cs.nott.ac.uk, gxk@cs.nott.ac.uk, gxw@cs.nott.ac.uk}

This paper presents a new heuristic algorithm for the two-dimensional irregular stock-cutting problem, which generates significantly better results than the previous state of the art on a wide range of established benchmark problems. The developed algorithm is able to pack shapes with a traditional line representation, and it can also pack shapes that incorporate circular arcs and holes. This in itself represents a significant improvement upon the state of the art. By utilising hill climbing and tabu local search methods, the proposed technique produces 25 new best solutions for 26 previously reported benchmark problems drawn from over 20 years of cutting and packing research. These solutions are obtained using reasonable time frames, the majority of problems being solved within five minutes. In addition to this, we also present 10 new benchmark problems, which involve both circular arcs and holes. These are provided because of a shortage of realistic industrial style benchmark problems within the literature and to encourage further research and greater comparison between this and future methods.

Subject classifications: search production/scheduling: cutting-stock/trim; production/scheduling: approximations/heuristic; computers/computer science: artificial intelligence.
Area of review: Optimization.
History: Received November 2003; revision received August 2004; accepted April 2005.

1. Introduction

The field of cutting and packing impacts several different industries and motivates many areas of research. Direct applications include the optimisation of layouts within the wood, textile, sheet metal, plastics, and glass industries. Cutting and packing algorithms can also be applied to other spatial problems such as floor plan layouts. The main objectives are to maximise space utilisation and minimise the computation time required. In addition to these *fundamental* objectives there are often *industry specific* requirements which are normally dictated by the material to be cut, the cutting method, and the required cut quality. For example, within the textile industry, consideration must be given to the weave of the cloth and printed designs which is in contrast to the sheet metal industry where heat distribution and material warping are of concern because plasma or oxy-fuel cutting processes are being employed. Research into automatic packing approaches has steadily increased over the years partly due to industrial competitiveness, but also due to greater academic interest from the scientific community (Sweeny and Paternoster 1992).

Gilmore and Gomory (1961) conducted some of the earliest research in the area and solved one-dimensional problems to optimality using linear programming. An example of a one-dimensional problem is the division of steel bars or rods into smaller lengths for fabrication or resale. Two-dimensional problems can be categorised into orthogonal problems (where pieces are rectangular) and irregular problems. Orthogonal problems have received greater attention from the academic community as they are less geometrically complex. A review of work in this area can be found in Hopper and Turton (2001). The best-known results for the established benchmark problems have been achieved using the *best-fit* heuristic which is presented, discussed, and evaluated by Burke et al. (2004). The irregular two-dimensional packing problem is the focus of this paper and is geometrically more complex; therefore, it is more difficult to implement and requires considerably more computational power. A review of approaches to this problem is included in Dowsland and Dowsland (1995). The best results for several benchmark problems (before the results presented here) are found in Gomes and Oliveira (2002), Hopper (2000), and Dowsland and Dowsland (1993). We will briefly discuss each paper that introduced one or more data sets and papers that improved on the best-known result for any of these problems.

Some of the earliest work on irregular packing was conducted by Albano and Sapuppo (1980) who approached the two-dimensional problem using a nesting algorithm which utilised the no-fit polygon (NFP) in addition to a profile simplification method to reduce the geometric complexity of the nesting process (the profile represents the leading edge of the shapes placed on the nest). Available space behind the leading edge is ignored, causing this method to perform as a bottom-left packing algorithm (i.e., with no hole-filling capabilities). The no-fit polygon defines the positions with which two polygons touch without intersecting. This was the first time that the term "no-fit polygon"

was used, but the concept had been introduced in Art (1966), who used the term "shape envelope" to describe the feasible nonintersecting positions for which two shapes can be placed. We further discuss the no-fit polygon in §2.1. Blazewicz et al. (1993) present an extension to the work performed by Albano and Sapuppo (1980). Their method produces a bottom-left-fill style algorithm which performs well against their chosen data sets. The approach attempts to back fill holes in the existing layout before attempting to place a shape on the leading edge of the nest. The technique utilises a tabu search method to produce moves from one nest to another. The authors note that the main problem is in the definition of an algorithm for checking the feasibility of the move produced by the process.

Marques et al. (1991) approached the problem by using a grid approximation to reduce the complexity of the nesting process. This, in combination with a simulated annealing search (to control compaction via Markov chains), produces a result for their data set in just over 24 hours. While it is difficult to determine the exact quality of the produced solution, we have estimated the length of sheet to be 100 units from the provided diagram giving a density of approximately 69%.

Fujita et al. (1993) present a two-part approach consisting of a genetic algorithm (GA) combinatorial search and a local minimisation function for nesting. The GA manipulates shape pairs which hold information about their positional relationship with one another. A hybrid crossover operator produces child chromosomes which are used to create a nest via the local minimisation function; the method then attempts to place these shape pairs in a bottom-left fashion. Overlapping solutions are not allowed but overhang from the required sheet width is acceptable, although penalised in the evaluation function. They only use convex shapes to reduce the time complexity of identifying shape intersections.

Oliveira and Ferreira (1993) discussed two approaches to the problem. Both are driven by a simulated annealing algorithm. The first approach is based on a raster representation of the shapes to be nested. This approximation allows the quick generation of solutions but suffers from inaccuracy, caused by the approximation inherent in the raster representation. The second approach uses a polygon-based representation where *D*-functions (Konopasek 1981) are used to identify and resolve overlap in the solutions. Both methods allow overlap in the solutions, the extent of which is penalised by the evaluation function. The algorithm aims to reduce the overlap to zero, but allows worse moves based on the cooling schedule applied. The second implementation produces feasible results but performs five times slower than the first, which is unable to produce results without overlap.

Dighe and Jakiela (1996) extend the work of Smith (1985), and add geometric extensions to cope with irregular polygons. Their use of a complex genetic algorithm chromosome which is able to avoid the generation of solutions with overlap significantly improves the performance of previous work. Their two-stage approach generates clusters of shapes in valid and tightly-packed positions, which are then manipulated through a tree structure by a GA searching for the best arrangement of those clusters. The generated solutions are evaluated by the area of the rectangular enclosure around the clusters. This paper presents a polygon assembly routine for the clustering of polygons based on similar principles to the overlap resolution routine which forms part of this paper. Dighe and Jakiela's problems are *jigsaw* in nature in that they have a known optimal as they were generated from a single rectangle through a process of subdivision.

Jakobs' (1996) work is aimed at the sheet steel stamping/punching industry. It develops work on rectangle packing from Baker et al. (1980), using a GA approach to improve on the orthogonal results of the earlier work. The paper then extends this to irregular polygons. Jakobs' approach uses polygons in their minimum bounding rectangle orientation and places them using the orthogonal method followed by a compaction phase. The compaction is a stepwise bottom-left algorithm that continues until no shape can be moved any further toward the bottom or left of the sheet, respectively. Jakobs discusses the idea of clustering several shapes, finding the minimum bounding rectangle of these clusters, and packing them orthogonally with the use of the same compaction phase to improve the layout but the results from this approach are not reported.

Bounsaythip and Maouche (1997) utilise an evolutionary algorithm approach which improves on the results from Fujita et al. (1993). The solution uses a comb-nesting algorithm to solve the geometric aspects of the problem. They approach the problem from a textile industry perspective, where the practicalities of cutting mean that strips of a user-defined length across the sheet of fixed width are preferable. This means that direct comparison of results is problematic. The nesting of shapes is again performed on two levels: the lower level uses the comb approach to find the minimum bounding rectangle for two shapes and the upper level employs an evolutionary algorithm to search through a tree of possible combinations. Ratanapan and Dagli (1997) describe an improved approximation method with the aim of reducing the computational time required to generate solutions. The authors report high densities achieved during runs with a higher resolution approximation.

Hopper (2000) was the first practitioner in this area to pull together numerous examples of problems from different papers and attempt to produce results for all the gathered data. A genetic algorithm in combination with bottom-left and bottom-left-fill approaches are reported in this work for both orthogonal and irregular nesting problems. Within the orthogonal field, both guillotine and nonguillotine problems are attempted. The irregular nesting approach also includes results gathered from commercial nesting software, which sometimes outperformed the presented methods. The work contains information on

many authors' approaches. Further to the collection of data from other authors, usually through a process of scanning shapes from illustrations in publications, Hopper has added nine new polygon-based problems to her results. These problems are randomly generated convex and concave polygons ranging from 15 to 75 in quantity.

Oliveira et al. (2000) approach the irregular nesting problem with a no-fit polygon solution that produces good results. They tackle several problems, most of which have been generated by the authors in previous papers (from the fabric-cutting industry). They also tackle one problem from the paper by Blazewicz et al. (1993). They produced new benchmarks (for the time) for the majority of the problems. Their approach involves generating an outline polygon for all shapes already nested onto a sheet. This polygon is then used in the generation of a no-fit polygon for the next shape to be nested. The algorithm is controlled with either the aim of producing a nest with the minimum bounding rectangle or minimum length. The shape will be nested in a nonoverlapping position touching at least one other shape already on the plate, allowing its addition to the profile of the shapes on the sheet before beginning the process again. Oliveira et al. (2000) compare their results against an implementation of the Albano and Sapuppo (1980) algorithm, and against results from Blazewicz et al. (1993) and Dowsland and Dowsland (1993). They report improvements on three of the five attempted problems, the best result being 6.2% better than any known solution and the worst being 4% worse than the best-known solution. The work also provides the data for these problems, in vertex form, as an appendix.

Two further papers have provided improved results for the data sets from the papers discussed above. These are Dowsland and Dowsland (1993) and Gomes and Oliveira (2002). Dowsland and Dowsland (1993) discuss simple geometric techniques for the identification of overlaps in a nested solution and they present several new algorithms which use the geometric techniques. They also present and discuss computational results for the various methods that they develop. One method, in particular, the jostle algorithm, produces excellent results when used to improve an existing nest. The jostle procedure iteratively shifts all shapes to the right and then left most boundaries of the plate. Hole filling is attempted on each move. The method gradually fills holes in the packing and improves the solution. A significant benchmark of 63 unit length was set for the problem SHAPES0 using this algorithm. This represents an improvement of 20.6% on the previous best solution from Oliveira and Ferreira (1993). Other algorithms presented in the work also surpass the previous best solution for SHAPES0 but not to the degree of the jostle algorithm, or with the speed of the jostle algorithm which is able to improve solutions within a "few minutes."

Gomes and Oliveira (2002) develop shape-ordering heuristics for an extended nesting algorithm similar to that in Oliveira et al. (2000). The nesting algorithm is improved by the introduction of the inner-fit rectangle which is the interior no-fit polygon for a shape about to be nested and the rectangle of the sheet on which the nest is generated. The vertices and intersection points of the inner-fit rectangle with the separate no-fit polygons of the shapes already nested, and the shape to be nested, produces all the nonoverlapping feasible positions for the shape. In addition to the extension of the geometric techniques, the paper introduces a 2-exchange heuristic for manipulating the order of shapes which are then nested onto a sheet. The paper presents data from numerous experiments which work from various initial ordering criteria, e.g., random, longest length, and greatest area, using the 2-exchange heuristic to develop better solutions over a number of iterations. The results for the improved geometry and new heuristic set new benchmarks for the shapes1, shapes2, shirts, and trousers data sets. Only the SHAPES0 data set is not improved upon; the best solution still standing at 63 units from Dowsland and Dowsland (1993). However, the authors improve on their own best solution for the problem (Oliveira et al. 2000).

2. The Proposed Approach

In this paper, we propose a new method for implementing a bottom-left-fill packing strategy which utilises new shape primitive overlap resolution techniques. These techniques allow us to quickly produce high-quality solutions by eliminating grid-based inaccuracy in the vertical axis of the packing sheet. Furthermore, the proposed approach allows problems containing circular arcs and shapes with holes to be nested. None of the practitioners discussed in this section present approaches capable of handling nonapproximated circular arcs. By combining the new techniques with both hill climbing and tabu local search methods, we test the proposed method against the 26 existing problems from the literature, then introduce and set benchmarks for 10 new problems, five of which contain circular arcs and three that include shapes with holes.

2.1. Motivation

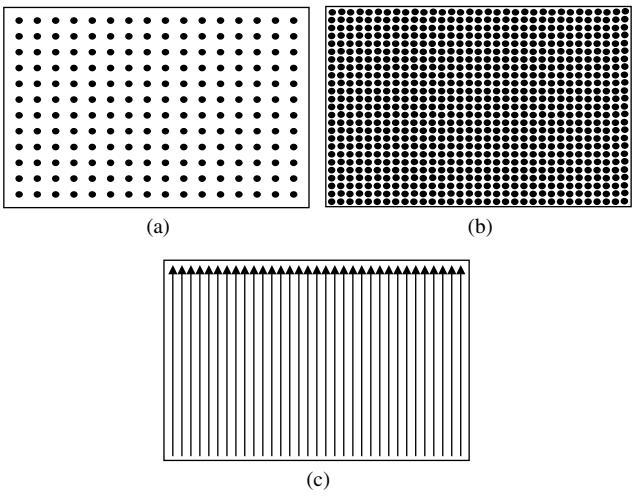
There have been many different approaches for producing solutions for the irregular two-dimensional stock-cutting problem. It should be noted that, in general, the approaches that have achieved the best-known results have used a no-fit polygon-based technique to generate potential placement positions and/or test for overlaps (Gomes and Oliveira 2002). While the no-fit polygon is a powerful geometric technique, there are several issues that make it limited in scalability for industrial applications. No-fit polygon techniques are notorious for the large quantity of degenerate cases that must be handled to be completely robust. There are several well-known cases for which no-fit polygon algorithms *can* fail which include the following: hole filling, shapes with concavities, and *jigsaw-type* shapes, where one shape fits exactly into a concavity from another

shape. Several different techniques have been implemented to generate no-fit polygons: sliding shapes using geometry intersection tests (Mahadevan 1984), convex partitioning of concave shapes (Avnaim and Boissonnat 1987), star-shaped region decomposition of concave shapes (Li and Milenkovic 1995), using Minkowski sums (Daniels and Milenkovic 1997, Flato and Halperin 2000, Agarwal et al. 2002, Bennell et al. 2001), and through the use of Φ-functions (Stoyan et al. 1996, Stoyan and Yaskov 1998, Stoyan et al. 2004). However, as far as the authors of this paper are aware, there have not been any implementations of the no-fit polygon that can successfully handle true arc geometry. Therefore, most methods and benchmark problems have approximated arcs using a sequence of lines. The problem with approximations is that there is an *accuracy to time* trade-off. That is, if we model the arc with fewer lines, then we reduce the accuracy of the approximation but the shape is less complex, and if we increase the quantity of lines, then we improve accuracy but make the resultant shape more complex. When performing translation or rotation operations, obviously shapes with a larger number of lines will take longer to manipulate than shapes with fewer lines. Ultimately, there will always be inaccuracy when modelling arcs as a series of lines, and in adding more lines to our approximation we increase the time required to operate on that approximation. Our geometry implementation is able to model shapes composed of both lines and nonapproximated arcs and is able to conduct fast shape intersection operations to an accuracy of 10^{-9} metres. The development of such a geometry system is a complex and time-consuming undertaking, and the authors would like to suggest that other practitioners may like to consider general geometry libraries such as LEDA and CGAL. Furthermore, continuing research is being conducted into computationally exact geometry systems. However, the additional accuracy obtained through computationally exact approaches is usually at the detriment of computation times. Further information on exact geometry systems can be found in Yap (1997) and rounding considerations in (Guibas and Marimont 1995, Goodrich et al. 1997, Milenkovic 2000, Halperin and Packer 2002).

In real-world industrial settings, and especially for profiling (sheet metal cutting) within the steel-cutting industry, it is imperative that we are able to handle arcs, concavities, and holes. These requirements mean that the no-fit polygon techniques that we have discussed are not currently suitable. Traditionally, industry has used a bottom-left-fill approach based on an iterative grid approximation in which arcs are represented by approximated lines. The grid approach aims to reduce the infinite number of potential positions (due to the continuous nature of space) to a fixed set of potential locations. The algorithm works as follows: when placing a shape on the sheet, we try the first grid location (bottom-left point) and then check for intersection with shapes already assigned to the sheet. If there are no intersecting shapes, then the shape is assigned to the sheet in its current position and we start from the first grid location with a new shape. However, if the current shape does intersect with another shape on the sheet, then we move it to the next grid position and test for intersection again continuing the process until a valid position is found. As we can see, this also means that using a lower resolution grid will, in the general case, adversely affect the quality of the solution because shapes are placed in later positions than they could otherwise be placed if a higher resolution grid was used. Once again, we can see that this causes an *accuracy to time* trade-off. The approach that we describe in the next section is not restricted to moving shapes by a fixed translation when intersecting with another shape, unlike the iterative grid approach. This is achieved by using the underlying geometric primitives of intersecting shapes to resolve the overlap. This has two main advantages: first, we can resolve intersecting shapes so that they touch exactly and, second, this accuracy is achieved in a smaller number of steps than the iterative grid mechanism. Although the grid method and our proposed approach follow a similar conceptual procedure when placing shapes, our proposed approach has both a faster and more accurate method of producing solutions.

Figure 1 shows the potential locations for two iterative grid approaches with differing resolutions and also our proposed overlap resolution method. In the two iterative grid approaches, there are a finite number of locations for which shapes may be placed. The iterative grid approach of Figure 1(b) has a higher resolution than that in Figure 1(a) and therefore has more potential placement locations and should result in more compact packings. The variable shift approach of Figure 1(c) has an infinite number of potential locations due to its continuous y-axis property and therefore has more chance of producing compact packings.

Figure 1. (a) Low resolution grid approach, (b) high resolution grid approach, and (c) variable shift approach.



3. The New Bottom-Left-Fill Algorithm

In this section, we explain the techniques used to resolve overlap. The overall algorithm can be found in §3.4.

3.1. Geometrical Definitions

To illustrate our new packing method, we first define what constitutes a "primitive," "loop," and "shape." For the purposes of our discussions, a "primitive" is defined as either an arc or a line. A line is represented by its start and end points, whereas an arc is circular and is represented through a centre point, radius, start angle, and offset angle. We define a "loop" as an anti-clockwise closed list of primitives, where each primitive's end point is the start point of the next primitive. We do not allow nonsimple polygons (O'Rourke 1998) as we are only interested in shapes that can be physically cut from a sheet of material. A shape is defined as one outer loop and $0 \ldots n$ internal loops which can be thought of as holes in the shape. Most of the problems from the current literature do not include shapes with either arcs or holes and numerous examples only contain convex shapes, where it is easier to detect overlaps. Furthermore, it is necessary for our algorithms to cope with floating-point data to establish high accuracy and realism on real-world problems. Our geometry library can cope with these extra complications. Figure 2 demonstrates an instance of overlap between two shapes, A and B. We can see that arc primitive a2 intersects with line primitive b4 and that the line primitives a3 and b3 also intersect.

3.2. Resolving Overlapping Primitives

This section shows how we can use the information about intersecting primitives to accurately resolve overlap between shapes. There are four possible cases that must be handled to resolve intersecting primitives. One of these primitives is part of the shape that we are trying to place, called the "free shape," and the other is part of a shape that has already been placed on the sheet, called the "locked shape." The techniques we describe in the following sections involve calculating the positive vertical distance required to translate the free shape such that the two primitives no longer intersect. While this resolves the overlap between the two intersecting primitives, the two shapes may still not be fully resolved in that other primitives belonging to the shapes may also intersect or the free shape may be entirely contained in the locked shape (discussed in §4.3). However, repeated application of these techniques will always resolve overlapping shapes with the smallest positive vertical distance required. There are four intersection cases which must be handled: (1) two lines, (2) line and arc, (3) arc and line, and (4) two arcs. It should be noted that throughout all of the cases, the locked shape's intersecting primitive, which we call the "locked primitive," has already been assigned to the sheet and its position may not change, whereas the intersecting primitive belonging to the shape that we are trying to place is termed the "free primitive." We will explain the steps required in resolving the intersections for each of these cases, but first introduce some terminology. The x span of a primitive can be thought of as the horizontal span of its bounding rectangle. Another concept we use is an "infinite vertical line." This is a vertical line that spans from negative infinity to positive infinity along the y-axis. The notations used within the diagrams and descriptions of the following subsections are presented in Table 1.

Having established the required terminology and notation, we explain each of the intersection cases outlined above.

3.2.1. Line and Line (Free Line Moving Through Locked Line). To resolve any two intersecting lines, we find the end points of each line, A and B, that are within the x span of the other. These points are known as the points in range (*pir*). We pass infinite vertical lines through each of the *pir* originating from line A and find the intersection points of these lines with line B. We calculate the distance between each *pir* from line A and its corresponding intersection point on line B by using formula (1):

$$\text{distance}pir\text{A} = \text{intersectionPointB}.y - pir\text{A}_y. \quad (1)$$

We also need to pass infinite vertical lines through each of the *pir* originating from line B to find their intersection

Figure 2. Overlapping shapes.

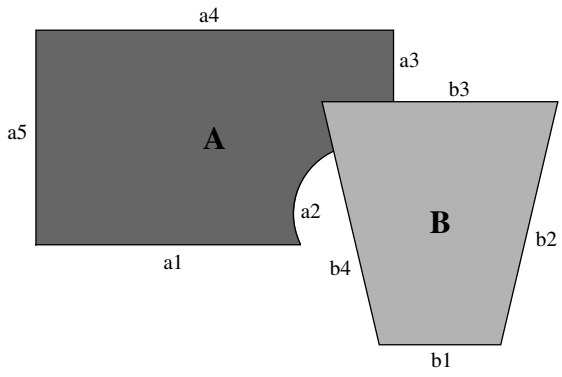


Table 1. Notation for diagrams and descriptions.

Symbol	Description
A1 → A2	Primitive A (the free primitive)
A1, A2	Start point (A1) and end point (A2) of primitive A
B1 → B2	Primitive B (the locked primitive)
B1, B2	Start point (B1) and end point (B2) of primitive B
CP	Centre point of an arc primitive
c1 . . . cn	Intersection points
t1, t2	Tangent points of a line on an arc
┆	Infinite vertical line (short-dashed vertical line)
– – – – –	Perpendicular to a line primitive (long-dashed line)
↑	Translation used to resolve overlap (bold vertical arrow)

points with line A. A different formula, formula (2), is used to calculate the distances when the *pir* originates from the locked primitive (line B):

$$\text{distance}pir\text{B} = pir\text{B}.y - \text{intersectionPointA}_y. \quad (2)$$

For our resolution method, the overlap must always be resolved by translating line A in the positive vertical direction. The distance formulae, given in formulas (1) and (2), may yield negative results, therefore, these results are not valid positive vertical movements and are eliminated. These distance formulae also form part of our strategies for resolving other cases. For intersecting lines, there always exists one valid positive result which can be used to vertically translate line A, thus resolving the overlap. An example of this approach is shown in Figure 3.

Here, point A1 is within the x span of B1 $\rightarrow$ B2, and B2 is within the x span of A1 $\rightarrow$ A2. We label these points *pir*A and *pir*B, respectively (see Figure 3(a)). We pass an infinite vertical line through *pir*A to create intersection point c1 and through *pir*B to create intersection point c2 (see Figure 3(b)). The distance between *pir*A and c1 is calculated using formula (1), and the distance between *pir*B and c2 is calculated using formula (2). In this example, the first distance yields a positive result while the second is negative. Formula (3) shows how we might combine the distance formulae (1) and (2) into a "max" function call:

$$y\text{Translation} = \max(\text{c1}.y - pir\text{A}_y, pir\text{B}.y - \text{c2}_y). \quad (3)$$

The result of formula (3) is shown pictorially as the bold arrow in Figure 3(b). Figure 3(c) shows how the overlap has been resolved by vertically translating line A by this positive translation. In practice, *all* of the primitives of shape A are translated, not just the line involved in the overlap.

Figure 3. Resolution of intersecting line primitives.

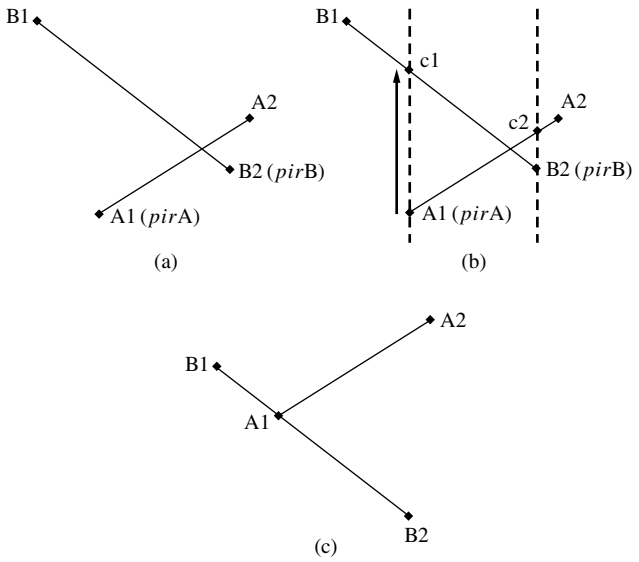
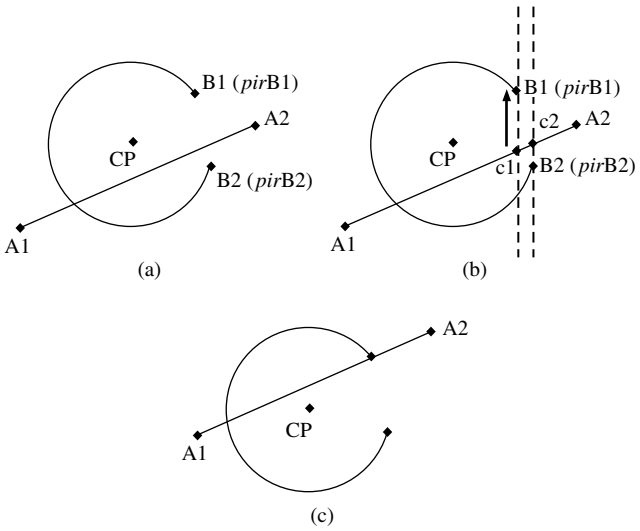


Figure 4. A line and arc example where points in range are insufficient to resolve overlap.

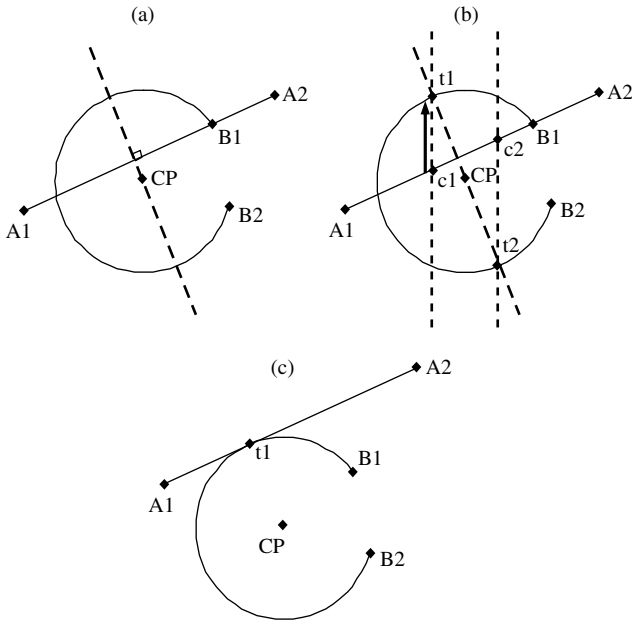


3.2.2. Line and Arc (Free Line Moving Through Locked Arc). In this case, where a line is intersecting with an arc, we must find the positive vertical translation with which the line should be translated to completely resolve its intersection with the arc. As with the Line and Line case (§3.2.1), we can utilise the points in range of each primitive. However, because an arc is involved, we may also need to use tangent points between the arc and line primitives. Figure 4 shows an example where applying the points in range method alone is not sufficient to resolve the overlap.

Figure 4(a) shows that the only points within range are B1 (*pir*B1) and B2 (*pir*B2) from the arc (both end points of the line, A1 and A2, are outside the x span of the arc and, therefore, are not in range). Once again, an infinite vertical line is passed through each *pir* and is intersected with the line A1 $\rightarrow$ A2. This creates intersection points c1 from *pir*B1 and c2 from *pir*B2 (see Figure 4(b)). The distance between each *pir* and its respective intersection point on the other primitive is calculated using the distance formulae (1) and (2). In this example, both *pir* originate from the locked primitive (arc B) and therefore both distances are calculated using formula (2). This yields one positive result that is shown by the bold arrow in Figure 4(b). Figure 4(c) shows that this vertical translation is not sufficient to resolve the overlap. Figure 5 shows how we can resort to the tangent points to fully resolve the overlap.

The tangent points can be found by translating the perpendicular (or "*normal*") of the line such that it passes through the centre point of the arc, CP, as shown in Figure 5(a). The intersection of the perpendicular with the arc gives the tangent point(s), t1 and t2 (see Figure 5(b)). These tangent points are then used in a similar manner to the point in range technique in that infinite vertical lines are passed through each tangent point and intersected with line

Figure 5. Using tangent points to resolve overlap with the line and arc case.



$A1 \rightarrow A2$ to give points c1 and c2. The translation distances can then be calculated by substituting each tangent point, t1 and t2, for *pir*B into formula (2). This gives

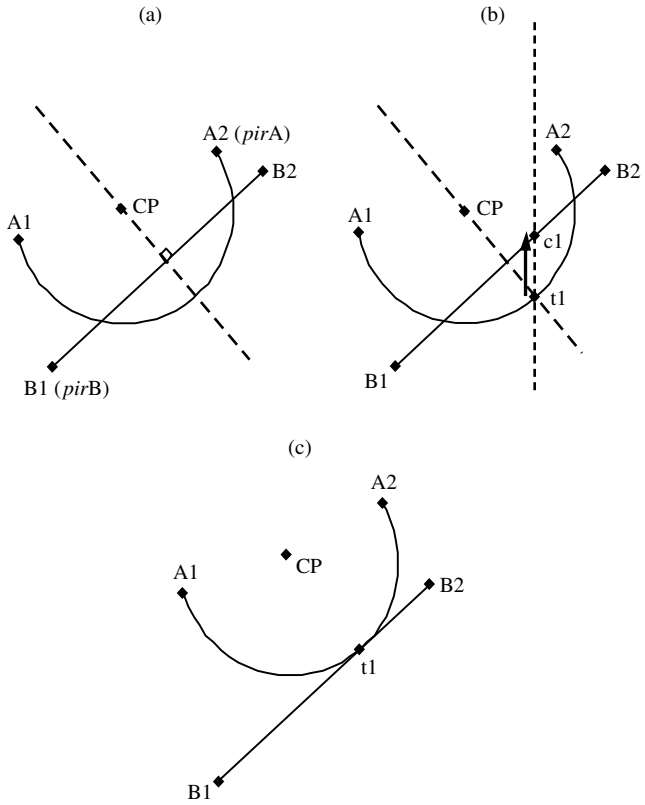
$$\text{distanceTangentB} = \text{tangentB}.y - \text{intersectionPointA}_y. \quad (4)$$

In the example, it can be seen that t1 would yield a positive translation, whereas t2 would give a negative translation distance. Therefore, translating the line $A1 \rightarrow A2$ by the distance given by t1 will resolve the overlap (see Figure 5(c)). It should be noted that if the intersection of the perpendicular line with the arc yields no tangent points or the tangent points result in negative translation distances using formula (4), then the point in range technique *must* be able to resolve the overlapping primitives.

3.2.3. Arc and Line (Free Arc Moving Through Locked Line). This case, where we have an arc moving through a locked line, involves a similar approach to the free line and locked arc case. Once again, the same technique for points in range applies and, therefore, will not be repeated here. However, because the arc is now the free primitive (arc $A1 \rightarrow A2$) and the line is now the locked primitive (line $B1 \rightarrow B2$), we must substitute calculated tangent points and their intersections into formula (1) instead of formula (2) (as was the previous case in §3.2.2). An example of this is shown in Figure 6.

Figure 6(a) shows that points A2 and B1 are the points in range, *pir*A and *pir*B. However, both of the *pir* produce negative translations (using formulae (1) and (2)), thus they cannot be used to resolve the intersection. We must resort to utilising the tangent points method again. In the example,

Figure 6. Resolving overlap using the tangent point method with the arc and line case.



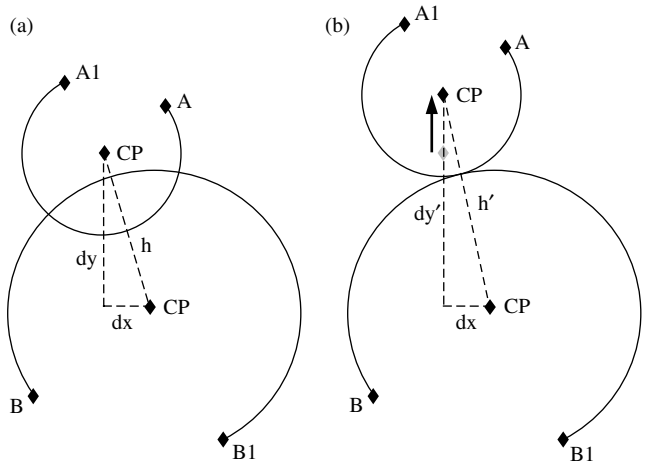
only one tangent point is found because the perpendicular line intersects the arc in only one place. Figure 6(b) shows that an infinite vertical line is passed through the tangent point, t1, and is intersected with line $B1 \rightarrow B2$ to produce point c1. The translation distances can then be calculated by substituting tangent points for *pir*A in formula (1). This gives

$$\text{distanceTangentA} = \text{intersectionPointB}.y - \text{tangentA}_y. \quad (5)$$

Using formula (5) in our example yields a positive translation distance as shown by the bold arrow in Figure 6(b). The intersection is resolved by translating the arc, $A1 \rightarrow A2$, by this vertical distance as shown in Figure 6(c). Once again, if the tangent points method does not find tangent points or does not yield a valid positive result, then the points in range method will be able to resolve fully.

3.2.4. Arc and Arc (Free Arc Moving Through Locked Arc). The arc through arc case initially uses the point in range technique. We shall not explain this in detail as it is identical to the technique used throughout the previous cases. For the situation where the point in range method is unable to resolve the intersection between the two arc primitives, we use two circle tangent methods that utilise the radii of the arcs and the Pythagorean Theorem.

Figure 7. Using the Pythagorean Theorem to resolve arc and arc intersections (method 1).



An example of intersecting arcs in opposite orientations is shown in Figure 7.

Given that rA is the radius of the free arc A1 → A2, and rB is the radius of the locked arc B1 → B2, we can make the following observations:

From Figure 7(a), when the arcs are intersecting, $(rB - rA) < h < (rA + rB)$, and

From Figure 7(b), when the intersection has been resolved, $h' = (rA + rB)$,

therefore,

$$y\text{Translation} = (dy' - dy),$$

$$\text{where } dy' = \text{sqrt}((h' * h') - (dx * dx)). \quad (6)$$

This intersection can then be resolved by translating arc A by the result of formula (6).

A further arc and arc case we may have to solve involves two arcs of similar orientation as shown in Figure 8.

Given that rA is the radius of the free arc A1 → A2, and rB is the radius of the locked arc B1 → B2, the following observations can be made:

From Figure 8(a), when the arcs are intersecting, $(rB - rA) < h < (rA + rB)$, and

From Figure 8(b), when the intersection has been resolved, $h' = (rB - rA)$,

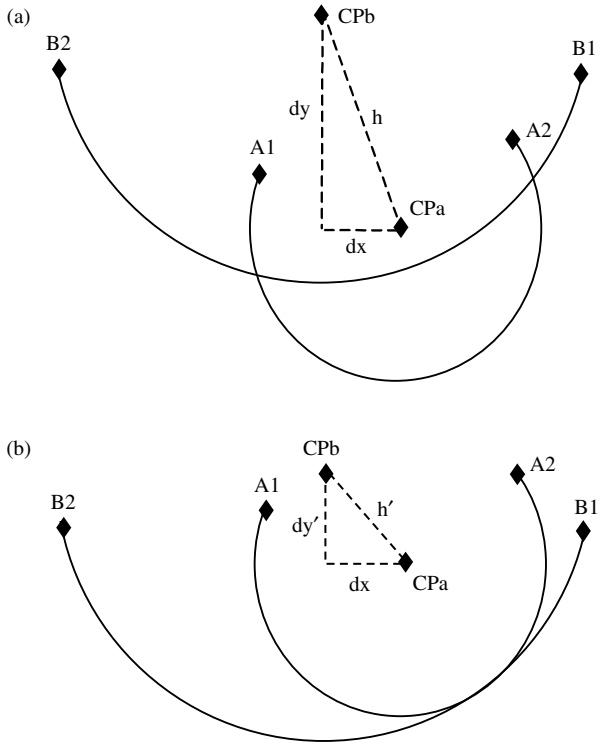
therefore,

$$y\text{Translation} = (dy - dy'),$$

$$\text{where } dy' = \text{sqrt}((h' * h') - (dx * dx)). \quad (7)$$

If the result of formula (7) is positive, applying this vertical translation to arc A will resolve the overlap. It can be seen that, whereas the first circle tangent resolution method translates the free arc to the exterior of the locked arc circle, the second method translates the free arc to be inside of the arc circle. This is imperative for the correct manipulation of both convex and concave arcs.

Figure 8. Using the Pythagorean Theorem to resolve arc and arc intersections (method 2).



3.2.5. Intersection Resolution Summary. We have detailed the four possible intersection cases and have shown that each case can be resolved by using the points in range method by using formulae (1) and (2). This will always resolve the intersection where two lines are involved (see formula (3)). If arcs are involved, the point in range method may not be sufficient to resolve intersections fully, and supplementary tangent-based techniques may be employed. When an arc and line are intersecting, we use the perpendicular of the line primitive, displace it through the arc's centre point, and intersect it with the arc to find tangent points. Where the arc is the "locked primitive" and the line is the "free primitive," we use formula (4) to calculate the vertical translation required. Formula (5) is utilised when the arc is the "free primitive" and the line is locked. The final case, where two arcs are intersecting, introduced two circle tangent methods that can resolve intersections. The first method is calculated by formula (6) and results in the respective arc's parent circles being separate. The second method results in the respective free arc's parent circle being contained by the locked arc's parent circle (formula (7)). The least expensive of the cases is where two lines are involved, as no extra tangent calculations are required. This presents optimisation possibilities (which we include in our implementation) whereby, if there are many intersecting pairs of primitives between two shapes, we resolve the line only cases first. Although repeated applications of these techniques may be necessary to fully resolve

the overlap between two shapes, this is more efficient and accurate than the iterative grid method outlined in §2.1.

3.3. Contained Shapes

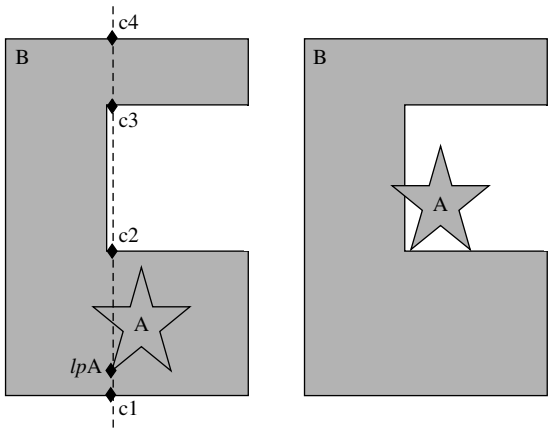
We now detail the special invalid case in which one shape is completely contained within another. This requires another resolution strategy as there are no intersecting primitives. During the nesting process, it is possible that a shape may become entirely contained within other already assigned shape. In this circumstance, there are no intersecting primitives to resolve so another approach is required. As the free shape A is contained by locked shape B, we use the lowest point on shape A, lpA, through which we then pass an infinite vertical line. The resulting intersection of the infinite vertical line and shape B gives us points c1, ..., cn. The translation we perform is defined by

$$y\text{Translation} = \min(\text{c1}.y - \text{lpA}_y, \ldots, \text{cn}.y - \text{lpA}_y), \quad \text{where } (\text{c}i.y - \text{lpA}_y) > 0,\ i = 1, \ldots, n. \quad (8)$$

Figure 9 shows an instance where employing this technique does not fully resolve overlap between the shapes. However, shape A is no longer contained by shape B and there are intersecting primitives once more allowing the techniques for resolving primitive intersection to be employed. This process continues until the shape intersection is completely resolved.

In resolving shape intersections, the vertical translations employed may cause the free shape to intersect with other already assigned shapes. These intersections must also be resolved until the shape does not intersect and can be assigned to the sheet, or until the shape has moved off the top of the sheet. In the latter case, the shape must be translated back to the bottom of the sheet in the next x coordinate, and the process continues until all of the shapes have been placed.

Figure 9. Contained shape where overlap is not fully resolved.



3.4. Summary—The Bottom-Left-Fill Algorithm

Given the functionality outlined in this section, we can describe the bottom-left-fill process using the following pseudo code:

```
Input Sequence: shape[1..m][1..n],
  where m = number of different shapes,
        n = number of rotations for given shape[m]
sheetshape[1..q], where array holds placed shapes on the sheet
q = total number of shapes assigned to the sheet
x = current x position
resolution = x increment step value

Begin
q = 1;
// place first shape
Place shape[1][1] at bottom left of sheet (0, 0);
sheetshape[q] = shape[1][1];
q++;   // increment shapes added counter
for (i = 2; i ⩽ m; i++)   // start remaining shape placement
{
  for (j = 1; j ⩽ n; j++)   // pack each allowable rotation, j
  {
    place shape[i][j] at bottom left of sheet;
    // find feasible position
    while (Overlap(shapes[i][j], sheetshape[1..q]))
    {
        Get Intersecting Primitives of shape[i][j] and
          sheetshape[1..q];
        Resolve Overlapping primitives;
        if (shape[i][j] off top of sheet)
        {
            x = x + resolution;
            place shape[i][j] at (x, 0);
        }
    }
    if (shape i in orientation j is the least costly
        orientation seen so far)
    {
        // record best orientation seen so far
        best orientation = j;
    }
  }
  // assign shape i in best orientation to sheet
  sheetshape[q] = shape[i][best orientation];
  q++;
}
Return Evaluation (total length of packing);
End
```

This bottom-left-fill placement algorithm takes a sheet size and an input sequence of shapes and their allowable rotations. The algorithm progresses packing by placing the first shape in the lower left corner of the sheet in its most efficient orientation (the orientation that yields the smallest bounding rectangle width within the set of rotation criteria). With subsequent shapes, if a copy of this shape has not been placed on the sheet, the shape starts at the lower left corner of the sheet. If a copy of the shape has previously been assigned to the sheet, then the new copy starts from where the previous copy of the shape was placed. A valid location for placement is found by testing for intersections and containment. If the shape is not intersecting or contained by (or containing) other already placed shapes,

then the location of the shape is valid and therefore can be assigned to the sheet. When a shape is in a position that intersects with already assigned shapes, we use the resolution techniques described earlier in this section to resolve the overlap in a positive vertical direction (up the y-axis of the sheet). If resolving overlap results in the shape moving off the top of the sheet, then it is returned to the bottom of the sheet and is incremented along the positive x-axis by a certain value (known as the resolution). The process continues as before with overlap/intersection tests and resolution until the shape does not intersect and can be placed. Packing is completed when all shapes have been assigned to the sheet and the solution can be returned to the user. Shapes are always packed in the order they appear in the input sequence.

3.5. Local Search

It is usual to apply some sorting criteria to the shapes of a given problem before packing, often by decreasing length or area. Although these often yield solutions of reasonable quality, further improvements can be found if a local search mechanism is applied to generate new input orderings. This approach was used during our experiments in §5 by applying hill climbing and tabu search for both area and length pre-sorted arrangements.

We use a standard hill-climbing algorithm during our experiments which simply applies operators to the current solution to find a neighbour of increased quality. If an improved neighbour is found, it is adopted as the current solution and the search continues. If the neighbour is not an improvement on the current solution, it is discarded and the search continues with other neighbours. The best solution is returned at the end of the search. The tabu search mechanism implemented for our experiments is similar in nature to that described in Glover (1993). The process generates a given number of neighbours and moves to the best solution in this subset of the neighbourhood. This best solution is then used to generate the next set of neighbours and the cycle continues. The use of a tabu list means that we will not revisit recently seen solutions within a given list length. Throughout this paper, we use a neighbourhood size of five solutions and a tabu list length of 200 solutions. These values were chosen from a set of possible values during initial experiments. The operators used throughout both search techniques are 1 Opt, 2 Opt, 3 Opt, 4 Opt, and N Opt. 1 Opt removes a randomly chosen shape and inserts it at a random location in the sequence. 2 Opt swaps two randomly chosen shapes in the order, although not two of the same type as this would produce the same result. This is extended to 4 Opt, where four randomly selected shapes are swapped. N Opt selects a random number of shapes to swap and is likely to produce a radically different solution, and thus diversify the search. The solution operator is chosen by means of a random number selected, within bounds, that is then compared to a weighted scale, which gives the particular operator to be used. Each operator has a different chance of selection—from 1 Opt which has the largest chance of being selected, to N Opt, which has a much lower chance of being selected. This is because the less radical operators allow us to concentrate our search and the highly radical operators, e.g., N Opt, enable us to escape local optima.

The following pseudocode shows how both of our local searches interface with bottom-left-fill:

```
INPUT Problem Shapes, Quantities and Allowable
        Rotations, Sheet Size
      current.ordering = Sort Ordering(decreasing area,
                                       decreasing length)
Begin
current.evaluation = Bottom-Left-Fill(current.ordering);
best = current;
// Stopping criteria can be based on the numbers of iterations
  or time
while (!Stopping Criteria)
{
  opt = Select Operator (1, 2, 3, 4, N Opt);
  if (TABU)
  {
    for (i = 0; i < neighbourhood size; i++)
    {
      neighbour[i].ordering
        = Generate NotTabu Neighbour(current ordering, opt);
      neighbour[i].evaluation
        = Bottom-Left-Fill(neighbour[i].ordering);
    }
    current = GetBestNeighbour(neighbour[ ]);
  }
  else if (HILL CLIMBING)
  {
    neighbour.ordering
      = Generate Neighbour(current.ordering, opt);
    neighbour.evaluation = Bottom-Left-Fill(neighbour.ordering);
    if (neighbour.evaluation ⩽ current.evaluation)
      { current = neighbour; }
  }
  if (current.evaluation < best.evaluation) { best = current; }
}
return best;
End
```

4. Benchmark Problems

4.1. Benchmark Problems from the Literature

To compare our new bottom-left-fill algorithm with the state of the art, we have gathered all the relevant test problems from the literature (of which we are aware). Some of these problems were first collected by Hopper (2000), and are now featured on the EURO Special Interest Group on Cutting and Packing (ESICUP) website (http://www.apdio.pt/sicup/).

Hopper (2000) introduced 10 new problems, nine of which were randomly generated and consist of varying quantities of similar polygonal shapes (note that for these nine "poly" problems, it is necessary to rotate the shapes into their minimum bounding rectangle orientation before applying the problems' rotation criteria). Oliveira et al. (2000) present five new problems, three of which are drawn

Table 2. Length evaluated benchmark problems from the literature.

Original authors	Problem name	No. of shapes	Rotational constraints	Sheet width	Best length	Best-result reference
Blazewicz et al. (1993)	Blasz1	28	0, 180 Absolute	15	27.3	Gomes and Oliveira (2002) [called SHAPES2]
Ratanapan and Dagli (1997)	Dagli	30	90 Incremental	60	65.6	Hopper (2000) [using NestLib]
Fujita et al. (1993)	Fu	12	90 Incremental	38	34	Fujita et al. (1993)
Jakobs (1996)	Jakobs1	25	90 Incremental	40	13.2	Hopper (2000) [using SigmaNest]
Jakobs (1996)	Jakobs2	25	90 Incremental	70	28.2	Hopper (2000) [S. annealing]
Marques et al. (1991)	Marques	24	90 Incremental	104	83.6	Hopper (2000) [Naïve evolution]
Hopper (2000)	Poly1A	15	90 Incremental	40	14.7	Hopper (2000) [using NestLib]
Hopper (2000)	Poly2A	30	90 Incremental	40	30.1	Hopper (2000) [using NestLib]
Hopper (2000)	Poly3A	45	90 Incremental	40	40.4	Hopper (2000) [using NestLib]
Hopper (2000)	Poly4A	60	90 Incremental	40	56.9	Hopper (2000) [using NestLib]
Hopper (2000)	Poly5A	75	90 Incremental	40	71.6	Hopper (2000) [using NestLib]
Hopper (2000)	Poly2B	30	90 Incremental	40	33.1	Hopper (2000) [using SigmaNest]
Hopper (2000)	Poly3B	45	90 Incremental	40	41.8	Hopper (2000) [using NestLib]
Hopper (2000)	Poly4B	60	90 Incremental	40	52.9	Hopper (2000) [using NestLib]
Hopper (2000)	Poly5B	75	90 Incremental	40	63.4	Hopper (2000) [using NestLib]
Hopper (2000)	SHAPES	43	90 Incremental	40	63	Hopper (2000) [Simple heuristic]
Oliveira and Ferreira (1993)	SHAPES0	43	0 Absolute	40	63	Dowsland and Dowsland (1993)
Oliveira and Ferreira (1993)	SHAPES1	43	0, 180 Absolute	40	59	Gomes and Oliveira (2002)
Oliveira and Ferreira (1993)	SHIRTS	99	0, 180 Absolute	40	63.13	Gomes and Oliveira (2002)
Oliveira et al. (2000)	SWIM	48	0, 180 Absolute	5,752	6,568	Hopper (2000) [using NestLib]
Oliveira et al. (2000)	TROUSERS	64	0, 180 Absolute	79	245.75	Gomes and Oliveira (2002)

Table 3. Density evaluated benchmark problems from the literature.

Original authors	Problem name	No. of shapes	Rotational constraints	Sheet width	Best density (%)	Best-result reference
Albano and Sapuppo (1980)	Albano	24	90 Incremental	4,900	86	Hopper (2000) [S. annealing]
Blazewicz et al. (1993)	Blasz2	20	90 Incremental	15	68.6	Blazewicz et al. (1993)
Dighe and Jakiela (1996)	Dighe1	16	90 Incremental	100	72.4	Hopper (2000) [using NestLib]
Dighe and Jakiela (1996)	Dighe2	10	90 Incremental	100	74.6	Hopper (2000) [Gen. algorithm]
Bounsaythip and Maouche (1997)	Mao	20	90 Incremental	2,550	71.6	Hopper (2000) [Gen. algorithm]

from the textile industry. Blazewicz et al. (1993), Jakobs (1996), and Dighe and Jakiela (1996) introduce two problems each to the collection. The remaining problems have been contributed by different practitioners. Table 2 shows 21 problems and provides the best-known results using a length-based evaluation. Table 3 contains five problems, where the best-known solution is evaluated by density measures.

4.2. New Benchmark Problems

We have created 10 new benchmark problems for the irregular stock-cutting problem to encourage further research and greater comparison between current and future methods. Table 4 gives the details of these new problems. The new benchmark data can be found in Appendix C of the Online Collection at http://or.pubs.informs.org/Pages/collect.html.

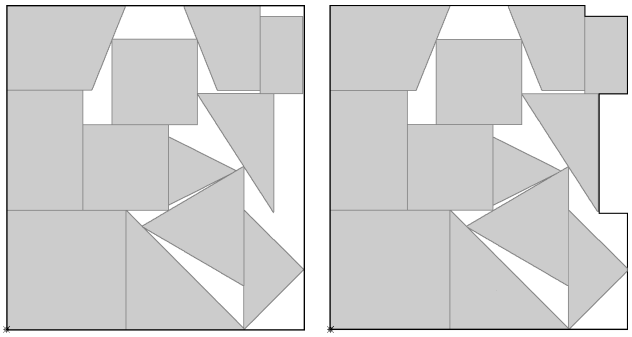
Line and Arc—Profiles1 to Profiles5. These new problems introduce arcs and holes to the set of benchmark problems. Some of these shapes, in particular Profiles1 and Profiles2, have been chosen from a library of standard shapes within the sheet metal profiling industry. All of these problems contain at least one shape consisting of one or more arcs and their optimal solutions are not known.

Line Only—Profiles6 to Profiles10. A further five problems have been created, some involving shapes with holes, which can be tackled by nonarc implementations. The optimal solutions are not known with the exception of

Table 4. New benchmark problems.

Problem name	No. of shapes	Rotational constraints	Sheet width	Optimal known
Profiles1	32	90 Incremental	2,000	No
Profiles2	50	90 Incremental	2,500	No
Profiles3	46	45 Incremental	2,500	No
Profiles4	54	90 Incremental	500	No
Profiles5	50	15 Incremental	4,000	No
Profiles6	69	90 Incremental	5,000	No
Profiles7	9	90 Incremental	500	Yes
Profiles8	18	90 Incremental	1,000	Yes
Profiles9	57	90 Incremental	1,500	No
Profiles10	91	0 Absolute	3,000	No

Figure 10. Density measures: (a) Straight line density measure (Density1), (b) Hopper (2000) density measure (Density2).



Profiles7 and Profiles8, which are "jigsaw" problems where the optimal length is 1,000 units for both problems. Profiles9 is a novelty data set involving a subset of letters from the English alphabet. Profiles10 combines polygons from numerous literature benchmark problems.

5. Experiments

For the purposes of our experiments, we use our proposed bottom-left-fill placement algorithm (§3.4) and local search techniques (§3.5) to generate input shape orderings. During the search process, the generated solutions are evaluated by the total length of the packing. We also record two different density measures: the first is a simple straight line density (Density1), while the second density measure, used by Hopper (2000), is based on the union of all individual shape bounding rectangles. This allows us to use a nonrectangular final density measurement (Density2). Both methods are pictured in Figure 10, where the thick line represents the containing area.

Density can then calculated by

$$\text{density} = \text{total shape area} / \text{containing area}.$$

All of the experiments conducted in this paper were performed on a PC with a 2 GHz Intel Pentium 4 processor and 256 MB RAM.

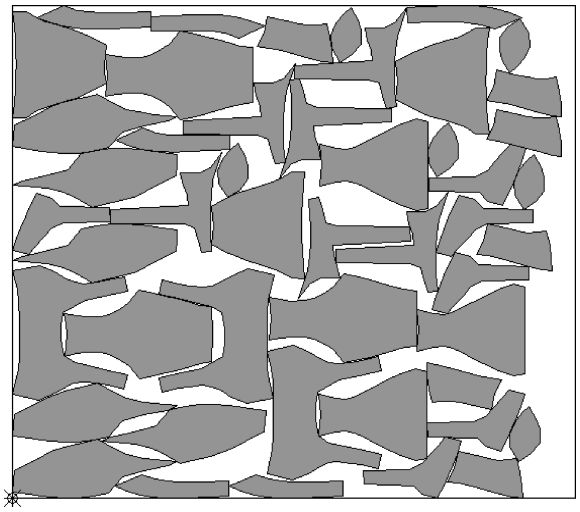
5.1. Experiments on Literature Benchmark Problems

We have divided the literature problems into two groups: (1) those for which the best-known result is measured using overall packing length, and (2) those which have been evaluated using density measures. In our results, we record both measures for the two different data sets to allow easier comparison for future researchers. Each problem was run 10 times using 100 iteration runs for both length and area initial orderings (resulting in a total of 40 runs for each problem). Due to space limitations, we only show a summary of the best results from these 40 runs in Table 5, where we detail the average times per nest, time taken

Table 5. Summary of the best results using 100 iteration runs.

Problem	Best literature	Best result			Time/Nest (s)	Time to best (s)	Percentage improvement (%)
		Length	Density1 (%)	Density2 (%)			
Blasz1	27.3	27.80	77.7	81.0	0.32	21	−1.83
Dagli	65.6	60.57	83.7	89.0	2.04	188.8	**7.66**
Fu	34	32.80	86.9	90.8	0.24	20.78	**3.53**
Jakobs1	13.2	11.86	82.6	92.6	0.74	43.49	**10.17**
Jakobs2	28.2	25.80	74.8	83.3	2.13	81.41	**8.51**
Marques	83.6	80.00	86.5	89.3	0.25	4.87	**4.31**
Poly1A	14.7	14.00	73.2	78.2	0.36	12.48	**4.76**
Poly2A	30.1	28.17	72.8	77.5	1.24	120.56	**6.42**
Poly3A	40.4	41.65	73.8	75.5	2.01	210.07	−3.10
Poly4A	56.9	54.93	74.6	75.9	2.43	203.17	**3.47**
Poly5A	71.6	69.37	73.9	75.7	5.04	475.63	**3.12**
Poly2B	33.1	30.00	75.4	77.5	2.50	179.86	**9.36**
Poly3B	41.8	40.74	74.9	77.1	4.26	417.67	**2.54**
Poly4B	52.9	51.73	74.8	77.4	8.24	95.66	**2.20**
Poly5B	63.4	60.54	75.8	77.2	14.70	676.61	**4.51**
SHAPES	63	59.00	67.6	69.1	0.60	31.36	**6.35**
SHAPES0	63	66.00	60.5	62.6	0.93	21.33	−4.76
SHAPES1	59	60.00	66.5	68.9	0.82	2.19	−1.69
SHIRTS	63.13	63.80	84.6	87.3	4.99	58.36	−1.06
SWIM	6,568	6,462.40	68.4	71.6	12.39	607.37	**1.61**
TROUSERS	245.75	246.57	88.5	90.1	7.89	756.15	−0.33
Albano	86.0%	10,292.90	84.6	86.5	1.18	93.39	**0.5**
Blasz2	68.6%	25.28	74.5	79.9	0.16	10.94	**11.3**
Dighe1	72.4%	1,292.30	77.4	78.9	0.22	8.87	**6.5**
Dighe2	74.6%	1,260.00	79.4	84.3	0.10	7.12	**9.7**
Mao	71.6%	1,854.30	79.5	82.9	0.38	29.74	**11.3**

Figure 11. "SWIM" 100 iteration solution (6,462.4 units, 607 seconds).

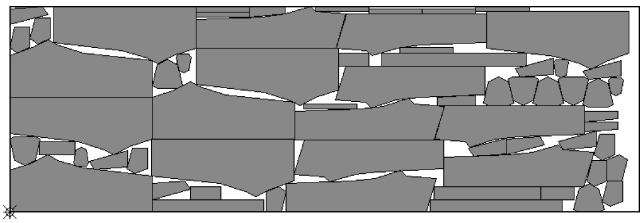


to find the best solution, and the percentage improvement on the best-known result from the literature. For problems where we have improved on the best-known solution, we have marked the percentage in bold. For the full results, see Tables D1 and D2 in online Appendix D.

From our initial experiments, the proposed approach has improved on 20 of the 26 available literature problems. The best solution for data set SWIM was obtained in 607 seconds and is shown in Figure 11. The average overall improvement over the 26 problems is slightly over 4% and is nearer 6% for the 20 problems where we have found the new best solution. On average, the best solutions have been obtained within a few minutes of execution time, although for many of the problems only a few seconds are necessary due to the high performance of the presented algorithm. The average time per nest compares favourably with other literature approaches utilising the no-fit polygon. For example, Gomes and Oliveira (2002) state that a solution for the problem SHAPES1 can be generated within 6.18 seconds on a 450 MHz CPU, in comparison with 0.82 seconds for a solution generated by our new nesting method.

We then extended our experiments on the problems for which we have not set new benchmarks, namely, Blasz1, SHAPES1, SHIRTS, and TROUSERS. For these extended runs, we have used the durations 551.73 s, 2,019.77 s, 6,367.57 s, and 13,613.67 s, respectively, as our upper limits for each run, as noted by the authors who presented the current benchmarks for these problems (Gomes and Oliveira 2002). For the problems SHAPES and Poly3A, we simply extended the experiment to 1,000 iterations per run, as guidance on search times is not available in the literature. We then ran these extension experiments for a further 10 runs. Table 6 shows the best results of these runs, including the time to best solution, the method that generated that solution, and percentage improvement on the best-known solution.

Figure 12. "TROUSERS" extended run solution (243.4 units, 3,612 seconds).



Of the six extended experiments, we set five new benchmarks for Blasz1, Poly3A, SHAPES1, SHIRTS, and TROUSERS. For the remaining problem, SHAPES0, our solution matches the work of Gomes and Oliveira (2002) (length of 65 units) but has not reached the best-known solution of 63 units (Dowsland and Dowsland 1993). Therefore, of the 26 problems from the literature, using our new nesting algorithm, we have produced 25 new best solutions. This is the first time any research paper in this area has attempted such a broad range of problems with such success. The solution obtained for TROUSERS is shown in Figure 12. All best solutions obtained for the literature problems are shown in online Appendix A.

5.2. Experiments on New Benchmark Problems

The next set of experiments we conducted involved setting benchmarks for our new problems, Profiles1–Profiles10. Each problem was run 10 times for 30 minute durations with both hill climbing and tabu search (length and area initial orderings) resulting in a total of 40 runs. Once again, due to space limitations, the full results of these experiments can be found in Table D3 in online Appendix D.

Table 6. Summary of the results from the extended experiments.

Problem	Best literature	Average length	Best result			Method	Time to best (s)	Percentage improvement (%)
			Length	Density1 (%)	Density2 (%)			
Blasz1	27.3	27.47	27.20	79.4	80.7	Hill climb/Length	501.91	**0.37**
Poly3A	40.4	41.45	40.33	76.2	77.3	Tabu/Area	1,515.49	**0.18**
SHAPES0	63	65.68	65.00	61.4	63.6	Hill climb/Area	332.39	−3.17
SHAPES1	59	60.53	58.40	68.3	71.5	Tabu/Area	1,810.14	**1.02**
SHIRTS	63.13	63.66	63.00	85.7	88.1	Tabu/Length	806.5	**0.21**
TROUSERS	245.75	246.40	243.40	89.6	91.1	Tabu/Area	3,611.99	**0.96**

Table 7. Summary of the best results for new benchmark problems.

	Best result		
Problem	Length	Density1 (%)	Density2 (%)
Profiles1	1,377.74	82.0	85.2
Profiles2	3,216.10	50.0	51.3
Profiles3	8,193.89	50.9	52.6
Profiles4	2,453.12	75.1	75.7
Profiles5	3,332.70	70.2	73.6
Profiles6	3,097.86	75.6	77.8
Profiles7	1,296.30	77.1	80.2
Profiles8	1,318.70	75.8	77.2
Profiles9	1,290.67	53.1	54.9
Profiles10	11,160.10	66.2	66.8

Figure 13. Best solutions for "Profiles1" (1,377.74 units) and "Profiles9" (1,290.67 units).

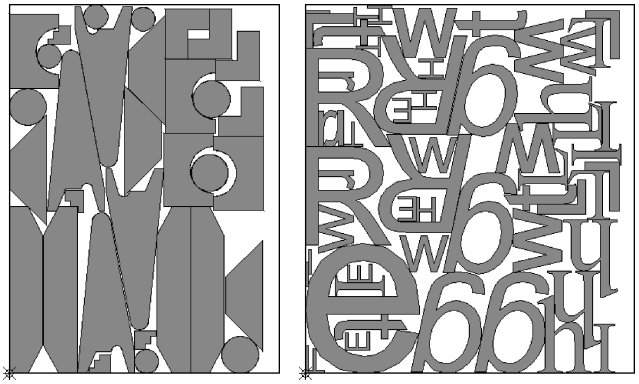


Table 7 shows a summary of the best results from these 40 runs. Figure 13 shows the best solutions for data sets Profiles1 and Profiles9. All best solutions obtained for these new problems are shown in online Appendix B.

6. Summary

In this paper, we introduced a new technique of primitive overlap resolution. We then demonstrated the steps required for implementing an efficient bottom-left-fill nesting algorithm that is able to handle profiles with both circular arcs and holes. This algorithm produces nests for realistic problems quickly and to a level of accuracy that makes it a strong candidate for industrial application. We have applied our algorithm to 26 problems drawn from over 20 years of cutting and packing research. The proposed algorithm, using only simple local search mechanisms, was able to produce the best-known results for 25 of the 26 problems. The majority of these best solutions were generated within 100 search iterations and yielded an average improvement of 5% over the existing best solutions from the literature.

To the authors' knowledge, this is the first paper in the field of two-dimensional irregular cutting and packing that has attempted such a broad range of problems from the literature. Furthermore, the paper introduces and sets initial benchmarks for 10 new problems, some of which involve profiles with both arcs and holes that have not previously been represented within the literature.

Acknowledgments

The authors thank the UK Engineering and Physical Sciences Research Council (EPSRC) for funding the project. They also thank the anonymous reviewers for their helpful and supportive comments.

References

Agarwal, P. K., E. Flato, D. Halperin. 2002. Polygon decomposition for efficient construction of Minkowski sums. *Comput. Geometry: Theory Appl.* **21** 39–61.

Albano, A., G. Sapuppo. 1980. Optimal allocation of two-dimensional irregular shapes using heuristic search methods. *IEEE Trans. Systems, Man Cybernetics* **SMC-10** 242–248.

Art, R. C. 1966. An approach to the two-dimensional irregular cutting stock problem. Technical report 36.008, IBM Cambridge Centre.

Avnaim, F., J. D. Boissonnat. 1987. Simultaneous containment of several polygons. *Proc. 3rd Annual ACM Sympos. Comput. Geometry*. Waterloo, Ontario, Canada, 242–250.

Baker, B. S., E. G. Coffman, R. L. Rivest. 1980. Orthogonal packings in two dimensions. *SIAM J. Comput.* **9**(4) 846–855.

Bennell, J. A., K. A. Dowsland, W. B. Dowsland. 2001. The irregular cutting-stock problem—A new procedure for deriving the no-fit polygon. *Comput. Oper. Res.* **28** 271–287.

Bischoff, E., M. Ratcliff. 1995. Load multiple pallets. Working paper, European Business Management School, University College of Swansea, Swansea, UK.

Blazewicz, J., P. Hawryluk, R. Walkowiak. 1993. Using a tabu search approach for solving the two-dimensional irregular cutting problem. F. Glover, M. Laguna, E. Taillard, D. De Werra, eds. *Tabu Search. Annals of Operations Research*, Vol. 41. J.C. Baltzer AG, Science Publishers, Basel, Switzerland, 313–325.

Bounsaythip, C., S. Maouche. 1997. Irregular shape nesting and placing with evolutionary approach. *Proc. IEEE Internat. Conf. Systems, Man Cybernetics* **4** 3425–3430.

Burke, E. K. B., G. Kendall, G. Whitwell. 2004. A new placement heuristic for the orthogonal stock cutting problem. *Oper. Res.* **52**(4) 655–671.

Daniels, K. M., V. J. Milenkovic. 1997. Multiple translational containment. Part 1: An approximation algorithm. *Algorithmica* **19**(Special Issue on Computational Geometry in Manufacturing) 148–182.

Dighe, R., M. J. Jakiela. 1996. Solving pattern nesting problems with genetic algorithms employing task decomposition and contact detection. *Evolutionary Comput.* **3** 239–266.

Dowsland, K. A., W. B. Dowsland. 1993. Heuristic approaches to irregular cutting problems. Working paper, EBMS/1993/13, European Business Management School, University College of Swansea, Swansea, UK.

Dowsland, K. A., W. B. Dowsland. 1995. Solution approaches to irregular nesting problems. *Eur. J. Oper. Res.* **84** 506–521.

Flato, E., D. Halperin. 2000. Robust and efficient construction of planar Minkowski sums. *Proc. 16th Eur. Workshop Comput. Geometry*. Eilat, Israel, 85–88.

Fujita, K., S. Akagi, N. Hirokawa. 1993. Hybrid approach for optimal nesting using a genetic algorithm and a local minimization algorithm. *Proc. 1993 ASME Design Automation Conf.* DE Vol. 65, Part 1. ASME, Albuquerque, NM, 477–484.

Gilmore, P. C., R. E. Gomory. 1961. A linear programming approach to the cutting stock problem. *Oper. Res.* **9** 849–859.

Glover, F., E. Taillard, D. De Werra. 1993. A user's guide to tabu search. F. Glover, M. Laguna, E. Taillard, D. De Werra, eds. *Tabu Search. Annals of Operations Research*, Vol. 41. J.C. Baltzer AG, Science Publishers, Basel, Switzerland, 3–28.

Gomes, A. M., J. F. Oliveira. 2002. A 2-exchange heuristic for nesting problems. *Eur. J. Oper. Res.* **141** 359–370.

Goodrich, M. T., L. J. Guibas, J. Hershberger, P. J. Tanenbaum. 1997. Snap rounding line segments efficiently in two and three dimensions. *Proc. 13th Sympos. Comput. Geometry*, Nice, France, 284–293.

Guibas, L. J., D. H. Marimont. 1995. Rounding arrangements dynamically. *11th Annual ACM Sympos. Comput. Geometry*. Vancouver, British Columbia, Canada, 190–199.

Halperin, D., E. Packer. 2002. Iterated snap rounding. *Comput. Geometry: Theory Appl.* **23**(2) 209–225.

Hopper, E. 2000, Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods. Ph.D. thesis, University of Wales, Cardiff, UK.

Hopper, E., B. C. H. Turton. 2001. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. *Eur. J. Oper. Res.* **128** 34–57.

Jakobs, S. 1996. On genetic algorithms for the packing of polygons. *Eur. J. Oper. Res.* **88** 165–181.

Konopasek, M. 1981. Mathematical treatments of some apparel marking and cutting problems. U.S. Department of Commerce Report 99-26-90857-10, U.S. Department of Commerce, Washington, DC.

Li, Z., V. J. Milenkovic. 1995. Compaction and separation algorithms for non-convex polygons and their application. *Eur. J. Oper. Res.* **84** 539–561.

Mahadevan, A. 1984. Optimisation in computer aided pattern packing. Ph.D. thesis, North Carolina State University, Raleigh, NC.

Marques, V. M. M., C. F. G. Bispo, J. J. S. Sentieiro. 1991. A system for the compaction of two-dimensional irregular shapes based on simulated annealing. *Proc. 1991 Internat. Conf. Indust. Electronics, Control and Instrumentation—IECON'*91, Vol. 3. Kobe, Japan, 1911–1916.

Milenkovic, V. J. 2000. Shortest path geometric rounding. *Algorithmica* **27**(1) 57–86.

Oliveira, J. F., J. S. Ferreira. 1993. Algorithms for nesting problems. Rene V. V. Vidal, ed. *Applied Simulated Annealing, Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, Berlin, Germany, 255–273.

Oliveira, J. F., A. M. Gomes, J. S. Ferreira. 2000. A new constructive algorithm for nesting problems. *OR Spektrum* **22** 263–284.

O'Rourke, J. 1998. *Computation Geometry in C*, 2nd ed. Cambridge University Press, Cambridge, UK.

Ratanapan, K., C. H. Dagli. 1997. An object-based evolutionary algorithm for solving irregular nesting problems. *Proc. Artificial Neural Networks in Engrg. Conf.* (*ANNIE'*97), Vol. 7. ASME Press, New York, 383–388.

Smith, D. 1985. Bin packing with adaptive search. *Proc. First Internat. Conf. Genetic Algorithms Appl.* Lawrence Erlbaum Associates, Hillsdale, NJ, 202–207.

Stoyan, Y. G., G. N. Yaskov. 1998. Mathematical model and solution method of optimization problem of placement of rectangles and circles taking into account special constraints. *Internat. Trans. Oper. Res.* **5**(1) 45–57.

Stoyan, Y. G., M. V. Novozhilova, A. V. Kartashov. 1996. Mathematical model and method of searching for a local extremum for the non convex oriented polygons allocation problem. *Eur. J. Oper. Res.* **92** 193–210.

Stoyan, Y., G. Scheithauer, N. Gil, T. Romanova. 2004. Phi-functions for complex 2D-objects. *4OR: Quart. J. Oper. Res.* **2**(1) 69–84.

Sweeny, P. E., E. R. Paternoster. 1992. Cutting and packing problems: A categorized application oriented research bibliography. *J. Oper. Res. Soc.* **43**(7) 691–706.