UNIVERSITY OF CANTERBURY

Department of Mechanical Engineering

Christchurch New Zealand

Minimizing Waste in the 2-Dimensional Cutting Stock Problem

by

Hamish T Dean

A thesis submitted in partial fulfillment of the
requirements for the Degree
of
Doctor of Philosophy in Engineering
at the University of Canterbury

August 2002

Table Of Contents

Abstract

The 2-dimensional cutting stock problem is an important problem in the garment manufacturing industry. The problem is to arrange a given set of 2-dimensional patterns onto a rectangular bolt of cloth such that the efficiency is maximised. This arrangement is called a marker. Efficiency is measured by pattern area / marker area. Efficiency varies depending on the shape and number of patterns being cut, but an improvement in efficiency can result in significant savings. Markers are usually created by humans with the aid of CAD software. Many researchers have attempted to create automatic marker making software but have failed to produce marker efficiencies as high as human generated ones.

This thesis presents a mathematical model which optimally solves the 2-dimensional cutting stock problem. However, the model can only be solved in a practical amount of time for small markers. Subsequently, two compaction algorithms based on mathematical modelling have been developed to improve the efficiency of human generated markers.

The models developed in this thesis make use of a geometrical calculation known as the no-fit polygon. The no-fit polygon is a tool for determining whether polygons A and B overlap. It also gives all feasible positions for polygons B with respect to polygon A, such that the two polygons do not overlap. For the case when both polygons A and B are non-convex, current calculation methods are either time consuming or unreliable. This thesis presents a method which is both computationally efficient and robust for calculating the no-fit polygon when polygons A and B are non-convex.

When tested on a set of industrial markers, the compaction algorithms improved the marker efficiencies by over 1.5% on average.

Acknowledgments

This research is part of a project funded by FRST (Foundation for Research, Science, and Technology). I would like to thank my supervisor Paul Tu, and Peter O'Halloran, Venkat, and Dennis Parker of Macpac Wilderness for arranging the project, and their assistance throughout the duration of the project.

I would also like to thank my associate supervisor John Raffensperger for his contribution to my thesis. Without his encouragement this thesis would most likely not have been completed. I would also like to thank him for his help with proof-reading the drafts of my thesis and other associated papers.

Thanks also go to my two external referees, Linus Schrage and Lawrence Wein, for their comments and their prompt evaluations of the thesis.

I also appreciate the assistance given by one of my fellow PhD students, Simon Ferguson, on creating the user interface code within my software. Simon also helped relieve stress and boredom throughout the course of my PhD with the odd game of draughts or Risk.

Chapter 1: Problem Description

1.1 Introduction

The cutting of 2-dimensional parts from a larger resource with minimal wastage is an important problem in many industries. This problem is known as the 2-dimensional cutting stock problem. This problem is applicable to industries including leather, glass, and sheet-metal cutting and garment manufacturing. Although the problem is similar in each industry, each has its own set of constraints and rules on how parts may be cut. Dyckhoff [22] gives a classification of cutting stock problems.

1.2 Problem Definition

The work in this thesis concentrates on the garment manufacturing industry. The 2-dimensional cutting stock problem for this industry is as follows:

Arrange a set S of 2-dimensional patterns on a bolt of cloth of fixed width W, such that the required length, ml, of the cloth is minimised. ml is the distance from the left edge of the cloth, to the right most point of the arrangement. Any feasible arrangement is called a *marker*. A example marker is shown in Figure 1.1. This marker contains 108 patterns.

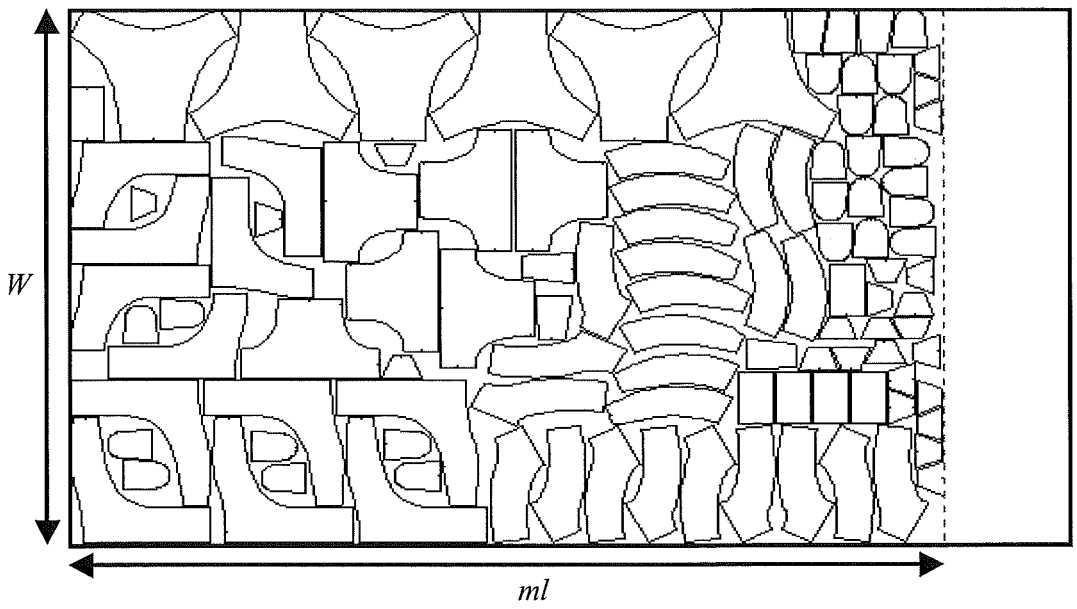

Figure 1.1: Example marker

Each pattern has its own set of possible orientations. The possible orientations are 0°, 90°, 180°, 270°. A pattern may also be flipped in the x or y plane. The available orientations for a particular pattern is usually dependent on factors such as the type of fabric being cut, and what type of garment is being produced.

In certain cases, a pattern can also be offset by up to 3° from any given orientation. This option has not been explored in this thesis, but may provide an opportunity for future research.

Due to the width and accuracy of the cutting blade, patterns must be separated by a small distance G. Any two polygons whose separating distance is less than G are considered to overlap. Some cutting machines do not require parallel edges from two patterns to be separated. In this case, $G = 0$.

1.3 Marker Efficiency

The efficiency of a marker is measured by the total pattern area / marker area. Efficiency varies depending of the shape and number of patterns being cut, but an improvement in efficiency can result in significant savings. Markers are usually created by humans with the aid of CAD software. Many researchers have attempted to create automatic marker making software but have failed to produce marker efficiencies as high as human generated ones.

1.4 Pattern Representation

A large proportion of the complexity of the 2-dimensional cutting stock problem is in pattern overlap computation. The complexity of the overlap computation depends on the pattern representation.

Pattern representation is split into two main approaches.

Firstly, an approximate polygon representation was developed by Adamowicz and Albano [1], and others such as Albano[5] and Cerny[14]. Each pattern is represented by a sequence of points, and the accuracy of approximation can be improved by choosing more points in the representation.

The second approach, proposed by Ismail and Hon [37], is a grid approximation represented by a number of connected squares of equal size. This is called a raster approximation. The advantage of the raster approximation is that it enables fast pattern

overlap computations. However, it is generally less accurate than the polygon approximation.

The two representations are shown below in Figure 1.2.

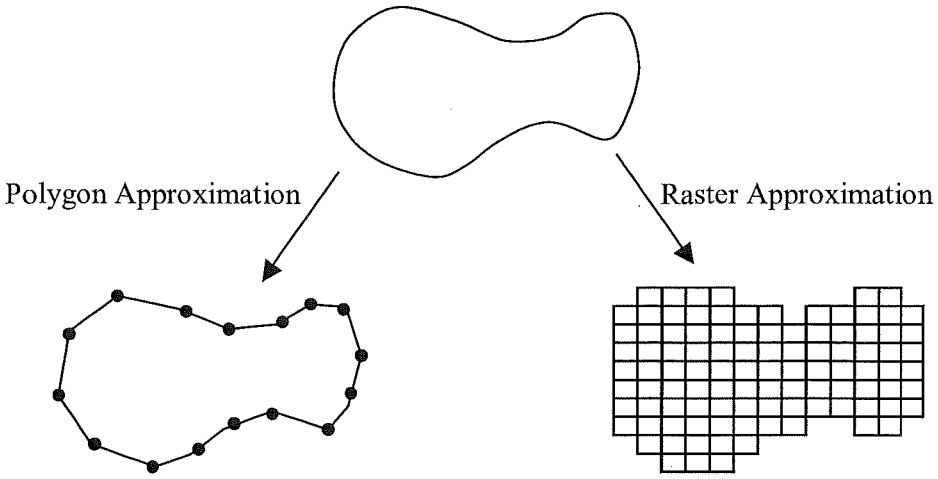

Figure 1.2: Pattern approximation

The method used in this thesis is the polygon approximation. The main reason for this is that the industrial marker data available is already in polygon form. The polygon approximation is also more accurate than the raster approximation.

1.5 Organisation of Thesis

The remainder of the thesis is organised as follows:

Chapter 2 will give a brief description of relevant literature for the 2-dimensional cutting stock problem.

Chapter 3 summarises the contribution of this thesis to the field of 2-dimensional cutting stock.

Chapter 4 gives an explanation of the no-fit polygon, and how it is relevant to 2-dimensional packing. A new method for calculating the no-fit polygon of two non-convex polygons is given. It uses a slope-based method, and is more robust and efficient than other slope-based methods. Other calculation techniques exist, however, these are time consuming when the complexity of the polygons is high.

Chapter 5 uses the concept of the no-fit polygon to create a global optimisation model for the 2-dimensional cutting stock problem. The model is a mixed integer

programming formulation, which cannot be solved in a practical amount of time for industrial sized problems.

Chapter 6 gives a model for compacting a pre-generated marker which does so by applying simultaneous translations to each polygon. The model is a restricted, linear version of the global optimisation model presented in chapter 6. The model is extended to an algorithm, which has successfully improved marker efficiency of industrial markers by almost 0.9% on average.

Chapter 7 extends the model of chapter 6 to allow polygons to simultaneously change orientation and translate at the same time. The model is a mixed integer programming formulation, and has successfully improved marker efficiency of industrial markers (when combined with the algorithm from chapter 6) by over 1.5% on average.

Chapter 8 gives a summary of a software application created from the work in this thesis.

Finally, chapter 9 summarises the results, and discusses the possibilities for future work.

Chapter 2: Related Work

2.1 Introduction

As the 2-dimensional cutting stock problem arises across a broad spectrum of industries, and is of significant financial value, a large amount of research has been done in this area. This chapter gives a brief summary of the main contributions to this field.

2.2 Problem Complexity

Milenkovic et al [48] show how the 2-dimensional cutting stock problem can be reduced to the one-dimensional bin packing problem. Coffman [17] has shown that the one-dimensional bin packing problem can be reduced to a problem of deciding whether a list of numbers can be partitioned into two sets with equal sums. This partitioning problem has been shown to be NP-Complete by Karp [40]. Therefore, the 2-dimensional cutting stock problem is at least NP-Hard.

Blazewicz et al [9] and Feng et al [23] claim the 2-dimensional cutting stock problem has been proven to be NP-Complete.

The following is a statement from Foulds [25] about the NP-Complete class of problems:

"It can be shown that the solution to any given NP-complete problem instance can be transformed into that for any other NP-complete problem in a number of steps which is of polynomial order. This means that if an algorithm of polynomial order can be found for any NP-complete problem, then all the NP-complete problems can be solved by polynomial algorithms. There is no evidence to suggest that such an algorithm exists."

Because of this, many researchers have developed solution procedures which find "good" solutions, not necessarily optimal ones.

2.3 Approximately Optimal Solution Procedures

One of the first solution procedures for the 2-dimensional cutting stock problem was proposed by Gilmore and Gomory [28]. They proposed a column generation technique, which required a huge number of columns in the LP model. Restrictions were placed to limit the number of columns. An example restriction was that the cutting had to be done

in stages, which in effect meant only allowing guillotine cuts. Guillotine cuts allow cuts only from one side of the marker to the other.

Gilmore and Gomory [29] applied the 2-dimensional cutting stock problem to the knapsack problem. This was done using a dynamic programming technique.

Adamowicz and Albano [1] developed a dynamic programming method for the 2-dimensional cutting stock problem when all patterns are of rectangular shape. This was done by laying the rectangles with a common dimension into strips. Adamowicz and Albano [2] furthered this work by considering how to cluster two arbitrary polygons into a rectangle. This idea can be extended to cluster any number of polygons together.

Albano and Orsini [3] extend the work of Adamowicz and Albano [1] by using a tree search heuristic algorithm. Here, the laying out of rectangles into strips is broken down into sub-problems. Albano and Osrini also state that even restricted versions of the 2-dimensional cutting stock problem can be solved only for medium sized problems.

Albano and Sappupo [4] reduce the 2-dimensional cutting stock problem to a search of an optimal path in a graph. Using a heuristic search method they implement an algorithm which produces an approximate solution which proves to be of good quality and efficient in terms of solution time.

Dori and Ben-Bassat [19] dealt with the problem of placing non-convex polygons onto a rectangular plane using a two phase approach. Firstly, the non-convex polygons were converted into convex polygons. Secondly, a polygon was found which could enclose each of the convex polygons, and also tile the plane.

Beasley [6] developed a mixed integer model to optimally solve the 2-dimensional cutting stock problem for rectangular patterns that were not restricted to guillotine cuts. This was the first time an exact algorithm for this type of model had been presented. It is capable of solving small problems to optimality.

Qu and Sanders [52] used an approach which either enclosed an arbitrary polygon in a rectangle, or broke it down into a number of rectangles. Greater accuracy could be achieved by using a smaller rectangles. A simple heuristic based on placing the rectangles into the lower left hand corner of the marker was used.

Jain et al [39] used a simulated annealing approach to create a marker of sheet metal patterns with minimal waste. The method used an integer grid technique to compute overlap between the patterns, and then applied the simulated annealing algorithm to

remove any overlap. This is done for only a few patterns, and then the configuration is repeated.

Fujita et al [26] proposed a hybrid approach which uses the genetic algorithm and a local minimisation algorithm. Genetic strings are represented by an ordered list of neighbouring polygons and their positions, The genetic algorithm is then used to exchange clusters of polygons. Finally, a local minimisation algorithm arranges the clusters and the remaining polygons by attempting to minimise the distance between neighbouring polygons. The process can produce good solutions, but the running time is very large even when the problem size is small.

Scheithauer and Terno [54] present a globally optimal mixed integer programming model for the 2-dimensional cutting stock problem with the constraint that the polygons cannot rotate. They use the concept of the no-fit polygon (see chapter 4) to generate non-overlapping constraints. Their model is not solved.

Blazewicz et al [8] proposed a two-stage approach. The first stage is to create an initial marker using a simple heuristic. The second stage then adjusts the initial marker using a tabu search algorithm.

Prasad [51] used a set of heuristics to create markers of irregularly shaped sheet-metal patterns. Heuristics were developed for nesting single pattern types into a single row, nesting single pattern types into multiple rows, and nesting multiple pattern types into a single row. The solutions compared favourably with the manual procedures in industry.

Ismail and Hon [38] used the raster approximation to represent patterns. Each pattern can then be represented as a binary string, containing information on the shape of the pattern, its x and y coordinates, and its orientation. The genetic algorithm is used to create the marker, using crossover, mutation and swap as the genetic operators. The objective function of the algorithm contains a penalty function for overlapping patterns. The method was tested on two small problems, and a large number of generations were required before good solutions were found.

Heckmann and Lengauer [33] proposed a simulated annealing approach with a dynamic cooling schedule. The approach starts by creating an initial "loose" marker, and then applies the simulated annealing algorithm to remove overlaps and to move the patterns around. Patterns are approximated by both raster and polygonal approximations. The polygonal approximation gave the best results.

Lamousin et al [43] used a leftmost lowest placement policy to determine the position of each irregularly shaped polygon. Orientation is determined by the smallest "shadow" created when firstly a light is shined from the left, then from above. Polygon selection is based upon a potential waste function comprising of two components, true waste and future waste. A tree-like search is developed to find the best solution. By searching for and expanding on the best node at each level, the majority of permutations are eliminated.

Han and Na [32] develop a two-stage approach which uses firstly neural networks, and then simulated annealing. The patterns are approximated by a number of rectangles and circles. The neural network is used to obtain a rough initial marker. The simulated annealing algorithm then improves the initial marker by translating each pattern. Positions of two patterns can also be swapped.

Grinde and Cavalier [30] investigated whether a single convex polygon could be translated and / or rotated to fit inside a fixed convex polygon. An algorithm is developed which uses parametric programming with a non-linear rotation parameter.

Grinde and Cavalier [31] extended their work from [30] to determine whether two convex polygons could fit inside a convex polygon when both polygons are allowed to translate and rotate. The method also uses parametric programming. This was the first work which dealt with the containment of two polygons.

Lai and Chan [42] present a simulated annealing algorithm which can be applied to the 2 or 3-dimensional cutting stock problems. The 2-dimensional case is concerned with placing different sized rectangles onto a large rectangle. The simulated annealing algorithm is used to generate a cutting order, whereby the rectangles are placed using a leftmost lowest placement policy.

Hong et al [36] present a nesting system with an analogical learning mechanism. The irregular patterns are processed in advance, and a rectangular or combination of rectangular enclosures are obtained. Then a solution is obtained using a heuristic search. In the solving mechanism of the system, an analogical reasoning mechanism is used. The system has a certain degree of learning, and the efficiencies and speed of nesting gradually increases as the total nesting time increases.

Dowsland et al [21] proposed a "jostle" algorithm. An initial marker is created using a leftmost placement policy, with a random ordering of polygons. Polygons are then re-

ordered in decreasing order of the x-coordinates of their rightmost points and the layout is created according to a rightmost placement policy. The polygons are then again re-ordered in increasing order of the x-coordinates of their leftmost points and the layout is created according to a rightmost placement policy. This is continued for a fixed number of iterations. The idea is based upon the observation that any unevenness in stored granular products can be removed by shaking the container up and down.

Oliveira et al [50] present a constructive algorithm called TOPOS for the problem when 180° rotation is allowed for irregular patterns. The marker is built by successively adding a new pattern to a partial solution, that is, to the set of patterns previously placed. Several criteria to choose the next pattern to place, and its orientation are proposed. 126 variants are produced by the combination of the placement and the orientation criteria. The results are compared over five data sets to the methods proposed by Dowsland et al [21], Blazewicz et al [8], and earlier work by Oliveira and Ferreira. TOPOS produces the best known solutions for two of the five data sets.

Heckmann and Lengauer [34] used an iterative greedy strategy to create a set of markers, from which the best is chosen. They then use their simulated annealing algorithm from [31] to improve the efficiency of the marker. They also used a branch-and-bound method to compute lower bounds on marker efficiency. The methods were tested on a variety of industrial data, and the difference between the upper and lower bounds ranged between 0.4% and 7.7%.

Milenkovic and Daniels [47] present an algorithm for determining whether k non-convex polygons can fit inside (without rotation) a non-convex container. The algorithm is generalised to find the minimum enclosing rectangle for k non-convex polygons. The approaches use mathematical programming principals. The algorithms are shown to outperform purely geometric containment algorithms. Containment with up to ten non-convex polygons have been solved using industrial data.

Burke and Kendall [10] examine evaluation functions used in applications of evolutionary algorithms (tabu search, simulated annealing, and the genetic algorithm) to the 2-dimensional cutting stock problem. Often evaluation functions are the most computationally expensive part of an algorithm. A new evaluation function is proposed based on the no-fit polygon (see chapter 4) which significantly increases the speed of the evolutionary algorithms.

Burke and Kendall [11] extend their work of [10] by developing a simulated annealing algorithm with a no-fit polygon based evaluation function to create markers for convex polygons. They show that simulated annealing out performs standard hill climbing techniques.

Burke and Kendall [12] then develop a genetic algorithm and a tabu search algorithm for the 2-dimensional cutting stock problem. These algorithms are compared to the simulated annealing algorithm developed in [11]. The tabu search algorithm is shown to produce the best results.

Burke and Kendall [13] apply a new type of evolutionary algorithm, the ant algorithm, to the 2-dimensional cutting stock problem. The results are then compared to previous applications of tabu search, simulated annealing, and the genetic algorithm. They find that the ant algorithm outperforms the genetic algorithm, matches simulated annealing, and is outperformed by tabu search.

For other surveys of the cutting stock problem, see Dowsland and Dowsland [20], Dyckhoff[22], and Cheng et al [16].

Chapter 3: Contribution

Marker making is a required process in many industries. Current automatic marker making software is unable to create markers whose efficiency match that of a human marker maker. For large companies, even a small improvement in marker efficiency can save a company millions of dollars per year.

The contribution of the work in this thesis will allow companies to improve their current marker efficiencies.

Firstly, a method for computing the no-fit polygon is given. The no-fit polygon is a tool for determining whether polygons A and B overlap. It also gives all feasible positions for polygons B with respect to polygon A, such that the two polygons do not overlap. For the case when both polygons A and B are non-convex, current calculation methods are either time consuming or unreliable. This thesis presents a method which is both computationally efficient and robust for calculating the no-fit polygon when polygons A and B are non-convex. It is an extension of the algorithm by Ghosh [27] for calculating the no-fit polygon of a convex polygon A, and a non-convex polygon B.

Secondly, a globally optimal model for the translational 2-dimensional cutting stock problem is given. This model is based on the no-fit polygon. It cannot be solved in a tractable amount of time for industrial sized problems. A restricted version of the model is developed, which, when given a starting marker (for example, a human generated marker), can improve the marker by applying a series of simultaneous translations to every polygon in the marker. This is called a compaction of the marker, and has improved the efficiencies of a set of 50 industrial markers by almost 0.9% on average.

Finally, the compacting model is extended to allow polygons to simultaneously change orientation, as well as simultaneously translate. This rotational compaction model further improves the set of industrial markers by over 0.6% on average.

The development of the compaction and rotational compaction models, in conjunction with fast, robust no fit polygon calculations, has enabled the creation of computer software for improving human generated markers in a practical amount of time.

Chapter 4: The No-Fit Polygon

4.1 Introduction

An issue in 2-dimensional packing is determining the set of feasible locations that one polygon may take with respect to another polygon, such that the polygons do not overlap. This set of locations is known as a *no-fit polygon* (NFP). The terms *Minkowski sum, Φ-function, hodograph, dilation, envelope* and *configuration space obstacle* have also been used by other researchers.

Let each polygon be represented by an ordered list of vertices. The location of each polygon i in the 2-dimensional plane is represented by a reference point, r_i. The reference point is located at point (0, 0) of a polygon's local coordinate system (see Figure 4.1.1).

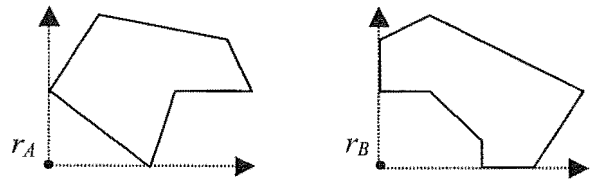

Figure 4.1.1: Local coordinate system

The perimeter of the no-fit polygon of polygon A and polygon B (denoted as NFP[A, B]) gives the points that r_B can take relative to r_A such that polygon B is touching polygon A. If r_B is located inside r_A + NFP[A, B] then the two polygons overlap. Conversely, if r_B is outside NFP[A, B] then the two polygons do not overlap, and do not touch (see Figure 4.1.2).

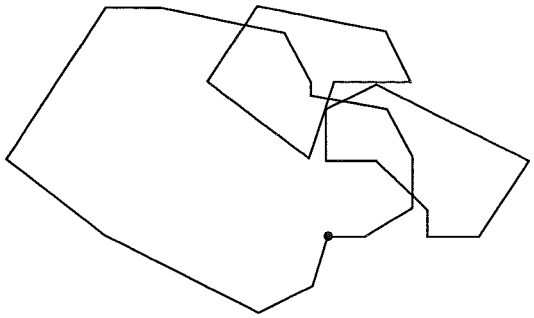

Figure 4.1.2: NFP[A, B]

Several publications (such as O'Rourke[49]) have shown the relationship between a form of vector addition known as the Minkowski sum and the NFP. If we let the vertices of polygons A and B be represented as vectors, then the Minkowski sum of A and B is defined in Equation (4.1):

$$A \oplus B = \{x + y \mid x \in A, y \in B\} \tag{4.1}$$

where $x + y$ is the vector sum of points x and y.

Geometrically, the *outer envelope* of the Minkowski sum of A and $-B$ is the equivalent of NFP[A, $-B$][1] (see Figure 4.1.3). $-B$ can be obtained by rotating polygon B by 180° or multiplying the vector representation of polygon B by -1.

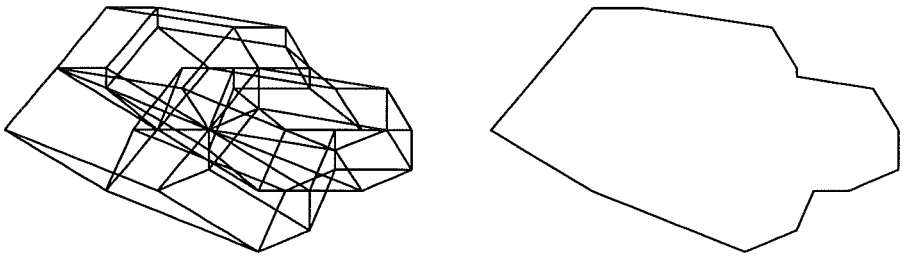

Figure 4.1.3: Outer envelope

In most 2-dimensional packing algorithms, many NFPs must be calculated. These calculations are computationally expensive. Most of the computational time is spent in calculating the outer envelope of the Minkowski sum. The number of edges in a Minkowski sum is $2mn$, where m and n are the number of edges on polygons A and B respectively. In industrial cases, polygons may have in excess of 100 edges. Polygons of this complexity result in Minkowski sums of very large size, and therefore require a

[1] From now on NFP[A, $-B$] will be referred to as NFP[A, B]

time consuming process to create the corresponding NFP. Because of this, many researchers have tried to calculate NFPs using methods other than the Minkowski sum.

Cunninghame-Green [18] showed that for the case when polygons A and B are convex, NFP[A, B] can be created by ordering the edges of A and $-B$ in increasing slope order. NFP[A, B]'s edges correspond exactly to this slope order (see Figure 4.1.4).

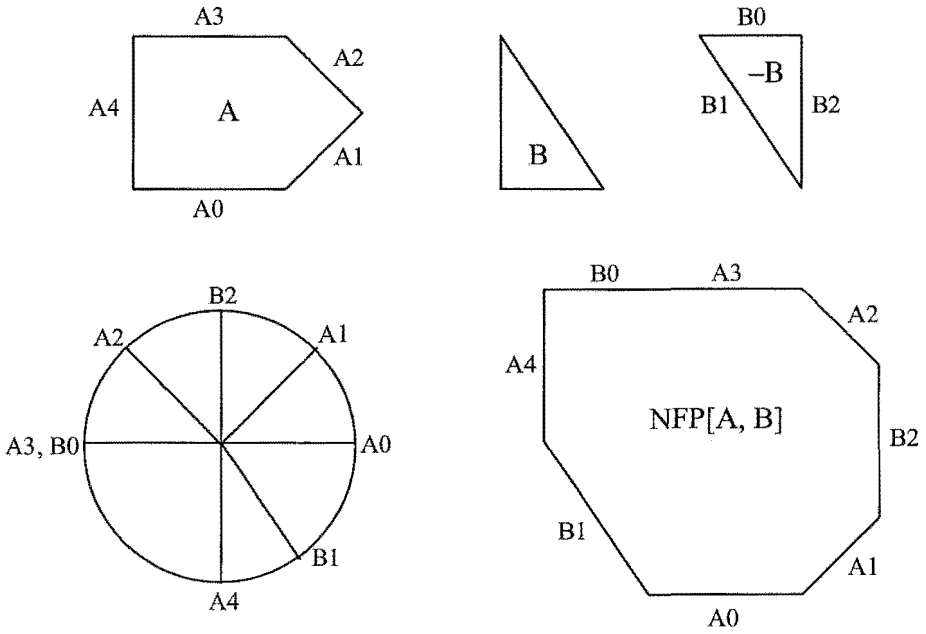

Figure 4.1.4: NFP for convex polygons

When one or more of the polygons are non-convex, a way of calculating the relevant NFP is to decompose each polygon i into a set of N_i convex sub-polygons ($CSP[i]_1 \rightarrow CSP[i]_{Ni}$). Overlap will occur between the two polygons if any sub-polygon of A overlaps any sub-polygon of B. NFP[A, B] is the union of $NFP[CSP[A]_i, CSP[B]_j]$, where $i = 1 \ldots N_A$ and $j = 1 \ldots N_B$.

There are two drawbacks to the polygon subdivision approach. Firstly, efficient algorithms are required for polygon decomposition and polygon composition. Secondly, it is possible that a non-convex polygon that has N edges in cavities (see Figure 4.1.5) can be decomposed into no less than N CSPs. The NFP of two of these polygons would require the composition of N^2 sub NFPs. Polygons used in industries such as garment manufacturing often have large numbers of edges in their curve-like cavities, and the sub-division technique becomes inefficient compared with the slope-based techniques described later in this chapter. A method of the convex subdivision technique is given by Lo Valvo [45].

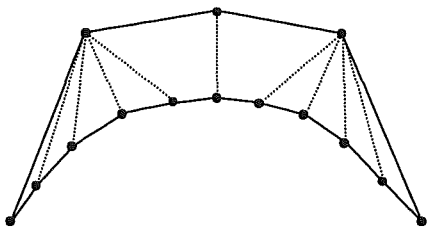

Figure 4.1.5: Convex subdivision

Mahadevan [46] describes a different method for calculating the NFP for two non-convex polygons. The basis of this algorithm is that one polygon orbits another, and at each position the reference point of the orbiting polygon is stored. These stored points become the vertices of the NFP. Kendall [41] found certain cases where Mahadevan's algorithm produces incorrect NPFs. Kendall gives a modified algorithm, which corrects these degenerate cases. The main drawback of the orbiting polygon approach is that as the orbiting polygon slides along an edge of the stationary polygon, a test must be performed in order to calculate the sliding distance. This is because for non-convex polygons, the sliding distance is not always equal to the stationary edge length. The test involves extending every vertex on the orbiting polygon in the direction of motion by the length of the sliding edge. Extended vertices are then checked for intersections with the stationary polygon. For polygons with a large number of vertices, this method can be computationally expensive. Also, an orbiting method may not detect that polygon B may be placed in a cavity of polygon A, when polygon B cannot slide in from the outside.

4.2 Ghosh's Approach

The method presented in this chapter is an extension of the algorithm given by Ghosh [27]. The method is based on the fact that the NFP of any two polygons is a function of their boundary edges. An outline of this algorithm is now given.

Firstly, we give some definitions and starting conditions used in Ghosh's method and the remainder of the chapter:

Condition 4.2.1: The edges of polygon A are ordered anti-clockwise, starting at the lowest, leftmost edge.

Condition 4.2.2: The edges of polygon B are ordered anti-clockwise, starting at the lowest, leftmost edge. B is then inverted to give $-B$. $-B = (-1)*B$.

Definition 4.2.1: If edge i extends from point D to point E, and edge $i + 1$ extends from point E to point F, then $\alpha(i) = DE \times DF$.

Definition 4.2.2: An edge i of a polygon is a *turning point* if the sign of $\alpha(i)$ is opposite to the sign of $\alpha(i+1)$.

Bennell et al [7] states that a polygon is convex if and only if it does not contain any turning points. Otherwise it is non-convex.

The initial stage of Ghosh's approach is to sort all the edges of polygon A and polygon B by slope into one list which we will call *MergeList*. If both polygon A and B are convex, then MergeList gives the edge order for NFP[A, B], and the method is equivalent to that of Cunninghame-Green [18].

Assuming polygon A is non-convex, and polygon B is convex, the method proceeds as follows:

Starting in MergeList at the first edge of polygon A, visit the edges of A in order, and add them to the list of edges (*NFPList*) which make up NFP[A, B]. If edge A is a turning point, then the direction of travel along MergeList is reversed. Any edges of B which are passed are added to NFPList. B edges are positive if the direction of travel forward, and negative if the direction is backward. This continues until the first edge of polygon A has been returned to. The resulting NFPList we will call *GhoshList*. The above algorithm is given in Pseudocode 4.2.1.

```
p = Element in MergeList which corresponds to A0
i = 0
Dir = 1
Loop{
    If MergeList[p].PolygonType = A Then
        If MergeList[p].PolygonIndex = i Then
            GhoshList = GhoshList + MergeList[p]
            If MergeList[p].IsTurningPoint = True Then Dir = Dir * -1
            i = i + 1 (If i > A.Size Then i = 0)
        End If
    Else
        GhoshList = GhoshList + MergeList[p] * Dir
    End If
    p = p + Dir
}While(i ≠ 0)
```

Pseudocode 4.2.1: Algorithm to find GhoshList

The process of finding GhoshList is seen easily with what Ghosh calls a slope diagram (see Figure 4.2.1). The points on the diagram are at the slope of the edges of polygons *A* and *B*.

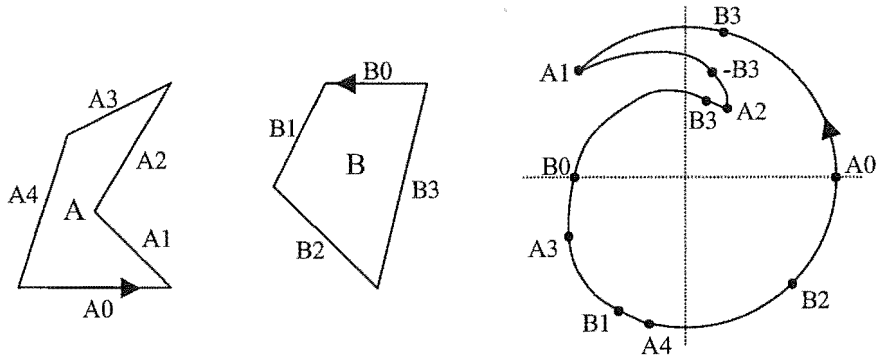

Figure 4.2.1: Slope diagram of *A* and *B*

Following around the slope diagram, starting and finishing at *A0*, mimics the process of traversing over MergeList. The outer envelope of NFPList gives NFP[*A*, *B*] (see Figure 4.2.2).

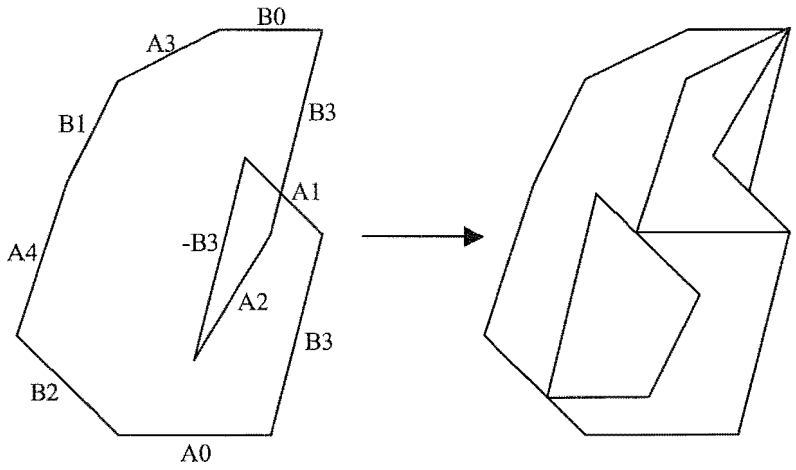

Figure 4.2.2: NFP[*A*, *B*]

Ghosh's method works for all simple polygons (no holes) when polygon *A* is non-convex and polygon *B* is convex. The method also works when both polygons are non-convex, as long as no two cavities from either polygon interfere which each other. This occurs when an interval of MergeList has wrongly ordered edges from both polygons. When this does occur, this interval must be traversed in two or more parallel paths.

Although the theory of traversal by parallel paths holds true for complex non-convex cases, there are considerable implementation problems in sorting out the paths. These difficulties led Bennell et al [7] to seek a different approach.

Their approach exploits the fact that the NFP of a non-convex polygon and a convex polygon can be easily and efficiently found by Ghosh's method. When both polygons are non-convex, the convex hull of polygon B (conv[B]) is used. Conv[B] can be regarded as a copy of B with its cavities replaced by dummy edges. Ghosh's method is then used to create an edge listing (GhoshList) for NFP[A, conv[B]]. GhoshList may include both positive and negative occurrences of the dummy edges.

For each type of dummy edge, GhoshList is split into segments containing a positive or negative dummy edge. Each occurrence of a dummy edge is then replaced by a combination (ReplaceList) of the B edges from which the dummy edge was derived (edges $B_{CavStart} \rightarrow B_{CavFin}$), and A edges within the segment. Starting at the dummy edge, all occurrences of $B_{CavStart}$ within the segment are "found" and added to ReplaceList before moving on to finding $B_{CavStart+1}$. Any A edges "passed" on the way are also added to ReplaceList. This is continued until all B_{CavFin} edges have been found, and the dummy edge has been returned to. The dummy edge in GhoshList is then replaced by ReplaceList.

Bennell's method works well when the edges in a B cavity occur in slope order. However, if the B edges within a cavity are out of slope order, an incorrect NFP is occasionally calculated.

The calculation difficulties of Bennell's method has motivated development of a more robust and efficient method of calculating NFPs. Like Bennell's method, it exploits the fact that the NFP of a non-convex polygon and a convex polygon can be easily and efficiently found by Ghosh's method. However, the new method does not use dummy edges to replace cavities of B.

4.3 A New Method

Intuitively, it would seem a good idea to modify Bennell's method to start "looking" for the next B edge of a cavity once an occurrence of the current B edge has been found, instead of continuing to look for the furthest occurrence of that B edge. However, if there is more than one occurrence of a B edge in any given segment then this approach will run into difficulties.

A solution to this is to make sure that each traversal segment contains only positive or negative occurrences of each *B* edge of a particular cavity. Replacing a *B* cavity with a dummy edge *D* will not guarantee this (see Figure 4.3.1).

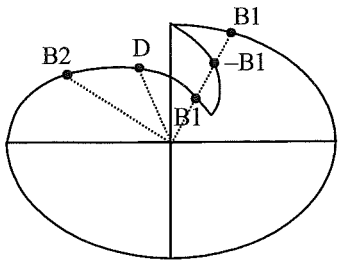

Figure 4.3.1: *B1* occurring more than dummy edge *D*

Figure 4.3.1 shows a dummy edge *D*, whose cavity is composed of edges *B1* and *B2*. In this example, Bennell's method would require only one segment which would contain a single occurrence of *D*. However this segment contains both positive and negative occurrences of *B1*.

To guarantee that there is only positive or negative occurrences of a given cavity *B* edge, we split the traversal of GhoshList using the algorithm given in Pseudo-code 4.3.1:

p = The element in GhoshList which corresponds to *A0*.
TravelDir = –1
CurrentSign = +1
TravelSign = *+1*
i = 1

Loop1{
    p = *p* + *TravelDir*
    If GhoshList[p].PolygonType = A **Then**
        If GhoshList[*p*].PolygonIndex = 0 **And** TravelDir = –1 **Then**
            Seg[i].*End* = *p*
            Exit Algorithm
        Else If GhoshList[*p*].IsTurningPoint = True **Then**
            TravelSign = *TravelSign** *–1*
            Seg[*i*].Start = *p*
        End If
    Else If GhoshList[*p*].*IsInCavity* = True **And** *CurrentSign* ≠ *TravelSign* **Then**
        TravelDir = *TravelDir* * –1
        p = *Seg*[*i*].*Start*
        TravelSign = *TravelSign** –1
        Exit Loop1
    End If
}

```
Loop2{
        p = p + TravelDir
        If p = Seg[1].Start Then
                Seg[i].End = p
                Exit Algorithm
        Else If GhoshList[p].PolygonType = A Then
                If GhoshList[p].IsTurningPoint = True Then
                        TravelSign = TravelSign* –1
                        Seg[i].End = p
                End If
        Else If GhoshList[p].IsInCavity = True And CurrentSign ≠ TravelSign Then
                i = i + 1
                Seg[i].Start = Seg[i–1].End
                CurrentSign = CurrentSign * –1
        End If
}
```

Pseudo-Code 4.3.1: Algorithm to split GhoshList into segments

The above algorithm splits GhoshList into i segments, separated at certain turning points of A. The segments are split so that each segment contains only positive or only negative occurrences of B edges which belong to cavities.

A cavity is an ordered list of at least three points, with only the first and last points being on the convex hull. The two cavities of polygon B are shown in Figure 4.3.2.

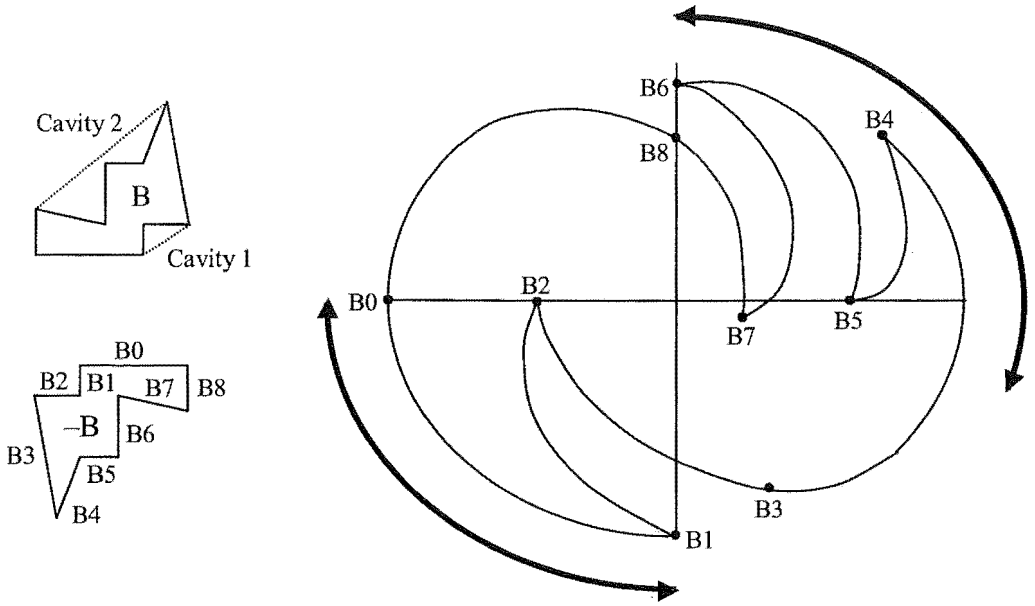

Figure 4.3.2: Spans of cavity 1 and cavity 2

The *span* of a cavity k is the arc spanning the farthest clockwise and farthest anticlockwise edges of k in the polygon's slope diagram. An example is shown in

Figure 4.3.2. Cavity 1's span is between *B1* and *B2*, and cavity 2's span is between *B7* and *B6*.

We say that a cavity *k* of *B* *interferes* with segment *i* if the span of cavity *k* intersects segment *i*.

Figure 4.3.3 shows the slope diagram for polygons *A* and *B*. It will be split into 4 segments, *A9→A1*, *A1→A3*, *A3→A6*, and *A6→A9* using the method described earlier. For each of the four segments, we count the cavities of *B* whose span intersects it. Cavity 1 intersects segments *A9→A1*, *A3→A6*, and *A6→A9*. Cavity 2 intersects *A9→A1*, *A1→A3*, and *A3→A6*. The algorithm must "process" a cavity for each segment that intersects the cavity.

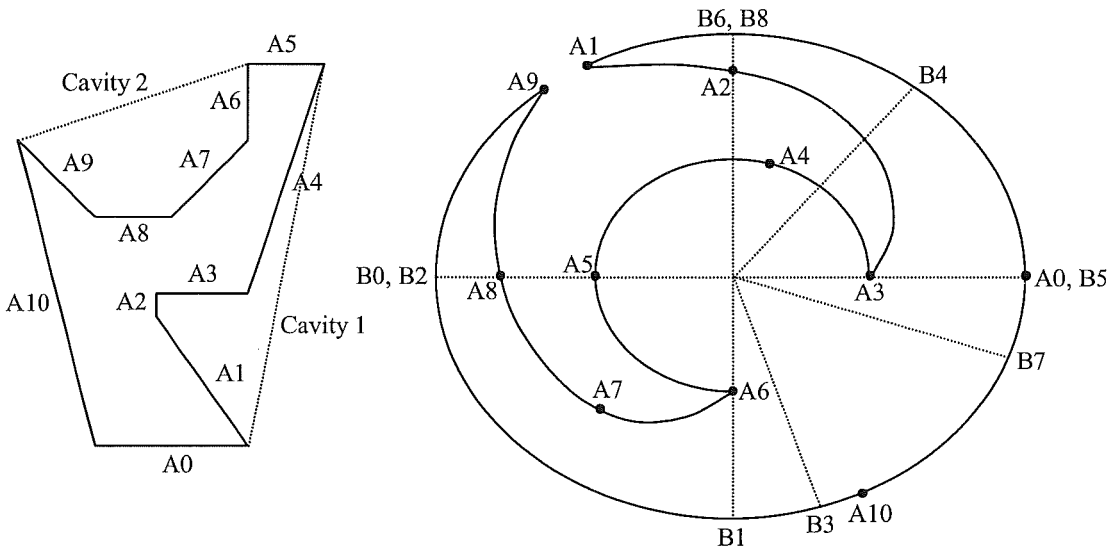

Figure 4.3.3: Slope diagram for *A* and *B*

To create NFPList with correct numerical ordering of both *A* and *B* edges, GhoshList and MergeList must be re-traversed together.

4.3.1 Finding the Starting Point

We start the re-traversal in GhoshList at a *B* edge which is either not part of a *B* cavity, or is the first edge of a *B* cavity. Because B0 starts at the lowest leftmost vertex of *B* (therefore its starting vertex is on conv[*B*]), at least one of these conditions will always hold for *B0*. *+B0* will appear in GhoshList at least once, and it is arbitrary which occurrence of *+B0* is selected as the starting point. The *A* segment is selected which contains the chosen starting point as the *current segment*. Our starting direction of

traversal is backwards (the same direction as travelling $A1 \rightarrow A0$) if $B0$ is a turning point, otherwise it is forwards.

4.3.2 Traversing Over A Segment

As stated earlier, all cavities which intersect the current segment must be processed. All non-cavity B edges which lie on the segment of GhoshList which corresponds to the current segment must also be processed. Once all intersecting cavities in the current segment and non-cavity B edges have been processed the algorithm moves onto the next segment. Starting at $B0$, we search for the edges of intersecting cavities and non-cavity B edges in numerical order by traversing back and forth (direction is reversed if the B edge is a turning point) between these edges, taking note of any A edges which are passed, and the direction that they are passed in.

From the example in Figure 4.3.3, if we choose $B0$ (on segment $A9 \rightarrow A1$) as our starting point, then we traverse from $B0$ to $B1$ in a forward direction. When $B1$ has been reached, we have entered cavity 1. All edges of this cavity must be found, even if this involved traversing off the boundaries of the current segment. We then turn backward to find $B2$. Cavity 1 has now been processed, as all its constituent edges have been found. We now turn forwards, heading for $B3$. Once $B3$ is found we continue forward towards $B4$, taking note of passing $A10$ and $A0$ in the forward direction. At $B4$ we turn backward to find $B5$, and then turn forward again to find $B6$. At $B6$ we turn backward to find $B7$, and in the process pass $A0$ in the backward direction. At $B7$ cavity 2 as been processed. We turn forward to find $B8$ and then reach $A1$ while in search of $B0$. Because $A1$ is the end of the current segment, and all cavities and non-cavity B edges intersecting the current segment have been found, we move on to the next segment, $A1 \rightarrow A3$.

Note: The segment $A1 \rightarrow A3$ runs in a clockwise direction. Any B edges found in a segment (or part segment) whose A edges run in a clockwise direction are negative, and the order they are found in is also reversed. So for segment $A1 \rightarrow A3$ we find the B edges $-B8 \rightarrow -B4$.

The edges added to NFPList in the traversal of the $A9 \rightarrow A1$ segment are $<B0, B1, B2, B3, A10, A0, B4, B5, B6, -A0, B7, A0, B8, A1>$.

This process is continued through all segments, until the initial segment is re-entered, and the starting point found.

4.4 Special Cases

The above method needs further explanation when either of the following situations occur: Traversal Moves Onto MergeList (sec. 4.4.1), or Cavity Before *B0* (sec. 4.4.2).

4.4.1 Traversal Moves Onto MergeList

If either the starting or ending limit, *L*, of the current segment is reached, and all relevant cavities and non-cavity *B* edges have not been processed, then the traversal shall continue on MergeList rather than on GhoshList. The traversal starts on MergeList at edge *L* and continues searching for the current *B* edge in the direction that the traversal on GhoshList was taking. Any *A* edges encountered on MergeList are ignored. The traversal will return back to GhoshList when *L* is passed in the *opposite* direction of which MergeList was entered. *L* is added to NFPList.

A degenerate case can occur which causes the traversal of MergeList to continue without ever returning to *L*. The happens directly after the last edge of the final cavity of the current segment has been added to NFPList, when the traversal does not head back to *L* in the opposite direction to which MergeList was entered. Because this is the final cavity of the current segment, after processing this cavity, we want to move onto the next segment. And because the final cavity of the current segment is also the first cavity of the next segment, we can make the transition from the current segment to the next segment within MergeList. After the last edge of the final cavity is added to NFPList, we add *L* to NFPList (opposite sign to previously added *L*). The next segment becomes the current segment, and we traverse back over MergeList starting at the last edge of the final cavity (now the first edge) and add *B* edges to NFPList in order until *L* is reached in the *opposite* direction of which MergeList was entered. *L* is added to NFPList.

4.4.2 Cavity Before B0

As stated earlier, we start our traversals of GhoshList and MergeList at an arbitrary occurrence of *+B0*. However, often the first cavity we come across is not actually the "first" cavity of the starting segment. That is, that cavity is not the first cavity which should be processed after processing all cavities in the previous section. *+B0* is used because it is either not part of a *B* cavity, or is the first edge of a *B* cavity. Instead of moving to the next segment when all the cavities and non-cavity *B* edges have been processed, for the first segment we move to the next segment when we have reached the

"final" cavity. That is, the cavity which should be processed directly prior to moving onto the next segment.

To determine the final cavity of the first segment, we process the cavities as per usual. If, during the traversal of MergeList, we start and finish processing a cavity which intersects the current segment without leaving MergeList, then the final cavity is the cavity which was processed directly prior to this. An example is shown in Figure 4.4.1.

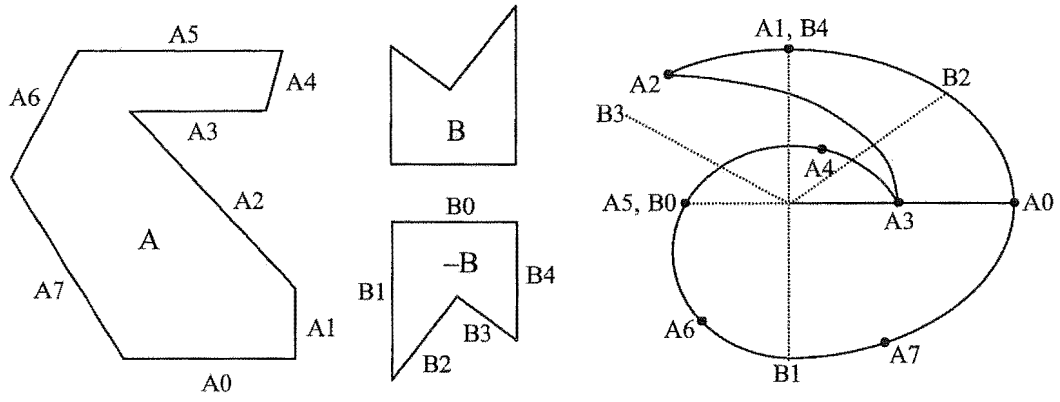

Figure 4.4.1: *+B0* located between two cavities

Here the *B* cavity involving *B2* and *B3* intersects the starting segment *A3→A2* twice. However, *+B0* is located between these two intersections. Starting at *+B0*, we pass *A6*, *B1*, *A7*, *A0*, *B2*, *A1*, *A2*, *B3*. At this point we have processed one intersection (CavA) of the *B* cavity. We continue our traversal past *–A2*, *B4*, and *A2*. We have now reached the end of our segment, but CavB has not yet been processed, so we move onto MergeList. We pass *B0*, *B1*, *B2*, and *B3*, without leaving MergeList. Now we have processed CavB, but it was done entirely within MergeList, so our final cavity of the starting segment is CavA.

So the initial partial traversal of segment *A3→A2* adds edges <*B0*, *A6*, *B1*, *A7*, *A0*, *B2*, *A1*, *A2*, *B3*, *–A2*, *B4*, *A2*> to NFPList, processing CavA. The traversal of segment *A2→A3* adds edges <*–B4*, *–A2*, *–B3*, *A2*, *–B2*, *A3*> to NFPList. The final partial traversal of segment *A3→A2* processes CavB, and adds edges <*B2*, *A4*, *B3*, *B4*, *A5*> to NFPList.

Once GhoshList has been split into segments (Pseudo-code 4.3.1), Pseudo-code 4.4.1 gives the remainder of the algorithm:

p = The element of GhoshList which corresponds to an arbitrary occurrence of *+B0*.
CurrSeg = *OrigSeg* = Segment containing the arbitrary occurrence of *+B0*.
BMulti = +1

```
Dir = +1
NextB = 0
Loop1{
    If GhoshList[p].PolygonType = B Then
        If GhoshList[p].PolygonIndex = NextB Then
            NFPList = NFList + GhoshList[p]*BMulti
            NextB = NextB + BMulti
            If GhoshList[p].IsTurningPoint = True Then Dir=Dir*–1
        End If
    Else
        NFPList = NFPList + GhoshList[p]*Dir
        If GhoshList[p] = Seg[CurrSeg].Start Then
            GoTo TraverseMergeList()
        Else If GhoshList[p] = Seg[CurrSeg].Fin Then
            If Seg[CurrSeg].CavitiesLeft = 0 Then
                CurrSeg = CurrSeg + 1
                BMulti = BMulti * –1
                NextB = NextB + BMulti
            Else
                GoTo TraverseMergeList()
            End If
        Else If GhoshList[p].IsTurningPoint = True Then
            BMulti = BMulti * –1
            NextB = NextB + BMulti
        End If
    End If
    p = p + Dir
    If GhoshList[p] = B0 And NextB = 0 And CurrSeg = OrigSeg And
    Seg[CurrSeg].CavitiesLeft = 0 Then Exit Algorithm
}

TraverseMergeList{
    OrigPos = Pos = Position of p in MergeList
    OrigDir = TotalDir = MergeDir = BMulti * Dir

    Loop2{
        Pos = Pos + MergeDir
        If MergeList[Pos].PolygonType = B Then
            If MergeList[Pos].PolygonIndex = NextB Then
                NFPList = NFPList + MergeList[Pos] * BMulti
                NextB = NextB + BMulti
                If MergeList[Pos].IsTurningPoint = True Then
                    MergeDir = MergeDir * –1
                End If
                If CurrSeg = OrigSeg And Condition1() = True Then
                    CavityBeforeB0 = True
                    Exit Loop2
                ElseIf MergeDir = OrigDir And
                Condition2() = True Then
                    Exit Loop2
                End If
```

```
                End If
            Else If Pos = OrigPos Then
                TotalDir = TotalDir + MergeDir
                If TotalDir = 0 Then
                    Dir = Dir * –1
                    NFPList = NFPList + MergeList[Pos] * Dir
                    Return To Loop1
                End If
            End If
        }
        If CavityBeforeB0 = True Then
            Remove from NFPList all B edges just added in TraverseMergeList
            up to the penultimate cavity added
        End If
        NFPList = NFPList + MergeList[OrigPos] * –1 * OrigDir
        BMulti = BMulti * –1
        Return To Loop1
}

Condition1{
        If every edge of the current cavity was found without leaving MergeList Then
            Return True
        Else
            Return False
        End If
}

Condition2{
        If the current cavity is the final cavity of the current segment Then
            Return True
        Else
            Return False
        End If
}
```

Pseudocode 4.4.1: Algorithm to find NFPList

4.5 Computing the Outer Envelope

Once we have found NFPList, the outer envelope of this must be found to find the final NFP. There are a number of algorithms available to do this, for example Hershberger[35]. However, because these algorithms have a complexity which is greater than linear, it is advantageous to use a divide and conquer strategy, splitting the problem into sub-problems.

Theorem 4.5.1: If an edge, E, of polygon A is a member of conv[A], then E is also a member of NFP[A, $-B$].

Proof: Edge E is a member of conv[A]. This implies that no other A edge, or part of edge, lies in half plane, H, bounded by the line tangential to E and containing the outward normal of E. Let Polygon B lie somewhere in H. Now translate the line tangential to E in the direction of the outward normal of E until a vertex V of polygon B is hit. If V is in contact with E, then every other point on polygon B is contained in H. By definition, the boundary of NFP[A, $-B$] are the points which the reference point of polygon B can take such that polygon A is touched. The points in H which satisfy this are edge E, as V can contact the entirety of E while remaining in H. Therefore NFP[A, $-B$] must contain edge E.

Theorem 4.5.2: If an edge, E, of polygon B is also a member of conv[B], then E is also a member of NFP[A, $-B$].

Proof: Bennell et al [1] show that NFP[A, $-B$] is equal to NFP[B, $-A$] rotated by 180°. Theorem 4.1 states that any edge E which is a member of conv[A] is a member of NFP[A, $-B$]. Because NFPList[B, $-A$] and NFPList[A, $-B$] are equivalent, every edge on conv[B] that is also on polygon B appears on NFP[A, $-B$].

Theorem 4.5.3: The outer envelope of the polygon described by NFPList[A, $-B$] can be constructed without negative edges of NFPList[A, $-B$].

Proof: The outer envelope of the Minkowski sum of A and $-B$ is equivalent to the outer envelope of polygon described by NFPList[A, $-B$]. Equation 4.1 states that the Minkowski sum of A and $-B$ is the result of vector additions of all combinations of points from A and points $-B$. Consequently, the outer envelope of NFPList[A, $-B$] can then be constructed using only positive edges of A and $-B$. Therefore, the outer envelope of the polygon described by NFPList[A, $-B$] can be constructed without negative edges of NFPList[A, $-B$].

Using theorems 4.5.1 and 4.5.2 we can reduce the number of calculations required to find the outer envelope because we know that edges that lie on the convex hull of their respective polygons occur at least once on the outer envelope. However, because construction of NFPList can give rise to multiple positive and negative copies of these edges, we need to establish some rules to determine which of these occurrences are actually members of the outer envelope.

If edge $E \subseteq$ conv(P), then it will occur on NFP[A, $-B$] if it is sliding along a convex vertex of the other polygon Q. For this to occur, the following conditions must hold:

Condition 4.5.1: E is non-negative.

Condition 4.5.2: E is a member of conv[P].

Condition 4.5.3: If the Q edge that precedes E goes from point s to point t, then t must be a point on conv[Q].

Condition 4.5.4: If the Q edge following E goes from point u to point v, then u must be a point on conv[Q].

Condition 4.5.5: The angle of E must be between the angles of the Q edges that precede and follow E. If those Q edges are not a member of conv[Q], then we use the convex edge that would replace them.

We can now divide the problem of finding the outer envelope of NFPList into sub-problems of finding the outer envelope of each set of edges between the edges already identified to lie on the outer envelope. We can further reduce the number of candidate edges for the outer envelope by removing negative edges using theorem 4.5.3.

An example of the reduction in calculation is shown in Figure 4.5.1.

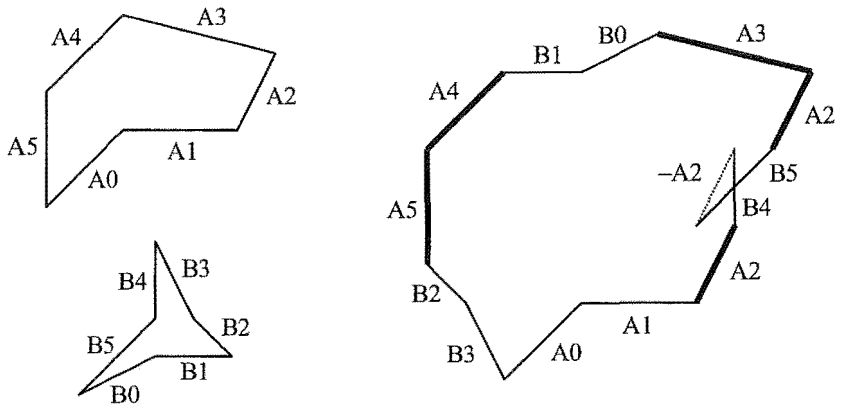

Figure 4.5.1: NFPList for NFP[A, B]

NFP[A, B] is the equivalent of OE[*B0, B1, A4, A5, B2, B3, A0, A1, A2, B4, –A2, B5, A2, A3*], where OE[x] represents the outer envelope of x. However, using theorems 4.5.1 and 4.5.2 we can identify that edges *A4, A5, A2, A2,* and *A3* are on the outer envelope. Using theorem 4.5.3 we can discard edge *–A2*. So the calculation of the edge list of NFP[A, B] can be simplified to <OE[*B0, B1*], *A4, A5*, OE[*B2, B3, A0, A1*], *A2*, OE[*B4, B5*], *A2, A3*>.

4.6 Adjusting for Parallel Edges

The method given in this chapter creates NFP[A, B], such that when the reference point of B is touching the perimeter of NFP[A, B], polygons A and B touch. As explained in chapter 1, often it is the case that polygons must be separated by a distance G. This is simple, and is just a case of buffering NFP[A, B] by G.

Some cutting machines do not require parallel edges from two polygons to be separated. In this case, $G = 0$ for the two parallel edges. Any given point on NFP[A, B] implies that polygons A and B are touching. However, no information is given about whether the contact is vertex-vertex, vertex-edge, or edge-edge (two parallel edges), so there is no way of determining whether $G = 0$, or $G \neq 0$.

Distinguishing areas of the NFP which arise from edge-edge contact can be achieved by "flagging" certain edges in the construction of NFPList. If an edge from A and an edge from $-B$ have the same slope, and are adjacent in NFPList, then this signifies a possible area of NFP[A, B] where the edges are parallel. These candidate edges are flagged. In construction of the outer envelope, if a candidate edge has been flagged and it appears in the outer envelope, then the points on the outer envelope containing this edge are allowed a buffer of $G = 0$, if applicable.

This method can be easily adjusted if there is a tolerance α on the parallelity of the edges. For example, if $\alpha = 0.1°$, and edges of A and $-B$ are 45° and 45.05°, then these edges are considered to be parallel.

4.7 Computational Results

In previously published papers on NFP calculation, the practice has been to test NFP algorithms on a common set of data. The data sets which have been used tend to contain simple polygons, which contain very few edges. However, in industries such as garment

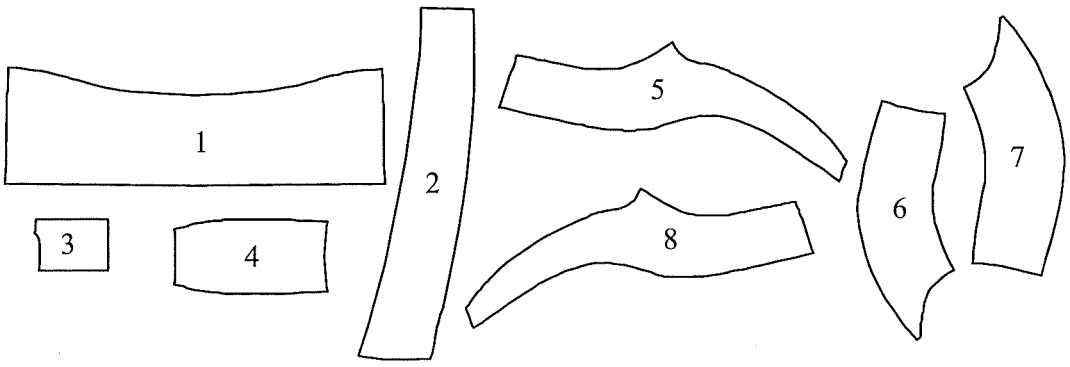

Figure 4.6.1: Data set for NFP calculations (due to the size of the polygon drawings, some small cavities may appear invisible)

manufacturing, individual polygons can have over 100 edges. The algorithms in this thesis were designed to improve the robustness and efficiency of calculating NFPs for polygons which have a large number of edges and many cavities. Therefore, instead of testing on the previously used data sets, we will set a performance benchmark on a new, more complex set of data (see Figure 4.6.1). Full details of this data set are available in Appendix A.

The properties of each polygon are given in Table 4.6.1. The calculation results of the NFPs for each polygon pair are given in Table 4.6.2.

Polygon	Edges	Cavities
1	65	9
2	43	2
3	21	3
4	58	17
5	128	9
6	64	3
7	64	3
8	141	11

Table 4.6.1: Polygon properties

Removing negative edges reduced the number of candidate edges for the outer envelope by over 34% on average.

The divide and conquer strategy for calculating the outer envelope reduced the total calculation time by 64.61%. Columns "Time 1" and "Time 2" of Table 4.6.2 show the calculation times for the standard outer envelope calculation, and the divide and conquer strategy respectively. This improvement increased as the number of candidate edges for each outer envelope calculation increased.

4.8 Extensions to the Algorithm

A polygon P is "star shaped" if there exists a point k in the interior of P that can "see" the entire polygon, and for any other point u in P, the entire segment ku lies inside P. Li et al [44] show that if two polygons are star shaped, then their NFP is also star shaped, that is, will not contain any internal holes.

The NFP algorithm described in this paper works for non-convex polygons, both star shaped, and non-star shaped. However, if one or more of the input polygons is not star shaped, then the NFP may contain internal holes. In this situation calculating the outer

envelope of NFPList becomes more difficult. Ramkumar [53] gives an algorithm to accomplish this.

4.9 Conclusion

In this chapter we have modified Ghosh's algorithm to calculate the NFP of a convex and a non-convex polygon so that the NFP of two non-convex polygons can easily and efficiently be calculated.

The time consuming process of finding the outer envelope from the list of candidate edges, NFPList, has been improved by reducing the size of NFPList, and dividing the problem into smaller sub-problems based on the convexity of the original two polygons.

Solutions to industrial 2-dimensional packing problems usually have time constraints in which they must adhere to. Most of these solutions involve the computationally expensive calculation of many NFPs. By increasing the speed in which these NFPs can be calculated, we open the possibility of more effective solution procedures to industrial packing problems.

A	B	Edges	Non-Negative Edges	Time 1 (s)	Time 2 (s)	% Impr
1	1	138	134	0.016	0.007	57.5%
1	2	132	120	0.017	0.007	60.5%
1	3	162	124	0.022	0.010	55.6%
1	4	265	189	0.032	0.012	61.3%
1	5	525	359	0.068	0.033	51.2%
1	6	421	275	0.036	0.018	49.5%
1	7	201	165	0.025	0.014	41.9%
1	8	506	356	0.070	0.032	53.7%
2	2	260	173	0.020	0.008	61.2%
2	3	104	84	0.006	0.002	57.1%
2	4	143	122	0.014	0.005	66.7%
2	5	249	210	0.037	0.014	61.3%
2	6	253	180	0.020	0.010	50.0%
2	7	319	213	0.024	0.012	51.7%
2	8	212	198	0.043	0.014	66.7%
3	3	54	48	0.002	0.001	40.0%
3	4	159	119	0.012	0.006	53.3%
3	5	277	213	0.037	0.018	52.2%
3	6	373	229	0.024	0.014	42.6%
3	7	125	105	0.010	0.005	45.8%
3	8	206	184	0.035	0.014	61.4%
4	4	254	185	0.024	0.008	66.1%
4	5	1080	633	0.125	0.055	55.8%
4	6	304	213	0.029	0.013	54.2%
4	7	218	170	0.024	0.010	55.9%
4	8	1157	678	0.144	0.050	65.6%
5	5	1532	894	0.244	0.067	72.5%
5	6	508	350	0.071	0.037	47.8%
5	7	340	266	0.054	0.033	38.1%
5	8	1609	939	0.291	0.081	72.3%
6	6	210	169	0.024	0.007	71.7%
6	7	500	314	0.049	0.027	45.1%
6	8	473	339	0.075	0.024	68.4%
7	7	216	172	0.021	0.008	64.2%
7	8	459	332	0.078	0.024	69.6%
8	8	2680	1481	0.444	0.102	77.0%
SUM		16624	10935	2.265	0.802	64.6%
AVERAGE						57.4%

Table 4.6.2: NFP calculation times

Chapter 5: Global Optimisation Model

5.1 Introduction

In this chapter, a mathematical model for global optimisation of the 2-dimensional cutting stock problem is formulated for the case when the polygons are in a fixed orientation. The model is a mixed-integer formulation that uses the no-fit polygon (see Chapter 4) to establish regions where one polygon can be placed without overlapping another polygon. These regions are expressed as constraints, and the objective function is to minimise the overall length of the marker.

The model is similar to the 2-dimensional non-convex model of Scheithauer et al [54] and the 3-dimensional model of Chen et al [15]. The main difference from Scheithauer et al and Chen et al is in the constraints which stop polygons from overlapping. These models use constraints of the form $y = mx + c$ to prevent polygon overlap. The equivalent constraints in this paper's model are constructed using cross products. The advantage of the cross product form is that it allows polygons to be separated by a distance G. G differs from marker to marker, and depends on factors such as cloth type, and cutting machine precision. Although not provided for in this model, G can be of differing size depending on how polygons contact one another, i.e. vertex-vertex contact, vertex-edge contact, or edge-edge contact.

In addition, the Scheithauer et al model has never been solved. The model given in this chapter is solved, and results are given.

Before presenting the model, the idea of the *convex feasible sub-region* (CFSR) must be discussed. CFSRs form a central part of the algorithms and models described in this thesis.

5.2 Convex Feasible Sub-Regions (CFSRs)

The area outside NFP[A, B] represents the region which reference point r_B can feasibly occupy, such that polygons A and B do not overlap. This area cannot be expressed by a set of linear constraints. It can, however, be decomposed into a number of convex feasible sub-regions (CFSRs), each of which can be expressed by a set of linear constraints (see Figure 5.1).

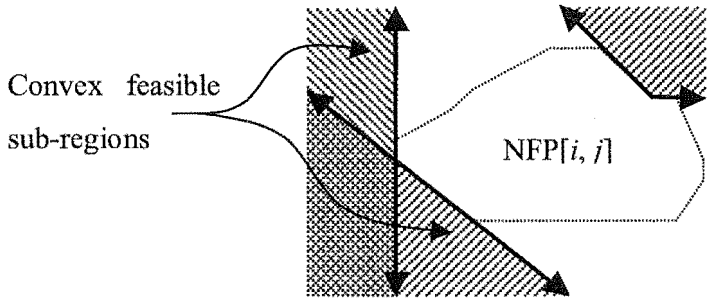

Figure 5.1: Example convex feasible sub-regions of NFP[i, j]

For each edge in NFP[A, B] that also appears in the convex hull of NFP[A, B], the corresponding CFSR is expressed by a single half-plane with its face collinear to that edge. The remaining edges of NFP[A, B] will form either convex or non-convex regions. If a region is convex, then it is a CFSR expressed by a polyhedron whose faces are collinear with the edges of that region. If a region is non-convex, then multiple CFSRs need to be formed from its edges. An algorithm for breaking a non-convex region into convex regions is given in Fernandez et al [24].

The polyhedrons representing the CFSRs are not always bounded. However, unlimited movement by a reference point in an unbounded polyhedron is avoided because all CFSRs are a subset of a polytope representing the edges of the resource cloth.

5.3 The Model

The following global optimisation model leads to an optimal solution of the translational 2-dimensional cutting stock problem. The model assumes that the width, W, of the marker is known; and the relevant NFPs for each pair of polygons (i, j) are also known.

5.3.1 Parameters

$|AB_{ijkt}|$ = The length of edge t in CFSR k of NFP[i, j].

Ax_{ijkt} = The x-coordinate of the start point of edge t in CFSR k of NFP[i, j].

Ay_{ijkt} = The y-coordinate of the start point of edge t in CFSR k of NFP[i, j].

Bx_{ijkt} = The x-coordinate of the end point of edge t in CFSR k of NFP[i, j].

By_{ijkt} = The y-coordinate of the end point of edge t in CFSR k of NFP[i, j].

G = The distance required between each polygon.

H_i = The height of polygon i.

L_i = The length of polygon i.

LB = The lower bound on the marker length.

UB = The upper bound on the marker length.

W = The width of the marker.

M_{ijkt} = A large number.

If $(Bx_{ijkt} - Ax_{ijkt}) \geq 0$ And $(Ay_{ijkt} - By_{ijkt}) \geq 0$ Then

$$M_{ijkt} = |AB_{ijkt}|\, G + (Bx_{ijkt} - Ax_{ijkt})\,(W - H_j) + (Ay_{ijkt} - By_{ijkt})\,(UB - L_j) - Ay_{ijkt}\, Bx_{ijkt} + Ax_{ijkt}\, By_{ijkt}.$$

If $(Bx_{ijkt} - Ax_{ijkt}) \geq 0$ And $(Ay_{ijkt} - By_{ijkt}) \leq 0$ Then

$$M_{ijkt} = |AB_{ijkt}|\, G + (Bx_{ijkt} - Ax_{ijkt})\,(W - H_j) + (Ay_{ijkt} - By_{ijkt})\,(L_i - UB) - Ay_{ijkt}\, Bx_{ijkt} + Ax_{ijkt}\, By_{ijkt}.$$

If $(Bx_{ijkt} - Ax_{ijkt}) \leq 0$ And $(Ay_{ijkt} - By_{ijkt}) \geq 0$ Then

$$M_{ijkt} = |AB_{ijkt}|\, G + (Bx_{ijkt} - Ax_{ijkt})\,(H_i - W) + (Ay_{ijkt} - By_{ijkt})\,(UB - L_j) - Ay_{ijkt}\, Bx_{ijkt} + Ax_{ijkt}\, By_{ijkt}.$$

If $(Bx_{ijkt} - Ax_{ijkt}) \leq 0$ And $(Ay_{ijkt} - By_{ijkt}) \leq 0$ Then

$$M_{ijkt} = |AB_{ijkt}|\, G + (Bx_{ijkt} - Ax_{ijkt})\,(H_i - W) + (Ay_{ijkt} - By_{ijkt})\,(L_i - UB) - Ay_{ijkt}\, Bx_{ijkt} + Ax_{ijkt}\, By_{ijkt}.$$

5.3.2 Variables

ml = The length of the marker.

rx_i = The x-coordinate of the reference point of polygon i.

ry_i = The y-coordinate of the reference point of polygon i.

z_{ijk} = 1 if the reference point of polygon i exists inside CFSR k of NFP[i, j]; otherwise 0.

5.3.3 Global ml

The problem is formulated as the following linear mixed integer programming model, Global ml:

Minimise ml (5.1)

subject to

$ry_i + H_i \leq W$, for all i (5.2)

$rx_i + L_i \leq ml$, for all i (5.3)

$\sum_k z_{ijk} = 1$, for all $i, j, j > i$ (5.4)

$$(Ax_{ijkt} - Bx_{ijkt})\,(ry_j - ry_i) + (By_{ijkt} - Ay_{ijkt})\,(rx_j - rx_i) + Ay_{ijkt}\,Bx_{ijkt} - Ax_{ijkt}\,By_{ijkt} + M_{ijkt}(1 - z_{ijk}) \geq |AB_{ijkt}|\,G, \text{ for all } i,j,k,t,j > i \quad (5.5)$$

$ml \geq LB$ (5.6)

$ml \leq UB$ (5.7)

$rx_i \geq 0,\ ry_i \geq 0$, for all i (5.8)

$z_{ijk} \in \{0, 1\}$, for all i, j, k (5.9)

5.3.4 Explanation of Constraints

(5.1) The objective function is to minimise the length of the marker in the x-direction.

(5.2) The y-coordinate of the reference point r_i of polygon i and the height of polygon i must lie within the marker width W.

(5.3) The x-coordinate of the reference point r_i of polygon i and the length of polygon i must lie within the marker length ml.

(5.4) For each pair of polygons (i, j), the reference point of polygon i must occupy exactly one CFSR k. If two or more CFSRs overlap and the reference point occupies the union of these regions, then the point may occupy an arbitrary CFSR.

(5.5) If $z_{ijk} = 1$, then the reference point of polygon i must lie to the right of each edge t in CFSR k (assuming NFP$[i,j]$ is directed counter-clockwise). This is shown diagrammatically in Figure 5.2. If the segments AB and Ar are treated as vectors then we can calculate the cross product $AB \times Ar$. Geometrically, if the cross product of the two vectors is positive, then its value is twice the area of triangle ABr and r is to the right of AB. $|AB|\,G$ is equal to twice the area of ABr. Equation (5.5) is the cross product translated by rx_i and ry_i.

M_{ijkt} is of minimum size such that if $z_{ijk} = 0$, (i.e. the reference point of polygon i does not occupy CFSR k of NFP[i, j]) then this constraint is loose regardless of the values of the remaining variables and parameters.

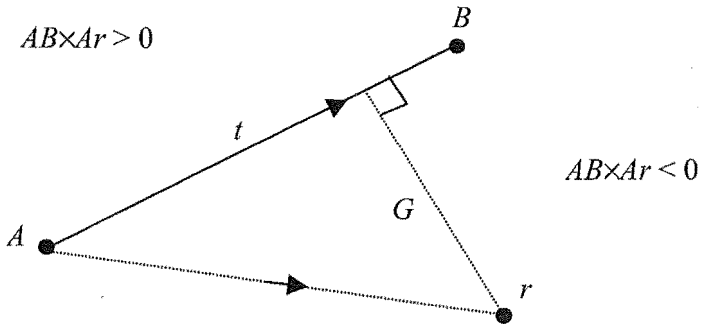

Figure 5.2: Geometry of equation (5.5)

(5.6) The lower bound on the variable *ml*.

(5.7) The upper bound on the variable *ml*.

(5.8) Non-negativity constraints for the location variables, rx_i and ry_i.

(5.9) Binary constraints for the variable z_{ijk}.

5.4 Numerical Results

The model presented in this chapter has been tested on a variety of problems. Results are given for problems[2] containing 3, 4, and 5 identical polygons. A representative polygon is shown in Figure 5.3.

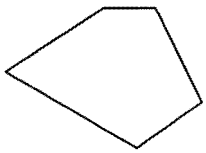

Figure 5.3: Polygon used in test cases for Global ml

The results in Table 5.1 were obtained solving models with Cplex 7.1 on a Pentium II-366 computer.

[2] Problem data can be found in Appendix B

Pieces	Lower Bound (LB)	Upper Bound (UB)	z	Time (sec)
3	50	150	96.4527	<1
3	60	140	96.4527	<1
3	70	120	96.4527	<1
3	80	120	96.4527	<1
3	90	110	96.4527	<1
3	95	100	96.4527	<1
4	120	200	122.3224	60
4	120	130	122.3224	52
4	100	150	122.3224	50
4	50	150	122.3224	42
4	50	200	122.3224	38
4	50	125	122.3224	36
5	110	180	142.9689	47748
5	130	180	142.9689	28055
5	140	160	142.9689	19126

Table 5.1: Global ml results

The optimal solution to the 4 polygon problem is shown in Figure 5.4.

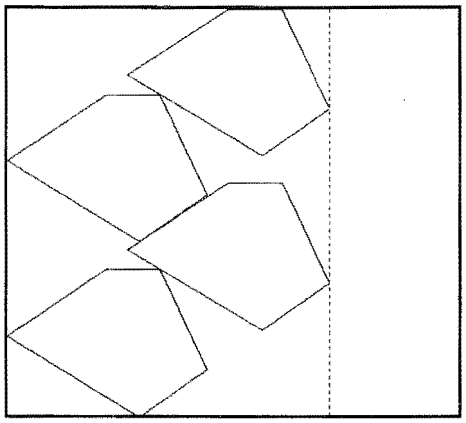

Figure 5.4: Optimal solution to the 4 polygon problem

5.4.1 Bounding

The results for the 5 polygon problem (see Table 5.1) show that, as the problem size increases, establishing good upper and lower bounds becomes increasingly important to solution times.

The reliance of good bounds for Global ml is a concern, as bounding is a difficult problem. Heckmann et al [34] gives a method for establishing lower bounds which are close to the optimal solution for certain problem types. However their technique is not robust enough for it to be applicable to generic markers.

5.5 Conclusion

This chapter has presented a model for the global optimisation of the translational 2-dimensional cutting stock problem. However, Global ml can be solved optimally in a practical amount of time only for small models. In practice, problems can contain over 200 polygons, and each polygon can contain over 100 edges.

The impracticality of the Global ml model has led to the development of an algorithm to compact an existing marker by fixing the z_{ijk} variables and therefore converting the model to a linear programming problem. This "compaction" model is given in chapter 6.

Chapter 6: Compaction Model

6.1 Introduction

Chapter 5 gave a global optimisation model for the translational 2-dimensional cutting stock problem. However, this approach is currently impractical for industrial sized problems.

The problems with the Global ml model led to the development of an algorithm to compact an existing marker by fixing the z_{ijk} variables and therefore converting the model to a linear programming problem.

Very few researchers have adopted approaches similar to this. Stoyan et al [55] present a method for compacting a pre-generated marker when the marker has defects, such as in the leather industry.

Heckmann et al [33] present a simulated annealing algorithm for compacting a marker. The algorithm makes slight placement modifications, however, it does not simultaneously modify all polygons at once.

Li et al [44] present an algorithm for compacting a pre-generated marker using a linear programming approach that utilises the concept of the no-fit polygon. Their algorithm attempts to maximise the total motion of all polygons in a desired direction. The compaction algorithm presented in this chapter is an improvement upon the method proposed by Li et al because, among other things, it allows polygons to simultaneously move in any direction, with the objective of minimising the length of the marker.

6.2 The Compaction Algorithm

The compaction algorithm takes a non-overlapping marker and attempts to improve the marker's efficiency by reducing its length via a series of simultaneous translations to each polygon. Each iteration of the algorithm involves solving a restricted continuous version of the Global ml model. This allows the solution to move to a local optima while remaining feasible.

By fixing the z_{ijk} variables in the Global ml model to either 1 or 0, the model becomes continuous. This could be done simply by fixing rx_i and ry_i for each polygon i, solving Global ml, and resolving with the z_{ijk} variables fixed and the (rx_i, ry_i) variables continuous. This method has at least two major drawbacks:

Firstly, we want to choose a CFSR for a particular reference point r that gives r the greatest freedom of movement, and this may not occur if the CFSR is chosen by a MIP solver. The *Find CFSR(k) Algorithm* (section 6.2.1) is used to find CFSR k from NFP[i, j] where z_{ijk} can equal 1.

Secondly, it involves constructing model Global ml, whereby NFPs and CFSRs for every polygon pair (i, j) must be calculated. Because r_i for each polygon i is restricted to a set of CFSRs, each polygon will come into contact with only a limited number of polygons. This issue is addressed in the section 6.2.2, *Limiting Polygon Movement.*

6.2.1 Find CFSR(*k*) Algorithm

For any given polygon pair (i, j), NFP[i, j] is located at (rx_i, ry_i), and, because the marker is non-overlapping, r_j occupies the area outside this NFP. The CFSR is found by starting with the edge of NFP[i, j] which has the smallest distance to r_j. NFP[i, j] is traversed by starting at this edge and moving along the perimeter until a vertex on the convex hull of NFP[i, j] is found in both the clockwise and counter-clockwise directions. The edges between these two vertices construct the CFSR. If the edge already lies on the convex hull of NFP[A, B] then the two vertices found will be the start and end points of this edge. This algorithm is shown in Pseudo-code 6.1:

```
Find CFSR(k) Algorithm{
        e = Edge with smallest distance to r_j
        f = Edge with smallest distance to r_j + 1
        Do Until (IsOnConvexHull(Vertex[e])
                e = e - 1
        Loop
        Do Until (IsOnConvexHull(Vertex[f])
                f = f + 1
        Loop
        For g = e To f - 1
                AddToCFSR(Edge[g])
        Next g
}
```

Pseudo-code 6.1: Find CFSR(k) Algorithm

If the marker is overlapping, then for at least one polygon pair (i, j), r_j will be inside NFP[i, j]. The above algorithm can still be used and the "closest" CFSR to r_j will be found. For many cases (when global overlap is small), the CFSR finding algorithm can

find a set of CFSRs which will allow the compaction algorithm to find a feasible non-overlapping layout.

6.2.2 Limiting Polygon Movement

Because human generated markers are tightly packed, and the compaction algorithm requires that the solution remain feasible, a specific polygon will contact only a local neighbourhood of polygons. NFPs and the corresponding CFSRs of a polygon need to be constructed only for the polygons in this local neighbourhood. The size of the neighbourhood is determined by a distance, D, that each polygon can move. The neighbourhood for polygon i is comprised of the remaining polygons that can possibly overlap polygon i when polygon i is free to move in a $2D \times 2D$ square centred at r_i and each remaining polygon j is free to move in a $2D \times 2D$ square centred at r_j. For each polygon i, the set of neighbouring polygons is denoted as N_i. Limiting the polygon movement is achieved by the following constraints:

$$rx_i = rx_i^* + dx_i, \text{ for all } i \tag{6.1}$$

$$ry_i = ry_i^* + dy_i, \text{ for all } i \tag{6.2}$$

$$-D \leq dy_i \leq D, \text{ for all } i \tag{6.3}$$

$$-D \leq dy_i \leq D, \text{ for all } i \tag{6.4}$$

where rx_i^* and ry_i^* are the starting coordinates of r_i, and dx_i and dy_i are the deviations in the x and y directions from this coordinate.

Determining the size of D is an interesting problem. Increasing D increases the amount a polygon can move. However this extra movement means potentially more polygons will come into contact. This results in a larger number of NFPs and CFSRs to be calculated. Apart from increasing the processing time, an increase in CFSRs may actually reduce the amount a polygon can move because r is more constrained. An example of this is shown in Figures 6.1 – 6.3. Figure 6.1 shows polygon P surrounded by polygons A, B, and C. Suppose moving P and B to the left would decrease the objective function. Suppose D is chosen such that only polygons A and B are in the local neighbourhood N_P, Figure 6.2 then shows the intersection of CFSRs that r_P can move in. However, if D is increased to a size where C is also in N_P, then this intersection of CFSRs decreases in size, because the CFSR of C "cuts into" the intersection of the CFSRs from B and C. This is shown in Figure 6.3.

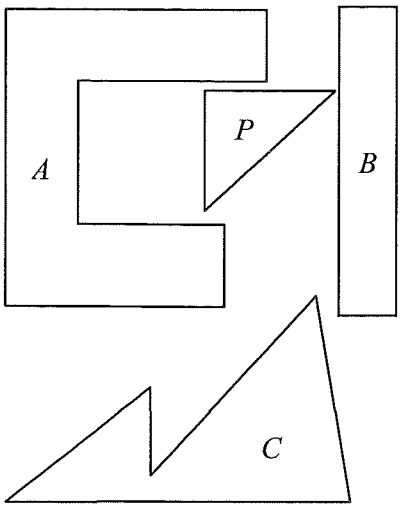

Figure 6.1

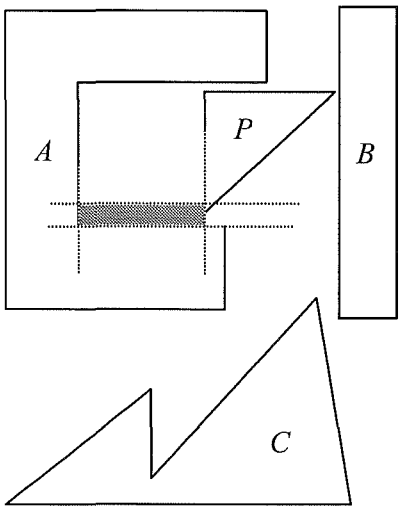

Figure 6.2

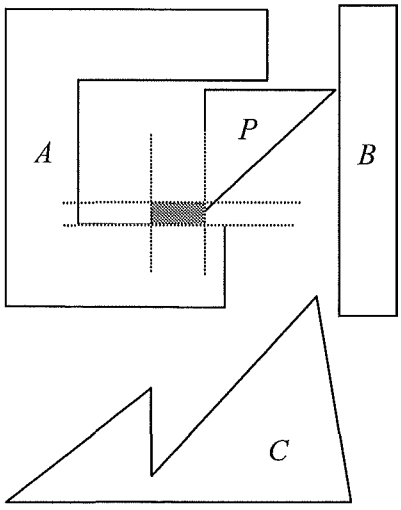

Figure 6.3

As yet, an effective way of determining a suitable D has not yet been found. The current method is to guess the maximum distance that a polygon will move, and set D equal to that.

The number of NFPs that the Global ml model requires is of order $O(n^2)$. The compaction algorithm needs NFPs only for neighbouring polygons, so the number of NFPs required may be of order $O(n)$, depending on the choice of D. As NFP calculations use the majority of time in the compaction process, this is a vast improvement.

Once the neighbourhoods of polygons have been established, and the relevant CFSRs found, equation (5.5) can be made continuous:

$$(Ax_{ijt} - Bx_{ijt})\,(ry_j - ry_i) + (By_{ijt} - Ay_{ijt})\,(rx_j - rx_i) + Ay_{ijt}\,Bx_{ijt} - Ax_{ijt}\,By_{ijt} \geq |AB_{ijt}|\,G, \text{ for all } i,j,t, j > i, j \in N_i \qquad (6.5)$$

6.2.3 Compact ml

The compaction model, *Compact ml*, is comprised of equations (5.1 – 5.3) and (5.8) from Global ml, and equations (6.1 – 6.5).

Minimise ml (5.1)

subject to

$ry_i + H_i \leq W$, for all i (5.2)

$rx_i + L_i \leq ml$, for all i (5.3)

$rx_i = rx_i^* + dx_i$, for all i (6.1)

$ry_i = ry_i^* + dy_i$, for all i (6.2)

$-D \leq dy_i \leq D$, for all i (6.3)

$-D \leq dy_i \leq D$, for all i (6.4)

$(Ax_{ijt} - Bx_{ijt})\,(ry_j - ry_i) + (By_{ijt} - Ay_{ijt})\,(rx_j - rx_i) + Ay_{ijt}\,Bx_{ijt} - Ax_{ijt}\,By_{ijt} \geq |AB_{ijt}|\,G$, for all $i,j,t, j > i, j \in N_i$ (6.5)

$rx_i \geq 0$, $ry_i \geq 0$, for all i (5.8)

6.2.4 Iterations of the Algorithm

Solving Compact ml compacts a non-overlapping marker by simultaneously translating each polygon i by some combination of dx_i and dy_i. However, the objective function may be improved if we re-run algorithm Find CFSR(k), and solve model Compact ml again. To see why, consider the example shown in Figure 6.4.

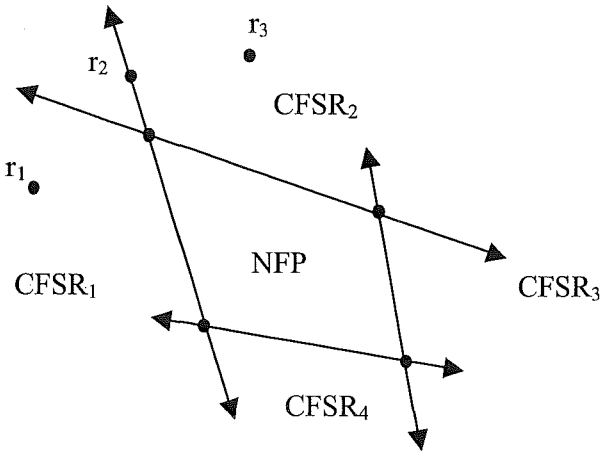

Figure 6.4: r moving from one CFSR to another

Figure 6.4 shows an NFP that has been decomposed into four CFSRs. Suppose r was originally located at r_1, and it was constrained by the CFSR algorithm to occupy $CFSR_1$. Suppose moving to r_3 would result in the lowest objective function value. After an iteration of model Compact ml, r can move only to r_2 due to the constraints of $CFSR_1$. However, moving to r_3 would result in the lowest objective function value. After recalculating the CFSRs based on the new polygon locations, it may be ambiguous whether r should be in $CFSR_1$ or $CFSR_2$ because r lies the same distance away from both corresponding edges on the NFP.

This ambiguity is resolved by considering the dual prices on each CFSR constraint. If a CFSR constraint has a non-zero dual-price, then the objective function will be lowered if r penetrates this constraint. This constraint has a corresponding edge, E, from the relevant NFP. If r is on E, then we do not need to change the CFSR that r occupies. However, if r does not lie on E, then a new CFSR k is found by the Find CFSR(k) algorithm starting with the NFP edge which is neighbouring E. Every polygon pair has their CFSRs updated via this method. Constraints (6.1) and (6.2) are updated with the new polygon locations and the model is resolved. The algorithm iterates over the above

process until all the constraints of type (6.3), (6.4) or (6.5) have dual prices of zero. When this stage is reached, the marker has reached a local optimum. From a local optimum, the efficiency cannot be improved by translation alone, unless the polygons are able to move by more than the distance D.

6.3 Numerical Results

The algorithms in this thesis have been tested on data from a small Christchurch manufacturing company. The company employs two full time workers to create markers and the results are measured by the efficiency percentage increase.

Table 6.1 gives the results for 50 markers, whose size range from 6 polygons, to 138 polygons. The average percentage improvement is 0.894%. However, because each marker has a different size, the actual percentage of fabric saved was 1.12%.

The compaction algorithm runs in a practical amount of time: the longest trial took 139 seconds for the 138 polygon marker.

Section 6.2.4 explains how the algorithm updates CFSRs from one iteration to another using dual prices on equations 6.5. The results for this algorithm are given in the "Dual Priced Iterations" section of Table 6.1. If the algorithm updated the CFSRs using the Find CFSR(k) Algorithm (section 6.2.1), then the results for this method are given in the "Standard Iterations" section of Table 6.1. The "% Diff" column, gives the percentage improvement between these two methods. The results have shown the dual price based iterations to be clearly superior.

Figures 6.5a and 6.5b show a marker (VAJ AB C=0220 1) before and after the compaction algorithm. The data for this marker can be found in Appendix C.

6.4 Conclusion

This chapter has given a model, Compact ml, which is a restricted, linear version of Global ml (see Chapter 5). Compact ml applies a series of simultaneous translations to each polygon in a pre-generated marker, in an attempt to decrease the length of the marker. The compaction algorithm applies iterations of Compact ml, such that when the algorithm is finished, the marker cannot be improved by translation alone, unless the polygons are able to move by more than the distance D.

The compaction algorithm increased marker efficiency by an average of 0.894% on 50 industrial markers.

		Dual Priced Iterations			Standard Iterations			
	Pieces	% Imp	Time (s)	Iter.	% Imp	Time (s)	Iter.	% Diff
AMP2 A=0220 BC=0440 131 1	52	1.199	12.152	3	1.199	19.498	7	0.000
AMP2 A=0220 BC=0440 131 2	24	4.818	11.569	13	2.511	11.430	6	2.307
ASX2 S3 ABCDEF=3 * 1	21	0.243	24.949	6	0.243	22.164	5	0.000
ASX2 S3 ABCDEF=3 * 2	69	0.977	12.713	6	0.785	9.992	4	0.192
ASX2 S3 ABCDEF=3 * 3	57	1.102	8.553	3	0.985	5.954	2	0.117
ASX2 S3 ABCDEF=3 * 4	24	0.879	4.597	8	0.686	4.197	5	0.193
ATV2 ABD=6 C=3 *	48	1.412	9.751	4	1.021	9.336	2	0.392
CLA2 LG A=6 *	24	1.284	3.792	6	0.732	2.653	6	0.552
CYC2 ABCDE=10 * 1	60	0.596	35.453	4	0.490	34.740	2	0.106
CYC2 ABCDE=10 * 2	80	0.496	22.324	9	0.477	19.470	7	0.019
DYN02 S2 ABC=20 *	100	0.599	57.793	8	0.599	50.067	4	0.000
EXP2 STD ABCDE=6 1	84	0.448	46.594	5	0.435	42.271	5	0.013
EXP2 STD ABCDE=6 2	42	0.696	21.106	5	0.629	18.035	4	0.067
EXP2 STD ABCDE=6 3	60	0.742	13.386	4	0.742	10.181	4	0.000
FAN2 ABCDE=6 183 *	36	0.243	26.143	2	0.243	26.591	3	0.000
FAN2 ABCDE=6 183 * 2	66	0.576	37.379	16	0.407	19.710	5	0.169
FAN2 ABCDE=6 183 * 3	21	0.000	1.430	1	0.000	1.131	1	0.000
FAN2 ABCDE=6 183 * 4	12	0.091	2.535	2	0.091	2.753	2	0.000
FAN2 ABCDE=6 183 * 5	24	0.975	18.552	3	0.949	19.233	3	0.025
FAN2 ABCDE=6 183 * 6	24	0.000	0.724	1	0.000	0.748	1	0.000
GAU2 ABC=053	64	0.317	29.388	12	0.186	21.452	5	0.132
GEC2 S1 AB=10 C=5 D=8 183 1	120	0.911	136.044	14	0.739	120.762	8	0.172
GEC2 S1 AB=10 C=5 D=8 183 2	45	0.241	15.761	7	0.233	16.975	4	0.008
GEN2 S3 ABD=6 C=3 1	138	0.763	139.419	13	0.257	86.385	2	0.506
GEN2 S3 ABD=6 C=3 2	48	0.205	8.335	7	0.174	9.435	4	0.031
HDP2 ABD=6 C=3 * 1	48	3.001	13.088	7	2.921	13.571	6	0.080
HDP2 ABD=6 C=3 * 2	12	0.295	4.284	3	0.295	5.390	3	0.000
HDP2 ABD=6 C=3 * 3	24	0.807	14.953	5	0.771	13.471	5	0.035
KOA2 STD ABEF=6 CD=3 * 1	60	0.679	27.134	5	0.439	24.499	2	0.240
KOA2 STD ABEF=6 CD=3 * 2	33	1.089	9.023	6	1.030	7.245	2	0.059
KOA2 STD ABEF=6 CD=3 * 3	72	0.471	100.147	7	0.461	91.745	5	0.010
KOA2 STD ABEF=6 CD=3 * 4	24	0.000	2.623	2	0.000	2.500	1	0.000
MCT2 ABD=111 1	12	0.479	3.186	3	0.416	0.859	2	0.063
MCT2 ABD=111 2	42	1.053	12.823	5	0.908	12.868	2	0.145
STL2 AB=1 CD=2 E=6 * 1	17	0.318	3.057	4	0.309	3.549	3	0.009
STL2 AB=1 CD=2 E=6 * 2	8	0.083	1.015	2	0.083	1.174	2	0.000
STR2 ABC=1221 1	18	2.599	10.480	7	2.562	8.154	4	0.038
STR2 ABC=1221 2	6	0.933	3.146	3	0.933	3.403	3	0.000
STR2 ABC=1221 3	48	2.254	29.030	9	2.254	34.435	7	0.000
TUA2 S1 AB=6 183 1	42	0.533	26.573	6	0.527	21.826	3	0.005
TUA2 S1 AB=6 183 2	48	0.465	26.786	5	0.465	28.258	7	0.000
VAJ ABC=2220 1	24	1.923	8.724	15	1.743	7.821	7	0.180
VAJ ABC=2220 2	12	0.825	3.919	9	0.823	3.937	4	0.002
ZEN2 ABCDE=122 1	35	0.556	11.885	8	0.340	11.895	4	0.216
ZEN2 ABCDE=122 2	35	1.268	16.343	5	1.268	15.983	5	0.000
ZEN2 ABCDE=122 3	45	2.269	12.144	12	1.682	10.911	6	0.587
ZEN2 ABCDE=122 4	40	0.395	7.454	4	0.165	5.090	2	0.230
ZEN2 ABCDE=122 5	10	0.003	9.951	2	0.003	7.163	2	0.000
ZOD2 ABCD=1221 1	24	1.704	1.350	6	1.655	2.190	5	0.049
Average	42.490	0.894	21.624	6.163	0.752	18.839	3.939	0.142

Table 6.1: Compaction results

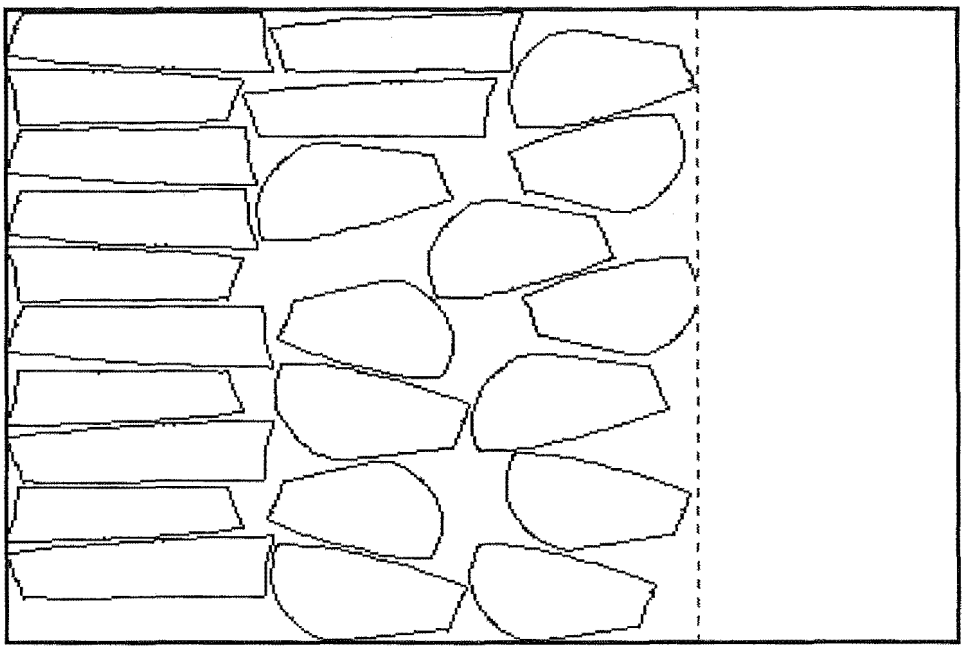

Figure 6.5a: Human generated marker

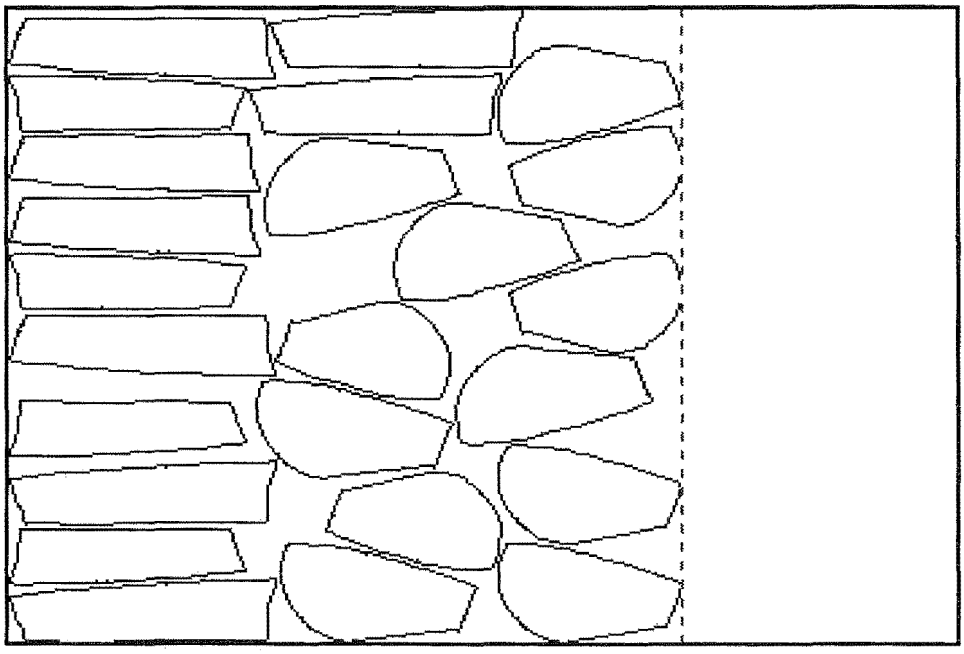

Figure 6.5b: After compaction algorithm

Chapter 7: Rotational Compaction

7.1 Introduction

From a local optimum (global optimum of Compact ml), the efficiency cannot be improved by translation alone, unless the polygons are moved by more than the distance *D*. One way of improving the efficiency further is by rotating polygons.

Figure 7.1 shows a marker which has been compacted via Compact ml. This marker cannot be improved by polygons translations. If the polygon with the arrow on it is allowed to change orientation (see Figure 7.2), then the marker efficiency can be improved by 0.3%.

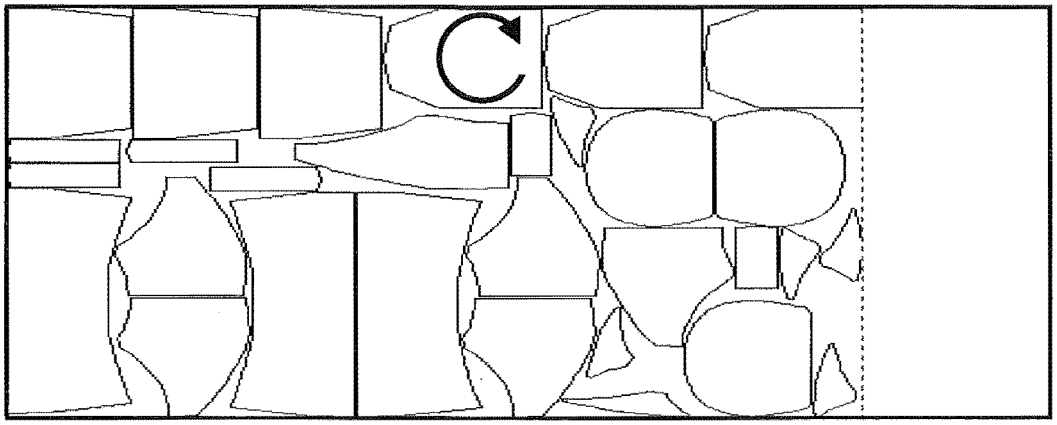

Figure 7.1: Before rotation

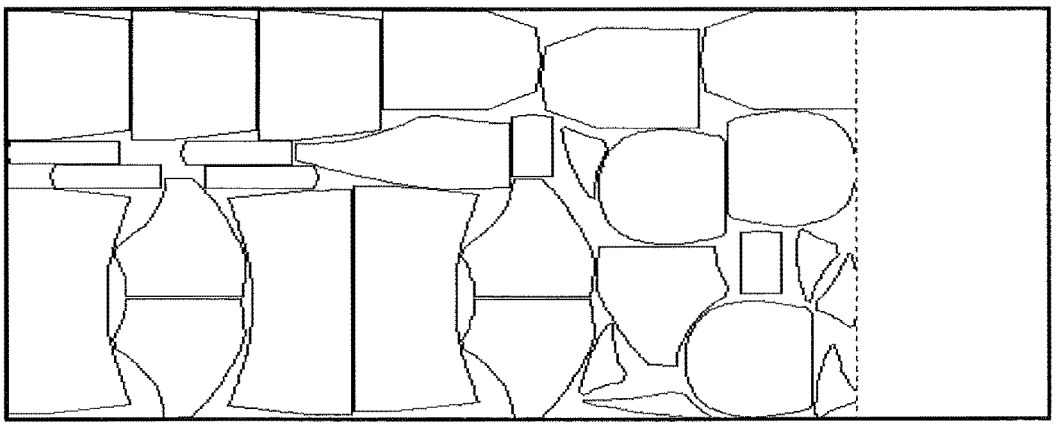

Figure 7.2: After rotation

This chapter extends the model of the previous chapter to allow polygons to change orientation. No references have been found on compacting a marker by rotating polygons.

7.2 Rotational Compaction Model

Each polygon in a marker has its own set of possible rotations. The possible rotations are 0°, 90°, 180°, 270°. A polygon may also be flipped in the x or y plane. This gives up to eight different orientations for any given polygon. The compaction model is relaxed to let some polygons change orientation. This chapter presents a model, *Rotate ml*, which extends model Compact ml to allow rotations. Variables and parameters are the same as in chapters 5 and 6 unless stated.

7.2.1 New Parameters

Ax_{irjst} = The x-coordinate of the start point of edge t in fixed CFSR k of NFP[i, j] when polygon i is in orientation r and polygon j is in orientation s.

Ay_{irjst} = The y-coordinate of the start point of edge t in fixed CFSR k of NFP[i, j] when polygon i is in orientation r and polygon j is in orientation s.

Bx_{irjst} = The x-coordinate of the end point of edge t in fixed CFSR k of NFP[i, j] when polygon i is in orientation r and polygon j is in orientation s.

By_{irjst} = The y-coordinate of the end point of edge t in fixed CFSR k of NFP[i, j] when polygon i is in orientation r and polygon j is in orientation s.

H_{ir} = The height of polygon i in orientation r.

L_{ir} = The length of polygon i in orientation r.

$M1_{ir} = \text{Min}(ry_i^* + D, W - H_{ir}) + H_{ir} - W$

$M2_{ir} = \text{Min}(rx_i^* + D, ml^* - L_{ir}) + L_{ir} - ml^*$, where ml^* is the starting length of the marker.

$M3_{irjst}$ = A large number.

If $(Bx_{irjst} - Ax_{irjst}) \geq 0$ And $(Ay_{irjst} - By_{irjst}) \geq 0$ Then

$$M3_{irjst} = |AB_{irjst}|G + (Bx_{irjst} - Ax_{irjst})\,(\mathrm{Min}(ry_j^* + D, W - H_{jr}) - \mathrm{Max}(ry_i^* - D, 0)$$
$$+ (Ay_{ijkt} - By_{ijkt})\,(\mathrm{Min}(rx_j^* + D, ml^* - L_{jr}) - \mathrm{Max}(rx_i^* - D, 0)$$
$$- Ay_{irjst}\,Bx_{irjst} + Ax_{irjst}\,By_{irjst}.$$

If $(Bx_{irjst} - Ax_{irjst}) \geq 0$ And $(Ay_{irjst} - By_{irjst}) \leq 0$ Then

$$M3_{irjst} = |AB_{irjst}|G + (Bx_{irjst} - Ax_{irjst})\,(\mathrm{Min}(ry_j^* + D, W - H_{jr}) - \mathrm{Max}(ry_i^* - D, 0)$$
$$+ (Ay_{ijkt} - By_{ijkt})\,(\mathrm{Min}(rx_i^* + D, ml^* - L_{ir}) - \mathrm{Max}(rx_j^* - D, 0)$$
$$- Ay_{irjst}\,Bx_{irjst} + Ax_{irjst}\,By_{irjst}.$$

If $(Bx_{irjst} - Ax_{irjst}) \leq 0$ And $(Ay_{irjst} - By_{irjst}) \geq 0$ Then

$$M3_{irjst} = |AB_{irjst}|G + (Bx_{irjst} - Ax_{irjst})\,(\mathrm{Min}(ry_i^* + D, W - H_{ir}) - \mathrm{Max}(ry_j^* - D, 0)$$
$$+ (Ay_{ijkt} - By_{ijkt})\,(\mathrm{Min}(rx_j^* + D, ml^* - L_{jr}) - \mathrm{Max}(rx_i^* - D, 0)$$
$$- Ay_{irjst}\,Bx_{irjst} + Ax_{irjst}\,By_{irjst}.$$

If $(Bx_{irjst} - Ax_{irjst}) \leq 0$ And $(Ay_{irjst} - By_{irjst}) \leq 0$ Then

$$M3_{irjst} = |AB_{irjst}|G + (Bx_{irjst} - Ax_{irjst})\,(\mathrm{Min}(ry_i^* + D, W - H_{ir}) - \mathrm{Max}(ry_j^* - D, 0)$$
$$+ (Ay_{ijkt} - By_{ijkt})\,(\mathrm{Min}(rx_i^* + D, ml^* - L_{ir}) - \mathrm{Max}(rx_j^* - D, 0)$$
$$- Ay_{irjst}\,Bx_{irjst} + Ax_{irjst}\,By_{irjst}.$$

7.2.2 New Variables

$r_{ir} = 1$ if polygon i is in orientation r; otherwise 0.

7.2.3 Rotate ml

The problem is formulated as the following linear mixed integer programming model, Rotate ml:

Minimise ml (5.1)

subject to

$ry_i + H_{ir} - W \leq M1_{ir}(1 - r_{ir})$, for all i,r, (7.1)

$rx_i + L_{ir} - ml \leq M2_{ir}(1 - r_{ir})$, for all i, (7.2)

$$\sum_r r_{ir} = 1 \text{ , for all } i, \tag{7.3}$$

$$(Ax_{irjst} - Bx_{irjst})\,(ry_j - ry_i) + (By_{irjst} - Ay_{irjst})\,(rx_j - rx_i) + Ay_{irjst}\,Bx_{irjst} - Ax_{irjst}\,By_{irjst}$$
$$+ M3_{irjst}(1 - r_{ir}) + M3_{irjst}(1 - r_{js}) \geq |AB_{irjst}|\,G, \text{ for all } i,j,r,s,t, j > i, \tag{7.4}$$

$$rx_i = rx_i^* + dx_i, \text{ for all } i \tag{6.1}$$

$$ry_i = ry_i^* + dy_i, \text{ for all } i \tag{6.2}$$

$$-D \leq dy_i \leq D, \text{ for all } i \tag{6.3}$$

$$-D \leq dy_i \leq D, \text{ for all } i \tag{6.4}$$

$$rx_i \geq 0,\ ry_i \geq 0, \text{ for all } i, \tag{5.8}$$

$$r_{ir} \in \{0, 1\}, \text{ for all } i, r. \tag{7.5}$$

7.2.4 Explanation of Constraints

(7.1, 7.2) These constraints are the equivalent of (5.2) and (5.3) from model Global ml. $M1_{ir}$ and $M2_{ir}$ are sized such that these constraints are loose if $r_{ir} = 0$.

(7.3) This constrains a polygon to be in exactly one orientation.

(7.4) This constrains a reference point r_B to lie in the appropriate CFSR for polygon i in orientation r, and polygon j in orientation s, if r_{ir} and $r_{js} = 1$. The constraint is loose if r_{ir} or $r_{js} = 0$.

7.3 Rotation Algorithm

The solution time for the above model depends on the number of polygons that can rotate, and the number of different orientations each polygon can have. For industrial sized markers, even rotating a few polygons at once is time consuming. For example, if two rotating polygons are in each other's local neighbourhood, up to 64 NFPs need to be calculated (8 different orientations for each polygon). Because in practice there are time constraints on creating a marker, a rotation algorithm has been developed.

Solving the rotation model with only one polygon rotating takes little more time than solving the compaction model for the same marker. In this case, Rotate ml will have at most 8 binary variables. The rotation algorithm attempts to rotate each polygon one at a time in a specific order (the *rotation list*) until no further improvement can be made. The order of the rotation list is based on finding polygons which, when moved, change the objective function the most (see section 7.3.1). We want to rotate these polygons first because as the efficiency of the marker improves there is less room for a polygon to

move around (and therefore it is less likely that a polygon can change orientation while keeping the marker feasible).

7.3.1 A Polygon's Effect on the Objective Function

The effect of polygon i on the objective function is measured by the sum of the dual prices on the constraints of type 7.1, 7.2, and 7.4 for polygon i.

Figure 7.3 shows polygons A and B in a marker after the completion of the compaction algorithm.

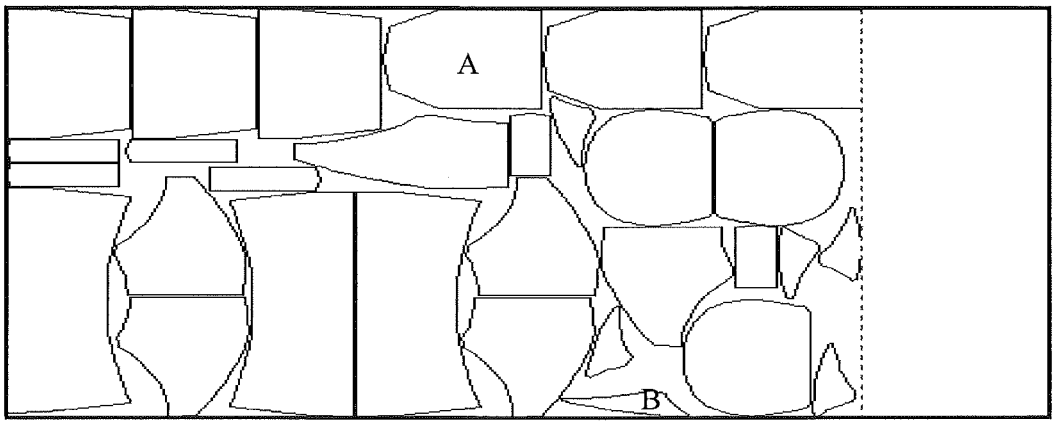

Figure 7.3

Figure 7.4 shows a close up of polygon A, and its surrounding polygons (C, D, E, F, G). Constraints are dotted if the dual price is zero. Of the constraints of type 7.4 for polygon A, only the ones involving polygon C and polygon G have a non-zero dual price. Although the constraint of type 7.1 is binding, the dual price on that constraint is zero.

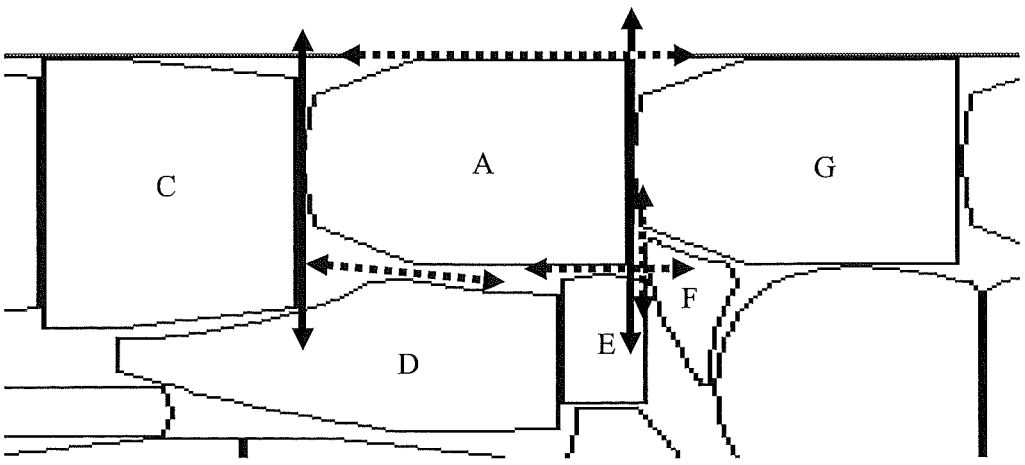

Figure 7.4: Effect of polygon A

The effect of polygon *A* on the objective function is then the sum of the dual prices on the constraints of type 7.4 for *A*/*C*, and *A*/*G*.

Figure 7.5 shows a close up of polygon *B*, and its surrounding polygons (*H*, *I*, *J*). All constraints of type 7.4 for polygon *B* have a dual price of zero. The dual price on the constraint of type 7.1 for polygon *B* is also zero. The effect of polygon *B* on the objective function is then zero. This is obvious, as polygon *B* is free to move without touching any of its neighbouring polygons or cloth boundaries.

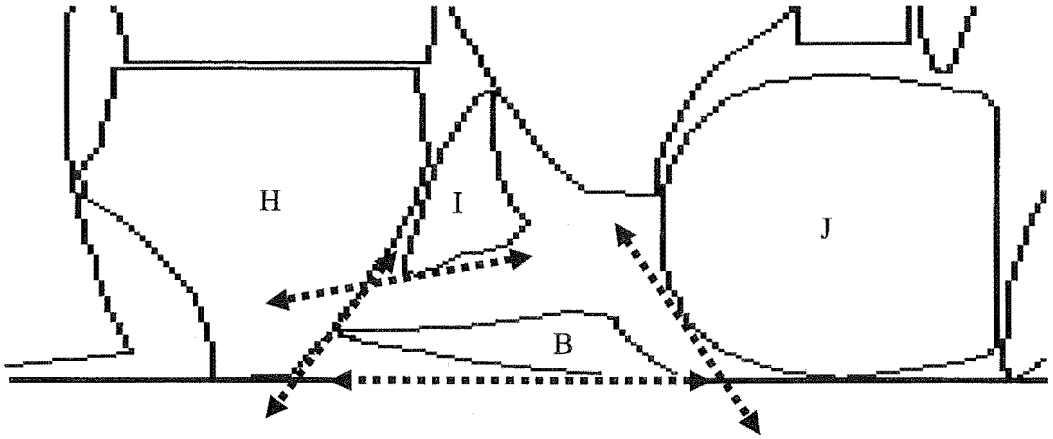

Figure 7.5: Effect of polygon *B*

The rotation list is then sorted from the polygon with the highest effect to the polygon with the lowest effect. Once the rotation list has been found, each polygon in the list is rotated via the rotation model until a rotation differing from a polygon's original rotation is found. The compaction algorithm is then applied, a new rotation list is created, and the process repeats until an entire rotation list is completed with no new rotations. Pseudo-code 7.1 gives the rotation algorithm.

Start:

Create *RotationList*
$i = 0$

Loop{
    CurrOrientation = *RotationList*[*i*].*Orientation*
    Solve Rotate ml, with *RotationList*[*i*] rotatable
    If *CurrOrientation* ≠ *RotationList*[*i*].*Orientation* **Then**
        GoTo Start
    Else
        $i = i + 1$
        If $i >$ *NumberOfPolygons* **Then Exit Algorithm**
    End If
}

Pseudo-code 7.1

After completion of the rotation algorithm, a marker cannot be improved by simultaneously translating polygons, or by changing the orientation of any one polygon.

7.4 Numerical Results

The algorithm in this chapter has been tested on the same set of data as used in chapter 6, after completion of the compaction algorithm. Table 7.1 gives the results of the rotation algorithm, in which a maximum of 10 minutes was used. The table also compares generating a rotation order based on dual prices, and a randomly generated rotation order.

The average percentage improvement in efficiency for the rotation algorithm (after compaction) was 0.624%. All markers could be compacted in a practical amount of time.

The "% Diff" compares the two methods of sorting the rotation list. The dual price sorted list outperforms the randomly sorted list by over 0.2%.

Figures 7.6a and 7.6b and 7.6c show a marker (VAJ AB C=0220 1) before any algorithm, after the compaction algorithm, and after the rotation algorithm.

7.5 Conclusion

This chapter has given a model, Rotate ml, which is a relaxation of Compact ml (see Chapter 6). Rotate ml allows n polygons to change orientation, whilst a series of simultaneous translations are applied to each polygon in a pre-generated marker, in an attempt to decrease the length of the marker. The rotation algorithm applies iterations of Rotate ml (n=1), such that when the algorithm is finished, the marker cannot be improved by translation or a change in orientation of any one polygon.

The rotation algorithm increased marker efficiency by an average of 0.624% on 50 industrial markers (markers already compacted by the compaction algorithm). When combined with the compaction algorithm, marker efficiency is increased by 1.518% on the 50 industrial markers.

		Dual Priced Sorted		Randomly Sorted		
	Pieces	% Imp	Time (s)	% Imp	Time (s)	% Diff
AMP2 A=0220 BC=0440 131 1	52	0.00	290.5	0.00	207.5	0.00
AMP2 A=0220 BC=0440 131 2	24	0.03	62.4	0.03	66.6	0.00
ASX2 S3 ABCDEF=3 * 1	21	0.99	450.9	0.24	378.4	0.75
ASX2 S3 ABCDEF=3 * 2	69	0.25	467.3	0.30	486.1	-0.05
ASX2 S3 ABCDEF=3 * 3	57	0.03	261.2	0.03	316.1	0.00
ASX2 S3 ABCDEF=3 * 4	24	0.00	66.6	0.00	62.1	0.00
ATV2 ABD=6 C=3 *	48	0.77	146.0	0.59	150.6	0.19
CLA2 LG A=6 *	24	0.88	28.8	0.09	21.5	0.79
CYC2 ABCDE=10 * 1	60	0.11	600.0	0.06	600.0	0.05
CYC2 ABCDE=10 * 2	80	0.26	600.0	0.00	600.0	0.26
DYN02 S2 ABC=20 *	100	0.20	600.0	0.39	600.0	-0.19
EXP2 STD ABCDE=6 1	84	0.25	600.0	0.08	600.0	0.17
EXP2 STD ABCDE=6 2	42	0.83	341.9	0.83	310.8	0.00
EXP2 STD ABCDE=6 3	60	0.01	423.5	0.00	202.5	0.01
FAN2 ABCDE=6 183 *	36	0.04	200.0	0.00	52.7	0.04
FAN2 ABCDE=6 183 * 2	66	0.16	402.0	0.13	357.0	0.04
FAN2 ABCDE=6 183 * 3	21	0.00	102.3	0.00	112.4	0.00
FAN2 ABCDE=6 183 * 4	12	0.03	73.5	0.09	89.7	-0.07
FAN2 ABCDE=6 183 * 5	24	0.09	180.6	0.09	104.9	0.00
FAN2 ABCDE=6 183 * 6	24	0.20	58.6	0.00	48.6	0.20
GAU2 ABC=053	64	0.30	488.3	0.31	407.7	-0.01
GEC2 S1 AB=10 C=5 D=8 183 1	120	0.02	600.0	0.01	600.0	0.01
GEC2 S1 AB=10 C=5 D=8 183 2	45	0.92	600.0	0.92	600.0	0.00
GEN2 S3 ABD=6 C=3 1	138	0.02	600.0	0.00	600.0	0.02
GEN2 S3 ABD=6 C=3 2	48	0.05	600.0	0.04	600.0	0.01
HDP2 ABD=6 C=3 * 1	48	0.14	600.0	0.13	600.0	0.01
HDP2 ABD=6 C=3 * 2	12	0.15	99.1	0.15	97.9	0.00
HDP2 ABD=6 C=3 * 3	24	0.54	305.5	0.51	250.2	0.03
KOA2 STD ABEF=6 CD=3 * 1	60	0.00	207.9	0.00	243.1	0.00
KOA2 STD ABEF=6 CD=3 * 2	33	1.71	470.4	1.22	327.7	0.49
KOA2 STD ABEF=6 CD=3 * 3	72	0.43	600.0	0.42	600.0	0.01
KOA2 STD ABEF=6 CD=3 * 4	24	1.26	35.1	0.36	32.9	0.90
MCT2 ABD=111 1	12	0.98	18.3	0.64	15.4	0.34
MCT2 ABD=111 2	42	0.00	116.8	0.03	150.1	-0.02
STL2 AB=1 CD=2 E=6 * 1	17	0.00	29.0	0.00	25.7	0.00
STL2 AB=1 CD=2 E=6 * 2	8	0.00	4.9	0.00	4.9	0.00
STR2 ABC=1221 1	18	1.38	53.6	1.31	50.1	0.07
STR2 ABC=1221 2	6	0.00	10.6	0.00	10.6	0.00
STR2 ABC=1221 3	48	2.10	324.4	0.92	245.6	1.18
TUA2 S1 AB=6 183 1	42	0.09	204.2	0.19	124.6	-0.10
TUA2 S1 AB=6 183 2	48	0.16	600.0	0.07	600.0	0.08
VAJ ABC=2220 1	24	3.93	98.9	3.22	50.3	0.71
VAJ ABC=2220 2	12	4.73	42.2	3.61	54.3	1.12
ZEN2 ABCDE=122 1	35	1.85	189.0	0.68	248.5	1.17
ZEN2 ABCDE=122 2	35	2.20	194.6	1.48	134.9	0.72
ZEN2 ABCDE=122 3	45	0.31	100.3	0.50	111.7	-0.19
ZEN2 ABCDE=122 4	40	0.11	98.6	0.11	123.8	-0.01
ZEN2 ABCDE=122 5	10	1.95	32.6	0.68	28.7	1.27
ZOD2 ABCD=1221 1	24	0.08	19.0	0.04	21.7	0.04
Average	42.49	0.62	271.4	0.42	251.6	0.21

Table 7.1: Rotation algorithm results

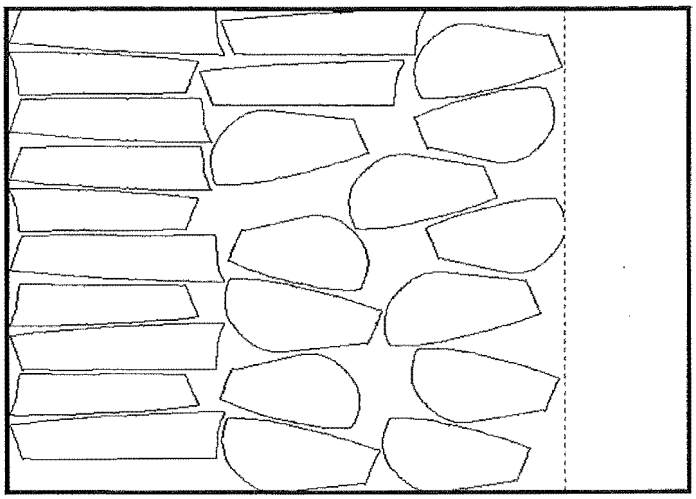

Figure 7.6a: Human generated marker

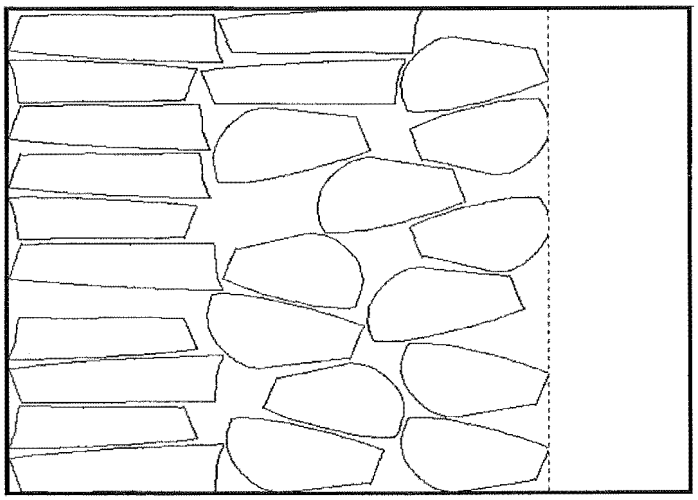

Figure 7.6b: After compaction algorithm

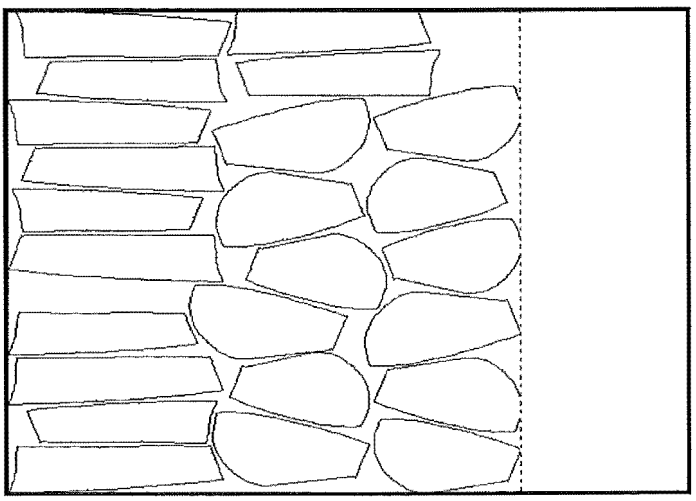

Figure 7.6c: After rotation algorithm

Chapter 8: Application

8.1 Introduction

The work in this thesis has been sponsored by a small Christchurch clothing manufacturer. A Windows application (programmed in C++) has been developed that can read data from their marker-making system. Compaction or rotation algorithms can then be performed on the markers and fed back into their system.

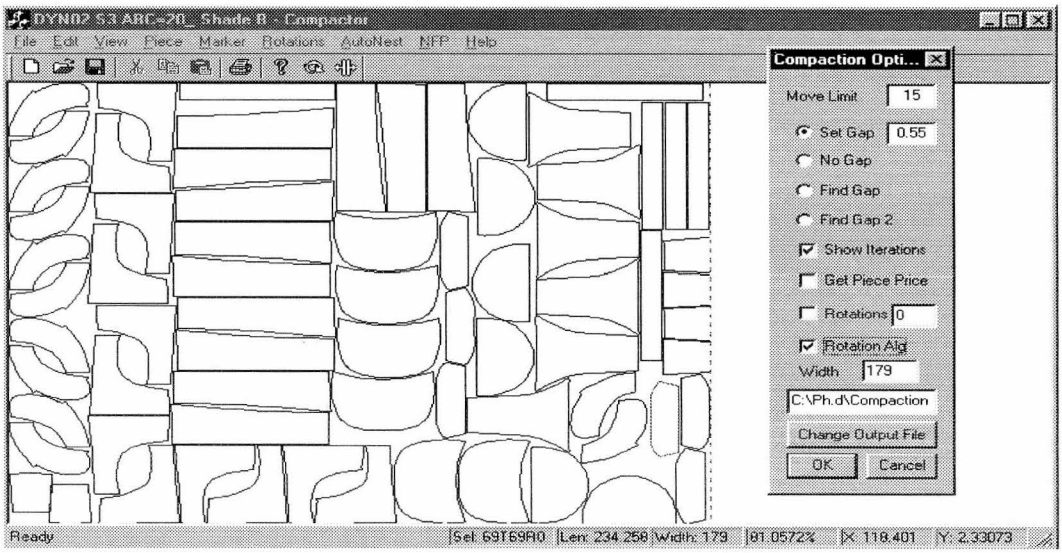

Figure 8.1: Screenshot of compaction software

8.2 Human Intervention

In practice, the compaction and rotation algorithms have been found to have other uses. For many markers with overlapping polygons, the compaction algorithm can separate the polygons without decreasing the efficiency of the marker. This can greatly speed up the marker making process by allowing human operators to roughly generate a marker, and then let the compaction algorithm remove overlaps, and compact the marker.

An operator can also quickly make changes to a marker that would otherwise be time consuming. For example, say an operator wishes to change the positions of polygons A, B, C, and D (Figure 8.2). A change of positions is not possible without also rearranging other polygons (Figure 8.3). However, after an application of the compaction algorithm, the overlaps are quickly removed (Figure 8.4), and the resulting efficiency of the marker in Figure 8.4 is 0.86% greater than that of the marker in Figure 8.2.

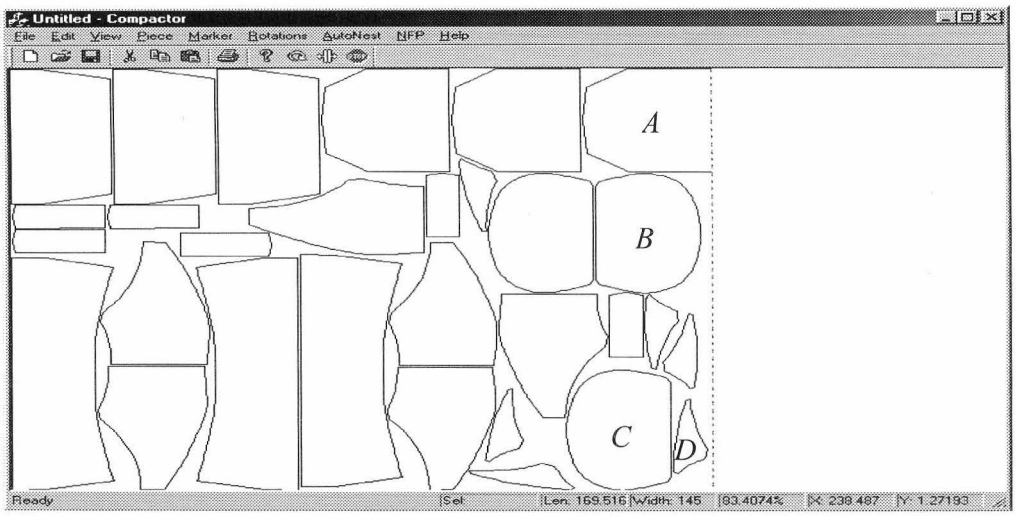

Figure 8.2: Before moving polygons

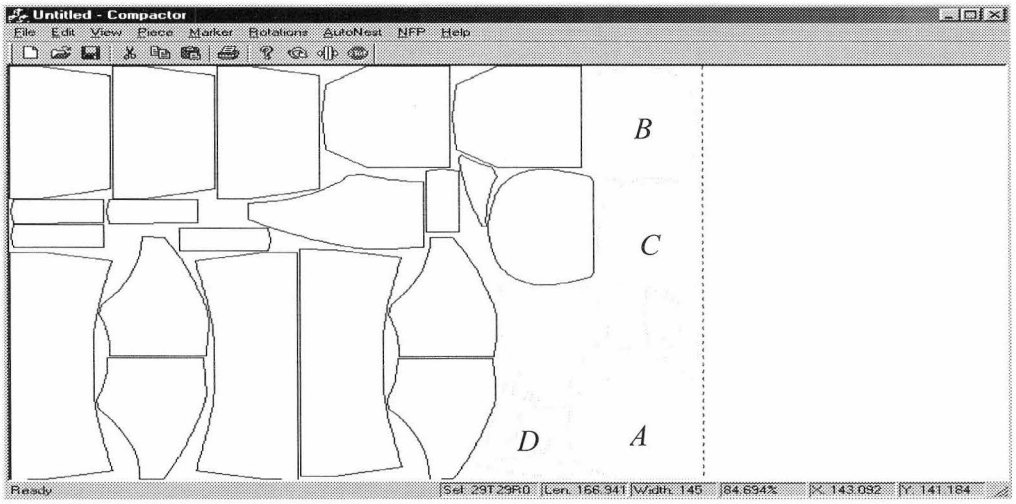

Figure 8.3: After moving polygons

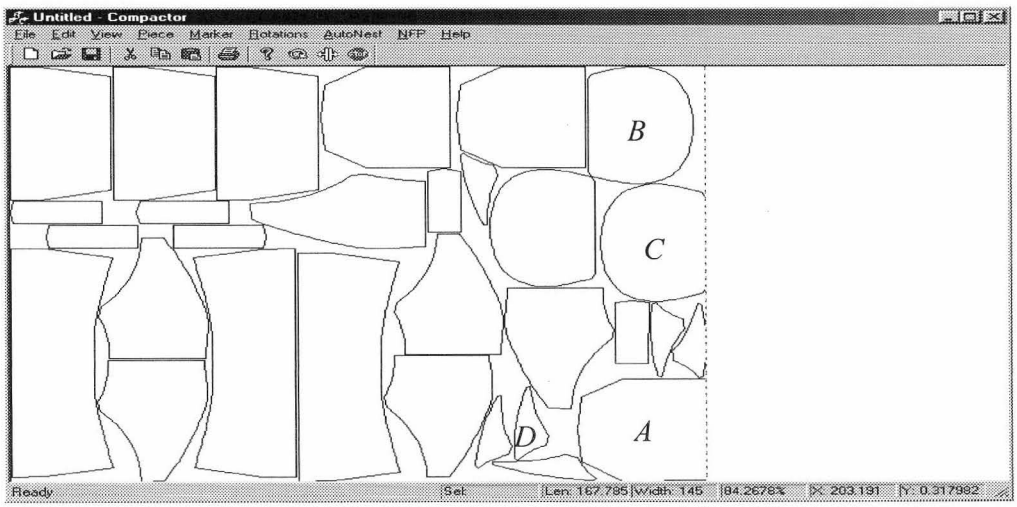

Figure 8.4: After compaction algorithm

In addition, bolts of cloth are often up to 10cm wider than a marker was made for, and the compaction and rotation algorithms can quickly adjust a marker to use this extra cloth area that would otherwise go unused.

8.3 Other Functions

The software can also perform functions other than compaction and rotational compaction. For example, an operator may wish to translate a polygon i as far as possible in a direction s without overlap (see Figure 8.5).

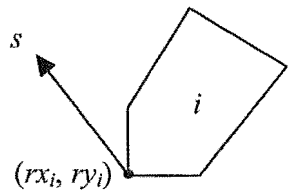

Figure 8.5: Translating polygon i

By changing the objective function (6.1) from Minimise ml to

$$\text{Maximise } -dx_i, \tag{8.1}$$

and constraining the deviation (dx_i, dy_i) from the original polygon position (rx^*_i, ry^*_i) to be in the direction s by the following constraints

$$dy_i - dy^*_i = s\,(dx_i - dx^*_i), \tag{8.2}$$

$$dy_i \geq 0, \tag{8.3}$$

$$dx_i \leq 0, \tag{8.4}$$

and constraining all other polygons to be stationary, we can maximise the movement of polygon i in direction s.

Note: Equations 8.3 and 8.4 guide the deviation in a left-and-upward direction, as opposed to a right-and-downward direction which would be possible without these two constraints. The objection function coefficient is negative because the horizontal component of s is negative.

This model is easily modified to allow other polygons to translate in an attempt to translate polygon i further in direction s.

Chapter 9: Conclusion and Future Work

9.1 Conclusion

This thesis has presented compaction and rotational compaction algorithms which can increase the efficiency of human generated markers significantly more than previously developed algorithms. Even a small efficiency improvement will result in large savings for industries in which the 2-dimensional cutting stock problem is important. The compaction and rotational compaction algorithms in the thesis have been able to improve human generated markers by over 1.5% on average.

By developing a robust and efficient NFP algorithm, there is no restriction on the type and size of problems that can be solved. The algorithm has allowed all of the industrial data sets tested to be solved in a practical amount of time. It has also opened the possibility of developing improved compaction algorithms, such as one which accounts for fine rotations, which have been previously impractical due to time constraints.

9.2 Future Work

This thesis has created many options for future research in improving human generated markers. The remainder of this chapter summarises a few potential research avenues.

9.2.1 Multiple Orientation Changes

After completion of the rotational compaction algorithm of chapter 7, a marker cannot be improved by simultaneously translating polygons or by changing the orientation of any one polygon. It may, however, be improved by changing of orientation of two or more polygons at once. The main problem with this is determining which combination of polygons should be rotated, and the time taken to calculate the possibly large amount of extra NFPs.

9.2.2 Switching of Polygons

A switch in position of two or more polygons could also result in a gain in efficiency. Again, choosing which combination of polygons to switch would be a problem.

In the clothing industry, often markers contain patterns of differing size (for example, S, M, L, and XL). If it was found that moving say an XL polygon would improve the efficiency of a marker, then this XL polygon could be switched with a smaller polygon

of the same pattern type. The smaller polygon would be selected such that replacing it with an XL polygon would not decrease the efficiency of the marker. This could be done by solving a modified version of Compact ml, with the objective of finding the smaller polygon with the maximum distance G between itself and neighbouring polygons, such that the efficiency of the marker remains constant. If G is large enough to accommodate the additional size of the XL polygon, then a switch could be made.

This could also be done with similarly shaped polygons.

9.2.3 Hole Finding

Often after the compaction or rotation algorithms, "holes" are created in the marker which could be filled by polygons whose current position is affecting the efficiency of the marker. Holes for a specific polygon P can easily (yet not so efficiently) be found by computing the union U of all NFPs[P, S] for the set S of remaining polygons. Any area of the marker which is not a part of U is a location for the reference point of P such that no overlap will occur with any other polygon. However, this method does not account for the slight movements that each polygon may be able to have without decreasing the efficiency of the maker, once P has been removed from its current location.

9.2.4 Fine Rotation

Another area for research is in fine rotation of polygons to improve the marker efficiency. Heckmann et al [33] use a simulated annealing algorithm for fine rotation of individual polygons, which has compacted markers by an average of 0.5%. This could be greatly improved, if a method for simultaneous fine rotation of polygons was developed.

9.2.5 Faster NFP Calculation

Even though this thesis has presented a robust method for calculating NFPs in a practical amount of time, NFP calculation still often takes the majority of time for the compaction and rotation algorithms.

Perhaps a hybrid method combining the slope based techniques presented in this thesis with a decomposition technique such as Lo Valvo [45] would be able to calculate significantly faster NFPs.

Appendix A: NFP Data

The following data is a set of 8 ordered lists of points in the form (x, y). Each ordered list of points represents a polygon.

Polygon 1:

29.716 , 0
59.432 , 0.33
59.407 , 3.022
59.331 , 10.794
59.305 , 11.353
59.305 , 11.988
59.28 , 12.648
59.254 , 13.385
59.229 , 14.147
59.204 , 15.01
59.178 , 15.899
59.127 , 16.839
59.077 , 17.855
59.051 , 18.465
57.832 , 18.414
56.664 , 18.312
55.495 , 18.185
54.353 , 18.058
53.235 , 17.855
52.143 , 17.652
51.076 , 17.398
50.035 , 17.118
49.019 , 16.788
48.638 , 16.661
47.165 , 16.229
45.565 , 15.823
43.787 , 15.468
41.857 , 15.163
39.799 , 14.909
37.564 , 14.68
35.177 , 14.502
32.662 , 14.401
29.97 , 14.299
29.564 , 14.299
26.872 , 14.401
24.357 , 14.553
21.97 , 14.731
19.735 , 14.96
17.678 , 15.239
15.747 , 15.569
13.995 , 15.925
12.369 , 16.331
10.922 , 16.788
10.541 , 16.915
9.525 , 17.245
8.483 , 17.525
7.417 , 17.779
6.325 , 18.007
5.207 , 18.211
4.064 , 18.363
2.896 , 18.49
1.727 , 18.591
0.508 , 18.642
0.483 , 18.033
0.432 , 17.042
0.381 , 16.077
0.331 , 15.188
0.305 , 14.35
0.254 , 13.563
0.229 , 12.826
0.204 , 12.166
0.204 , 11.556
0.178 , 10.997
0.051 , 3.2
0 , 0.508

Polygon 2:

5.638 , 0
6.959 , 0
8.331 , 0.05
9.753 , 0.127
11.251 , 0.254
11.429 , 1.041
11.582 , 1.854
11.759 , 2.666
11.937 , 3.504
12.14 , 4.343
12.318 , 5.181
12.547 , 6.044
12.75 , 6.908
12.978 , 7.771
13.131 , 8.203
14.096 , 12.546
14.934 , 16.839
15.696 , 21.08
16.356 , 25.245
16.89 , 29.36
17.347 , 33.424
17.677 , 37.411
17.931 , 41.348
18.084 , 45.208
18.084 , 45.615
17.931 , 55.418
9.702 , 55.571
9.626 , 53.285
9.626 , 52.879
9.524 , 48.383
8.864 , 42.567
8.102 , 36.878
7.264 , 31.316
6.299 , 25.88
5.232 , 20.597
4.089 , 15.416
2.819 , 10.362
1.448 , 5.46
0 , 0.685
0.991 , 0.457
2.057 , 0.279
3.2 , 0.127
4.394 , 0.05

Polygon 3:

0.685 , 0
9.6 , 0.025
10.007 , 0.025
11.53 , 0
11.53 , 8.077
0.177 , 8.077
0.177 , 7.772
0.152 , 7.594
0.152 , 7.391
0.127 , 7.162
0.076 , 6.654
0.025 , 6.375
0 , 6.07
0.279 , 5.791
0.508 , 5.41
0.685 , 4.927
0.838 , 4.369
0.889 , 3.683
0.914 , 2.895
0.889 , 2.032
0.812 , 1.067

Polygon 4:

12.801 , 0
13.969 , 0
15.264 , 0.025
16.661 , 0.051
18.185 , 0.101
19.81 , 0.152
21.538 , 0.203
23.392 , 0.279
24.052 , 0.33
23.976 , 0.965
23.442 , 5.689
23.442 , 6.095
23.976 , 10.845
24.052 , 11.48
23.392 , 11.505
21.538 , 11.581
19.81 , 11.658
18.185 , 11.708
16.661 , 11.759
15.264 , 11.784
11.734 , 11.784
10.794 , 11.759
10.388 , 11.759
10.007 , 11.784
9.194 , 11.784
8.737 , 11.759
7.772 , 11.708
7.238 , 11.658
6.705 , 11.632
6.121 , 11.556
5.562 , 11.505
5.003 , 11.429
4.419 , 11.353
3.2 , 11.099
2.565 , 10.946
1.93 , 10.769
1.27 , 10.565
0.584 , 10.362
0 , 10.235
0.025 , 9.753
0.025 , 2.057
0 , 1.574
0.584 , 1.447
1.27 , 1.219
1.93 , 1.016
3.2 , 0.711
3.81 , 0.558
4.419 , 0.457
5.003 , 0.355
5.562 , 0.279
6.121 , 0.228
6.705 , 0.178
7.772 , 0.076
8.737 , 0.025
10.007 , 0.025
10.388 , 0.051
10.794 , 0.051
11.734 , 0.025

Polygon 5:

53.184 , 0
53.514 , 0.508
53.844 , 1.752
54.327 , 3.124
53.997 , 3.606
53.667 , 4.114
52.956 , 5.079
51.813 , 6.527
51 , 7.441
50.72 , 7.746
49.298 , 9.016
47.825 , 10.235
46.327 , 11.378
44.752 , 12.495
43.126 , 13.537
41.476 , 14.553
39.748 , 15.492
37.996 , 16.381
36.167 , 17.22
35.812 , 17.397
35.405 , 17.575
35.024 , 17.753
34.618 , 17.931
34.212 , 18.083
33.831 , 18.21
33.424 , 18.337
32.612 , 18.54
32.205 , 18.616
31.367 , 18.718
30.58 , 18.896
29.869 , 19.15
29.208 , 19.455
28.649 , 19.81
28.141 , 20.242
27.71 , 20.75
27.329 , 21.309
27.049 , 21.918
26.516 , 21.639
25.424 , 20.928
24.408 , 20.318
23.392 , 19.734
22.452 , 19.226
21.538 , 18.794
20.649 , 18.439
19.811 , 18.134
19.024 , 17.88
18.262 , 17.727
13.334 , 17.727
11.963 , 18.032
10.642 , 18.312
9.398 , 18.591
8.204 , 18.845
7.086 , 19.074
6.02 , 19.277
5.029 , 19.48
4.089 , 19.658
3.226 , 19.81
2.616 , 19.886
2.515 , 19.328
2.438 , 19.048
2.134 , 18.134
1.88 , 17.474
1.727 , 17.143
0.203 , 12.394
0 , 11.784
0.635 , 11.632
2.438 , 11.175
4.14 , 10.768
5.715 , 10.387
7.188 , 10.057
8.559 , 9.752
9.829 , 9.473
10.972 , 9.244
12.014 , 9.041
12.953 , 8.889
13.334 , 8.787
14.096 , 8.635
14.503 , 8.533
14.96 , 8.457
15.417 , 8.406
15.925 , 8.33
16.458 , 8.254
16.992 , 8.203
17.576 , 8.152
17.931 , 8.127
18.693 , 8.076
19.1 , 8.076
19.481 , 8.051
21.132 , 8.051
21.538 , 8.102
21.767 , 8.102
22.173 , 8.152
22.401 , 8.152
22.605 , 8.178
23.036 , 8.178
23.443 , 8.229
23.57 , 8.229
23.824 , 8.279
23.925 , 8.305
24.56 , 8.432
25.754 , 8.838
26.872 , 9.194
27.913 , 9.473
28.853 , 9.727
29.691 , 9.93
30.478 , 10.083
31.164 , 10.184
31.748 , 10.235
32.256 , 10.235
32.586 , 10.26
33.424 , 10.286
34.339 , 10.21
35.304 , 10.032
36.37 , 9.778
37.488 , 9.448
38.682 , 8.991
39.952 , 8.457
41.298 , 7.848
42.72 , 7.136
42.974 , 6.959
43.838 , 6.4
44.777 , 5.79
45.768 , 5.13
46.835 , 4.393
47.977 , 3.631
49.171 , 2.793
50.441 , 1.93
51.787 , 0.99

Polygon 6:

9.245 , 0
9.778 , 0.432
9.778 , 1.016
9.804 , 1.6
9.905 , 2.667
9.956 , 3.175
10.032 , 3.658
10.134 , 4.115
10.235 , 4.521
10.337 , 4.953
10.489 , 5.791
10.743 , 6.604
11.099 , 7.366
11.531 , 8.077
12.064 , 8.737
12.674 , 9.372
13.41 , 9.956
14.198 , 10.49
15.112 , 10.972
14.502 , 12.191
13.944 , 13.385
13.461 , 14.528
13.029 , 15.62
12.674 , 16.661
12.369 , 17.652
12.14 , 18.617
11.963 , 19.531
11.861 , 20.395
11.861 , 24.306
12.242 , 26.008
12.598 , 27.583
12.902 , 29.081
13.156 , 30.453
13.359 , 31.723
13.512 , 32.865
13.639 , 33.907
13.715 , 34.847
13.74 , 35.685
12.75 , 35.761
11.734 , 35.888
10.667 , 36.04
9.601 , 36.218
8.483 , 36.447
7.365 , 36.726
6.197 , 37.005
5.003 , 37.361
3.784 , 37.742
2.946 , 34.897
2.21 , 32.281
1.6 , 29.894
1.067 , 27.71
0.635 , 25.779
0.33 , 24.052
0.127 , 22.579
0 , 21.309
0 , 20.268
0.229 , 17.931
0.66 , 15.62
1.295 , 13.309
2.108 , 11.048
3.149 , 8.788
4.368 , 6.553
5.791 , 4.343
7.416 , 2.159

Polygon 7:

12.039 , 0
12.877 , 2.87
13.588 , 5.562
14.223 , 8.051
14.731 , 10.337
15.112 , 12.445
15.417 , 14.376
15.595 , 16.103
15.671 , 17.627
15.645 , 18.947
15.391 , 21.487
14.934 , 24.001
14.274 , 26.516
13.41 , 28.98
12.369 , 31.443
11.125 , 33.881
9.651 , 36.294
8.026 , 38.682
6.172 , 41.069
5.639 , 40.663
5.613 , 40.053
5.562 , 39.444
5.512 , 38.885
5.359 , 37.818
5.283 , 37.336
5.181 , 36.853
5.054 , 36.421
4.953 , 35.989
4.775 , 35.126
4.496 , 34.288
4.14 , 33.5
3.683 , 32.764
3.124 , 32.078
2.489 , 31.418
1.753 , 30.834
0.914 , 30.275
0 , 29.767
0.61 , 28.395
1.168 , 27.049
1.651 , 25.779
2.083 , 24.535
2.438 , 23.367
2.743 , 22.224
2.972 , 21.157
3.149 , 20.141
3.251 , 19.176
3.251 , 14.757
2.87 , 12.775
2.515 , 10.947
2.21 , 9.245
1.956 , 7.696
1.753 , 6.274
1.6 , 5.004
1.473 , 3.886
1.397 , 2.896
1.372 , 2.057
2.515 , 1.981
3.657 , 1.854
4.826 , 1.702
5.994 , 1.524
7.188 , 1.295
8.382 , 1.016
9.601 , 0.711
10.82 , 0.381

Polygon 8:

1.143 , 0
2.566 , 0.965
3.912 , 1.905
5.182 , 2.793
6.375 , 3.606
7.518 , 4.394
8.585 , 5.105
9.575 , 5.79
10.515 , 6.4
11.379 , 6.959
11.633 , 7.111
13.055 , 7.822
14.401 , 8.457
15.671 , 8.991
16.865 , 9.422
17.982 , 9.778
19.024 , 10.032
20.014 , 10.184
20.928 , 10.261
21.767 , 10.235
22.605 , 10.235
23.189 , 10.159
23.875 , 10.057
24.637 , 9.93
25.5 , 9.727
26.44 , 9.473
27.456 , 9.168
28.573 , 8.838
29.792 , 8.432
30.173 , 8.356
30.275 , 8.33
30.91 , 8.203
31.291 , 8.178
31.52 , 8.178
31.748 , 8.152
31.951 , 8.152
32.18 , 8.127
32.383 , 8.127
33.196 , 8.025
34.466 , 8.025
34.872 , 8.051
35.253 , 8.051
35.659 , 8.076
36.04 , 8.102
36.396 , 8.102
36.777 , 8.127
37.361 , 8.203
38.428 , 8.305
39.393 , 8.457
40.257 , 8.61
40.637 , 8.686
40.993 , 8.787
41.399 , 8.864
42.339 , 9.041
43.381 , 9.219
44.523 , 9.473
45.793 , 9.727
47.165 , 10.032
48.638 , 10.388
50.213 , 10.743
51.914 , 11.175
53.718 , 11.607
54.353 , 11.759
54.149 , 12.369
52.625 , 17.118
52.473 , 17.474
52.346 , 17.804
52.194 , 18.109
52.092 , 18.439
51.965 , 18.744
51.864 , 19.023
51.762 , 19.328
51.686 , 19.607
51.584 , 19.861
51 , 19.785
50.568 , 19.709
50.111 , 19.632
49.654 , 19.531
49.171 , 19.429
48.155 , 19.226
47.622 , 19.124
47.063 , 18.997
46.479 , 18.896
46.098 , 18.845
45.59 , 18.718
44.574 , 18.515
43.457 , 18.261
42.873 , 18.134
42.263 , 18.007
41.628 , 17.855
40.993 , 17.728
36.091 , 17.702
35.329 , 17.88
34.542 , 18.109
33.704 , 18.413
32.815 , 18.794
31.901 , 19.226
30.935 , 19.734
29.945 , 20.293
28.904 , 20.928
27.837 , 21.639
27.303 , 21.893
26.999 , 21.283
26.643 , 20.725
26.211 , 20.217
25.703 , 19.785
25.119 , 19.404
24.484 , 19.099
23.773 , 18.845
22.986 , 18.667
22.122 , 18.515
21.716 , 18.464
21.309 , 18.388
20.903 , 18.286
20.497 , 18.159
20.065 , 18.032
19.659 , 17.855
19.227 , 17.702
18.795 , 17.499
18.363 , 17.27
18.008 , 17.093
16.103 , 16.28
14.249 , 15.391
12.496 , 14.451
10.82 , 13.461
9.22 , 12.419
7.721 , 11.327
6.274 , 10.184
4.902 , 8.991
3.607 , 7.746
3.353 , 7.441
2.947 , 6.984
2.54 , 6.502
2.134 , 6.044
1.753 , 5.562
1.042 , 4.597
0.686 , 4.089
0.331 , 3.606
0 , 3.098
0.508 , 1.752
0.838 , 0.482

Appendix B: Global Data

$W = 155$ for all problems. The data is for the polygon used in the testing of Global ml. The data is an ordered lists of points, (rx, ry).

50.795 , 0
76.193 , 17.779
58.414 , 55.875
38.096 , 55.875
0 , 30.477

Appendix C: VAJ AB C=0220 1 Data

The data below is a set of 24 ordered lists of points in the form (x, y). Each ordered list of points represents a polygon in VAJ AB C=0220 1. $W = 141$, and the original length is 132.12.

Polygon 1:

13.1307 , 0
21.1818 , 0
24.1026 , 0.0253979
27.1503 , 0.0761937
29.2584 , 0.0761937
29.6393 , 0.0761937
49.0687 , 0
49.1703 , 0.83813
49.2465 , 1.65086
49.3227 , 2.4382
49.4243 , 3.96207
49.4751 , 4.69861
49.5005 , 5.40975
49.5005 , 6.78124
49.5513 , 7.36539
49.6275 , 8.00033
49.7545 , 8.63528
49.8814 , 9.32102
50.0592 , 10.0322
50.2878 , 10.7687
50.5418 , 11.5306
50.8212 , 12.3434
51.1259 , 13.1815
51.2021 , 13.6641
48.7131 , 13.5879
46.3511 , 13.5117
44.0653 , 13.4355
41.8811 , 13.3593
39.7731 , 13.2831
37.7921 , 13.2069
35.9126 , 13.1307
34.1094 , 13.0291
32.4077 , 12.9529
32.0013 , 12.9275
31.8236 , 12.9275
31.6458 , 12.9021
31.2902 , 12.9021
31.1124 , 12.8767
30.7568 , 12.8767
30.5791 , 12.8513
30.4267 , 12.8513
30.0203 , 12.8259
26.7948 , 12.6735
20.3437 , 12.2672
17.1182 , 12.0132
13.8926 , 11.7084
10.6417 , 11.4037
7.41618 , 11.0481
4.19065 , 10.6417
0.939722 , 10.2353
0.558753 , 10.1592
0 , 9.98137
1.90484 , 4.08906
2.00643 , 3.70809
2.51439 , 2.18422
2.64138 , 1.80325
3.17474 , 0.0761937
3.75889 , 0.279377
4.01287 , 0.228581
6.12089 , 0.152387
8.33051 , 0.101592
10.6671 , 0.0507958

Polygon 2:

2.00643 , 12.9021
7.79715 , 12.9021
9.6258 , 12.9275
11.429 , 12.9783
13.1815 , 13.0291
14.9086 , 13.0799
16.5848 , 13.1815
16.9658 , 13.1815
17.2452 , 13.1815
17.3722 , 13.2069
18.0071 , 13.2069
18.1595 , 13.2323
18.3373 , 13.2323
18.7436 , 13.2577
21.9184 , 13.4101
25.0169 , 13.5625
28.0139 , 13.7657
30.9346 , 13.9688
33.7538 , 14.1974
36.4968 , 14.4514
39.1381 , 14.7054
41.7033 , 14.9848
44.1923 , 15.2895
44.5733 , 15.3657
45.1574 , 15.5435
43.405 , 20.9279
43.278 , 21.3088
42.8208 , 22.6803
42.7192 , 23.0613
42.2113 , 24.6106
41.7033 , 24.4328
41.4747 , 24.4836
39.6715 , 24.5852
37.7413 , 24.6613
35.7094 , 24.7375
33.6014 , 24.8137
31.3664 , 24.8645
29.0552 , 24.8899
26.617 , 24.9153
24.0772 , 24.9153
21.4612 , 24.8899
18.0579 , 24.8137
17.6515 , 24.8137
2.59058 , 25.2201
2.46359 , 24.4836
2.33661 , 23.7724
2.13342 , 22.3501
2.03183 , 21.6898
1.95564 , 21.0294
1.85405 , 19.7596
1.80325 , 19.15
1.75245 , 18.6167
1.65086 , 18.0579
1.54927 , 17.4737
1.1937 , 16.2039
0.96512 , 15.5435
0.406366 , 14.1212
0.0761937 , 13.3593
0 , 12.9529

Polygon 3:

45.8178 , 25.982
45.9194 , 26.744
46.021 , 27.5313
46.1226 , 28.2678
46.1988 , 29.0044
46.2749 , 29.7155
46.3257 , 30.4267
46.3511 , 31.087
46.4019 , 31.7474
46.4273 , 32.4077
46.4781 , 32.9665
46.5543 , 33.5506
46.6813 , 34.1602
46.8337 , 34.8205
47.0115 , 35.5062
47.2401 , 36.192
47.494 , 36.9285
47.7734 , 37.6905
48.1036 , 38.4778
48.1798 , 38.935
45.9194 , 38.935
43.7606 , 38.9096
41.6525 , 38.8588
39.5953 , 38.808
37.6397 , 38.7572
35.7348 , 38.7064
33.9316 , 38.6302
32.1791 , 38.554
30.4775 , 38.4778
30.0965 , 38.4524
29.6139 , 38.4524
29.4615 , 38.427
29.1314 , 38.427
28.9536 , 38.4016
28.598 , 38.4016
28.1917 , 38.3762
24.9915 , 38.2238
21.8422 , 38.046
18.7436 , 37.8428
15.6705 , 37.6143
12.6227 , 37.3603
9.6512 , 37.0555
6.70504 , 36.7507
3.80968 , 36.4206
0.939722 , 36.065
0.558753 , 35.9888
0 , 35.811
1.80325 , 30.1727
1.93024 , 29.7917
2.4128 , 28.344
2.53979 , 27.9631
3.04775 , 26.3376
3.5811 , 26.5154
3.83508 , 26.4646
5.79072 , 26.3884
7.87334 , 26.3122
10.0576 , 26.236
12.3434 , 26.1852
14.7308 , 26.1598
17.2452 , 26.1344
19.8611 , 26.1344
22.5787 , 26.1598
25.4233 , 26.1852
28.1663 , 26.2106
28.5726 , 26.2106

Polygon 4:

45.8178 , 39.9255
45.9194 , 40.7128
46.021 , 41.4747
46.1226 , 42.2113
46.1988 , 42.9478
46.2749 , 43.659
46.3257 , 44.3701
46.3511 , 45.0558
46.4019 , 45.7162
46.4273 , 46.3511
46.4781 , 46.9099
46.5543 , 47.494
46.6813 , 48.129
46.8337 , 48.7639
47.0115 , 49.4497
47.2401 , 50.1608
47.494 , 50.8974
47.7734 , 51.6593
48.1036 , 52.4466
48.1798 , 52.9038
45.9194 , 52.8784
43.7606 , 52.853
41.6525 , 52.8022
39.5953 , 52.7768
37.6397 , 52.726
35.7348 , 52.6498
33.9316 , 52.599
32.1791 , 52.5228
30.4775 , 52.4212
30.0965 , 52.3958
29.4615 , 52.3958
29.2838 , 52.3704
28.9536 , 52.3704
28.7758 , 52.345
28.598 , 52.345
28.1917 , 52.3196
24.9915 , 52.1673
21.8422 , 51.9895
18.7436 , 51.7863
15.6705 , 51.5577
12.6227 , 51.3037
9.6512 , 51.0244
6.70504 , 50.7196
3.80968 , 50.364
0.939722 , 50.0084
0.558753 , 49.9576
0 , 49.7799
1.80325 , 44.1161
1.93024 , 43.7352
2.4128 , 42.2875
2.53979 , 41.9065
3.04775 , 40.281
3.5811 , 40.4588
3.83508 , 40.4334
5.79072 , 40.3318
7.87334 , 40.2556
10.0576 , 40.1795
12.3434 , 40.1541
14.7308 , 40.1033
17.2452 , 40.1033
19.8611 , 40.0779
22.5787 , 40.1033
25.4233 , 40.1287
28.1663 , 40.1795
28.5726 , 40.1795

Polygon 5:

3.98747 , 52.4974
5.91771 , 52.4974
7.79715 , 52.5228
9.6258 , 52.5482
11.429 , 52.5736
13.1815 , 52.6244
14.9086 , 52.7006
16.5848 , 52.7768
16.9658 , 52.8022
17.6769 , 52.8022
17.8293 , 52.8276
18.3373 , 52.8276
18.7436 , 52.853
21.9184 , 53.0054
25.0169 , 53.1832
28.0139 , 53.361
30.9346 , 53.5641
33.7538 , 53.7927
36.4968 , 54.0467
39.1381 , 54.3261
41.7033 , 54.6055
44.1923 , 54.9102
44.5733 , 54.961
45.1574 , 55.1388
43.405 , 60.5486
43.278 , 60.9295
42.8208 , 62.301
42.7192 , 62.682
42.2113 , 64.2313
41.7033 , 64.0535
41.4747 , 64.0789
39.6715 , 64.1805
37.7413 , 64.282
35.7094 , 64.3582
33.6014 , 64.409
31.3664 , 64.4598
29.0552 , 64.4852
26.617 , 64.5106
21.4612 , 64.5106
18.0579 , 64.409
17.6515 , 64.4344
2.59058 , 64.8408
2.33661 , 63.3677
2.23501 , 62.6566
2.03183 , 61.2851
1.95564 , 60.6248
1.85405 , 59.3549
1.80325 , 58.7453
1.75245 , 58.212
1.65086 , 57.6532
1.54927 , 57.069
1.37149 , 56.4595
1.1937 , 55.8246
0.96512 , 55.1388
0.685743 , 54.4531
0.406366 , 53.7165
0.0761937 , 52.98
0 , 52.5482
2.00643 , 52.5228

Polygon 6:

15.6959 , 65.8059
18.3881 , 65.8059
21.1818 , 65.8313
24.1026 , 65.8567
27.1503 , 65.9075
29.2584 , 65.9075
29.6393 , 65.9075
49.0687 , 65.8313
49.1703 , 66.6694
49.2465 , 67.4568
49.3227 , 68.2695
49.4243 , 69.7934
49.4751 , 70.5299
49.5005 , 71.2411
49.5005 , 72.6126
49.5513 , 73.1967
49.6275 , 73.8317
49.7545 , 74.4666
49.8814 , 75.1523
50.0592 , 75.8635
50.2878 , 76.6
50.5418 , 77.362
50.8212 , 78.1747
51.1259 , 79.0128
51.2021 , 79.4954
48.7131 , 79.4192
46.3511 , 79.343
44.0653 , 79.2668
41.8811 , 79.1906
39.7731 , 79.1144
37.7921 , 79.0382
35.9126 , 78.962
34.1094 , 78.8604
32.4077 , 78.7842
32.0013 , 78.7588
31.8236 , 78.7588
31.6458 , 78.7334
31.2902 , 78.7334
31.1124 , 78.708
30.7568 , 78.708
30.5791 , 78.6826
30.4267 , 78.6826
30.0203 , 78.6573
26.7948 , 78.5049
20.3437 , 78.0985
17.1182 , 77.8445
13.8926 , 77.5397
10.6417 , 77.235
7.41618 , 76.8794
4.19065 , 76.473
0.939722 , 76.0667
0.558753 , 75.9905
0 , 75.8127
1.90484 , 69.9204
2.00643 , 69.5394
2.51439 , 68.0155
2.64138 , 67.6346
3.17474 , 65.9075
3.75889 , 66.1107
4.01287 , 66.0599
6.12089 , 65.9837
8.33051 , 65.9075
10.6671 , 65.8821
13.1307 , 65.8313

Polygon 7:

2.53979 , 80.0795
17.5753 , 80.3843
17.9817 , 80.3843
20.0897 , 80.3589
22.9851 , 80.3081
25.7535 , 80.2827
28.3948 , 80.2573
30.9092 , 80.2573
33.2712 , 80.2827
35.5062 , 80.3081
37.6143 , 80.3843
39.5699 , 80.4351
41.424 , 80.5367
42.1351 , 80.3843
42.643 , 81.9336
42.77 , 82.3145
43.2272 , 83.686
43.3542 , 84.067
45.1574 , 89.4514
44.5733 , 89.6291
44.1923 , 89.6799
41.7033 , 90.0101
39.1381 , 90.3149
36.4968 , 90.5943
33.7538 , 90.8736
30.9346 , 91.1276
28.0139 , 91.3562
25.0169 , 91.5594
21.9438 , 91.7626
18.769 , 91.9403
18.3627 , 91.9403
18.1849 , 91.9657
18.0071 , 91.9657
17.8547 , 91.9911
16.9912 , 91.9911
16.5848 , 92.0165
14.934 , 92.1181
13.2069 , 92.1943
11.4544 , 92.2705
9.6512 , 92.3213
7.82255 , 92.3721
5.91771 , 92.3975
2.03183 , 92.3975
0 , 92.3721
0 , 91.6864
0.330173 , 90.9752
0.634947 , 90.3149
0.914324 , 89.6545
1.1429 , 89.0196
1.34609 , 88.41
1.52387 , 87.8259
1.65086 , 87.2417
1.72706 , 86.7084
1.77785 , 86.175
1.82865 , 85.5655
1.85405 , 84.9305
1.93024 , 84.2956
2.00643 , 83.6352
2.08263 , 82.9495
2.18422 , 82.2638
2.28581 , 81.5526
2.3874 , 80.8161

Polygon 8:

51.1767 , 91.28
51.2021 , 92.0419
50.872 , 92.8039
50.5672 , 93.5658
50.3132 , 94.3023
49.9068 , 95.6738
49.7545 , 96.3342
49.6529 , 96.9691
49.5767 , 97.5787
49.5259 , 98.1628
49.5259 , 99.5343
49.5005 , 100.245
49.4751 , 100.982
49.3735 , 102.506
49.3227 , 103.319
49.2465 , 104.106
49.1449 , 104.944
29.7155 , 105.02
29.3346 , 105.02
27.2265 , 105.046
24.1788 , 105.122
21.258 , 105.173
18.4643 , 105.198
15.7721 , 105.223
13.2069 , 105.223
10.7433 , 105.198
8.4067 , 105.173
6.19708 , 105.122
4.08906 , 105.046
3.25093 , 105.223
2.69218 , 103.496
2.56519 , 103.115
2.05723 , 101.592
1.93024 , 101.236
0 , 95.3437
0.558753 , 95.1405
0.939722 , 95.0897
4.19065 , 94.6325
7.41618 , 94.2262
10.6417 , 93.8452
17.0928 , 93.1848
20.3183 , 92.9055
23.5438 , 92.6515
26.7694 , 92.4483
29.9949 , 92.2705
30.4013 , 92.2451
30.5791 , 92.2451
30.7568 , 92.2197
30.9346 , 92.2197
31.1124 , 92.1943
31.468 , 92.1943
31.6458 , 92.1689
31.7982 , 92.1689
31.9759 , 92.1435
32.3823 , 92.1181
34.084 , 92.0419
35.8872 , 91.9403
37.7667 , 91.8388
39.7477 , 91.7372
41.8557 , 91.6356
44.0399 , 91.534
46.3003 , 91.4578
48.6877 , 91.3562

Polygon 9:

2.53979 , 106.189
17.5753 , 106.468
17.9817 , 106.468
20.0897 , 106.468
22.9851 , 106.417
25.7535 , 106.366
30.9092 , 106.366
33.2712 , 106.392
35.5062 , 106.417
37.6143 , 106.468
39.5699 , 106.544
41.424 , 106.646
42.1351 , 106.493
42.643 , 108.043
42.77 , 108.424
43.2272 , 109.77
43.3542 , 110.151
45.1574 , 115.535
44.5733 , 115.738
44.1923 , 115.789
41.7033 , 116.119
39.1381 , 116.424
36.4968 , 116.703
33.7538 , 116.983
30.9346 , 117.237
28.0139 , 117.465
25.0169 , 117.668
21.9438 , 117.846
18.769 , 118.024
18.3627 , 118.049
18.1849 , 118.075
17.7023 , 118.075
17.5499 , 118.1
16.9912 , 118.1
16.5848 , 118.126
14.934 , 118.227
13.2069 , 118.303
11.4544 , 118.354
9.6512 , 118.43
7.82255 , 118.456
5.91771 , 118.481
3.98747 , 118.507
2.03183 , 118.507
0 , 118.481
0 , 117.77
0.330173 , 117.084
0.634947 , 116.424
0.914324 , 115.764
1.1429 , 115.129
1.34609 , 114.519
1.52387 , 113.91
1.65086 , 113.351
1.72706 , 112.792
1.77785 , 112.259
1.82865 , 111.649
1.85405 , 111.04
2.08263 , 109.059
2.18422 , 108.347
2.28581 , 107.662
2.3874 , 106.925

Polygon 10:

51.1767 , 117.364
51.2021 , 118.126
50.872 , 118.913
50.5672 , 119.675
50.3132 , 120.386
50.11 , 121.097
49.9068 , 121.783
49.7545 , 122.443
49.6529 , 123.078
49.5767 , 123.688
49.5259 , 124.272
49.5259 , 125.643
49.5005 , 126.354
49.4751 , 127.091
49.3735 , 128.615
49.3227 , 129.402
49.2465 , 130.215
49.1449 , 131.053
29.7155 , 131.129
29.3346 , 131.129
27.2265 , 131.155
24.1788 , 131.205
21.258 , 131.256
18.4643 , 131.307
10.7433 , 131.307
8.4067 , 131.282
6.19708 , 131.231
4.08906 , 131.155
3.25093 , 131.307
2.69218 , 129.605
2.56519 , 129.224
2.05723 , 127.701
1.93024 , 127.32
0 , 121.427
0.558753 , 121.25
0.939722 , 121.199
4.19065 , 120.742
7.41618 , 120.335
10.6417 , 119.954
13.8672 , 119.599
17.0928 , 119.294
20.3183 , 119.014
23.5438 , 118.761
26.7694 , 118.557
29.9949 , 118.38
30.4013 , 118.354
30.5791 , 118.329
30.7568 , 118.329
30.9346 , 118.303
31.2902 , 118.303
31.468 , 118.278
31.6458 , 118.278
31.7982 , 118.253
31.9759 , 118.253
32.3823 , 118.227
34.084 , 118.126
35.8872 , 118.024
37.7667 , 117.922
39.7477 , 117.846
41.8557 , 117.745
44.0399 , 117.643
46.3003 , 117.541
48.6877 , 117.465

Polygon 11:

53.5895 , 119.37
55.215 , 119.395
56.8151 , 119.472
58.4405 , 119.599
60.066 , 119.802
61.6915 , 120.081
63.3169 , 120.437
64.9424 , 120.843
66.5679 , 121.3
66.9488 , 121.402
69.2092 , 122.011
71.4443 , 122.672
73.6793 , 123.358
75.9143 , 124.069
78.1239 , 124.856
80.3335 , 125.669
82.5431 , 126.507
84.7273 , 127.421
86.9116 , 128.361
87.2163 , 128.539
87.8005 , 128.716
87.2163 , 130.393
87.0893 , 130.774
86.5814 , 132.272
86.4544 , 132.628
84.4734 , 138.52
83.8892 , 138.342
83.6606 , 138.393
64.8154 , 140.907
63.7487 , 140.984
62.7074 , 140.984
61.6915 , 140.857
60.7263 , 140.653
59.7866 , 140.323
58.8977 , 139.892
58.0342 , 139.358
57.2214 , 138.723
56.4341 , 137.987
56.1801 , 137.758
55.2658 , 136.844
55.0626 , 136.615
54.8848 , 136.387
54.6816 , 136.183
54.5039 , 135.955
54.2753 , 135.65
53.1324 , 134.126
52.2181 , 132.526
51.5069 , 130.875
51.0244 , 129.174
50.7196 , 127.396
50.6434 , 125.567
50.7704 , 123.662
51.1259 , 121.707
51.6847 , 119.675
51.9133 , 119.675
51.9895 , 119.446

Polygon 12:

92.0165 , 119.345
93.5912 , 119.345
95.1913 , 119.421
96.7405 , 119.548
98.2898 , 119.751
99.8391 , 120.005
101.363 , 120.335
102.861 , 120.742
104.36 , 121.199
104.741 , 121.3
106.976 , 121.91
109.16 , 122.545
111.268 , 123.205
113.351 , 123.916
115.357 , 124.627
117.313 , 125.415
119.243 , 126.202
121.097 , 127.04
122.9 , 127.904
123.205 , 128.082
123.789 , 128.285
123.231 , 129.885
123.104 , 130.266
122.621 , 131.663
122.494 , 132.044
120.589 , 137.682
120.03 , 137.479
119.776 , 137.53
102.43 , 140.907
101.363 , 140.984
100.322 , 140.984
99.3057 , 140.857
98.3406 , 140.628
97.4009 , 140.298
96.512 , 139.866
95.6484 , 139.333
94.8357 , 138.698
94.0484 , 137.961
93.7944 , 137.733
93.5658 , 137.479
93.1086 , 137.022
92.9055 , 136.793
92.7023 , 136.59
92.4991 , 136.361
92.3213 , 136.133
92.1435 , 135.929
91.8895 , 135.599
90.7974 , 134.075
89.9339 , 132.501
89.2736 , 130.85
88.8418 , 129.148
88.6386 , 127.37
88.664 , 125.542
88.918 , 123.637
89.4006 , 121.681
90.0863 , 119.649
90.3403 , 119.649
90.4165 , 119.395

Polygon 13:

70.8855 , 100.576
71.9268 , 100.626
72.9427 , 100.753
73.9078 , 100.982
74.8222 , 101.312
75.7365 , 101.744
76.5746 , 102.277
77.4128 , 102.938
78.1747 , 103.674
78.6826 , 104.131
79.1398 , 104.588
79.5462 , 105.046
79.724 , 105.274
79.9017 , 105.477
80.0795 , 105.706
80.3335 , 106.011
81.4002 , 107.535
82.2384 , 109.135
82.8225 , 110.786
83.1781 , 112.487
83.2797 , 114.265
83.1527 , 116.094
82.7971 , 117.999
82.1876 , 119.954
81.3494 , 121.986
81.0954 , 121.986
81.0193 , 122.215
79.4192 , 122.291
77.8699 , 122.291
76.3206 , 122.215
74.8222 , 122.088
73.3491 , 121.884
71.876 , 121.63
70.4537 , 121.326
69.0822 , 120.945
67.7108 , 120.488
67.3298 , 120.386
65.1202 , 119.802
63.0122 , 119.167
61.0057 , 118.532
59.0755 , 117.872
57.2722 , 117.16
55.5452 , 116.449
53.9197 , 115.687
52.3958 , 114.925
50.9736 , 114.113
50.6434 , 113.935
50.0846 , 113.757
50.5926 , 112.233
50.745 , 111.852
51.2021 , 110.532
51.3291 , 110.151
53.1578 , 104.766
53.7165 , 104.944
53.9705 , 104.919
69.8188 , 100.652

Polygon 14:

54.3515 , 78.5049
55.9515 , 78.5049
57.577 , 78.5811
59.2025 , 78.7334
60.8025 , 78.9366
62.428 , 79.216
64.0535 , 79.5462
65.6789 , 79.9525
67.3298 , 80.4351
67.7108 , 80.5367
69.9458 , 81.1462
72.2062 , 81.7812
74.4158 , 82.4669
76.6508 , 83.2035
78.8604 , 83.9654
81.0701 , 84.7781
83.2797 , 85.6417
85.4639 , 86.5306
87.6481 , 87.4703
87.9783 , 87.6481
88.537 , 87.8513
87.9783 , 89.5275
87.8513 , 89.9085
87.3179 , 91.3816
87.1909 , 91.7626
85.2099 , 97.6549
84.6512 , 97.4517
84.3972 , 97.5025
65.5519 , 100.017
64.4852 , 100.118
63.4439 , 100.093
62.428 , 99.9915
61.4629 , 99.7629
60.5486 , 99.4581
59.6596 , 99.0264
58.7961 , 98.493
57.9834 , 97.8581
57.196 , 97.1215
56.9421 , 96.8929
56.2309 , 96.1818
56.0277 , 95.9532
55.8246 , 95.75
55.6214 , 95.5214
55.4436 , 95.2929
55.2658 , 95.0897
55.0118 , 94.7595
53.8689 , 93.2356
52.9546 , 91.661
52.2688 , 90.0101
51.7609 , 88.3084
51.4815 , 86.5306
51.4053 , 84.7019
51.5323 , 82.7971
51.8625 , 80.8415
52.4212 , 78.8096
52.6752 , 78.8096
52.726 , 78.5557

Polygon 15:

102.15 , 76.5746
103.217 , 76.6762
122.062 , 79.1906
122.316 , 79.2414
122.875 , 79.0382
124.856 , 84.9305
124.983 , 85.3115
125.516 , 86.7846
125.643 , 87.1655
126.202 , 88.8418
125.643 , 89.045
125.313 , 89.2228
123.129 , 90.1625
120.945 , 91.0514
118.735 , 91.9149
116.525 , 92.7277
114.316 , 93.4896
112.081 , 94.2262
109.871 , 94.9119
107.611 , 95.5468
105.376 , 96.1564
104.995 , 96.258
103.344 , 96.7405
101.719 , 97.1469
100.093 , 97.4771
98.4676 , 97.7565
96.8421 , 97.9596
95.2421 , 98.112
93.6166 , 98.1882
92.0165 , 98.1882
90.3911 , 98.1374
90.3403 , 97.8835
90.0863 , 97.8835
89.5275 , 95.8516
89.1974 , 93.896
89.045 , 92.0165
89.1466 , 90.1625
89.426 , 88.3846
89.9085 , 86.683
90.6197 , 85.0321
91.534 , 83.4575
92.6769 , 81.9336
92.9309 , 81.6034
93.1086 , 81.4002
93.2864 , 81.1716
93.4896 , 80.9431
93.6928 , 80.7399
93.896 , 80.5113
94.6071 , 79.8002
94.8611 , 79.5716
95.623 , 78.835
96.4612 , 78.2001
97.2993 , 77.6667
98.2136 , 77.235
99.1279 , 76.9302
100.093 , 76.7016
101.109 , 76.6

Polygon 16:

72.9173 , 60.066
73.9586 , 60.0914
74.9492 , 60.2184
75.9143 , 60.4724
76.854 , 60.8025
77.7429 , 61.2343
78.6065 , 61.7677
79.4192 , 62.4026
80.2065 , 63.1391
81.1462 , 64.0789
81.5526 , 64.536
81.7558 , 64.7392
81.9336 , 64.9678
82.086 , 65.1964
82.3399 , 65.5011
83.4067 , 67.025
84.2448 , 68.5997
84.8289 , 70.2506
85.1845 , 71.9776
85.3115 , 73.7301
85.1845 , 75.5841
84.8035 , 77.4636
84.194 , 79.4446
83.3559 , 81.451
83.1019 , 81.451
83.0511 , 81.705
81.451 , 81.7558
79.8763 , 81.7558
78.3525 , 81.705
76.8286 , 81.578
75.3555 , 81.3748
73.9078 , 81.1208
72.4856 , 80.8161
71.0887 , 80.4351
69.7172 , 79.9779
69.3362 , 79.8763
67.1266 , 79.2668
65.0186 , 78.6573
63.0122 , 78.0223
61.1073 , 77.3366
59.2787 , 76.6508
57.577 , 75.9143
55.9515 , 75.1777
54.4277 , 74.3904
52.98 , 73.6031
52.6752 , 73.4253
52.0911 , 73.2221
52.6244 , 71.6982
52.7514 , 71.3173
53.2086 , 69.9966
53.3356 , 69.6156
55.1642 , 64.2313
55.723 , 64.4344
55.9769 , 64.3836
71.8252 , 60.1422

Polygon 17:

93.4896 , 42.3637
94.5563 , 42.4399
111.903 , 45.8178
112.157 , 45.8686
112.716 , 45.6654
114.621 , 51.3037
114.748 , 51.6847
115.23 , 53.107
115.357 , 53.4625
115.916 , 55.088
115.357 , 55.2658
115.027 , 55.4436
113.224 , 56.3071
111.37 , 57.1452
109.439 , 57.958
107.484 , 58.7199
105.477 , 59.4564
103.395 , 60.1422
101.287 , 60.8279
99.1025 , 61.4629
96.8675 , 62.047
96.4866 , 62.1486
94.9881 , 62.6058
93.4896 , 63.0122
91.9657 , 63.3423
90.4165 , 63.5963
88.8672 , 63.7995
87.3179 , 63.9519
85.7433 , 64.0027
84.1432 , 64.0281
82.5431 , 63.9519
82.4669 , 63.7233
82.213 , 63.7233
81.5272 , 61.6915
81.0447 , 59.7358
80.8161 , 57.831
80.7907 , 56.0023
80.9685 , 54.2245
81.4002 , 52.4974
82.0606 , 50.872
82.9241 , 49.2719
84.0162 , 47.748
84.2702 , 47.4432
84.448 , 47.2147
84.6258 , 47.0115
85.2353 , 46.3257
85.6925 , 45.8686
85.9464 , 45.64
86.175 , 45.4114
86.9624 , 44.6495
87.8005 , 44.0145
88.6386 , 43.4812
89.5529 , 43.0494
90.4673 , 42.7446
91.4324 , 42.5161
92.4483 , 42.3891

Polygon 18:

60.1168 , 29.5885
61.1581 , 29.5885
62.2502 , 29.6901
81.0701 , 32.2045
81.324 , 32.2299
81.8828 , 32.0521
83.8892 , 37.9444
84.0162 , 38.3254
84.5242 , 39.7985
84.6512 , 40.1795
85.2353 , 41.8557
84.6512 , 42.0589
84.3464 , 42.2367
82.1622 , 43.1764
79.9525 , 44.0653
77.7683 , 44.9289
75.5587 , 45.7416
73.3237 , 46.5035
71.1141 , 47.2401
68.8791 , 47.9258
66.6441 , 48.5608
64.3836 , 49.1703
64.0027 , 49.2719
62.3772 , 49.7545
60.7517 , 50.1608
59.1263 , 50.491
57.5008 , 50.7704
55.8753 , 50.9736
54.2499 , 51.1259
52.6498 , 51.2021
51.0244 , 51.2021
49.4243 , 51.1259
49.3481 , 50.8974
49.0941 , 50.8974
48.5608 , 48.8655
48.2052 , 46.9099
48.0782 , 45.0051
48.1544 , 43.1764
48.4338 , 41.3986
48.9417 , 39.6969
49.6529 , 38.046
50.5672 , 36.4714
51.6847 , 34.9475
51.9387 , 34.6173
52.1165 , 34.4141
52.2942 , 34.1856
52.7006 , 33.7284
52.9292 , 33.4998
53.1324 , 33.2712
53.3864 , 33.0426
53.6149 , 32.8141
53.8689 , 32.5855
54.6562 , 31.8489
55.469 , 31.214
56.3325 , 30.6806
57.2214 , 30.2489
58.1612 , 29.9187
59.1263 , 29.7155

Polygon 19:

93.5404 , 15.1371
93.5658 , 15.8483
93.2356 , 16.5848
92.9309 , 17.3214
92.6769 , 18.0071
92.4483 , 18.6674
92.2451 , 19.3278
92.0927 , 19.9373
91.9657 , 20.5469
91.8895 , 21.131
91.8388 , 21.6898
91.8134 , 22.3247
91.788 , 22.9851
91.7626 , 23.6708
91.7118 , 24.3566
91.6356 , 25.0931
91.5594 , 25.8043
91.4832 , 26.5662
91.3816 , 27.3281
91.28 , 28.1155
74.0348 , 28.0139
73.6285 , 28.0139
71.5458 , 28.0139
68.5743 , 28.0901
65.7043 , 28.1155
62.9868 , 28.1409
60.3962 , 28.1663
57.9326 , 28.1409
55.596 , 28.1155
53.361 , 28.0647
51.2783 , 28.0139
49.2973 , 27.9377
48.51 , 28.0901
47.9766 , 26.4646
47.8496 , 26.0836
47.3671 , 24.6359
47.2401 , 24.255
45.386 , 18.6167
45.9448 , 18.4389
46.3511 , 18.3627
49.1957 , 17.9817
52.0911 , 17.6261
55.0372 , 17.296
58.0088 , 16.9912
61.0311 , 16.7118
64.1043 , 16.4578
67.2282 , 16.2292
70.3775 , 16.0261
73.5777 , 15.8483
73.984 , 15.8229
74.1618 , 15.8229
74.3142 , 15.7975
74.6698 , 15.7975
74.8222 , 15.7721
75.3047 , 15.7721
75.4571 , 15.7467
75.8635 , 15.7467
77.5397 , 15.6451
79.2922 , 15.5435
81.1208 , 15.4673
83.0003 , 15.3911
84.9813 , 15.3149
87.0132 , 15.2641
89.1212 , 15.2133
91.3054 , 15.1625

Polygon 20:

98.5946 , 0.431764
98.62 , 1.1683
98.2898 , 1.90484
97.985 , 2.61598
97.7311 , 3.30173
97.5025 , 3.98747
97.2993 , 4.62242
97.1469 , 5.25736
97.0199 , 5.86691
96.9437 , 6.42567
96.8929 , 6.98442
96.8421 , 8.30511
96.8167 , 8.99085
96.7659 , 9.67659
96.6897 , 10.3877
96.6136 , 11.1243
96.5374 , 11.8862
96.4358 , 12.6481
96.3342 , 13.4355
79.089 , 13.3085
78.6826 , 13.3339
76.6 , 13.3339
73.6285 , 13.3847
70.7839 , 13.4355
68.0409 , 13.4609
62.9868 , 13.4609
60.6502 , 13.4355
58.4151 , 13.3847
56.3325 , 13.3339
54.3515 , 13.2577
53.5895 , 13.4101
53.0308 , 11.7592
52.9038 , 11.3783
52.4212 , 9.95597
52.3196 , 9.575
50.4402 , 3.93667
50.999 , 3.73349
51.4053 , 3.68269
54.2499 , 3.30173
57.1452 , 2.94615
60.0914 , 2.61598
63.0629 , 2.31121
66.1107 , 2.03183
69.1584 , 1.77785
72.2824 , 1.54927
75.4317 , 1.34609
78.6319 , 1.1683
79.0382 , 1.1429
79.216 , 1.11751
79.5462 , 1.11751
79.724 , 1.09211
80.2065 , 1.09211
80.3589 , 1.06671
80.5113 , 1.06671
80.9177 , 1.04131
82.5939 , 0.939722
84.3464 , 0.863528
86.175 , 0.787334
88.0799 , 0.711141
90.0355 , 0.634947
92.0673 , 0.584151
94.1754 , 0.533356
96.3596 , 0.48256

Polygon 21:

109.008 , 4.49543
110.074 , 4.57162
127.421 , 7.94954
127.675 , 8.00033
128.234 , 7.79715
130.139 , 13.4355
130.266 , 13.8164
130.774 , 15.2133
130.901 , 15.5943
131.434 , 17.1944
130.875 , 17.3976
130.545 , 17.5753
128.742 , 18.4389
126.888 , 19.277
124.983 , 20.0897
123.002 , 20.8517
120.996 , 21.5882
118.913 , 22.2739
116.805 , 22.9597
114.621 , 23.5946
112.411 , 24.1788
112.005 , 24.2804
110.506 , 24.7375
109.008 , 25.1439
107.484 , 25.4741
105.935 , 25.7281
104.385 , 25.9312
102.836 , 26.0836
101.261 , 26.1344
99.6613 , 26.1344
98.0612 , 26.0836
98.0104 , 25.855
97.7565 , 25.8296
97.0453 , 23.8232
96.5882 , 21.8676
96.3342 , 19.9627
96.3088 , 18.1341
96.512 , 16.3562
96.9183 , 14.6292
97.5787 , 13.0037
98.4422 , 11.4037
99.5597 , 9.87978
99.7883 , 9.575
99.9661 , 9.34642
100.144 , 9.14324
100.753 , 8.4575
101.211 , 8.00033
101.719 , 7.54317
102.506 , 6.78124
103.319 , 6.14629
104.182 , 5.61293
105.071 , 5.18117
105.985 , 4.851
106.95 , 4.64781
107.966 , 4.52082

Polygon 22:

124.043 , 23.1121
125.618 , 23.1121
127.218 , 23.1629
127.269 , 23.4169
127.523 , 23.4169
128.361 , 25.4233
128.97 , 27.4043
129.351 , 29.2838
129.478 , 31.1378
129.351 , 32.8903
128.996 , 34.6173
128.412 , 36.2682
127.574 , 37.8428
126.507 , 39.3667
126.278 , 39.6715
126.101 , 39.9001
125.923 , 40.1033
125.313 , 40.789
124.373 , 41.7287
123.586 , 42.4653
122.773 , 43.1002
121.91 , 43.6336
121.021 , 44.0653
120.081 , 44.3955
119.116 , 44.6241
118.126 , 44.7765
117.084 , 44.8019
115.992 , 44.7257
100.144 , 40.4842
99.9153 , 40.4334
99.3311 , 40.6366
97.5025 , 35.2269
97.3755 , 34.8713
96.9183 , 33.5252
96.7913 , 33.1696
96.2834 , 31.6458
96.8421 , 31.4426
97.1469 , 31.2648
98.5946 , 30.4775
100.118 , 29.6901
101.744 , 28.9536
103.446 , 28.2171
105.274 , 27.5313
107.179 , 26.8456
109.186 , 26.2106
111.294 , 25.5757
113.503 , 24.9915
113.884 , 24.8899
115.256 , 24.4328
116.652 , 24.0518
118.075 , 23.747
119.522 , 23.493
120.996 , 23.2899
122.519 , 23.1629

Polygon 23:

126.685 , 55.215
128.259 , 55.215
129.859 , 55.2658
129.91 , 55.5198
130.164 , 55.5198
131.002 , 57.5262
131.612 , 59.5072
131.993 , 61.3867
132.12 , 63.2407
131.993 , 64.9932
131.637 , 66.7202
131.053 , 68.3711
130.215 , 69.9458
129.148 , 71.4696
128.92 , 71.7744
128.564 , 72.2316
128.361 , 72.4348
127.955 , 72.8919
127.015 , 73.8317
126.227 , 74.5682
125.415 , 75.2031
124.551 , 75.7365
123.662 , 76.1683
122.723 , 76.4984
121.757 , 76.727
120.767 , 76.8794
119.726 , 76.9048
118.634 , 76.8286
102.785 , 72.5872
102.557 , 72.5364
101.973 , 72.7395
100.144 , 67.3298
100.017 , 66.9742
99.5597 , 65.6281
99.4327 , 65.2726
98.9248 , 63.7487
99.4835 , 63.5455
99.7883 , 63.3677
101.236 , 62.5804
102.76 , 61.7931
104.385 , 61.0565
106.087 , 60.32
107.916 , 59.6342
109.82 , 58.9485
111.827 , 58.3135
113.935 , 57.6786
116.145 , 57.0944
116.525 , 56.9929
117.897 , 56.5357
119.294 , 56.1547
120.716 , 55.85
122.164 , 55.596
123.637 , 55.3928
125.161 , 55.2658

Polygon 24:

98.7978 , 98.8994
100.372 , 98.8994
101.947 , 98.9756
103.522 , 99.1025
105.071 , 99.3057
106.62 , 99.5597
108.144 , 99.8899
109.643 , 100.296
111.141 , 100.753
111.522 , 100.855
113.757 , 101.465
115.941 , 102.099
118.049 , 102.76
120.132 , 103.471
122.138 , 104.182
124.094 , 104.969
126.024 , 105.757
127.878 , 106.595
129.682 , 107.458
129.986 , 107.636
130.571 , 107.839
130.012 , 109.439
129.885 , 109.82
129.402 , 111.217
129.275 , 111.598
127.37 , 117.237
126.786 , 117.033
126.558 , 117.084
109.211 , 120.462
108.144 , 120.538
107.103 , 120.538
106.087 , 120.411
105.122 , 120.183
104.182 , 119.853
103.293 , 119.421
102.43 , 118.888
101.617 , 118.253
100.83 , 117.516
100.576 , 117.287
100.347 , 117.033
99.8899 , 116.576
99.6867 , 116.348
99.4835 , 116.145
99.2803 , 115.916
99.1025 , 115.687
98.9248 , 115.484
98.6708 , 115.154
97.5787 , 113.63
96.7151 , 112.055
96.0548 , 110.405
95.623 , 108.703
95.4199 , 106.925
95.4453 , 105.096
95.6992 , 103.192
96.1818 , 101.236
96.8675 , 99.2041
97.1215 , 99.2041
97.1977 , 98.9502

References

[1] Adamowicz M, Albano A. A solution of the rectangular cutting-stock problem. IEEE Trans. Syst., Man and Cybernetics, SMC-6, 1976, pp 302-310.

[2] Adamowicz M, Albano A. Nesting two-dimensional shapes in rectangular modules. Computer Aided Design, v8, 1976, pp 27-33.

[3] Albano A, Orsini R. A heuristic solution of the rectangular cutting stock problem. The Computer Journal, v23, 1980, pp 338-343.

[4] Albano A, Sappupo G. Optimal allocation of two-dimensional irregular shapes using heuristic search methods. IEEE Trans. Syst., Man and Cybernetics, SMC-10, 1980, pp 242-248.

[5] Albano A, A method to improve two-dimensional layout. Computer Aided Design, v9, 1977, pp 48-52.

[6] Beasley J.E, An exact two-dimensional non-guillotine cutting tree search procedure. Operations Research, v33, 1985, pp 49-64.

[7] Bennell J.A, Dowsland K.A, and Dowsland W.B. The irregular cutting-stock problem – a new procedure for deriving the no-fit polygon. Computers and Operations Research, v28, 2001, pp 271-287.

[8] Blazewicz J, Hawryluk P, Walkowiak R. Using tabu search approach for solving the two-dimensional irregular cutting problem. Tabu Search (eds. Glover F, Laguna M, Taillard E, Werra D), v41 of Annals of Operations, Baltzer J.C.AG.\, 1993.

[9] Blazewicz J, Walkowiak R. A local search approach for two-dimensional irregular cutting. OR Spektrum, v17, 1995, pp 93-98.

[10] Burke K.E, Kendall G. 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.

[11] Burke K.E, Kendall G. 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.

[12] Burke K.E, Kendall G. Applying evolutionary algorithms and the no fit polygon to the nesting problem. Proceedings of IC-AI '99 : The 1999 International Conference on Artificial Intelligence, Las Vegas, Nevada, USA, 28 June – 1 July 1999, pp 51-57.

[13] Burke K.E, Kendall G. Applying ant algorithms and the no fit polygon to the nesting problem. Proceedings of the 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.

[14] Cerny V, Thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. Journal of Optimization Theory and Applications, v45(1), January 1985, pp 41-51.

[15] Chen C.S, Lee S.M, Shen Q.S. An analytical model for the container loading problem. European Journal of Operational Research, v80, 1995, pp 68-76.

[16] Cheng C.H, Feiring B.R, Cheng T.C.E. The cutting stock problem – A survey. International Journal of Production Economics, v36, 1994, pp 291-305.

[17] Coffman E.G, Lueker G.S, Rinnooy A.H.G. Asymptotic methods in the probabilistic analyisis of sequencing and packing heuristics. Management Science, v34, 1988, pp 266-290.

[18] Cunninghame-Green R. Geometry, shoemaking and the milk tray problem. New Scientist 12 August 1989; 1677, pp 50-53.

[19] Dori D, Ben-Bassat M. Efficient nesting of congruent convex figures. Communications of the ACM, v27, 1984, pp 228-235.

[20] Dowsland K.A, Dowsland W.B. Solution approaches to irregular nesting problems. European Journal of Operational Research, v84 ,1995, pp 506-521.

[21] Dowsland K.A, Dowsland W.B, Bennell J.A. Jostling for position: local improvement for irregular cutting patterns. Journal of the Operational Research Society, v49(6), 1998, pp 647-658.

[22] Dyckhoff H. A typology of cutting and packing problems. European Journal of Operational Research, v44, 1990, pp 145-159.

[23] Feng E, Wang Xilu, Wang Xiumei, Teng H. An algorithm of global optimisation for solving layout problems. European Journal of Operational Research, v114, 1999, pp 430-436.

[24] Fernandez J, Canovas L, Pelegrin B. Algorithms for the decomposition of a polygon into convex polygons. European Journal of Operational Research, v121, 2000, pp 330-342.

[25] Foulds L.R. Graph Theory Applications. Springer-Verlag. 1992, pg 152.

[26] Fujita K, Akagi S, Hirokawa N. Hybrid approach for optimal nesting using a genetic algorithm and a local minimization algorithm. Advances in Design Automation, v1, ASME 1993, pp 477-484.

[27] Ghosh P.K. An algebra of polygons through the notion of negative shapes. CVGIP: Image Understanding, v54 (1), 1991, pp 119-144.

[28] Gilmore P.C, Gomory R.E. Multistage cutting stock problems of two and more dimensions. Operations Research, v13, 1965, pp 94-120.

[29] Gilmore P.C, and Gomory R.E. The theory and computation of knapsack functions. Operations Research, v14, 1966, pp 1045-1074.

[30] Grinde R.B, Cavalier T.M. Containment of a single polygon using mathematical programming. European Journal of Operations Research, v92, 1996, pp 368-386.

[31] Grinde R.B, Cavalier T.M. A new algorithm for the two-polygon containment problem. Computers and Operations Research, v24(3), 1997, pp 231-251.

[32] Han G.C, Na S.J. Two-stage approach for nesting in two-dimensional cutting problems using neural network and simulated annealing. Proc Instn Mech Engrs, v210, 1996, pp 509-519.

[33] Heckmann R, Lengauer T. A simulated annealing approach to the nesting problem in the textile manufacturing industry. Annals of Operations Research, v57, 1995, pp 103-133.

[34] Heckmann R, Lengauer T. Computing closely matching upper and lower bounds on textile nesting problems. European Journal of Operations Research, v108, 1998, pp 473-489.

[35] Hershberger J. Finding the upper envelope of n line segments in O(nlogn) time. Information Processing Letters, v33, 1989, pp 169-174.

[36] Hong L, Guangzhou Z, Zongkai L. A system of optimising nesting with analogical learning mechanism. Computers and Industrial Engineering, v32(4), 1997, pp 713-725.

[37] Ismail H.S, Hon K.K.B. New approach for the nesting of two-dimensional shapes for press tool design. International Journal of Production Research, v30(4), 1992, pp 825-837.

[38] Ismail H.S, Hon K.K.B. The nesting of two-dimensional shapes using genetic algorithms. Proc Instn Mech Engrs, v209, 1995, pp 115-124.

[39] Jain P, Fenyes P, Richter R. Optimal blank nesting using simulated annealing. Journal of Mechanical Design (Transactions of the ASME), v114, March 1992, pp 160-165.

[40] Karp R.M. Reducibility among combinatorial problems. Complexity of Computer Computations, Miller R.E, Thatcher J.W (eds.), Plenum Press, New York, 1972, pp 85-103.

[41] Kendall, G. Applying meta-heuristic algorithms to the nesting problem utilising the no fit polygon. PhD thesis, University of Nottingham, 2000.

[42] Lai K.K, Chan J.W.M. Developing a simulated annealing algorithm for the cutting stock problem. Computers and Industrial Engineering, v32(1), 1997, pp 115-127.

[43] Lamousin H.J, Waggenspack W.N, Dobson G.T. Nesting of complex 2-d parts within irregular boundaries. Journal of Manufacturing Science and Engineering, v118, November 1996, pp 615-622.

[44] Li Z, Milenkovic V. Compaction and separation algorithms for non-convex polygons and their applications. European Journal of Operational Research, v84, 1995, pp 539-561.

[45] Lo Valvo, E. A new simple and quick algorithm for the calculation of "no fit polygon" of generic polygons. Working paper.

[46] Mahadevan, A. Optimisation in computer-aided pattern packing. PhD thesis, North Carolina State University, 1984.

[47] Milenkovic V.J, Daniels K. Translational polygon containment and minimal enclosure using mathematical programming. International Transactions in Operations Research, v6, 1999, pp 525-554.

[48] Milenkovic V, Daniels K, Li Z. Automatic marker making. Proceedings of the 3rd Canadian Conference for Computational Geometry, 1991, pp 243-246.

[49] O'Rourke, J. Computational Geometry in C. Cambridge University Press, 1998.

[50] Oliveira J.F, Gomes A.M, Ferreira J.S. TOPOS – A new constructive algorithm for nesting problems. Accepted for ORSpektrum, 1998.

[51] Prasad Y.K.D.V. A set of heuristic algorithms for optimal nesting of two-dimensional irregularly shaped sheet-metal blanks. Computers in Industry, v24, 1994, pp 55-70.

[52] Qu W, Sanders J.L. A nesting algorithm for irregular parts and factors affecting trim losses. International Journal of Production Research, v25, 1987, pp 381-397.

[53] Ramkumar G. D. An algorithm to compute the minkowski sum outer-face of two simple polygons. Proceedings of the 12th Annual Symposium on Computational Geometry FCRC 96, 1996.

[54] Scheithauer G, Terno J. Modelling of packing problems. Optimization, v28, 1993, pp 63-84.

[55] Stoyan Y.G, Novozhilova M.V, Kartashov A.V. Mathematical model and method of searching for a local extremum for the non-convex oriented polygons allocation problem. European Journal of Operational Research, v92, 1996, pp 193-210.