A New Approach to Packing Non-Convex Polygons Using the No Fit Polygon and Meta-Heuristic and Evolutionary Algorithms

Edmund Burke and Graham Kendall

The University of Nottingham, School of Computer Science & IT, Jubilee Campus, Wollaton Road, Nottingham NG8 1BB, UK
{ekb, gxk}@cs.nott.ac.uk

Abstract

Earlier work by the authors has presented a method for packing convex polygons, using a construction known as the no fit polygon. Although the method was shown to be successful, it could only deal with convex polygons. This paper addresses the issue by showing how a non-convex, no fit polygon algorithm can (and should) produce better quality solutions. We demonstrate the approach on two test problems that the authors have used in previous work. The new algorithm does, however, have an increased computational complexity. We tackle this by presenting an algorithm that allows the non-convex, no fit polygon to be approximated. This means that much more of the search space can be considered and computational results show that better quality solutions can be achieved, in less time than when using the full no fit polygon algorithm.

1. Introduction

Cutting material from stock sheets is a key process in a number of important manufacturing industries such as the sheet metal, paper, garment and glass industry.

Cutting out the material in the most effective way usually has a financial incentive. If a company is able to minimise the amount of waste it produces then there is a quantifiable saving in the cost of raw material. In addition, the company may be able to reduce its stock holding which can make additional savings through improved cash flow. The company may also be able to reduce its warehousing capacity, resulting in further savings.

Although the normal motivation for effective stock cutting is financial, companies may have other objectives in implementing efficient stock cutting procedures. For example, there may be a requirement to meet certain orders within a given time. Under these circumstances the order in which shapes are cut may be more important than the packing so as to meet the deadline.

In previous work Burke and Kendall [1, 2, 3] have shown that evolutionary and meta-heuristic approaches can give good quality solutions to the nesting problem. However, in these papers only convex (i.e. where all interior angles are less than 180°) pieces were used. The evaluation function relied on an idea known as the no fit polygon (described in section 2). Calculating the no fit polygon for convex

pieces is trivial, but for non convex polygons the algorithm is much more complex and is still the subject of current research [4, 5].

In the nesting problem it is necessary to place a number of shapes onto a larger shape. In doing so, the shapes must not overlap and they must stay within the confines of the larger shape. The usual objective is to minimise the waste of the larger shape. Only two dimensions, height and width, of the shapes are considered and the larger piece is sometimes considered to be of infinite height so that only the width of the placements needs be checked. This is a realistic assumption for the real world as the larger shapes are sometimes rolls of material which can be considered as being of infinite length for the purposes of the placement procedure. This assumption is made in this paper., although it remains the aim of the evaluation function to minimise this height. Another assumption that we make is that only one bin is used (that is, there is no possibility of filling a bin and having to start another). This nesting problem has been shown to be NP-Complete by [6].

Art [7] introduced the no fit polygon (described in section 2). It is also used by Adamwicz and Albano [8] to calculate the minimum enclosing rectangle for two irregular shapes. Albano and Sappupo [9] used a heuristic search method to place irregular pieces onto stock sheets. Their search is based on the A* algorithm but some restrictions are introduced due to the size of the search space. The approach makes use of the no fit polygon when deciding where pieces should be placed. More recently Olivera et. al. [10] also used the no fit polygon. Pieces are placed onto a stock sheet one at a time. The location of the next piece is calculated using the no fit polygon. Once the best placement has been found the piece is added to the partial solution and the next piece is placed. Oliveira experimented with three evaluation functions when considering the placement of the next piece.

In this paper we employ various meta-heuristic and evolutionary algorithms to solve instances of the nesting problem using non-convex polygons. The evaluation function utlises the no fit polygon.

2 No Fit Polygon

The no fit polygon determines all arrangements that two arbitrary polygons may take so that the polygons touch but do not overlap. If we can find the no fit polygon for two given polygons then we know we cannot move the polygons closer together in order to obtain a tighter packing. In order to find the placements we proceed as follows (see fig. 1). One of the polygons (P_1) remains stationary. The other polygon, P_2, orbits around P_1, staying in contact with it but never intersecting it. Both polygons retain their original orientation. That is, they never rotate. As P_2 moves around P_1 one of its vertices (the reference point, shown as a filled circle) traces a line (this becomes the no fit polygon).

Fig. 1 shows the starting (and finishing) positions of P_1 and P_2. The no fit polygon is shown as a dashed line. It has been slightly enlarged in the figure to make it visible. In fact, some of the edges will be identical to some of the edges of P_1 and P_2. Once we have calculated the no fit polygon for a given pair of polygons we can place the reference point of P_2 anywhere on an edge or vertex of the no fit polygon in the knowledge that P_2 will touch but not intersect P_1. In order to implement a no fit polygon algorithm for convex polygons it is not necessary to

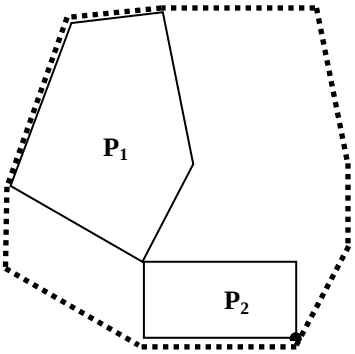

Fig. 1. No Fit Polygon

emulate one polygon orbiting another. Cunninghame-Green [11] and Cunninghame-Green and Davis [12] present an algorithm that works on the assumption that the no fit polygon has a number of edges equal to the number of edges of P_1 and P_2. In addition, the edges of the no fit polygon are simply copies of the edges of P_1 and P_2, suitably ordered. To build the no fit polygon it is a matter of taking the edges of P_1 and P_2, sorting them and building the no fit polygon using the ordered edges.

The assumptions made in the algorithm in [11, 12] are only valid for convex polygons. Once one of the polygons is not convex these algorithms no longer work. However, the notion of one polygon orbiting another still holds and an algorithm based on these principles is still valid but there are many degenerate cases that need to be addressed in order to produce a correct no fit polygon. Mahadevan [13], details the various conditions that can arise and how to deal with them. This work relies heavily on D-functions which are a set of low level computational geometry functions. Our work [5] extended the analysis of Mahadevan so that our algorithm can deal with more degenerate cases.

An alternative method is to use minkowski sums, which have been widely employed in robot motion planning [14, 15, 16, 17, 18, 19] and in image analysis [20]. In chapter 3 of [21] "The Theory of Minkowski Sum and Difference" is presented, with particular reference to compaction algorithms. In this work the author states

> *"Although the Minkowski sum and difference has been extensively used in robot motion planning via the configuration space approach, we have seen very few efforts in applying it to packing problems."*

The work proceeds to show how to calculate the minkowski sum and difference for intersection and containment problems, in particular proving a crucial property with respect to star shaped polygons which results in a simplified algorithm for computing the minkowski sum for these type of polygons.

Bennell et. al. [4] states

> *"However, unless all the pieces are convex, it is widely perceived as being difficult to implement [the no fit polygon], and its use has therefore been somewhat limited."*

Both of these papers indicate that the calculation of the no fit polygon for non-convex polygons is still an active research area. Bennell's et. al. paper presents a revised and simplified method for deriving the no fit polygon using minkowski sums and derives a set of simple rules that are easier to implement than previous approaches. In this paper, we use the method we derived from Mahadevan, but, if accessible at the time, we could have employed the algorithm presented by Bennell et. al..

3 Evaluation

3.1 Placing the Polygons

In order to place the polygons within a bin, we proceed as follows. The first polygon to be placed is chosen and becomes the stationary polygon (P_1 in the example above). The next polygon (P_2 from the above example) becomes the orbiting polygon. Using these two polygons the no fit polygon is constructed. The reference point of P_2 is placed on each vertex of the no fit polygon and for each position the convex hull (i.e. the smallest convex polygon that will enclose a given polygon) for the two polygons is calculated. Once all placements have been considered the convex hull that has the minimum area is returned as the best packing of the two polygons. Note, that when the polygons are packed, it is not the convex hull that is used to build the solution as this would lead to very sub-optimal solutions and, in effect, we would be emulating the convex case described in [1, 2, 3]. In fact, we use a *combine* operation to join two polygons together. This operation is defined in chapter 7 of [5]. The larger polygon, produced from the combine operation, now becomes the stationary polygon and the next polygon is used as the orbiting polygon. This process is repeated until all polygons have been processed. As each large polygon (i.e. output from the combine operation) is created, its width is checked. If this exceeds the width of the bin, then a new row within the bin is started. In this case the polygon which forced the width of the bin to be exceeded becomes the stationary polygon. That is, the large polygon built thus far forms one row and the next row is constructed using a single polygon as a starting point.

There are a number of points that need to be considered when using the above evaluation method. A more complete discussion is provided in [22], with the main points summarised below

A number of computational geometry algorithms have been implemented in order to effectively manipulate polygons. Executing computational geometry algorithms is computationally expensive. This makes the evaluation function the slowest part of the algorithm. In view of this a cache has been implemented which retains *n* (user defined) previous evaluations. If a solution (partial or complete) is in the cache then the results are retrieved from the cache rather than executing the evaluation function. To further improve the algorithm the concept of a polygon type has been introduced. This allows polygons which are identical to be classified together so that the cache can work at a higher level of abstraction when deciding if a partial or complete solution is in the cache. Once the no fit polygon has been calculated there may be a number of optimal placements for the two polygons being evaluated (that is, the convex hulls have equal area). In this case, the placement to use is selected at random. However, all the optimal solutions are stored in the cache so that if the solution (partial or complete) is seen again, one of the other placements could be selected.

If *n* polygons have been evaluated the solutions in the cache for the n^{th} polygon will be influenced by earlier evaluations. However, even though polygon placements may seem optimal at the time they are placed, future placements may show that these earlier decisions were not the best choice once other polygons are

added to the solution. Therefore, it may be beneficial to re-evaluate a complete solution even if it is stored in the cache. In order to accommodate this a re-evaluation parameter was introduced which, with some probability, forces a solution to be re-evaluated even if it is in the cache.

3.2. Evaluating the placement

Our cost function is based on that used in [23]. It can be stated as follows

$$(\sum_{1}^{n}((1-(UsedRowArea/TotalRowArea))^2)*k)/n \qquad (1)$$

Where *UsedRowArea* is the total area of the polygons placed in that row
TotalRowArea is the total area of the bin occupied by that row
k if a factor simply to scale the result. We set *k*=100
n is the number of rows in the bin

In essence, we are trying to minimise the area used by each row.

This evaluation function is preferable to the more obvious method of simply measuring the bin height otherwise many solutions will map to the same evaluation function. This makes it much more difficult to effectively explore the search space. In previous work [2], we have compared these two different types of evaluation and we find that the above function (1) produces quality solutions; which is the same experience as reported in [23].

4 Experiments

4.1 Comparison of Test Problems with Convex Results

In previous work [1, 2, 3, 5] a variety of search algorithms have been tested in order to find good quality solutions to the nesting problem by utilising the no fit polygon. The algorithms we have used are, hill climbing, tabu search, simulated annealing, a genetic algorithm, an ant algorithm and a memetic algorithm (a population based approached with a local search). For a full discussion of these algorithms the reader is referred to the above references which discuss the algorithms in detail and show how the various parameters, that control the algorithms, were selected. All the algorithms were searching for an ordering of polygons, that is the order in which they will be packed. In previous work these approaches have been applied to two test problems (test data 1 and 2). The best results were obtained using a memetic algorithm (genetic algorithm with hill climbing) and are summarised in table 1.

Table 1 - Best Results for Test Data 1 & 2 using Convex No Fit Polygon Algorithm

Test Data	Best Evaluation
1	283.07
2	890.51

These results were obtained using experiments that carried out 8000 evaluations. On a Pentium PII (laptop) with 16MB main memory this took about ten minutes. Identical runs were carried out using the non-convex no fit polygon algorithm and the number of evaluations were adjusted so that the algorithms ran for about the same amount of time. To achieve this the searches were only allowed to carry out 300 evaluations. All the same searches were conducted as in previous work [1, 2, 3, 5]. The results are shown in tables 2 and 3. All results are averaged over 10 runs. The results are sorted in ascending order of evaluation.

Table 2 - Results for Test Data 1 using Non-Convex No Fit Polygon Algorithm

Search Type	Parameters	Evaluation	Row Height	Standard Deviation
Memetic Algorithm (genetic algorithm with hill climbing)	Popsize = 6, Generations = 8, Hill Climbing Iterations = 2, Neighbourhood Size = 4	129.77	156.00	22.39
Simulated Annealing	Schedule = {40, 0, 4, 30)	129.87	156.50	36.44
Tabu Search	Iterations=30, List Size = 20, Neighbourhood Size = 10	132.09	157.00	57.50
Hill Climbing	Iterations = 30, N/H Size = 10	134.78	157.00	71.71
Memetic Algorithm (genetic algorithm with hill climbing)	Popsize = 10, Generations = 10, Hill Climbing Iterations = 3, Neighbourhood Size = 1	147.24	158.00	41.04
Simulated Annealing	Schedule = {200, 0, 20, 30)	156.83	158.00	51.03
Tabu Search	Iterations=30, List Size = 10, Neighbourhood Size = 10	168.80	158.80	52.31
Hill Climbing	Iterations = 300, Neighbourhood Size = 1	171.49	160.20	44.84
Ant Algorithm	Ants=13, Iterations=12, Trail=1, Visibility=20, Evaporation=0.5, Q=100	174.54	159.60	72.15
Simulated Annealing	Schedule = {60, 0, 4, 20}	179.02	159.40	73.61
Memetic Algorithms (ant algorithm with hill climbing)	Ants = 13, Iterations = 6, Hill Climbing Iterations = 4, Neighbourhood Size = 1	185.54	162.40	59.75
Genetic Algorithm	Pop Size = 18, Generations = 30	191.80	160.60	81.16

Table 3 - Results for Test Data 2 using Non-Convex No Fit Polygon Algorithm

Search Type	Parameters	Evaluation	Row Height	Standard Deviation
Memetic Algorithm (genetic algorithm with hill climbing)	Popsize = 6, Generations = 8, Hill Climbing Iterations = 2, Neighbourhood Size = 4	857.80	26663.00	146.27
Tabu Search	Iterations =30, List Size = 20, Neighbourhood Size = 10	871.67	26867.63	197.90
Memetic Algorithm (genetic algorithm with hill climbing)	Popsize = 10, Generations = 10, Hill Climbing Iterations = 3, Neighbourhood Size = 1	872.03	26987.12	129.90
Memetic Algorithm (ant algorithm with hill climbing)	Ants = 13, Iterations = 6, Hill Climbing Iters = 4, Neighbourhood Size = 1	925.12	27185.91	136.49
Ant Algorithm	Ants=13, Iterations =12, Trail=1, Visibility=20, Evaporation=0.5, Q=100	933.15	27268.53	128.75
Simulated Annealing	Schedule = {2000, 0, 200, 30}	944.15	27367.30	99.07
Tabu Search	Iterations =30, List Size = 10, Neighbourhood Size = 10	964.77	27533.27	152.52
Hill Climbing	Iterations = 30, Neighbourhood Size = 10	994.82	26874.97	152.34
Simulated Annealing	Schedule = {1000, 0, 100, 30}	1000.79	27675.30	170.94
Simulated Annealing	Schedule = {1000, 0, 200, 60}	1003.50	27740.35	113.76
Genetic Algorithm	Pop Size = 18, Generations = 30	1026.82	27935.78	190.51
Hill Climbing	Iterations = 300, Neighbourhood Size = 1	1030.82	27896.10	221.15

Not surprisingly, better results are achieved using the non-convex algorithm than with the convex algorithm. This was to be expected as pieces are able to fit into concavities that are not available when only working with convex shapes.

In addition, both sets of results support the findings from earlier work [1, 2, 3]) in that a memetic algorithm (based on a genetic algorithm) gives better quality results than a single algorithm used in isolation. It is also interesting to note, especially for test data 1, that the standard deviation of the evaluation function shows that the memetic algorithm produces good quality results on a more consistent basis than some of the other search algorithms.

4.2 Approximating the No Fit Polygon

Utilising the no fit polygon does lead to good quality solutions for the problems that have been addressed above. However, the no fit polygon algorithm is computationally expensive. It would be beneficial if the no fit polygon could be approximated, without loss of solution quality. One way to achieve this is to take each vertex of the orbiting polygon and place it on each vertex of the stationery polygon. At each placement, the polygons are tested for intersection. If they intersect, the algorithm continues, to check more vertices. If no intersection is detected, the polygons are combined and the convex hull calculated. The combined polygon with the minimum convex hull area is returned as the best placement.

This algorithm can be demonstrated more formally as follows.

```
S = Stationery Polygon
O = Orbiting Polygon
sv = Random Vertex on Stationery Polygon
start = sv
ov = Random Vertex on Orbiting Polygon
do {
Align ov on sv
          if no intersection
            C = Combine(O, S)
            if ConvexHull(C) is smallest so far
              P = C
            end if
          end if
          sv = sv +1 [using modulus arithmetic]
} while(start != sv)
return P
```

Notice that the starting vertices are chosen randomly. This is to ensure that if two placements (for the two given polygons) have more than one convex hull which has the minimum area then each has a chance of being selected. If placements always started at, say, the zero'th vertex then the algorithm would always return the same placement, which may not be the best placement later in the nesting procedure (see [22] for a discussion as to why this is important).

Another possible performance improvement is to stop the evaluation of a particular permutation of polygons, once the evaluation exceeds the current minimum. Intuitively this seems like a good idea because if the value for the polygons being evaluated exceeds the minimum evaluation found thus far, it is pointless continuing as the current evaluation cannot improve on the best nesting found so far. However, some of our search algorithms require a full evaluation to be carried out due to the way they operate. Our simulated annealing algorithm needs to carry out a full evaluation as it needs the change in evaluation so that it can use this value in the acceptance function in deciding whether to move to the solution under consideration. Our ant algorithm also need to carry out a full evaluation so that each ant can decide how much pheromone to deposit on each

edge. Finally, our genetic algorithm also requires a full evaluation as the parent chromosomes are selected for breeding based on the fitness.

Experiments were run to test the two ideas presented above. Four initial experiments were run, using all the possible combinations. They can be summarised as follows; (1) Use the no fit polygon algorithm and do not stop evaluations when the minimum is exceeded, (2) Use the no fit polygon algorithm but stop evaluations when the minimum is exceeded, (3) Use the approximated no fit polygon algorithm and do not stop evaluations when the minimum is exceeded and (4) Use the approximated no fit polygon algorithm but stop evaluations when the minimum is exceeded.

The test data for these tests was taken from [24], using their first set of test data in category 1. This data comprises 16 rectangles which are to be nested in a bin of width 20.

The results obtained are shown in table 4, with the results being averaged over ten runs. The test numbers refer to the numbers given above for the four proposed tests, with the description being given for ease of reference.

Table 4 - Approximating the No Fit Polygon (NFP) and Using Partial Evaluation

Test Number	Test Type	Evaluation	Time (seconds)
1	Normal NFP/Complete Eval.	128.52	2495
2	Normal NFP/Partial Eval.	131.10	1872
3	Approx. NFP/Complete Eval.	178.90	614
4	Approx. NFP/Partial Eval.	180.75	475

At first sight the results are not encouraging. Using test 1 as a baseline, this uses the no fit polygon algorithm that has been described earlier (section 2) and carries out complete evaluations. When partial evaluations are used, with the no fit polygon algorithm described in section 2 (test 2), the results show that the algorithm runs faster with no significant difference in solution quality. However, if the approximated no fit polygon algorithm is used (test 3 and 4), there is a significant speed improvement but also a reduction in solution quality. This is due to the fact that the normal no fit polygon algorithm allows vertices of the orbiting polygon to rest on edges on the stationery polygon. Using the approximation algorithm means that at least one vertex of the orbiting polygon must be aligned with a vertex of the stationery polygon.

Therefore, at first sight, it would seem advisable to implement partial evaluation and to continue using the slower, but better quality, no fit polygon algorithm described in section 2. However, further investigation shows that this may not have to be the case.

The tests were run using tabu search. Thirty iterations were done, using a list size of ten and a neighbourhood size of ten. Using the approximated no fit polygon and partial evaluation the algorithm speed can be improved from 2495 seconds to 475 seconds, an improvement of over 500%, but it leads to a solution quality that is approximately 40% worse.

However, there could be a trade off between these two extremes. It is possible to run the algorithm for longer, using the faster methods, achieve good quality

solutions but still save on the execution time of the algorithm. Table 5 shows the results of these experiments.

Table 5 - Using an Approximation of the No Fit Polygon (NFP) to Reduce Run Times

No.	Iterations, List Size, N/H Size	Test Type	Eval	Time (seconds)
1	30,10,10	Normal NFP/Complete Eval.	128.52	2495
2	30, 10, 10	Approx. NFP/Partial Eval.	180.75	475
3	60, 10, 10	Approx. NFP/ Partial Eval.	110.83	970
4	75, 30, 20	Approx. NFP/ Partial Eval.	105.32	2287

The first two rows of table 5 are the best and worst results from table 4, to allow easier comparison. Row three doubles the number of iterations from the original tests. The result is an improvement of solution quality but in significantly less time than using the original no fit polygon algorithm with complete evaluation (row 1).

Adjusting the tabu search parameters (see row 4) so that the algorithm, using the approximate no fit polygon and partial evaluation, runs for a similar amount of time as the tabu search in row 1 produces a better quality solution but with a lower execution time.

Based on these results it seems sensible to use the partial evaluation as this improves the execution time of the algorithm, with no loss of solution quality.

It also seems appropriate to use the approximated no fit polygon algorithm. Over the same number of evaluations, the original no fit polygon does lead to better quality solutions. However, the approximated no fit polygon allows more of the search space to be explored and leads to better quality solutions in shorter execution times.

5 Summary and Discussion

When packing shapes by utilising the no fit polygon, the computational overheads in calculating the no fit polygon becomes a bottleneck of the algorithm. It is quite often the case that the evaluation function is the slowest part of the algorithm and previous work [22] has already presented some techniques to reduce the amount of time spent in the evaluation process without reducing solution quality. This paper has shown that the non-convex no fit polygon can be approximated without loss of solution quality. Indeed, it can lead to better quality solutions as the algorithms are able to explore more of the search space, given the time they are allocated. This has an additional benefit which is not immediately apparent from the work presented above. The calculation of the no fit polygon is still an open research area and the algorithms developed by Mahadevan [13], Kendall [5] and Bennell et. al. [4] all have to deal with a variety of degenerate cases. We do not believe that any of these authors would claim that their algorithm can cope with all cases that they may come across. Hopefully, this will change and an extremely robust algorithm will be reported in the near future. In the meantime, the approximation of the no fit polygon, for the problems with which we are concerned, is not only faster, but the

algorithm to calculate the no fit polygon is also simpler and less error prone when it is being implemented.

Although not tackled in this paper, the majority of packing problems are likely to benefit from some form of pre-processing. If some of the shapes are packed prior to employing the search strategy, then there will be less shapes to pack. For example, rectangles (and squares) with a common dimension could be pre-packed. Similarly, triangles with identical dimensions can be tightly packed if one of them is first rotated through 180°. The authors believe that pre-processing of shapes will lead to even better quality solutions and in shorter times. However, deciding which shapes to pack together, and to what extent the shapes should be pre-packed, is a difficult problem in itself but the authors are planning to investigate this area as part of their on-going research.

References

1 Burke E.K, and Kendall, G. 1999. Applying Ant Algorithms and the No Fit Polygon to the Nesting Problem, Proceedings of 12th Australian Joint Conference on Artificial Intelligence, Sydney, Australia, 6-10 December 1999, Lecture Notes in Artificial Intelligence (1747), Foo, N. (Ed), pp 453-464.

2 Burke E.K, and Kendall, G. 1999. Applying Simulated Annealing and the No Fit Polygon to the Nesting Problem, Proceedings of WMC '99 : World Manufacturing Congress, Durham, UK, 27-30 September, 1999, pp 70-76.

3 Burke E.K, and Kendall, G. 1999. Applying Evolutionary Algorithms and the No Fit Polygon to the Nesting Problem, in proceedings of IC-AI'99 : The 1999 International Conference on Artificial Intelligence, Las Vegas, Nevada, USA, 28 June - 1 July 1999, pp 51-57.

4 Bennell J.A., Dowsland K.A, and Dowsland W.B. 2001. The irregular cutting stock problem - a new procedure for deriving the no-fit polygon, Computers and Operations Research 28 pp 271-287.

5 Kendall G. 2000. Applying Meta-Heuristic Algorithms to the Nesting Problem Utilising the No Fit Polygon PhD Thesis, Department of Computer Science & IT, The University of Nottingham, UK.

6 Karp, R.M. 1972. Reducibility Among Combinatorial Problems. Complexity of Computer Computations, Miller, R.E., Thatcher, J.W. (eds.), Plenum Press, New York, pp 85-103.

7 Art, R.C., 1966. An Approach to the Two-Dimensional Irregular Cutting Stock Problem. Technical Report 36.008, IBM Cambridge Centre.

8 Adamowicz, M., Albano, A. 1976. Nesting Two-Dimensional Shapes in Rectangular Modules. Computer Aided Design, 8, 27-33.

9 Albano, A., Sappupo, G. 1980. Optimal Allocation of Two-Dimensional Irregular Shapes Using Heuristic Search Methods. IEEE Trans. Syst., Man and Cybernetics, SMC-10, pp 242-248.

10 Oliveira, J.F., Gomes, A.M., Ferreira, S. 1998. TOPOS A new constructive algorithm for nesting problems. OR Spektrum 22 (2000) 2, 263-284.

11 Cunninghame-Green, R. 1989. Geometry, Shoemaking and the Milk Tray Problem. New Scientist, 12, August 1989, 1677, pp 50-53.

12 Cunninghame-Green and R., Davis, L.S. 1992. Cut Out Waste! O.R. Insight, Vol 5, iss 3, pp 4-7.

13 Mahadevan, A. 1984. Optimisation in Computer-Aided Pattern Packing. PhD thesis, North Caroline State University.

14 Canny, J. 1987. The Complexity of Robot Motion Planning. MIT Press, Cambridge, MA.

15 Ghosh, P. K. 1993. A Unified Computational Framework for Minkowski Operations. Computers and Graphics, Vol. 17, No. 4, pp 357-378.

16 Lozano-Pérez, T., Wesley, M. A. 1979. An Algorithm for planning collision-free paths among polyhedral obstacles. Communications of the ACM 22, pp 560-570.

17 O'Rourke, J. 1998. Computational Geometry in C. Cambridge University Press.

18 Ramkumar, G. D. 1996. An Algorithm to Compute the Minkowski Sum Outer-Face of Two Simple Polygons. In proceedings of 12th Annual ACM Symposium of Computational Geometry, pp 234-241.

19 Schwartz, J. T., Sharir, M. 1990. Algorithmic Motion Planning in Robotics, in J. van Leeuwen (ed), Algorithms and Complexity, Handbook of Theoretical Computer Science, Vol A, Elsevier, Amsterdam, pp 391-430.

20 Serra, J. 1982. Image Analysis and Mathematical Morphology, Vol. 1, Academic Press, New York.

21 Zhenyu, Li. 1994. Compaction Algorithms for Non-Convex Polygons and Their Applications. PhD Thesis, Computer Science, Harvard University, Cambridge, Massachusetts.

22 Burke, K. E. and Kendall, G. 1999d. Evaluation of Two Dimensional Bin Packing Problem using the No Fit Polygon, Proceedings of the 26th International Conference on Computers and Industrial Engineering, Melbourne, Australia, 15-17 December 1999, pp 286-291.

23 Falkenauer, E. 1998. Genetic Algorithms and Grouping Problems. John Wiley and Sons.

24 Hopper E. and Turton B.C.H. 2000. An Empirical Investigation of Meta-Heuristics and Heuristic Algorithms or 2D Packing Problem. European Journal of Operational Research (EJOR) 128 (1), 34-57.