UNIVERSITY OF SOUTHAMPTON

FACULTY OF LAW, ARTS AND SOCIAL SCIENCE

SCHOOL OF MANAGEMENT

Solving Two-Dimensional Layout Optimization Problems with Irregular Shapes by Using Meta-Heuristic

By

Kumaran Ramakrishnan

Thesis for the degree of Master of Philosophy in Management Science

December 2008

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF LAW, ARTS AND SOCIAL SCIENCE
SCHOOL OF MANAGEMENT

Master of Philosophy

Solving Two-Dimensional Layout Optimization Problems with Irregular Shapes by Using Meta-Heuristic

by Kumaran Ramakrishnan

The focus of this thesis is developing methodologies for 2-dimensional problems that involve irregular shapes, where the objective is to find an arrangement of the irregular pieces in order to minimise waste material. There are two main approaches popular with researchers; in this research project we will group them into iterative constructive heuristics (ICH) and those heuristics which search over the physical layout (SOL). ICH seeks to generate good layouts by placing pieces on the stock sheet piece by piece according to a placement rule. The orders the pieces are placed is controlled by a search algorithm. SOL works with the physical layout and tries to improve the solution by moving the placement position of pieces. Overlap is often permitted and penalised in the cost function in order to generate new solutions.

Both approaches are competitive and each new publication brings better results with respect to the benchmark data sets. Although this can be credited to better algorithm design, it could also be argued that researchers are getting better at incorporating sophisticated specific features in their algorithms to handle the 17 benchmark data sets. In this research we intend to investigate the two representations of the problem and establish some principles of the strengths and weaknesses of each method with respect to data type. In order to conduct this research the algorithms will be developed using only the basic principles of both approaches and discarding any special features found in the literature. The aim is to deduct from the experimental results an understanding of what solution representation should be applied given the data type, performance requirements and number of pieces.

Contents

Contents

Contents

Contents

List of Figures

List of Tables

Acknowledgements

Acknowledgments

First and foremost I offer my sincerest gratitude to my supervisor, Dr Julia Bennell, who has supported me throughout my thesis with her patience and expert knowledge whilst allowing me the room to work in my own way. One simply could not wish for a better or friendlier supervisor.

My special thanks to the School of Management's Research Secretary, Ms. Jayne Cook for her dedication in assisting me with essential paperwork and other day-to-day related matters.

A big thanks to my loving wife, Rajes, without your love, support, encouragement and sacrifices this Masters degree and this thesis, too, would not have been completed or written. Thank you so much, I love you.

1 Introduction

Cutting and packing problems has attracted the interest of researches for many years. This problem occurs across a great variety of industries for example textile, paper, wood, metal and glass. Many of the industries involved in cutting and packing generate huge volumes of sales to fulfill the accelerating global demand of products made from textile, paper, wood, metal and glass. For example all of us need clothing in our daily lives, the furniture we use may be made from wood or metal and all sorts of household and industrial items will have components cut from various materials.

Cutting and packing problems arise in a wide variety of industries and include many problem types such as cutting lengths of pipe (1-dimensional problem), cutting sheets of glass or loading pallets (2-dimensional) or loading a container (3-dimensional). The focus of this proposal is developing methodologies for 2-dimensional problems that involve irregular shapes, where the objective is to find an arrangement of the irregular pieces in order to minimise waste material. Examples of this problem type include garment manufacturing, leather hide cutting, ship building, and tool manufacturing. Although commercial software exists for generating layouts, the underlying algorithms are generally unsophisticated and can not compete with solutions generated by experts.

Classification of cutting and packing problem type can be found in Dyckhoff (1990) and Waescher et al. (2007). Our research interest in on the variant of packing problem commonly known as 2-dimensional strip packing problem in the literature or as 2-dimensional irregular open dimension problems (2D irregular ODP) as classified by Waescher et al. (2007). We will be using 2D irregular ODP to refer to this problem type throughout this thesis, 2D irregular refers to the dimensions and shapes of the pieces to be packed and OD refers to the stock sheet. An assumption is made that sufficient space is available to pack all the pieces on the stock sheet. For example in the garment manufacturing industry the stock sheet (i.e. rolls of textile) is rectangular in shape, it has a constrained width but unconstrained length so that all the pieces that are required to

manufacture the garment can be conveniently placed on the stock sheet. There are two key solution approaches used to solve the 2D irregular ODP problems; iterative construction heuristics (e.g. Gomes and Oliveira, 2002; Dowsland et al., 2002; Burke et al., 2006), and searching over the layout (Bennell and Dowsland, 2001; Oliveira and Ferreira, 1993; Heckmann and Lengauer, 1995). The focus of this research is not to compete with the best results published in the literature with respect to the benchmark data sets but rather to investigate the two solution approaches and gain some insights into the strengths and weaknesses of each solution approach with respect to the benchmark data. We aim to do this by stripping the special features that are used to enhance performance often embedded in these solution approaches and creating a focus on the basic behaviour of the two key solution approaches. Hence a level playing field for computational results comparison is created.

The next section is an introduction to cutting and packing problems and some of the issues that need to be considered. Subsequently we will discuss the research problem that we will be investigating and how we intend to approach it. In the final section we will set our goals for this project.

1.1 The problem

Bennell and Oliveira, 2008 define 2D irregular ODP to be a problem, *"where more than one piece of irregular shape must be placed in a configuration with the other pieces(s) in order to optimise an objective"*. The 2D irregular ODP is NP-hard (e.g. Egeblad et al., 2003) hence exact methods cannot be used to solve this problem. The geometric properties of the shapes to be packed or cut add another dimension of complexity to this problem. Huge cost savings could be derived from a good quality cutting or packing solution. The reasons given above make this problem a very important research area in academia as well as in industry. Figure 1.1 (generated from author's computational experiment run) is a good example of an irregular packing layout in the garment manufacturing industry; it shows the cutting pattern for pairs of trousers.

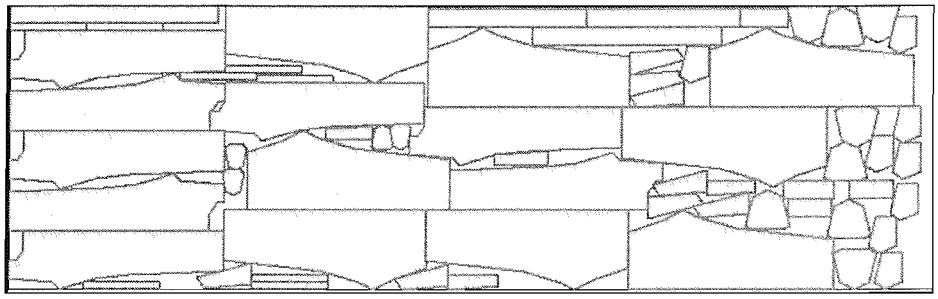


Figure 1.1: An example layout for pairs of trousers

Referring to Figure 1.1 there are in total 52 pieces that needs to be packed onto the stock sheet and the objective is to pack them as close together as possible in order to minimize the wastage of material. In the literature the quality of packing layouts produced is measured in terms of the shortest strip length or highest area utilization. The cutting and packing layout is constrained by shape of the stock sheet. For example in the textile industry, rolls of textile will have a fixed width and we normally assume sufficient length of material is available to pack all the required pieces, although in reality this may not be the case. In the literature this type of problem is referred to as a strip packing problem. Whereas in the leather industry the packing layout is restricted by the size and shape of the leather hide, hence more than one may be required to pack all the pieces. The problem we are interested in is the strip packing problem variant.

One of the main obstacles for researchers working with 2D irregular ODP is implementing the computational geometric tools to prevent pieces from overlapping one another as they are placed on the layout. In regular shapes like rectangle or circles the overlap calculation can be trivial but detecting overlap between the irregular shapes requires non-trivial computational geometric techniques. The next chapter will cover the details of the overlap detection techniques. Once the overlap detection tool is in place we can investigate the optimization methodologies. There are two main solution approaches popular with researchers. In this research we will group them into iterative constructive heuristics (ICH) and those which search over the physical layout (SOL). ICH applies a placement policy to pieces in a predetermined order of pieces. The placement policy

seeks to find good positions for placing a piece on the layout. The predetermined order can be generated using a ranking criteria or using local search. This approach aims to produce a locally optimum ordering of pieces that would lead to an optimized layout. Overlap is not permitted while placing the pieces on the layout. SOL starts with an initial layout and seeks to improve this layout by moving pieces around within the layout. Usually overlap is permitted when placing the pieces on the layout but it is penalized with a cost function. The aim is for the final optimized layout to be free from overlaps. Due to the computational intensiveness of the solution methods many researches embed enhancements and special features in their algorithm to improve the solution quality and the computational speed. Further, it is common for researches to add features to their implementations to exploit certain problem specific properties. As a result it is difficult to judge which representation of local search method performs best.

1.2 Aims and objectives

The purpose of this research is to investigate the effectiveness of the ICH (e.g. Gomes and Oliveira, 2002; Dowsland et al., 2002; Burke et al., 2006), and SOL (Bennell and Dowsland, 2001; Oliveira and Ferreira, 1993; Heckmann and Lengauer, 1995) from a neutral standpoint. Both approaches are competitive and each new publication brings better results with respect to the benchmark data sets. Although this can be credited to better algorithm design, it could also be argued that researchers are getting better at incorporating sophisticated specific features in their algorithms to handle the 17 benchmark data sets available from the EURO Special Interest Group on Cutting and Packing (ESICUP) website.

In this research we intend to investigate these two key solution approaches of the problem (ICH and SOL) and establish some principles of the strengths and weaknesses of each method with respect to data type. In order to conduct this research the algorithms will be developed using only the basic principles of both approaches and discarding any special features found in the literature. The aim is to deduct from the experimental results an

understanding of what solution representation should be applied given the data type, performance requirements and number of pieces.

1.3 Structure of Thesis

This thesis will be divided into literature review, methodology, implementation, experimental results and conclusion chapters. In the literature review we will discuss the geometric aspects of irregular packing problem and will discuss some important published work from both solution approaches. In our methodology chapter we will discuss our strategy for algorithm design for ICH and SOL. In our implementation chapter we will be setting our test parameters for computational experiments and in the results chapter we will discuss the findings from our computational runs and deduct a conclusion based on our empirical results.

1.4 Summary

In this chapter we introduced the problem we will be investigating, discussed the aims and objectives of our research project and presented the outline for our thesis. In the next chapter we will review the important published work that use ICH and SOL and also discuss the geometric aspect of the irregular packing problems.

2 Literature Review

Our literature survey will concentrate on 2D irregular ODP as this is the area of our research focus. Although there exists a substantial amount of literature in cutting and packing, most concentrates on the regular or rectangular packing problem (Dowsland and Dowsland, 1992; Dyckhoff, 1992). Dowsland and Dowsland (1995) have written a survey that focuses exclusively on 2D ODP involving irregular shapes. A more recent survey was written by Lodi et al. (2002), (2003). The earliest work on 2D irregular ODP was done by Art (1966). However, it was only in the late 90's 2D irregular ODP started getting more attention from researches. Researchers have noted that the complex computational geometry involved in such problems has stifled academic research in this area (Bennell and Oliveira, 2008). Some of the research work was aimed at solving specific problems from the packing industries. Adamowicz and Albano (1976) laid the foundation for solving packing problem in textile industry which was followed by Heckmann and Lengauer (1995). Heistermann and Langauer (1995) produced a specialized algorithm for the leather industry. Prasad et al. (1991), (1995); Nye (2001) and Jain (1992) focused on optimal pattern nesting of metal blanks. Reference to algorithms applied to solve 2D irregular ODP problems can be found in Hopper and Turton (2001) and in the introductory section of Bennell and Dowsland (1999).

In the two key solution approaches used to solve the 2D irregular ODP problems; iterative construction heuristics (e.g. Gomes and Oliveira, 2002; Dowsland et al., 2002; Burke et al., 2006), and searching over the layout (Bennell and Dowsland, 2001; Oliveira and Ferreira, 1993; Heckmann and Lengauer, 1995). Iterative constructive heuristics attempt to generate a good layout based on a selection and placement rule to arrange the pieces on the layout. The selection rule selects the next piece to be packed from the pool of available pieces and the placement rule identifies the best position for this piece on the layout. In these solution approaches the order in which the pieces are selected to be packed has a direct impact on the solution quality thus local search techniques are normally used to continuously improve the solution, by reordering the pieces. In contrast, searching over the layout starts from an initial layout with pieces already arranged on the

stock sheet and seeks to design a better layout by moving these pieces around within the stock sheet. An important difference in this approach is that pieces are allowed to overlap one another on the layout. This is done to allow greater freedom for positioning pieces anywhere on the layout. A penalty value is then assigned to each piece in an overlap situation. This penalty value is used in a cost function which will guide the search for a better placement position on the layout with the objective of removing all overlaps within the layout. The final improved layout will be free from any overlaps; otherwise we have an infeasible solution.

In section 2.3 and 2.4 we will discuss the literature published on both solution approaches in more detail. Although this will not be a complete survey, it will cover most of the best known methods and solutions published so far. We will discuss the variants of the local search techniques and the geometric techniques to gain a better understanding of the underlying foundations used to solve 2D irregular ODP. Once we have covered the geometric and search techniques, understanding the concepts used in the literature survey in section 2.3 and 2.4 will become much easier.

2.1 Search Methods

2.1.1 Basic Local Search

A local search is an iterative search where the algorithm moves from one solution to another based on a predefined move criteria. This move criteria explores the solution space in a systematic manner to reach a better solution whilst escaping local minima. It is common practice to define the solution space in terms of a neighbourhood structure when using these search methods. A move is usually defined as the change made to a current solution that will result in another solution. All solutions that can be reached by a move are called neighbours. For example, consider an initial order list of 5 pieces sorted by their decreasing area size. The initial solution is the sequence in which these pieces will be packed onto the layout. A neighbour can be generated by swapping two pieces in sequence, e.g. if the initial sequence is (1,2,3,4,5), a neighbour could be (2,1,3,4,5). The

neigbourhood size is the maximum number of neighbours that could be generated in this way, for this example the resulting neighbourhood size would be 4, and the respective neighbours are (2,1,3,4,5), (1,3,2,4,5) and (1,2,4,3,5) and (1,2,3,5,4). Depending on the evaluation criteria, one of the four neighbours, usually the one that improves on the initial solution will be selected as the current best solution. The search then continues by using the current best solution to generate a new set of neighbours, evaluate these neighbours and choose the best one as the current best solution. Every time a current best solution is chosen it will be checked against the previous best solution and the better one will be stored in the memory as the best solution which will be returned when the search algorithm terminates. This search cycle is repeated again and again until the stopping criteria predefined in the search algorithm is reached. Generally there are two types of search strategies; 1) those that attempt to visit all the solutions within a neighbourhood and select the best, 2) those that select a neighbouring solution randomly and stop the search upon finding an improving solution. The former strategy is known as steepest descent search and the latter as random descent search.

The main disadvantage of using a basic local search is the likelihood of getting stuck in a local optimum. This is because these searches generally do not accept non-improving solutions. More advanced local search techniques, popularly known as metaheuristic, aim to diversify the search away from local optimum. This is done by allowing non-improving solutions in a controlled manner with the aim of discovering better solutions and getting closer to a global optimum. We will look into metaheuristic which have been applied successfully to solve 2D Irregular OPD in the following subsections. A pseudocode for a generic basic local search is given in Figure 2.1 (Reeves, 1993).

Generic Local Basic Search for a minimization problem
with solution space S, neighbourhood size N and cost function f

Step 1: Select a starting solution $s_o \in S$

Set $s_{best} = s_o$

Step 2: Repeat

Select s such that $f(s) < f(s_o)$

Replace s_o by s

until $f(s) > f(s_o)$ for all $s \in N(s_o)$

Step 4: If $f(s) < f(s_{best})$

$s_{best} = s$

Step 3: If stopping condition not true

Return to Step 2

Otherwise stop

Figure 2.1 Pseudocode of a Basic Local Search Algorithm (Reeves, 1993)

2.1.2 Tabu Search

Tabu search was formalized by Glover in 1986. Tabu search uses a memory system to restrict the choice of the next solution from the neighborhood of a given solution. When this memory system is applied to the steepest descent local search, it will be transformed into a tabu search. The basic structure of a Tabu search has the following components:

- Local search procedure
- Neighborhood structure
- Aspiration conditions
- Form of tabu moves
- Tabu list
- Maximum size of tabu list
- Stopping rule

A tabu list records forbidden moves, which are referred to as tabu moves. It is common practice to store attributes of a neighbourhood structure as elements of a tabu move, for example the neighbourhood move that resulted in that solution. In essence we want to prevent the search from recycling through a few good solutions. The longer the tabu list the more aggressive the search will be for making non-improving moves, thus visiting more local optima. However tabu restrictions can also be broken. When a tabu move has an evaluation where it would result in a solution better than any visited so far, then its tabu restriction is overridden. A condition that allows such an override to occur is called an aspiration criterion. Figure 2.2 (Reeves, 1993) shows the standard pseudocode for a generic tabu search algorithm and Figure 2.3 shows the flow chart of a standard tabu search algorithm.

Generic Tabu search for a minimization problem
with solution space S, neighbourhood size N and cost function f
$N(s_o,h)$ denotes neighbouring solutions not contained within the tabu list

Step 1: Select a starting solution $s_o \in S$
Select tabu list length l
Set tabu history h empty
Set $s_{best} = s_o$
Step 2: Repeat
Select s such that $f(s) < f(s_o)$
Replace s_o by s
until $f(s) > f(s_o)$ for all $s \in N(s_o,h)$
Step 4: If $f(s) < f(s_{best})$
$s_{best} = s$
Step 3: If stopping condition not true
Return to Step 2
Otherwise stop

Figure 2.2 Pseudocode of a Simple Tabu Search Algorithm (Reeves, 1993)

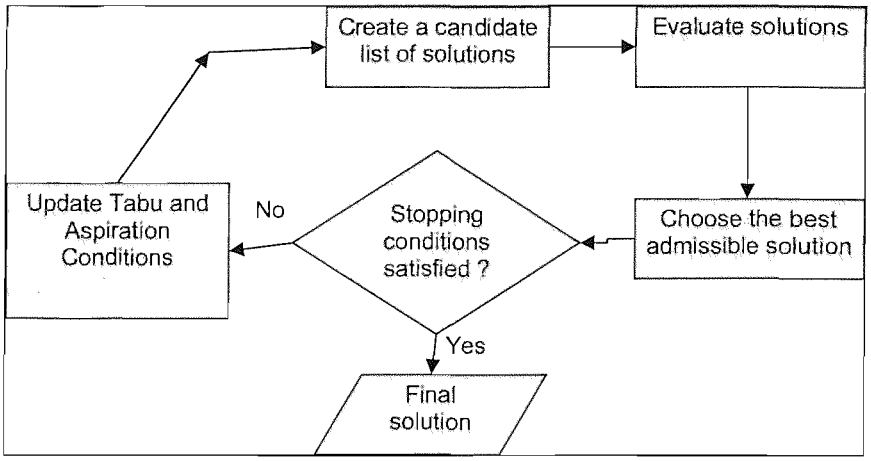


Figure 2.3 Flow Chart of a Tabu Search Algorithm

The application of tabu search to 2D irregular OPD is scarce. The main papers on the use of tabu search for solving irregular packing problems are Blazewicz et al. (1993), (1995).

2.1.3 Simulated Annealing

The concept of simulated annealing (SA) was first published by Metropolis et al (1953). The annealing process is modeled based on the cooling of a material in a heat bath. Material, when cooled slowly, will turn into crystals but if fast cooling is implemented the crystals formed will contain some imperfections. The increase or decrease in energy during the stages of the cooling has a direct impact on the resulting state of the material. Kirkpatrick (1983) used SA which mimics the annealing process to solve optimization problems. The law of thermodynamic states that at temperature t, the probability of an increase in an energy of magnitude δE is given by,

$$P(\delta E) = exp(\delta E/kt), \tag{2.4}$$

where k is a physical constant known a Boltzmann's constant. The main idea of SA is to use a probabilistic approach to avoid getting stuck at a local optimum by performing controlled uphill moves. It uses a temperature parameter that controls the search. The

temperature parameter usually starts off high and is slowly "cooled" or lowered in each iteration. This is called a cooling schedule. During each iteration if energy has decreased Metropolis' SA accepts the corresponding neighbourhood move and if the energy has decreased then the corresponding neighbourhood move is accepted according to the probability given in equation 2.4. At each temperature a predetermined number of iterations are preformed to diversify the search within the restricted neighbourhood structure. Simulated annealing has a similar search strategy as of a random descent local search (Reeves 1993). A random descent local search samples a neighbourhood at random and chooses the first improving solution, whereas simulated annealing differs with random local search by probabilistically accepting worse solutions. The probability of doing so is directly dependent on the temperature. This process sometime helps to identify a new region of the search space in hope of finding a better local optimum. Figure 2.4 (Reeves, 1993) provides a mapping of the problem specific parameters of all local search procedures with SA. Figure 2.5 (Reeves, 1993) shows the standard pseudocode for a generic SA algorithm. One of the earliest references that utilizes SA for solving irregular packing problem is Dagli and Hajakbari (1990). Lutfiyya (1992), Marques et al (1991), Oliveira and Ferreira (1993), Jain (1992), Theodoracatos and Grimsley (1995), Heckmann and Langauer (1995), Han et al. (1996), Burke and Kendall (1999), Faina (1999) and Hifi (2003) also successfully implemented SA to solve the irregular packing problem.

System States	→	Set of Feasible Solutions
Energy	→	Cost
Change in State	→	Neighbourhood Solution
Temperature	→	Control Parameter
Frozen State	→	Heuristic Solution

Figure 2.4 Mapping of attributes of SA with that of a Local Search (Reeves, 1993)

Generic Simulated Annealing Search for a minimization problem with solution space S, neighbourhood size N and cost function f

Step 1: Select a starting solution $s_o \in S$

Select and initial temperature $t_o > 0$

Select a temperature reduction function α

Set $s_{best} = s_o$

Step 2: Repeat

Randomly select $s \in N(s)$

$\delta = f(s) - f(s_o)$

If $\delta < 0$

Replace s_o by s

Else

Generate random number x in the range (0,1)

If $x < exp(-\delta/t)$

Replace s_o by s

until iteration_count = $nrep$

Set $t = \alpha(t)$

Step 4: If $f(s) < f(s_{best})$

$s_{best} = s$

Step 3: If stopping condition not true

Return to Step 2

Otherwise stop

Figure 2.5 Pseudocode of a Simple SA Search Algorithm (Reeves, 1993)

2.1.4 Genetic Algorithm

A Genetic algorithm (GA) is also a probabilistic approach adopting a similar search strategy as a random descent local search. Although genetic algorithms were first conceived by Fraser in 1957 and Bremermann in 1958, it was made popular by Holland in 1975. Genetic algorithms are inspired by models of natural evolution of species and use the principle of natural selection which favors individuals that are more adapted to a specific environment for survival and further evolution. In a similar way, in finding better solution to complex problems, we can apply this principal to combine pieces of existing solutions. Each individual in a GA algorithm typically represents a solution with an associated *fitness value*. The three main operators used are selection, mutation, and recombination. Selection prefers fitter individuals to be chosen for the next generation and for the application of the mutation and recombination operator. Mutation is a unary operator that introduces random modifications to an individual. Recombination combines the genetic material of two individuals, also called parents, by means of a crossover operator to generate new individuals, called *offsprings*.

In the first GA applications, the solutions or individuals were usually represented as string of 0s and 1s, which is commonly known as a *binary* representation. This type of representation however proved to be insufficient to efficiently attack certain types of combinatorial problems, like permutation problems (permutation problems are problems in which a solution may be represented by a permutation of the numbers, which are naturally encoded in other ways. Therefore, for such problems usually more general, problem specific encodings are applied. For example in 2D ODP the sequence of pieces to be placed on the strip layout is usually encoded as the representation of the chromosomes, this is commonly known as a *permutation* representation. The crossover operator is usually understood as the main operator driving the search in genetic algorithms. The idea of crossover is to exchange useful information between two individuals and in this way to generate a hopefully better *offspring*. Mutation is understood as a background operator which introduces small, random modifications to an

individual. The selection operator is used to keep the population at a constant size, choosing preferably individuals with higher fitness (survival of the fittest). Each individual encodes a solution to the problem, and its *fitness value* corresponds to the objective function value of that solution. The complete cycle of recombination, mutation and selection is called generation. The basic steps in a GA are given in Figure 2.6.

Among the papers that investigated the suitability of GA with irregular packing problem are Fujita et al. (1993), Ismail et al.(1995), Dighe and Jakiela (1996), Jakobs (1996), Bounsaythip et al. (1996), Jain et al.(1998), Hopper (1999), Cheng and Rao (2000), Ramesh Babu and Ramesh Babu (2001), (1998), Tay et al. (2002), Yeung et al. (2003).

Basic Genetic Algorithm Framework

Step 1 Initialization

Choose an initial population (usually randomly) containing P solutions

Step 2 Evaluation

For each member of the current population, evaluate its fitness

Step 3 Selection

Create a mating pool by selecting solutions in the current population, where the probability of selection is based on the fitness function.

Step 4 Crossover

Forms pairs of elements from the mating pool and perform crossover on each pair to create offspring

Step 5 Mutation

Mutate the elements of the strings obtained from crossover with a given probability

Step 6 Recombination

Form a new generation from the offspring obtained from cross and mutation.

Step 7 Termination test

Test whether the algorithm should terminate, go to the evaluation step

Figure 2.6 Basic Framework of a Genetic Algorithm

2.2 Geometric Solutions

Why do we need geometric computation? It is because the computer is not like the human eye. Imagine two arbitrarily placed shapes on the computer screen, it is trivial for the human eye to detect if these two pieces overlap, touch each other, or are placed a distance apart from one another but how can we build this intelligence into an algorithm? Below some of the popular geometric approach will be discussed, most of the information given below was extracted from the detailed description of the geometric tutorial given by Bennell and Oliveira (2008).

Since the focus of this research is on the irregular variant, let us start by defining irregular shapes. "Irregular shapes are defined as simple polygons, and in some cases, polygons that may contain holes" (Bennell and Oliveira, 2008). An irregular shape needs some complex geometrical technique to detect overlap between them, for example a circle cannot be classed as an irregular shape as the overlap between two circles can be easily calculated by using the center and radius values of both circles.

2.2.1 Pixel/Raster Method

The pixel/raster method digitizes the stock sheet into a continous array of pixels representing grid points on the layout. These grid points based on pixels is a standard with any computer aided drafting tools, e.g. AutoCAD. Once this has been done placing a shape on this grid will result in some pixels or grid points being occupied by the shape and some not. This information will be used and represented in a coding scheme. For example a simple coding scheme proposed by Oliveira and Ferreira (1993) uses a value of 1 to code the existence of a piece and a value of 0 to represent empty space on the layout as illustrated in Fig 2.7. Every time a piece occupies a grid point a value of 1 is added to the memory buffer of this particular grid pint, hence if 2 pieces occupies the same grid point the value in the memory buffer will be 2. Once this information is

available detecting overlap is as simple as checking the value in the memory buffer where a value greater than 1, indicates an overlap on the layout.

Other more complex coding schemes were proposed by Segenreich and Braga (1986) and Ramesh Babu and Ramesh Babu (2001). The advantage of the pixel/raster method is the simplicity of implementing it in a computer program to detect and resolve overlap between two pieces on the layout. However the disadvantage of using this method is the memory intensive nature of its implementation and inability to accurately represent shapes with non-orthogonal edges.

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Figure 2.7 0 -1 *Pixel/Raster Representation of a Shape*

2.2.2 Direct Trigonometry and D functions

This method enables accurate representation of the shapes. The shapes are approximated as polygons. Overlap between two shapes can be detected using their relative position on the layout. This is done by using functions of direct trigonometry to test for intersections between the edges of these two shapes. The tests are broken down into high level and low level tests. The high level test aims to detect non-overlap situations at a very early stage using relatively easy calculations based on direct trigonometry as shown in Figure 2.8. Figure 2.8 (a) shows that if two polygons are in an overlapping position, the bounding

boxes of the two polygons must overlap. The same principle applied to the intersecting edges of the two polygons shows that the bounding box of the edges must also intersect as shown in Figure 2.8 (b). The direct trigonometry consists of a series of test on line intersection and point inclusion. The low level test is a computational intensive test to analyze polygonal edges between the two shapes and find out if they intersect, hence overlap. D-function proposed by Konopasek (1981) is used during the edge analysis phase to accurately test for an intersection between two edges. D-function analyses the edges between two shapes with respect to their orientation and vertex and translate this information into finding if they intersect or not.

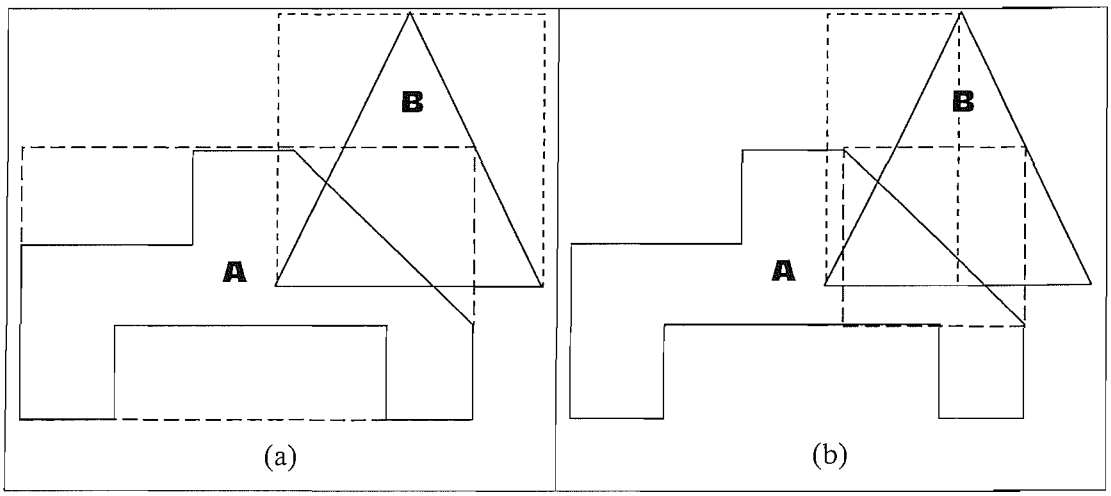


Figure 2.8 (a) If two polygons overlap, then the rectangle bounding boxes of the pieces must overlap (b) If two edges intersect, then the rectangle bounding boxes of the edges must intersect.

The D-function can be defined as follows:

$$D_{ABP} = ((X_A - X_B)(Y_A - Y_P) - (Y_A - Y_B)(X_A - X_P)). \quad (1)$$

Given an oriented edge AB and a point P, the D-function identifies the relative position of this point to the oriented edge as shown in Figure 2.9.

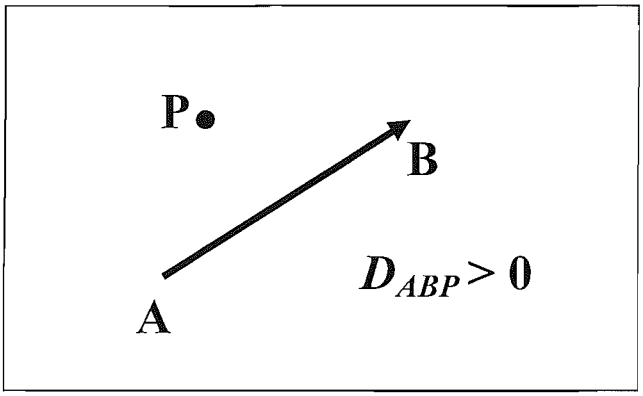


Figure 2.9 Analysis of D-Function

The following analysis is used for the D-function: if $D_{ABP} > 0$ then point P is on the left side of the supporting line of edge AB; if $D_{ABP} < 0$ then point P is on the right side of the supporting line of edge AB; if $D_{ABP} = 0$ then point P is on the supporting line of edge AB. During the edge analysis phase of the low level test, six different conditions will be tested out using D-functions to accurately identify the relative position of the two oriented edges. This method can accurately represent the shapes of the polygons, however because of the rigorous computational calculations that must be performed for every overlap test; it makes this method a less efficient choice for the 2D ODP problem when it is solved using iterative search methods.

2.2.3 The Nofit Polygon (NFP)

There are three approaches found in the literature to develop the NFP generator; the orbiting algorithm (Mahadevan, 1984; Burke et al., 2007), minkowski sums (Milenkovic et al., 1991; Bennell and Song, 2008) and decomposition into star shapes polygons (Li and Milenkovis, 1995) or convex polygon (Watson and Tobias, 1999; Agarwal et al., 2002). We will not go into details of these approaches as it is beyond the scope of this thesis. However we will briefly describe the Minkowsi sums approach since the NFP for this thesis work was provided by the authors of Bennell and Song (2008). The concept is as follows: given two arbitrary point sets, A and B, the Minkowski sum of A and B is defined by the following:

$A \Phi B = \{a + b: a \in A, b \in B\}$

In order to produce no-fit polygons the Minkowski difference can be used, $A \Phi -B$. This is equivalent to the two input polygons in opposing orientations and is easily shown through simple vector algebra (Bennell et al., 2001).

The NFP is a polygon formed as a result of combining components of two polygons together. It works together with a vector algebra that uses the relative positions of two polygons and calculates the vector difference between them. The result of this vector difference is then used to find out if the calculated vector point is in, outside or on the NFP which can be interpreted as the two polygons overlapping, separated or touching each other. Given two polygons A and B the construction of the NFP of B in relation to A can be found in the following way. The NFP is derived by placing two polygons (A and B) so that they touch each other but do not intersect and moving one of them (B) around the other, respectivelys. The resulting polygon that is generated as B moves around A is denoted as $NFP_{A.B.}$ While this is happening both A and B maintain a fixed orientation, touch each other but do not overlap. We need to define the origin on A, B and $NFP_{A,B}$ and the reference point on B before $NFP_{A,B}$ is generated. Bennell (1998) defined the bottom left corner of the enclosing rectangle of both polygons as the origin of A and B. As a result the origin of $NFP_{A,B}$ will be the top right corner of the enclosing rectangle of the tracing polygon B. The reference point can be any point on polygon B, we have chosen this point as the bottom left corner of enclosing rectangle of polygon B. As polygon B moves along polygon A in a clockwise direction the reference point on B will trace the path it follows, this will be called $NFP_{A,B}$. Figure 2.10 show the step by step process in obtaining the no-fit-polygon, origin for polygon A,B, $NFP_{A,B}$ and reference point on polygon B is defined as described above.

Having derived $NFP_{A,B}$ detecting overlap between piece A and B is achieved using a point inclusion geometric test to find if a given point on the layout is inside, outside or on $NFP_{A,B.}$ Before the point is used in the test, it has to translated by $(-x,-y)$, where (x,y) refers to the relative origin of polygon A on the layout.

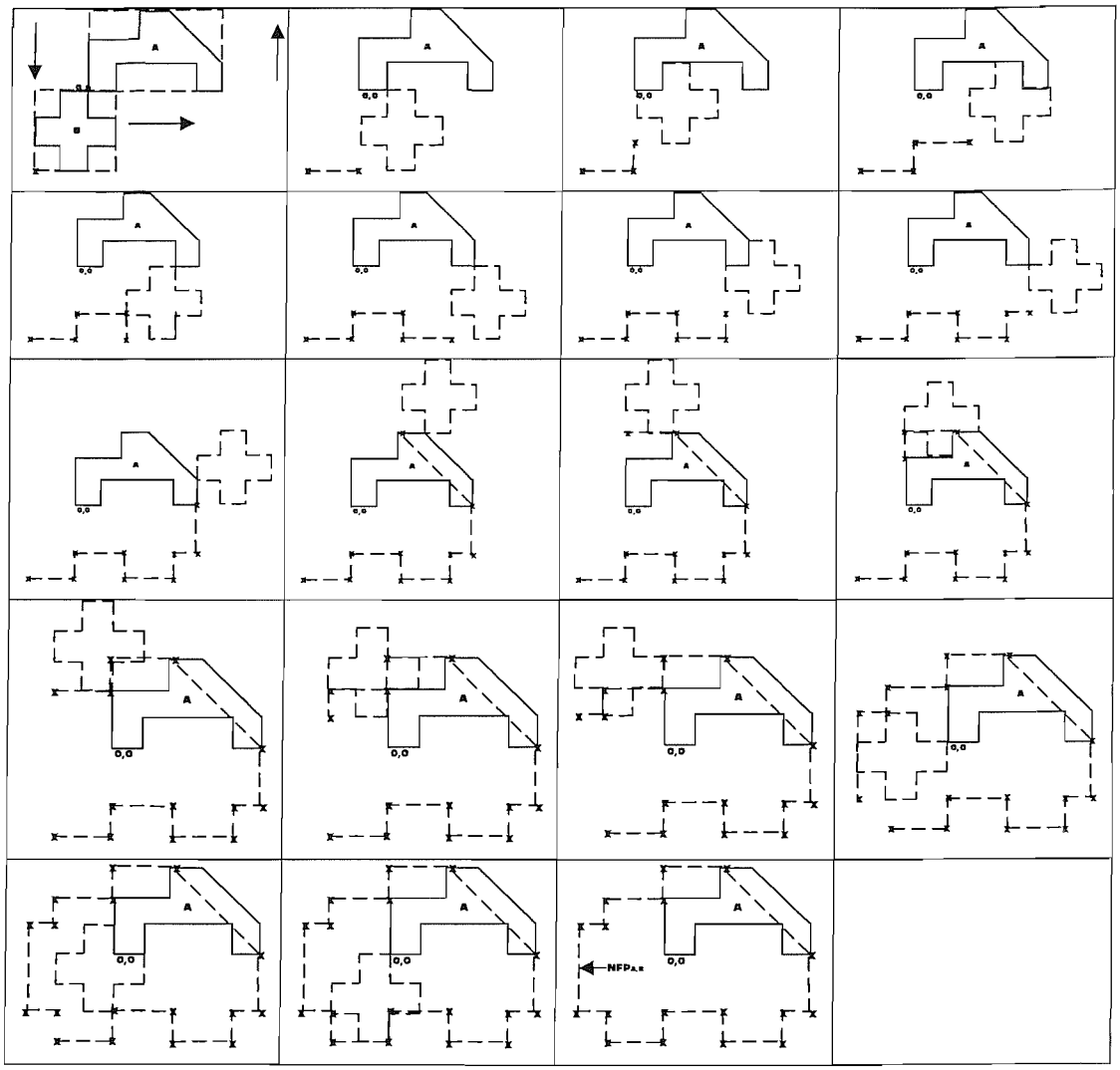


Figure 2.10 Step By Step Generation of NFP

If this translated point is inside $NFP_{A,B}$ this indicates an overlap situation, if it is outside then they don't overlap and if it is on $NFP_{A,B}$ both polygons touch each other but do not overlap. Figure 2.11 illustrates this process, given piece A is already positioned at point (0,3) on the layout and piece B is about to be placed at point (4,3). The resultant of vector difference, $u - v$ gives us point (4, 0) which lies inside $NFP_{A,B}$ indicating an overlap in the layout. This illustration helps us to understand that detecting overlap using the NFP is not just a simple point inclusion test but it has the complexity of $O(n)$, where n represents the number of edges in the NFP.

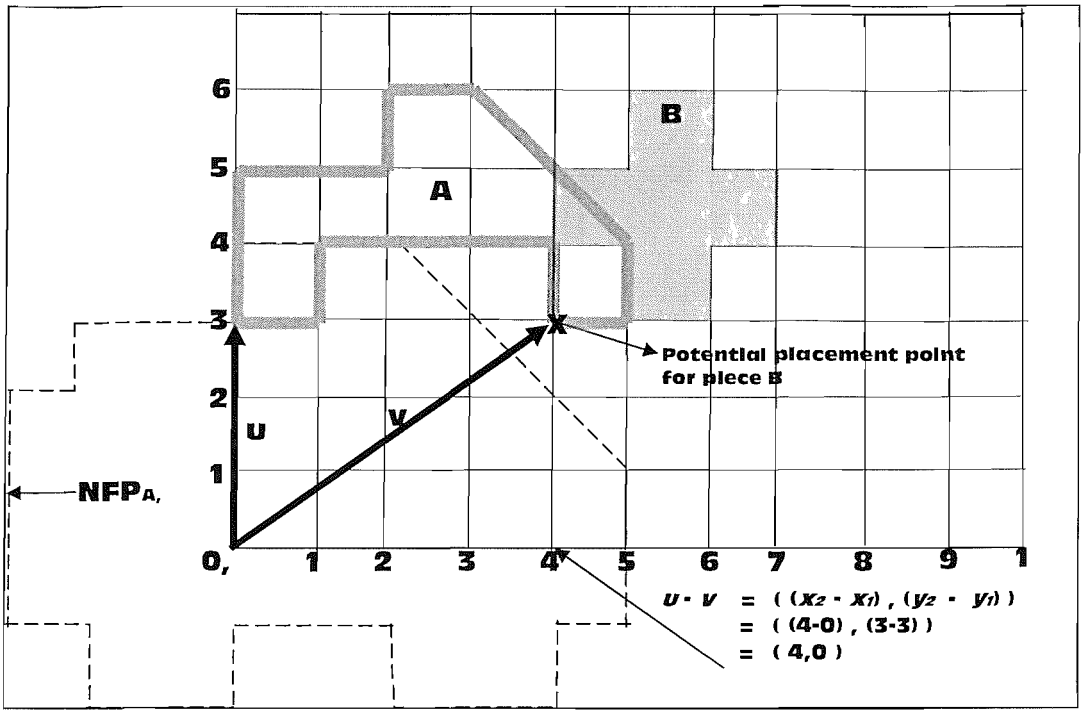


Figure 2.11 Detecting overlap in the layout using the NFP

The advantage of using $NFP_{A,B}$ is the quality of layouts obtained by accurately representing the shapes as polygons. By preprocessing the no-fit-polygons the execution speed of overlap detection can be increased. However the complexity involved in calculating and generating the no-fit-polygon hinders many researchers from implementing it.

2.2.4 Phi Functions

The phi-function is another method in overlap detection between two polygons (Stoyan et al., 2004). It is made up of a mathematical function that defines the relationship between two polygons with regards to their position in the layout. The calculated phi value is greater than zero in a non-overlap situation, equal to zero if they are touching each other and less than zero if they overlap. However this method is not widely adopted due to the lack of algorithmic know how by the wider community.

2.3 Iterative Constructive Heuristics (ICH)

Having surveyed the geometric techniques for detecting overlaps in the layout, we will now concentrate on the heuristics routine. In this section we will focus on approaches that we will call Iterative Constructive Heuristic (ICH). We will be reviewing the literature published on ICH but first we will explain some of the key steps of ICH. This involves a selection rule to select pieces to be packed on the layout from a pool of available pieces, a placement rule to find the best position for this piece on the layout with regards to the objective function and a search strategy to continuously improve the packing quality.

In these heuristics pieces are selected or ordered and then packed one by one onto the strip layout. When selecting or ordering the pieces to be packed a specific rule is used. In earlier research a sophisticated approach was used to choose the next piece to be packed dynamically in order to minimize wastage of area (Art, 1966, Albano and Sapuppo, 1980). More recently local search techniques have become increasingly popular to select a good ordering of the pieces by beginning with an initial presorted order of pieces and searching through this predefined solution space based on a neigbourhood structure (Gomes and Oliveira, 2002, Burke et al., 2006).

Having selected the piece to be packed next, there are several ways of finding the feasible placement points on the layout to position this piece. Feasible placement points are generated based on a resolution of grid points on the layout or using the boundary of no-fit-polygon. In ICH pieces are not allowed to overlap one another on the layout. Some algorithms allow hole filling, e.g. placing pieces in between gaps of already placed pieces. We have now selected the piece to be packed and have identified several feasible locations on the layout, now this piece will be positioned on the layout based on a placement rule. This placement rule will select the best position in the layout to achieve the defined objective function.

Next is the iterative optimization phase, as the name suggests several different packing orders are generated depending on the specific rules embedded in the algorithm and the solution that produces the best layout measured in terms of minimizing wastage, usually the shortest strip length or highest area utilization will be returned when the algorithm terminates. In this iterative optimization phase metaheuristic search techniques (e.g. tabu search, simulated annealing, genetic algorithm) have become quite popular in recent times to prevent the search from getting stuck in a local optimum. The algorithms terminates based on a termination criteria, which is normally the number of iterations or elapsed time. To illustrate the phases in Iterative Constructive Heuristic, figure 2.12 below will be helpful.

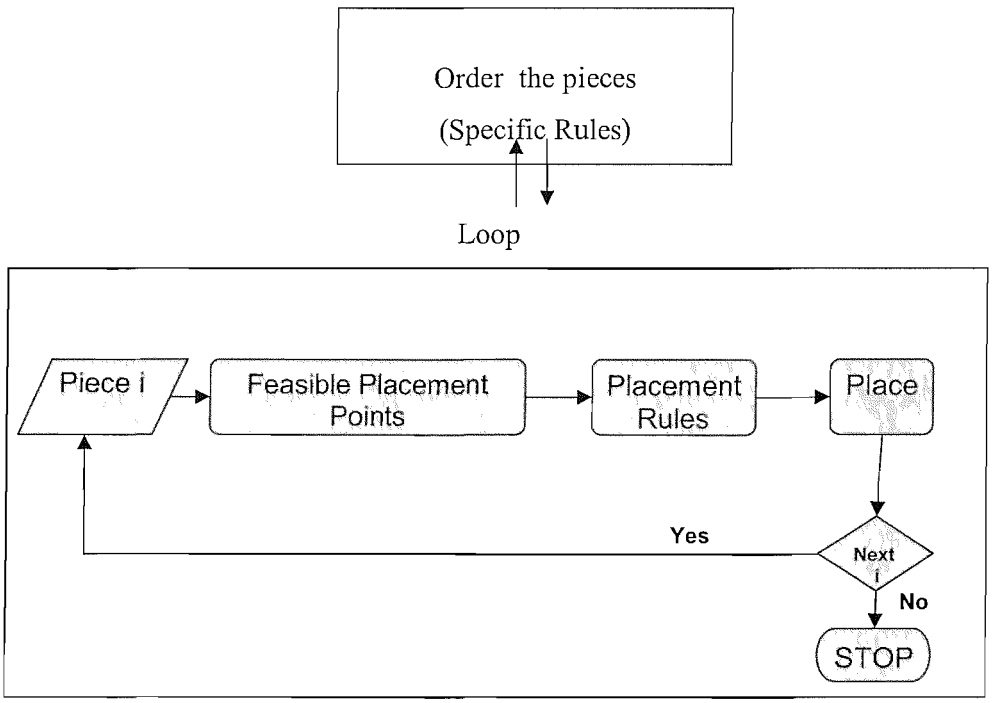


Figure 2.12 Phases of Iterative Constructive Heuristics

The literature will be sectioned into describing the placement heuristic first, identifying the commonly used approaches and then moving into the search techniques to get a good overview of the various ways research has been carried out to solve this problem.

2.3.1 Placement Rules

Art (1966) introduced the concept similar to today's well known no fit polygon (NFP) and called it 'shape envelope'. The *'shape envelope'* defines the feasible positions on the stock sheet for placing polygons without overlaps. This concept later became a standard for generating feasible placement positions on the strip layout (e.g. Albano and Sapuppo, 1980, Gomes and Oliveira, 2002, Dowsland et al., 2002, and Burke et al., 2006) these are positions on the layout where pieces can be placed without overlapping one another.

Art (1966) also introduced the bottom left placement heuristic for positioning the pieces on the layout. This will be the choice of location when placing the pieces on the layout, as close as possible to the bottom and left of the strip layout. Bottom left heuristics have became a popular placement rule for positioning pieces on the strip layout.

The jostle approach (Dowsland, et al., 1998) places the pieces by jostling between a left most and right most placement policy. A leftmost/rightmost policy places pieces towards the left/right of the stock sheet according to a predefined set of rules. Unlike Art (1966) biasness is not just towards the bottom instead pieces are positioned as close a possible towards either the bottom or top of the stock sheet. Hole filling is permitted by generating a sequence of feasible positions, x co-ordinates from zero onwards and y co-ordinates from zero to W-w, where W is the width of the stock sheet and w is the width of the current piece being placed. This algorithm was inspired by the fact that by shaking up and down a granular product stored in a container will let them settle into their natural position, removing any unevenness in the initial form.

Oliveira et al. (2000) worked on an algorithm which they named TOPOS. In this work the pieces are placed one by one onto the strip layout building a partial solution that grows from a floating point origin. Every time a new piece is placed on the layout, an NFP is formed using the external contour around the pieces already placed in the layout. This will be used to determine feasible placement points on the layout. When choosing

where to position this piece a best fit criteria is used based on 3 measure; i) minimizing the area of rectangular enclosure of two pieces, ii) minimizing the length of rectangular enclosure of two pieces and iii) maximizing the overlap between the rectangular enclosure of two pieces. Once this new piece is positioned in the layout a geometric algorithm is used to merge this piece with the existing pieces on the layout to form a new external contour.

Bennell and Song (2007) modified the work carried out by Oliveira et al. (2000) on TOPOS. The changes in the new TOPOS include an NFP generator based on the concept of Minkowski sums (Bennell and Song, 2008) that retained the gaps between polygons as they are merged.

Dowsland, et al. (2002) describe a step by step procedure for implementing the bottom left strategy with hole filling for packing irregular shapes. They begin the search for a feasible placement point by finding the leftmost point from the set of the NFP vertices, which will be the required bottom-leftmost point on the layout for the piece selected to be packed, where ties are broken by choosing the lowest y co-ordinates. If this selected position passes all the NFP point inclusion tests then a feasible placement point is found otherwise a standard trigonometry test is used to get all the intersection points between a vertical line generated from this point and the NFP edges. This will generate test positions at intermediate points on the NFP edges. All these positions are tested in their ascending order. If none of these positions are overlap free the search resumes again from the next bottom-leftmost point in sequence. This is a computationally demanding test so some important aspects were given to improve the algorithm efficiency. The algorithm was implemented on four datasets used in Dowsland et al. (1998) together with the dataset used by Blazewicz and Walkowiak (1993) and compared with the basic BL algorithm. The results reported improvement in layout length and suggested some minor performance improvements were achieved when some piece ordering/sorting strategy was introduced to the experiments.

Gomes and Oliveira (2002) used a bottom left placement heuristic similar to that of (Dowsland, et al., 2002). A step by step procedure is given to reduce the infinite non-convex feasible placement points to a discrete and finite set of points; hence the search for bottom left placement points becomes quite easy. This was accomplished by following this set of rules. Given any two shapes i and k, the no-fit polygon of i and k ,denoted $NFP_{i,k}$ defines the shape of area where placement of k will result in overlap with i. For $piece_k$ to be placed among other pieces already on the layout: 1) Get the vertex coordinates of $NFP_{i,k}$ and the inner-fit-rectangle IFR_k, 2) Get the intersection points between two edges of two no-fit-polygon $NFP_{i,k}$ and $NFP_{j,k}$ and 3) Get the intersection points between the edges of inner-fit-rectangle IFR_k and an edge of a no-fit-polygon $NFP_{i,k}$. The inner-fit-rectangle is derived using the NFP and it represents the feasible placement points that will contain this piece within the layout. Sort this collection of points according to their lowest x coordinate, ties are broken by selecting the smaller y coordinate and start the search for a feasible placement point moving through this sorted list.

Burke et al. (2006) introduced an alternative new bottom-left-fill heuristic with hole filling. This heuristic places the first shape at the lower left corner in the layout. It then tests the allowable rotation for this piece and determines the best rotation based on the defined performance measure. For the next shape in the packing order it starts from the lower left corner if this piece type is not already on the layout, otherwise it starts from where this same piece type was placed last on the layout and attempts to find a feasible placement point in a positive vertical direction (up the y axis). If this piece intersects with already placed pieces it uses an overlap resolution technique during the search process for a feasible placement position. If overlap is not resolved the search is continued by moving along the positive x axis direction by one increment (known as resolution) and proceeding to search vertically again. The search continues in this fashion until a feasible location is found to place the shapes.

Having identified the most popular placement rule for positioning pieces on the strip layout let us now consider the various different rules for selecting the pieces to be packed next on the layout as this is often the key to achieving better layouts.

2.3.2 Selection Rule

Art (1966) used single pass heuristic with a selection rule based on three criteria to choose the piece to be packed next. These criteria are used successively to narrow down the choice of piece selection step by step until finally choosing the best one. The first criteria select the pieces that could produce lowest x values on the envelope. This x value is within a defined tolerance limit and is determined by the placement heuristic according to the probable position of this piece on the layout. The second criteria will be applied to the pieces selected from the first criteria. It will select the pieces based on their area. The pieces with area greater than a fixed fraction will be chosen. The fixed fraction is calculated as the area of the chosen piece over the maximum area, which is given by the piece with the largest area among pieces selected under criteria one. The final criteria will select from these pieces the ones that produce a minimum "probable waste" on the left hand side of the layout, this waste is calculated according to a defined formulation. Ties are broken with pieces having the lowest y value on the envelope.

Albano and Sapuppo (1980) discuss a procedure which claims to produce a locally optimal arrangement of irregular pieces. The best ordering of pieces is achieved by performing a tree search to find the locally optimal path from an "initial state" to a "goal state". Backtracking through the search path is allowed in order to not restrict the search. A cost function is used to find the cost of the path between two states. This is calculated based on the waste generated by the piece selected to be packed next. The waste is calculated by examining the net change to the area on the right hand side of the profile. Therefore the best solution is the best arrangement of a set of given pieces achieved by finding the minimum cost from the "initial state" to the "final state". Hole filling

capability is not included in this algorithm. The main application discussed is that of cloth layout and leather cutting.

Blazewicz, et al. (1993) extended the work done by Albano and Sapuppo (1980). Additional feature included is the capability to fill holes in the layout. The initial solution is generated by sorting the pieces according to decreasing area and then packing them one by one onto the strip layout. Tabu search is used to improve the quality of the initial solution. The method is implemented in such a way that a single piece is chosen and its position is changed at each step of the algorithm. The algorithm terminates if no improvement to solution quality is achieved after a predetermined number of iterations.

The jostle approach algorithm (Dowsland, et al., 1998) is based on a single pass algorithm ("combines piece ordering with a placement policy"). It starts by placing the initially randomly ordered pieces by following a left most placement policy. Once the first pass is complete the pieces are re-ordered in decreasing order of their right-most points (x coordinates) and packed using the right-most policy. Once this packing is complete the pieces are re-ordered in increasing order of their left-most points (x coordinates) and packed again using the left-most policy and the process is continued for a fixed numbers of iterations. This method was tested with D-functions and NFP, with NFP reporting significant savings in computational time.

Oliveira et al. (2000) worked on an algorithm which they named TOPOS. The criteria used for choosing the piece to be placed next is an initial sort based on a set of criteria or is determined by local search heuristics which iterates through the unique pieces left to pack and chooses the one based on a best fit measure. The TOPOS algorithm produces two approaches for adding a new piece to the layout: initial sorting or local search. In the initial sort pieces are ordered based on one of the following criteria: i) decreasing length, ii) decreasing area, iii) decreasing concavity, iv) increasing rectangularity (based on the difference between the piece area and the area of the respective enclosing rectangle) and iv) total area (based on numbers of similar pieces available). The local search heuristic evaluates the partial solution using a best fit measure. The best fit measure is based on

three formulated criteria; i) Waste, ii) Overlap and iii) Distance. Taking into account the combination of all criteria from both approaches a total of 126 variants were used for the computational tests It was observed that the best result could not be associated with a particular variant of the algorithm. Furthermore it was concluded that the shape's geometric properties have a strong effect on the results.

Gomes and Oliveira (2002) developed a 2-exchange heuristics. Neighbourhoods are generated by exchanging pair of pieces in a sequence. The solutions are evaluated based on i) first better solution found ii) best solution found and iii) randomly chosen solution among the better solution found. Computational results support the probabilistic approach with a longer neighbourhood size (distance between swapped pieces = 3). The ranking criteria used to build the initial layout are randomness, area, length, width, irregularity, and non-rectangularity but was reported that the choice strongly depended on the particular dataset and has no dominance over different data sets. The placement heuristic was reported to generate a layout in less than 20 seconds and was effective at filling holes at any stage of placement.

Burke et al. (2006) used standard hill climbing and tabu search mechanism over an initial area or length pre-sorted arrangement of pieces in their experiments. A neighbourhood size of 5 solutions and a tabu list length of 200 were used throughout the experiments. The search technique uses a numerical operator between 1 to 4 and a random number N, 1 removes a randomly chosen shape and inserts it at a random location in the sequence, 2,3,4 and N swaps the equivalent number of shapes in the order. A new technique introduced in this paper is their ability to handle arced edges in the pieces without approximating by the polygon edges. The algorithm improved 20 of the 26 available literature problems. The average improvements for the 20 best solutions were in the range of 4% to 6%.

Bennell and Song (2007) modified the work by done by Oliveira et al. (2000) on TOPOS. They introduced 6 new formulation criteria and utilised a beam search heuristic for selecting the next piece to be placed. A beam search heuristic works in a similar manner

to a branch and bound algorithm where the tree is searched breadth first and aggressively pruned at each level. This allows for many parallel partial solutions to be evaluated. The best packing lengths were equal to, or better, than the best results reported in the literature in 7 out of 15 cases. Since multiple factors are involved, no clear conclusion could be made as to which could be selected as the best criteria.

2.4 Searching Over Layout Heuristics

For these methods an initial layout is formed and then pieces are repeatedly selected and moved to some other location on the layout. During this phase overlap is often allowed. To choose the piece to be moved and, the position to move it to, a cost function is used. The cost function used to choose the piece aims to select the piece with the maximum amount of overlap while for positioning the piece it aims to minimize the total amount of overlap on the layout. A feasible solution is found when overlap is totally eliminated from the layout. Several different solutions are generated depending on the neighbourhood structure and the solution that produces the best feasible, or zero overlap layout, measured in terms of minimizing waste, usually the shortest strip length or highest area utilization, is returned at the end of the desired number of iterations. The algorithm terminates based on a stopping criteria, which is usually number of iterations performed or duration of lapsed time. The solution space, with this method is infinite, meaning any piece can be placed in any location on the stock sheet. This can be seen as the major strength of SOL compared with ICH. Several different alternatives are available for defining the cost function and reaching better solutions from one layout to another. This will be discussed later in the literature review

Simulated Annealing (SA) appears to be the most popular search engine applied by many researches in this approach. This maybe because it is better suited to infinite neighbourhoods.

Jain (1992) used SA as a search engine. The application discussed was that of the metal blanks stamping. The shapes to be laid out on the metal stock sheet was restricted to not

more than three due to die stamping machine constraint. The focus was to cluster no more than 3 pieces together, finding the optimum arrangement of these pieces and repeating this pattern horizontally along the stock sheet. The placement rule as to how these pieces may best fit together is given by fixing the position of one of pieces and allowing the others to slide freely over the identified corner locations of a predefined bounded region in which the shapes may be placed and minimizing waste during placement. The aim was to form a "lock and key" arrangement. Overlap is allowed during placement. The cost function used in the position placement rule is scrap cost added to the weighted penalty of the overlap area. The overlap area is calculated by finding the boundary of the overlap region and then computing its area.

Heckmann and Lengauer (1995) also used SA and specifically tailored the algorithm towards the requirement of the textile industry. Constraints are subdivided into three main categories, i) layout legality ii) technology limitations and iii) constraints for waste reduction and cutting time minimization. The search points available to SA are generated from a number of possible moves types. A move type consists of the allowed movements (translation, rotation, and exchange of two pieces on the layout) and a real number to denote how much to move, e.g. in a translation move type this will be the distance of the translation. The rationale behind this strategy is that as the SA temperature decreases short distance translations will be preferred. In each step of the SA a piece must be selected for movement and a move type must be chosen, these are controlled by assigning selection probability to each move type and this probability depends on the cost function. As the temperature decreases the cost function reduces the possibility of the selection of a move type. The cost function is a sum of three cost measures, which are i) weighted length, ii) weighted overlap and iii) weighted row and column load. The length of a generated layout is defined as the difference between the largest and smallest x-coordinate of all the pieces, overlap is defined as the sum of overlap areas of all the pieces and row and column load are measures of the overlap produced by the pieces covering a specific row or column. The weights are preset initially and changed dynamically during the course of the algorithm. The annealing uses a dynamic cooling schedule and is used in 4 different stages in order to guarantee the final layout is free of

overlaps. The first stage is a rough placement, the second removes the remaining overlap, the third compacts layout to the left and last stage is a fine placement with original pieces. After the first stage overlap can still be present and a decision must be made as to which piece to choose and in what direction should this piece be moved. Moving this piece to the right of the layout will obviously eliminate the overlap but might result in an increase in the layout length. The following strategy was devised to tackle this problem, heavily penalize large overlaps in the horizontal direction and move the piece with the largest penalized overlap value in the vertical direction to the empty space. The empty space is reserved at the top and bottom margin of the layout during the first stage of annealing. An interesting claim made is that large overlap in horizontal direction implies small overlap in y direction and this is the rationale for translating the pieces in the vertical direction to remove this overlap. An interesting point to note here is the heavy intricacies involved in designing this algorithm specific to textile data types and the fine tunings made to guarantee the success of eliminating the overlaps.

A hybrid algorithm approach was used by Gomes and Oliveira (2006). The generation of the initial layout was formed using the greedy bottom left placement heuristic which places the pieces one by one from a given sequence to the most bottom-left position on the strip layout. The NFP was used to prevent overlap in the layout. This was the same heuristic that was used previously by the same author, Gomes and Oliveira (2002), however a new criterion based on random weighted length was introduced for choosing the next piece to be packed. SA was used to guide the search towards better solution. Neighbourhood structure which they call LOCALCOMPACT was generated for 2 pieces chosen to be swapped; the two pieces are selected according to the 2 exchange heuristics proposed by the same author in a previous work, Gomes and Oliveira (2002). LOCALCOMPACT generates a number of neighbours based on all possible combinations of swapping the 2 pieces and their allowable orientations. The combination that leads to the best final layout is selected. A swap would lead to exchanging the positions of two pieces in the layout and would usually result in overlap on the layout. Overlap is removed by applying a linear programming model for layout separation and then using layout compaction model the placed pieces are moved as close as possible to

each other. The linear programming model for layout separation and layout compaction model has also been used by other authors (Li and Milenkovic, 1995; Bennell and Dowsland, 2001). Overlap is not permitted. For cases where the separation model could not resolve the overlaps, two different pieces are swapped and tried again. The computational test results were outstanding, it reported improvements averaging between 6.8% - 8.84% against best result published on all problem instances.

Bennell and Dowsland (1999) used a variant of the tabu search called the simple tabu thresholding (STT). STT, although very similar to tabu search, is based on a simple candidate list strategy and avoids extensive use of memory. A new approach is proposed to estimate overlap between the pieces on the layout. Overlap is estimated as the minimum horizontal distance a piece has to move to remove overlap. To avoid heavily penalizing small pieces, the respective widths of pieces in overlap situation are also included in the overlap function and the minimum of these are then selected as the overlap cost. This will give preference to move smaller pieces. The width and length of the stock sheet is fixed and an attempt is made to remove overlap within this constrained layout. Placement positions are generated by randomly selecting a vertical grid line on the layout. Using this as a column the algorithm starts to evaluate positions alternating from the lowest y coordinate, followed by the highest, next lowest etc. approaching towards the middle and stopping at the first found non overlapping position. If no zero overlap position exists then the search repeats itself from a new randomly generated vertical grid line on the layout. The starting solution is generated by randomly ordering the pieces and packing them into columns on the layout. While forming the columns, successive pieces are placed one above the other until reaching the top of the stock sheet. A new column is then generated and this process is repeated until all pieces are placed on the layout. The y coordinate is then fixed as given by the piece position and a randomly generated x coordinate is assigned to each piece. This will create overlap in the layout. The decisions that control the STT search were initially set based on observing the behaviour of the algorithm with respect to improving the layout, escaping local optima and minimizing overlaps. Among the observation made were the tendency to cycle within a few good positions, small pieces clustering together becoming sub-optimal and a

tendency to place pieces in large holes. Based on the success of the initial experiments further investigations were carried to explore the parameter space. Five variants of the algorithm, that combine two different special cost function and three different neighbourhoods, were derived and tested. This shows that problem specific knowledge has to be derived and applied to make the search mechanism effective.

The authors continued their work in Bennell and Dowsland (2001), this time incorporating a layout compaction/separation routine in the optimization phase. Here the length of the stock sheet is not fixed. Instead the total length of the solution is evaluated and incorporated into the cost function so that both overlap and layout length is reduced simultaneously. The idea of using grid points as search points is abandoned and this time NFP edges were used instead. The properties of NFP were also used to aid in overlap detection and compaction routine. The LP formulation for the layout compaction was based on that of Li and Milenkovic (1995). The compaction/separation phase will start once the overlap value falls below a give threshold. The authors reported significant improvements over their previous work

Egeblad, et al. (2007) presented a heuristic method based on simple local search and the meta-heuristic method Guided Local Search (GLS). The generation of the initial layout was based on randomly ordering the pieces and packing on the layout using some fast heuristic, e.g. a bottom left bounding box placement algorithm. The initial strip length is calculated and then reduced by some percentage. Pieces that are no longer contained within the reduced strip length are moved back into the feasible packing region. This will obviously create overlaps on the layout. The overlaps are iteratively removed using a simple local search. If the search gets stuck in a local minimum, i.e. there are no neighbouring solutions which could reduce the overlap, then the GLS mechanism will diversify the search away from this local minima. In each iteration the search may use one of the four options to move a piece depending on the given problem instance, i.e. horizontal translation, vertical translation, rotation or flipping. This piece is then moved and evaluated at each new position on the layout with respect to minimizing total amount of overlap in the layout. If minimizing overlap positions are found, this piece is then

moved to the position with least overlap and the search quits otherwise it chooses from the remaining available options for movement and repeats itself. The algorithm terminates when overlap in the layout is totally eliminated. GLS works by assigning a penalty value to a cost function. The penalty is based on the number and amount of pair wise overlaps for a given piece. Overlap is defined as the actual area of intersection between two pieces on the layout and the cost function is given by the total sum of pair wise overlap for a given piece. It was reported that most of the results from the experimental runs are also the current best published for problem instance tested.

2.4 Summary

We started this chapter by defining the cutting and packing problem and providing examples of the problem types. We classified this problem as two dimensional irregular open dimensional problems. We reviewed the geometric and search techniques popularly used in this problem type. We highlighted the two main approaches used for 2D irregular ODP problems; iterative construction heuristics and searching over the layout. Finally we reviewed some research carried out in this area with respect to the two main approaches.

3 Methodologies

Having surveyed the literature and identified the components and tools for the irregular packing algorithm, we will now focus on the methodology adopted to develop our packing tools. The purpose of this research is to investigate the effectiveness of the two representations; iterative constructive heuristic and searching over layout heuristic from a neutral standpoint and not to compete with the best results found in the literature. In this research we intend to investigate the two approaches of the problem and establish some principles of the strengths and weaknesses of each approach with respect to data type. In order to conduct this research the algorithms will be developed using only the basic principles of both approaches and discarding any special features found in the literature. The aim is to deduct from the experimental results an understanding of what solution approach should be applied given the data type, performance requirements and number of pieces. Thus first we have to decide on the geometric tools to detect overlap, and the local search method to avoid getting stuck in local optima. We will also discuss piece selection and placement with respect to the Iterative Constructive Heuristic (ICH) and piece placement and movement with respect to the Searching Over Layout (SOL).

3.1 Solution Methods

3.1.1 Placement and Geometry

The data of shapes to be packed and their corresponding stock sheet dimension has been gathered from the EURO Special Interest Group on Cutting and Packing (ESICUP) website. The shapes are made of simple polygons and represented by a set of vertices. Some of data sets allow the shapes to be rotated, usually by 90 or 180 degrees. The stock sheet, on which to place the pieces, will normally have a constrained width and unconstrained length. The geometric evaluations are handled using the no-fit-polygon (NFP) both for identifying overlap and selecting feasible placement positions. The no-fit-polygons of all pairs of shapes, represented as sets of polygonal vertices, were provided

by the authors of Bennell and Song (2008). Having stored the NFP data of all the different shapes in the memory of the computer program we will use them to test overlap. The overlap test is a point inclusion test based on a ray crossing algorithm; the code was made available from a geometric library, (O'Rourke, 1998). This test will tell us if a selected point on the layout is inside or outside of the NFP. If the search point is an infeasible point (will result in overlap) it is inside the NFP and feasible if it is either on or outside the NFP. For placing the pieces onto the layout we have selected the bottom left placement heuristic. The bottom left placement heuristic places pieces as close as possible to the bottom of the layout, followed by the left most feasible position, as shown in Figure 3.1 for packing piece A , B and C onto the layout.

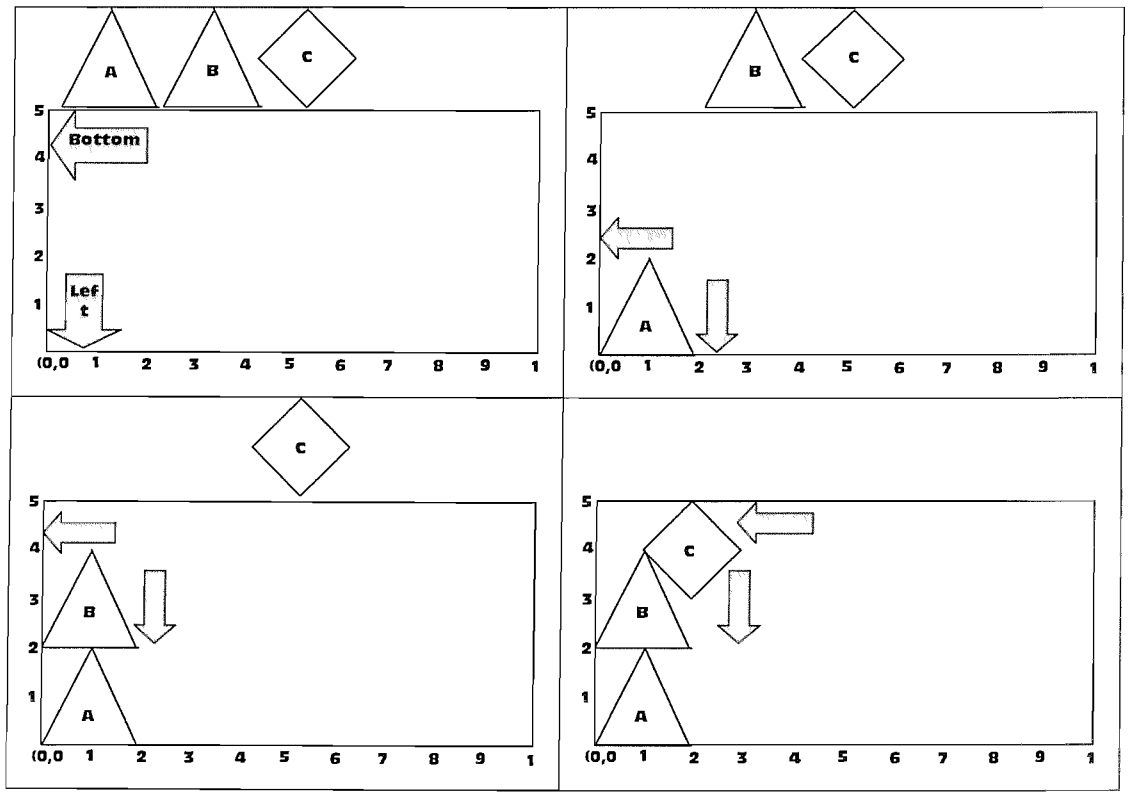


Figure 3.1 Illustration of Bottom Left Heuristic

If the shapes have more than one allowable orientation, each shape is placed by the heuristic in all allowable orientation and the one the produces the most bottom left placement is chosen. Fig 3.2 shows two pieces being evaluated for their best position in the layout in their allowable orientations.

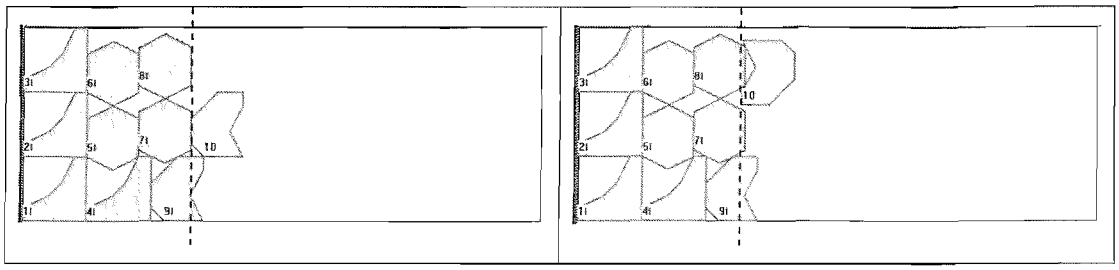


Figure 3.2 Placing Pieces in their Allowable Orientations

The shapes input data represents the shapes of the pieces to be packed on the layout. We have adopted a standard way to represent this piece on the layout. Each shape is imagined to be contained within a rectangle and the origin of the shape is always the bottom left corner of this rectangle as show in Figure 3.3, thus the initial origin of any irregularly shaped piece will always be (0,0). Using this origin positioning this piece anywhere on the layout is as simple as adding the (x,y) coordinates of the selected position on the layout with the vertices input data of this shape.

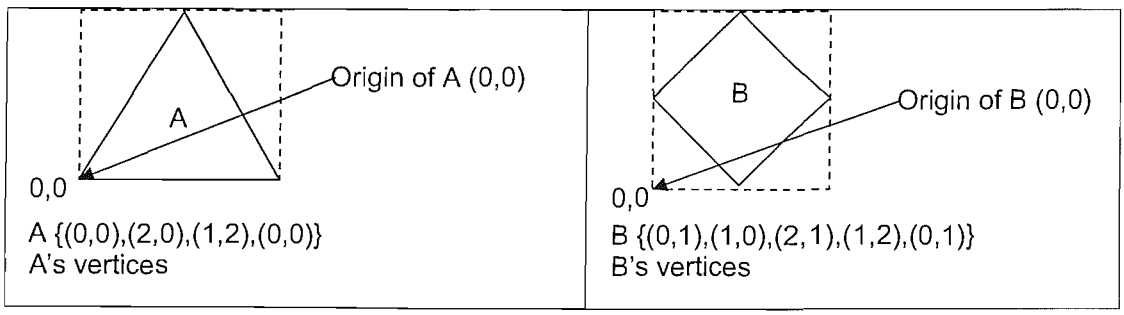


Figure 3.3 Origin Definition of a Piece

In the geometry section we have described that during the NFP generation phase we need to define the origin of the static piece, NFP and a reference point on the moving piece. We have chosen to fix both the origin and reference point as the bottom left point of the shape's enclosing rectangle. We can now utilize the data generated by the NFP generator more effectively.

We will first discuss the point inclusion test for detecting overlaps and how we use it, followed by an explanation of how we find intersections between two NFPs.

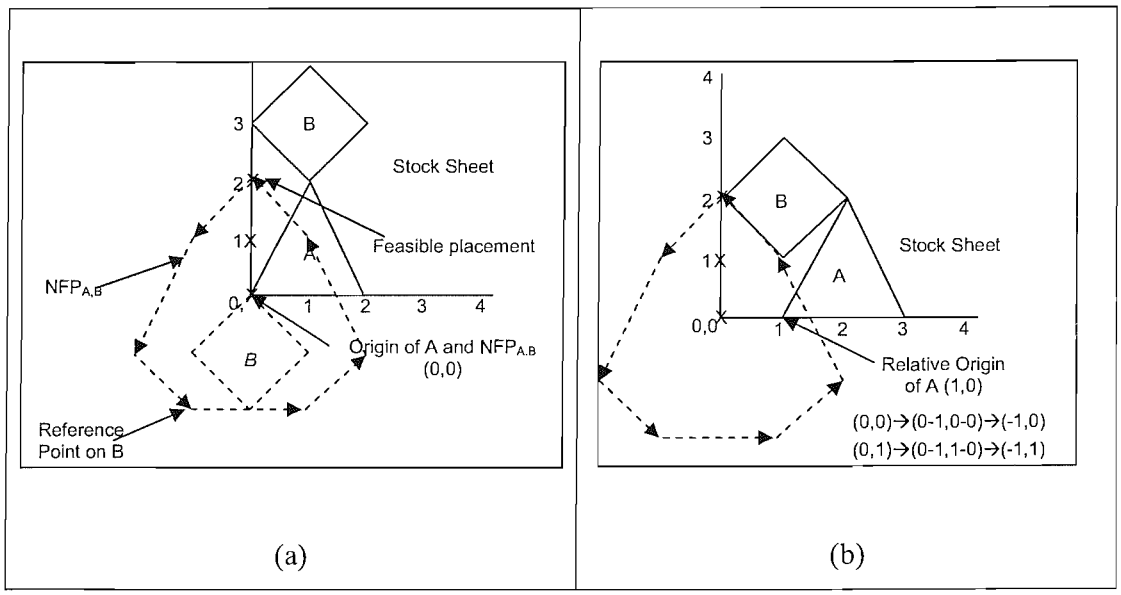


Figure 3.4 Searching for Feasible Placement Point using the NFP

Assume piece A is fixed at position at (0,0) and we need to determine the best position for piece B using NFP. Referring to figure 3.4 (a) the points (0,0), (0,1) and (1,0) are inside the $NFP_{A,B}$ so these are infeasible search points and points (0,2), (0,3), (2,0) are feasible points as it is either on or outside $NFP_{A,B}$; the point inclusion test is based on a ray crossing algorithm described in O'Rourke (1998). This test will tell us if a given point is inside or outside the given polygon $NFP_{A,B;}$ note that search points do not have to be integers. Since the most bottom left feasible point is point (0,2), piece B will be placed here. Now assume piece A is fixed at position at (1,0) as in figure 3.4(b) and using the

same NFP we have to determine feasible positions for piece B. In this case the placement points have to be translated appropriately before testing them with $NFP_{A,B}$. This translated placement point is then used in the point inclusion test; the explanation of this was given in chapter 2, section 2.2.3 where we talked about using the resultant of vector difference $u - v$, where $u = \{x_1, y_1\}$ and $v = \{x_2, y_2\}$. For example placement point (0,0) will be translated by (0-1, 0-0); where $\{x_2, y_2\}$ the relative origin of piece A is (1,0). This will give us the translated placement point (-1,0) and this new point is inside $NFP_{A,B}$. Likewise for placement point (0,1) the translated point is (-1,1) and this new point is feasible as it is on the vertex of $NFP_{A,B}$. Thus for any search point we use this convention for vector translation before testing it with the respective NFP; where $\{x_2, y_2\}$ is the relative origin of the piece already on the layout.

To find intersection between two NFP we will need to define the relative position of their NFP on the layout first. In figure 3.5 (a) we can see $NFP_{A,B}$ for piece A on the layout, this preprocessed NFP data has been entered into the computer program. In figure 3.5 (b) piece B is similar to piece A but the relative origin of piece B on the layout is (2,0), so we will translate the coordinates of $NFP_{A,B}$ to relative origin of piece B on the layout. This will give us the $NFP_{A',B}$ as shown in figure 3.5 (b). With these two NFP's in their relative position on the layout we can calculate their intersection point, the intersection points (circled) are shown in figure 3.5 (b). The intersection calculation is based on line segment to segment intersection algorithm as described by O'Rourke (1998). We will explain how we use these intersection points when we describe the approach of Gomes and Oliveria (2002) to generate feasible placement points on the stock sheet.

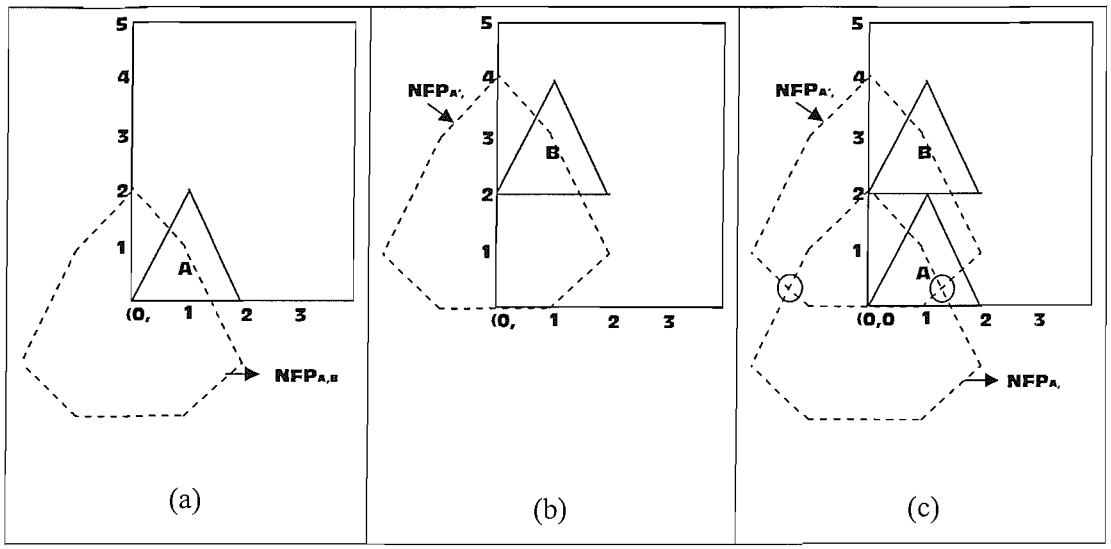


Figure 3.5 Finding Intersection between two NFP

We can start positioning the pieces onto the layout as we have defined the input data for vertices of the pieces to be packed and have generated their corresponding NFPs. We will discuss two strategies for generating our search points to position the pieces on the layout, one based on the grid point based search and the other adopted the approach of Gomes and Oliveria (2002), who used a specific method to generate a set of search points and limit the search within these points. We will explain the grid based search first, followed by the alternative method.

The packing layout represented as grid points is shown in Fig 3.6 (a). The resolution of the grid points can be controlled easily by multiplying the numbers of grid point by a certain factor to increase the precision of the available points on the layout.

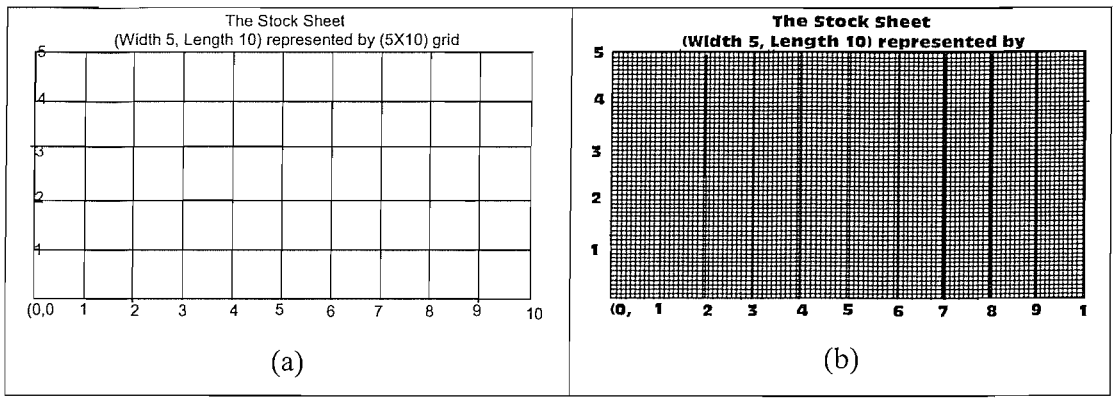


Figure 3.6 Grid Based Placement Points

In Fig 3.6(b) precision to the nearest 2 decimal point is obtained by multiplying the (x,y) grids with a factor of 10. Having defined the positions available on the layout to position the incoming pieces we will look into how we pack these pieces onto this grid based layout. Let us assume the two pieces to be packed on the layout are A and B and the packing order is pack A first then B. Assuming the bottom left most placement, we will position piece A to (0,0), this is the absolute origin of this piece and we have the pre-defined input data of this piece, so no translation needed, we can use the vertices input data of this piece to represent it on the layout as shown in figure 3.7.

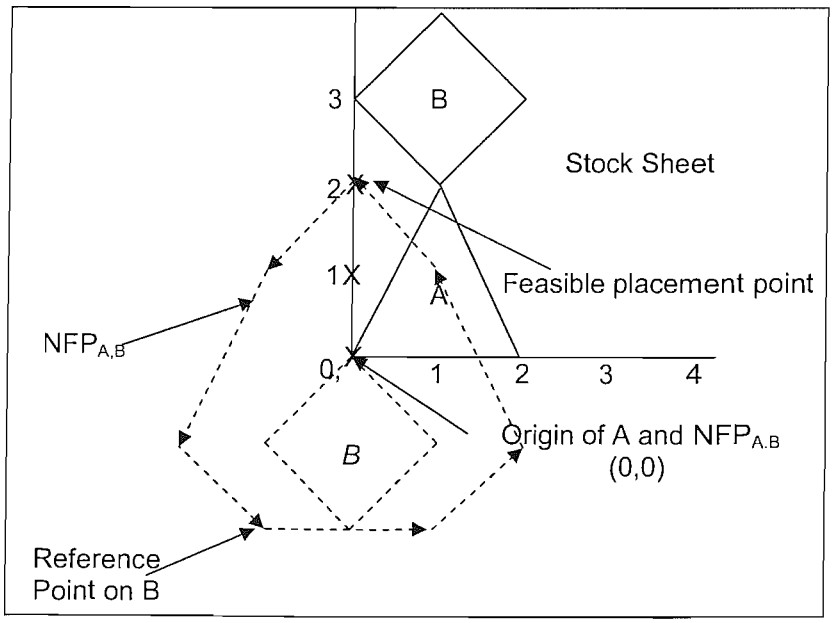


Figure 3.7 Placing Pieces on a Grid based Layout

The next piece to be packed is B, we will now use $NFP_{A,B}$, the pre-processed vertices input data of $NFP_{A,B}$ has been pre-calculated and stored. We have a stock sheet divided into discrete grid points indicating the placement points available on the layout, we need to formulate an efficient way to search through the grid points on the layout and locate the best placement for a given piece. Assuming we are implementing the bottom left most placement rule in our algorithm, thus the search for a bottom left most point will start from point (0,0) on the grid layout and increment in the direction of y axis until reaching the top of the stock sheet, if during the search a non-overlapping position is found, the search terminates. Otherwise the search increments 1 grid unit to the right of the layout and continues up the *y* axis again. This pattern is repeated until the whole area of search points in the stock sheet is searched. Referring to figure 3.7 the first search point is coordinate (0,0), obviously this point is inside the $NFP_{A,B}$, thus it is a non-feasible point. Incrementing up the *y* axis the next coordinate is (0,1), this too is inside $NFP_{A,B.}$ Next is point (0,2), this is on the vertex of $NFP_{A,B}$, we have found a feasible placement point. Using the pre-defined input coordinates of piece and translating the coordinates by (0,2) will position piece B to its bottom left most position on the layout as shown in figure 3.7. The grid based search algorithm is relatively easy to implement and is quick for low resolution grid. The quality of the solution produced depends directly on the resolution of the grid point but on the other hand using a finer grid may mean deteriorating the computational speed of the algorithm as we now have to search a bigger area of search points.

Thus in addition to the grid based search point we adopted the approach of Gomez and Oliveria (2002). They define a set of search points based on a collection of points derived from the no-fit-polygon, inner-fit-rectangle (IFR) and intersections between the lines of these polygons. We will be using the figures below to explain this concept more clearly.

The inner-fit-rectangle is a rectangle derived by sliding the shape to be packed within the stock sheet as shown in figure 3.8 (a). Placing shape C within the IFR will ensure that this piece is contained within the stock sheet. Hence our first set of search points are the points from the vertex of IFR_C as shown in figure 3.8 (a). Our second set of points will be

extracted from the vertices of the NFPs' of the pieces already placed on the stock sheet. Assuming pieces A and B are already placed on the layout, we will position $NFP_{A,C}$ and $NFP_{B,C}$ translated to the relative origin (x,y) of piece A and piece B on the layout. Our third sets of points are the intersection points between edges of $NFP_{A,C}$, $NFP_{B,C}$ and IFR_C as shown in figure 3.8 (c). To find the intersection points we pick an edge on the NFP of a piece and cycle through all edges on the NFP of the other pieces and IFR on the layout and calculate their intersections.

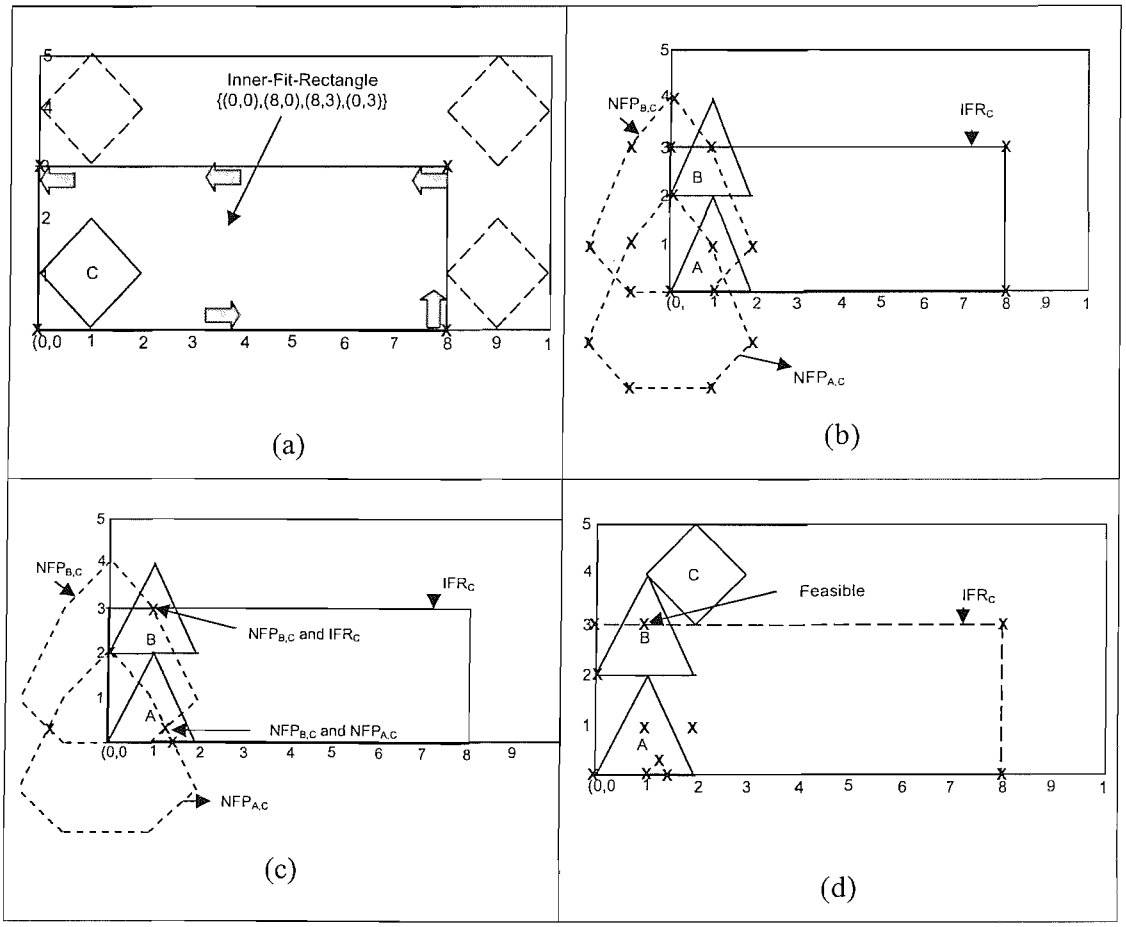


Figure 3.8

Generating Placement Points using the approach of Gomes and Oliveria (2002)

This process will cover every edge on every NFP. Now that we have a collection of search point we will eliminate the search points which are not contained within IFR_C using the point inclusion test. This will leave us with the remaining search point as shown in figure 3.8 (d). We sort these points by their lowest x coordinate, breaking ties with the lowest y coordinate following the bottom most heuristic placement rule. The search for a feasible position will start from this sorted list of search points and will be subjected to the point inclusion test using piece C's corresponding NFPs as explained earlier in this section. Thus piece C will be positioned to its feasible bottom most position, see figure 3.8 (d). The flow chart in figure 3.9 gives the steps required to generate the set of search points as explained above.

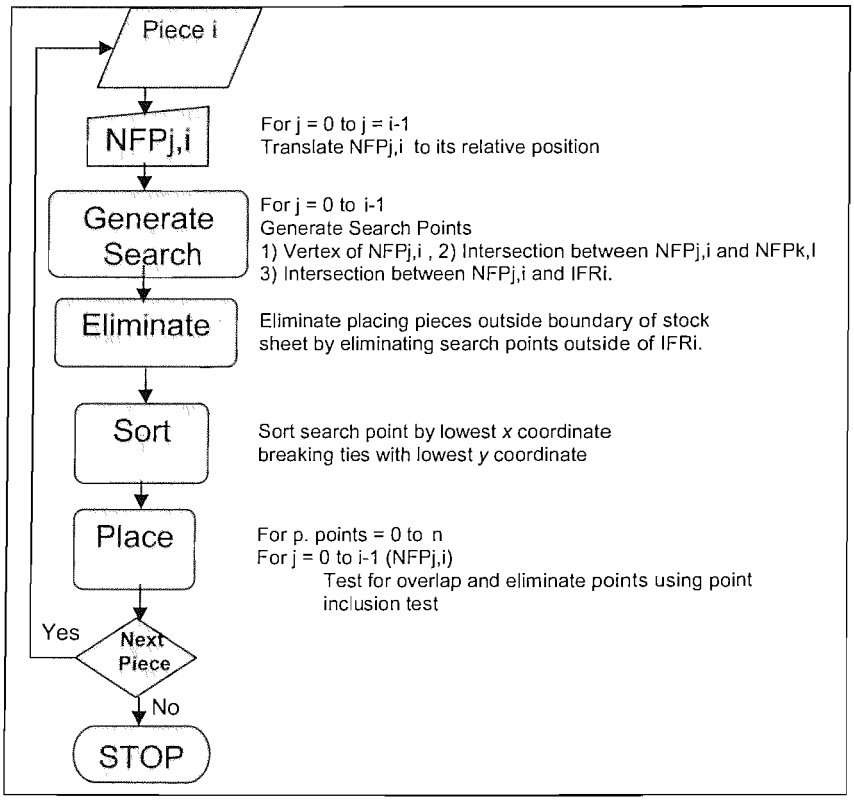


Figure 3.9

Flow Chart showing steps to generate feasible placement points

In our SOL we permit overlaps and we have to determine how we measure the layout between two pieces. The literature highlighted two main techniques, measuring exact overlap between the pieces or an approximation of overlap. We selected the later as our choice of approach as it is simple yet powerful in terms of quick computation time. Our approach is taken from Bennell and Dowsland (1999) where overlap is estimated as the minimum horizontal distance a piece has to move to remove overlap. We observed that using this approach it is possible to have more that one piece with the same overlap measure thus we made a slight modification to the overlap measure, we introduced overlap measure as the sum of minimum extent in both x and y direction as shown in figure 3.10, ties are broken randomly. We will be using this overlap measure in our cost function to select and position the pieces on the layout.

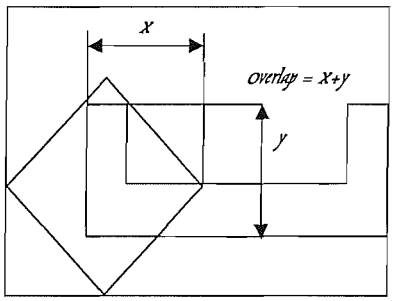


Figure 3.10 Measure of Overlap

The overlap cost function, for a given piece from a set of pieces, $\Phi = \{s1,...,sn\}$ is described in the equation shown below;

$$f(i,p) = \sum_{j=1}^{n} Oij(i,p) \quad , \text{where } i,j \in \Phi \text{ and } Oij(i,p) = 0 \text{ for } i = j; \qquad (3.1)$$

$Oij(i,p)$ is a measure of the overlap between pieces si and sj at placement p as described in figure 3.10. A placement p such that $f(i,p) = 0$ implies that at position p piece si contains no overlap.

3.1.2 Tabu Search for ICH

To recap ICH from our previous chapter; ICH involves a selection rule to select pieces to be packed on the layout from a pool of available pieces, a placement rule to find the best position for this piece on the layout with regards to the objective function and a search strategy to continuously improve the packing quality. With the geometric and placement heuristic algorithm in place, we will consider the components of tabu search which will produce the best order in which the pieces are to be packed.

In the literature two metaheuristic techniques were popular among the researches; tabu search and simulated annealing. We have chosen tabu search for our experiments as this search technique was proven successful in both iterative constructive and searching over layout approach from the literature review (Burke et al., 2006; Bennell and Dowsland, 1991). Tabu search uses a flexible memory system to restrict the next solution choice to some sub set of neighborhood of current solution.

In order to use tabu search we need to define a neighbourhood structure. This is because the combinatorial nature of the problem makes it impossible to search through all possible combinations; hence we need to limit the search within local neighbours with the objective of improving the solution and reaching a local optimum. The tabu search will search through the defined neighbourhood and return the best solution within this neighbourhood. Let us start by defining our neighbourhood and solution representation. A neighbourhood is defined as a small change made to the solution representation, which will lead us to another solution. A solution representation in our case is a sequence of pieces and the corresponding solution is the length of the final layout achieved once all the pieces in the current sequence are packed onto the layout using a placement rule; in our case we are using bottom left most heuristic as our placement rule.

We start by formulating a swap based neighbours and limit the distance of the swap to five consecutive pieces in sequence. For example consider 10 pieces in the sequence: 1,2,3,4,5,6,7,8,9,10. The possible sequences obtained by swapping piece 1 with the next 5

piece in original sequence will give us the following set of 5 sequences : { (2,1,3,4,5,6,7,8,9,10), (3,2,1,5,6,7,8,9,10), (4,2,3,1,5,6,7,8,9,10), (5,2,3,4,1,6,7,8,9,10), (6,2,3,4,5,1,7,8,9,10) }. If we continue to swap in this manner, we will get 5 new sequences for each piece in the original sequence but notice that when we reach piece 6 there will not be enough pieces left to get 5 swaps, there is no piece 11 to swap with, we tackled this by allowing the swap to be taken from the start of the sequence. The resulting set of 5 sequences : {(1,2,3,4,5,7,6,8,9,10), (1,2,3,5,8,7,6,9,10), (1,2,3,4,5,9,7,8,6,10), (1,2,3,4,5,6,10,8,9,6), (6,2,3,4,5,1,7,8,9,10)}. However we are aware that there might be more than one identical piece type in the sequence and swapping these pieces will give us the same solution, so to improve the algorithm we eliminated these types of neighbours from the neighbourhood. We shall refer to this search operator as *OptN1(d)*, where *d* defines the swap distance.

In addition to this we introduced a smaller sized neighbourhood structure; here we restrict the number of swaps in the following manner. Using the same sequence example above with a swap distance of five, we start by swapping piece 1 with the next 5 pieces in the sequence. Unlike the previous search operator, we do not proceed to swap for all other pieces in the sequence but stop here for the first iteration. The reason for doing this is to enable the search algorithm to make more tabu moves. In the next iteration we start swapping piece 2 with the next 5 pieces in the sequence, in the following iteration we start swapping piece 3 with the next 5 piece in sequence. By proceeding in this manner we give the search a chance to sample the solution space more dynamically. If there are not enough pieces left to get five swaps we allow the swap to be taken from the start of the sequence similar to our representation one. We also do not allow swaps of identical pieces in the sequence similar to *OptN1(d)*. We shall call this search operator as *OptN2(d)*, where *d* defines the swap distance.

In addition to this two search operators, we have also introduced an alternative neighbourhood structure where we apply a random sampling of 10% on neighbours

generated using search operator *OptN1(d)*. We shall call this third search operator as *OptN3(d)*, where *d* defines the swap distance.

During the search for a locally optimum solution, at each iteration all solution representation generated by the specific neighbourhood structure will be evaluated with respect to the objective of minimizing the length of the packing layout and the solution that produces the best strip length will be chosen according to the criteria determined by the local search used. This best solution representation will be called a move. There are many types of local search strategies that could be used during this stage; in a single descent local search, a first found improve type will stop once a first improving solution is found within the neighbourhood structure while a best found improve type will search the entire solution neighbourhood and choose the best. Once we have completed with the first iteration and found the best solution within this set of neighbours, we can use this current solution to start the next iteration and evaluate the next set of solution choosing the best each time. We proceed in this way until the search gets stuck in a local minimum, i.e. no improving moves can be found after a fixed number of iterations or computational runs. The following describes our tabu search algorithm for ICH.

Algorithm 1 Tabu search for ICH
Solution space *S*
Neighbourhood size *N*
Cost function *f(s)* = Length of Layout
N(so,h) denotes neighbouring solutions not contained within the tabu list
Bottom-Left-Heuristic is implemented according to steps given in Figure 3.9
Input: Problem Shapes and their NFPs, Sorted Orderings, Quantities, Allowable Rotations and Sheet Size

Step 1:

Build an initial solution, *so* = *Bottom-Left-Heuristic(Sort_Ordering)*
Select tabu list length, *l*
Set tabu history *h(l)* empty

Set *sbest* = *so*

Initialize aspiration criteria = *sbest*

Step 2:

Select Search Operator = {*OptN1(d), OptN2(d), OptN3(d)*}

Repeat

Apply the selected search operator for defining the neighbourhood to explore

Explore this neigbourhood using the *Bottom-Left-Heuristic*

Select *s* such that *f(s)* < *f (so)*

Replace *so* by *s*;

Until *f(s)* > *f(so)* for all *s* € *N(so,h)* or the aspiration criteria is met

Step 3:

sbest = *s*

Tabu move = sequence of pieces swapped to get *sbest*

Update tabu history *h(l)* = tabu move

Update aspiration criteria = the best *sbest* found so far

Step 4:

If stopping condition not true

Return to Step 2

Otherwise stop

Our tabu search is based on the best found improve method but unlike the best found improve we accept worse solution from successive iterations (see Step 4) and carry on with the search for a fixed duration of time. Applying the principles of tabu search to our neighbourhood of solutions representation will mean that using the best improving solution representation found a tabu move is the record of pieces swapped to generate this solution representation. The tabu list will record this forbidden move, e.g. if (**2**,**1**,3,4,5,6,7,8,9,10) is the best solution representation from one of the iterations then the tabu list will record {(**2**,**1**)}, the pieces swapped as one of its forbidden move. In the next successive iteration when the possible solution representations are generated according to the defined neighbourhood structure this pieces will not be allowed to swap as they are in

the tabu list unless the aspiration criteria is met, a move from the tabu list can result in a solution better than the best solution found so far. In all iterations, from the best solution representation of that iteration, the swapped positions will be stored in the tabu list. Once the maximum size of tabu list is reached the oldest tabu move in this list will be removed and the new tabu move will occupy its place. We can experiment with different tabu list length sizes to determine the effects this length has on the final solution achieved. Figure 3.11 below demonstrates a sample run from tabu search for ICH.

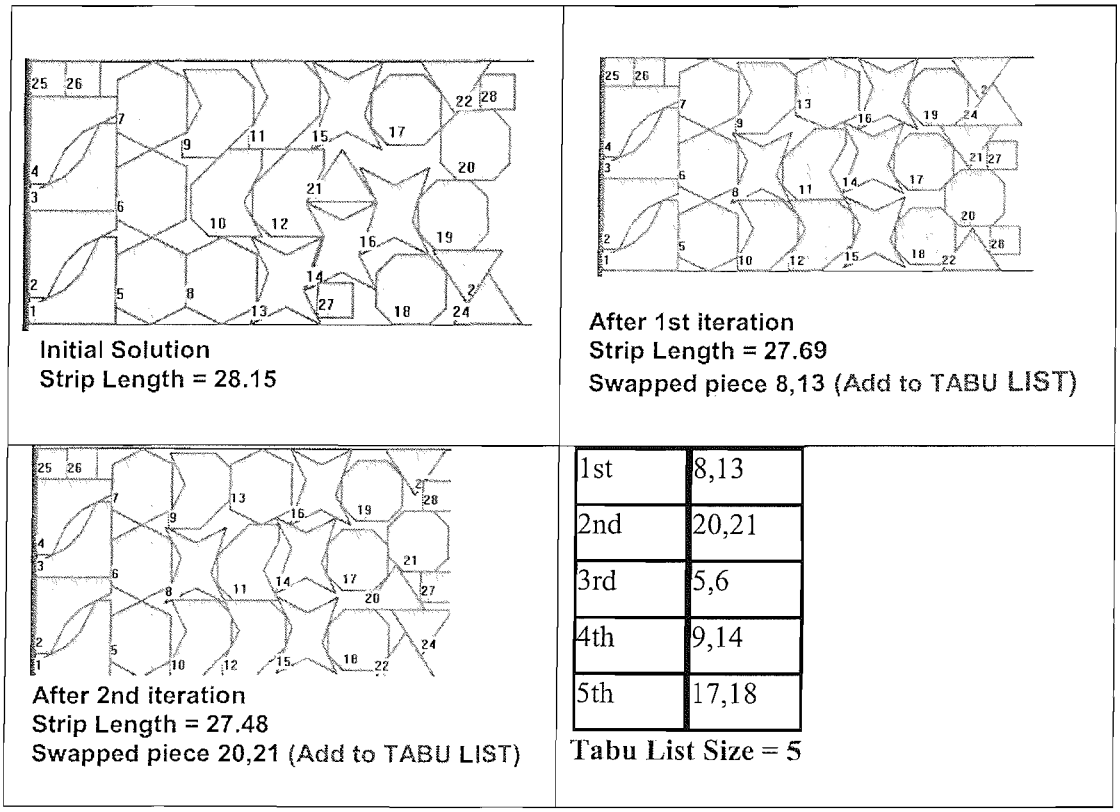


Figure 3.11 Stages during tabu search execution

3.1.3 Tabu search for SOL

To recap SOL from our previous chapter; SOL starts from an initial layout with pieces already arranged on the stock sheet and seeks to design a better layout by moving these pieces around within the stock sheet. An important difference in this approach is pieces are allowed to overlap one another on the layout. Having defined our overlap measure let us move on to steps involved in generating an initial solution for SOL.

The literature reviewed many ways of how the initial solution may be generated. Our initial solution is similar to that of Egeblad et al. (2007). Here we randomly order the pieces and pack the pieces on the layout using the bottom-left most placement heuristic. Then we find out the strip length for this initial layout, reduce the strip length by some value and move all the pieces not contained within this strip length to a random position on the layout within this new strip length, this will result in overlap on the layout. We will then attempt to eliminate overlap by moving pieces around this constrained layout. The following describes our tabu search algorithm for SOL.

Algorithm 2 Tabu search for SOL

Search positions p are generated according to the algorithm in Figure 3.8

$f(i,p)$ = cost function, calculates overlap measure of each piece with other pieces in layout, see equation 3.1

$N(po,h)$ denotes neighbouring solutions not contained within the tabu list

Bottom-Left-Heuristic is implemented according to steps given in Figure 3.9

Input: Problem Shapes and their NFPs, Sorted Orderings, Quantities, Allowable Rotations and Sheet Size

Step 1:

Build an initial solution, so = *Bottom-Left-Heuristic(Sort_Ordering)*

Select tabu list length, j

Set tabu history $h(j)$ empty

Set $sbest = so - 1$, strip length

Set aspiration criteria = 0 (zero overlap in entire layout)

Step 2:

Move pieces not contained within the new layout length into the layout

Repeat

Calculate total overlap cost, $f(i,p)$ for each piece on the layout

Choose the piece with the maximum cost, $f(i,po) = f(i,p)$

Rotate Piece = true or false (determined at random)

Generate search positions for this chosen piece

Explore these search positions

Select a search position such that $f(i,p) < f(i,po)$ for all $p \in N(po,h)$ or the aspiration criteria is met

Move this piece to this new position on the layout

Tabu move = piece type and the old position in layout

Update tabu history $h(j)$ = tabu move

Until overlap cost of all the pieces on the layout = 0

Step 3:

If stopping condition not true

$sbest = so - 1$

Return to Step 2

Otherwise stop

To identify the pieces that are in overlapping positions we use the point inclusion test by cycling through NFPs' of this piece with already placed pieces on the layout. Once an overlapping piece is selected we need to formulate where this piece will be relocated. The literature suggests many ways of doing this as we have seen in section 2.4. We have introduced a new approach. Our set of search points generated for this piece will be based on collection of points derived from the no-fit-polygon, inner-fit-rectangle (IFR) and intersections between them as explained in Section 3.1.1.

Referring to figure 3.12c let's say piece 41 is the piece selected to be move within the restricted layout, search points will be generated for this piece similar to the approach given by Gomes and Oliveria (2002), assuming that all other pieces are already placed on

the layout and piece 41 is the next piece to be packed. We decided these as our placement positions because in these positions piece 41 will be contained within the restricted layout and will be touching with at least one other or more piece in the layout. The search attempts to eliminate the overlaps within this fixed strip length. Once an overlap free layout is obtained the strip length will be reduced again and the same process continues again and again to reach better solutions.

Unlike in ICH where the neighbourhood structure was based on the positions of pieces in the predefined sequence, here the neighbourhood will be based on the generated placement points on the layout. Thus in our search all the possible placement positions will be considered. The first step of any search algorithm is formulating the initial solution, we construct our initial solution by ordering the pieces either randomly or presorted by decreasing area size of the pieces and packing them one by one onto the strip layout (Figure 3.12a).

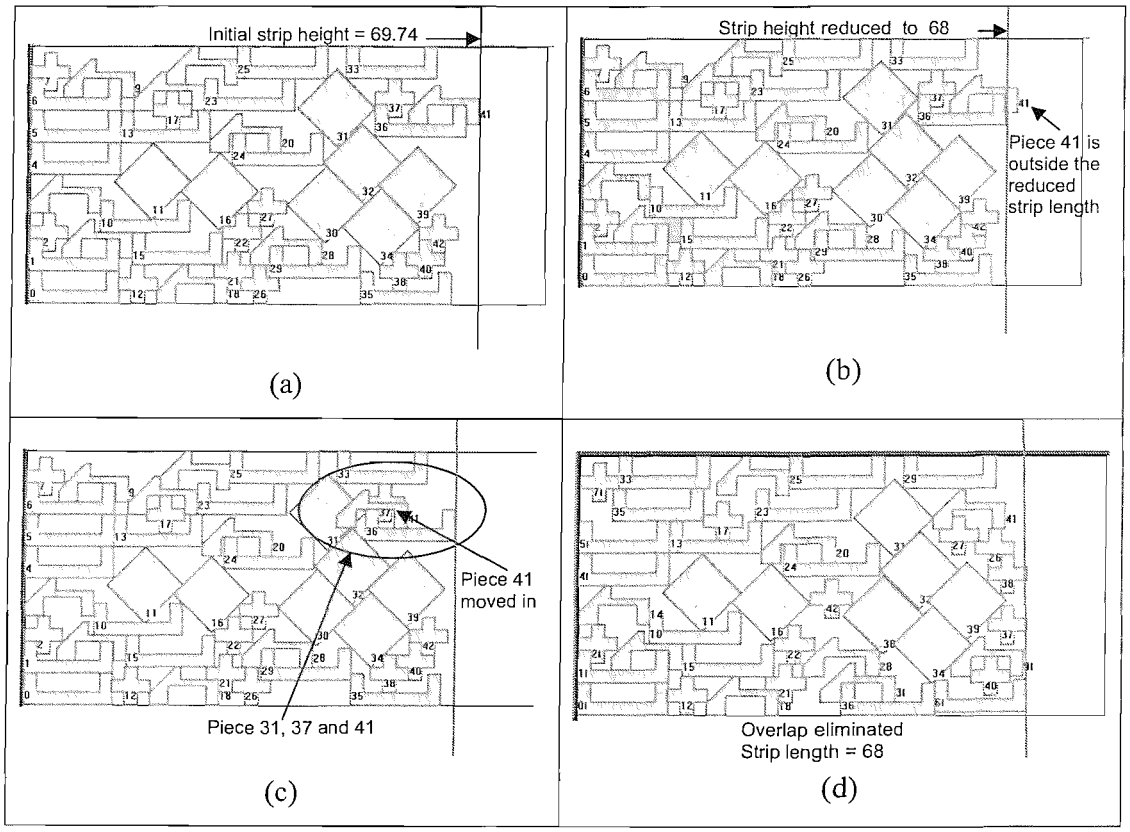


Figure 3.12 SOL in Execution

Once we have this initial solution we will attempt to improve the initial strip length, we reduce the initial strip length by some value (usually by one) and translate pieces no longer contained within these reduced length back into the layout (Figure 3.12a and c), this will create overlaps in the layout. We allow overlap and use our tabu search to reduce and eliminate the overlaps within the layout which brings us to the second component of the tabu search, the evaluation function.

We will be introducing a cost function to evaluate and choose the piece to be relocated on the layout and where it will be positioned. This cost function is described by equation 3.1 in section 3.1.1. For choosing the piece to be moved we will select the piece having the largest total sum of overlap in the layout, this will be the total sum of overlap between this piece and the rest of the piece on the layout as given in equation 3.1. Referring to figure 3.12c, the total sum of overlap for piece 41 will be the sum of overlap measure of this piece with piece 31 and 37. After having selected a piece we randomly rotate this piece to its allowable orientation or use the non-rotated form. We then evaluate where to position this piece to within the layout. Since our neighbourhood structure is all available placement points on the layout, generated using Gomes and Oliveira (2002) approach, all of these positions will be evaluated individually and the position that yields the smallest total sum of overlap will be selected. This piece will then be moved to this new position. The objective of this search is therefore to attempt to reduce and ultimately eliminate any overlaps in the layout. If there are no more pieces in overlap positions then the first iteration terminates otherwise the same cycle is repeated again and again till there are no more overlap in the layout.

The reasons for choosing maximum and minimum overlap measure as our cost function for selecting and moving the piece within the layout is because this is the widely popular choice used in SOL (e.g. Egeblad et al. (2006), Bennell and Dowsland (1999)). Once overlap is eliminated the strip length will be reduced again and the whole process is repeated again and again till the search gets stuck and can no longer eliminate overlaps within the layout.

If there are no restrictions as to which piece or search position could be selected we could end up in a situation where the solution will keep cycling by picking the same piece or position again and again, we would like to avoid this in our algorithm by using an appropriate tabu list. Tabu list contains record of the forbidden move; in our case we selected piece and its position on the layout as this forbidden move. Recall from the previous section that we choose the piece and move it around in the layout based on our cost function, once the piece is selected and relocated to a new position on the layout, this piece and its old position (before being moved) will be stored into the tabu list. The piece and its position recorded in our tabu list will not to be selected in the next successive iterations until they expire from the tabu list unless the aspiration criteria is met, a move from the tabu list can result in a overlap free layout. The pieces and its corresponding position recorded in our tabu list will prevent pieces on the tabu list from being positioned back to their historical position on the layout as long as they are still on the tabu list. By controlling the tabu list length we can easily experiment with the behaviour of the algorithm, a shorter tabu list length possibly restrict movements within overlapping pieces only and a longer tabu list might force pieces in non overlapping positions to be moved, introducing a big perturbation to the search. Similar to the tabu search introduced with ICH we can experiment with different initial solution to start the search and tabu list sizes to determine the effects this has on the final solution achieved.

3.2 Summary

In this chapter we have introduced the geometric computation routines common to irregular packing problems and how we have tackled them. We also described our placement strategy and our search strategy for ICH and SOL. In the next chapter we will be looking at the test parameters that we will be defining for ICH and SOL implementations.

4 Implementation

We have given a detailed description of the methodologies and tools that we have developed for solving the packing of irregular shapes in the previous chapter. As we have mentioned earlier the shapes data were drawn from the EURO Special Interest Group on Cutting and Packing (ESICUP) website and their corresponding NFP data provided by Bennell and Song (2008). With this we are ready to test the two mainstream approaches used to solve a two dimensional irregular packing problem and we will investigate their performances using the packing tools that we have developed. In this section we will look into the experimental design of our algorithm implementation to be used on the benchmark datasets that we have gathered from ESICUP. We have outlined this section into looking at our experimental data sets, placement heuristic, search heuristic, evaluation function, solution space and neighbourhood structure, and our starting solution with respect to our ICH and SOL.

The programming language we have chosen for this project is Microsoft Visual C++.Net 2003 edition as this software provides the platform for graphical utilities which is useful in displaying the final solution of the packed layout graphically. The coded algorithm was executed on a desktop PC which uses a Pentium IV, 2.8 GHz and 504MB of RAM.

4.1 Experimental Data

Appendix A shows the 10 data sets used for the experimental study, their corresponding author name, year published, the problem name as known in the literature, number of pieces and types, stock sheet width and their permitted rotations. These data sets can be divided into 3 groups; the first group (see Appendix A.1) is an artificial data set, group 2 is a real data set from the industry (see Appendix A.2) and the third group (see Appendix A.3) is a jigsaw puzzles whose optimum solutions are known.

4.2 Iterative Constructive Heuristics (ICH)

4.2.1 Placement Heuristic

Given the set of input data comprising the shapes of pieces to be packed, their corresponding stock sheet sizes and their NFPs' we can start positioning the pieces onto the layout. We have implemented two representations for generating placement points on the layout, one is the grid based placement point and the other is the set of points derived from collections of points from the no-fit-polygon, inner-fit-rectangle (IFR) and intersections between the lines of these polygons as described in the methodology section. However we decided to use the latter in analysis of our final results. This was based on our initial investigation where the latter proved to be more efficient in terms of computational speed and quality of solution especially when the data sets are of highly irregular shapes and require double digit decimal precision results. These characteristic are typical of industrial data as shown in Appendix A.2.

The placement heuristic will be based on bottom most heuristic. We have described in our methodology section. We have discussed how bottom most heuristic became the most popular choice for placement heuristic in chapter 2.

4.2.2 Search Heuristic

The main search method used will be Tabu Search as we have mentioned in the previous chapter. We experimented with five different tabu list length; these are 5, 10, 20, 50 and 100. As is normal in tabu search we accept non-tabu move even it is non-improving. This prevents cycling and diversifies the search away from its current best solution, hoping our search will explore into other regions of the solution space.

4.2.3 Evaluation Function

Our evaluation function is based on the minimum strip length in that particular iteration or neighbourhood of solution once all the pieces are packed onto the layout. Our strip length can be found by keeping track of the relative position of the pieces as they are placed on the layout and locating the vertex with the highest x coordinates and storing it in the memory.

4.2.4 Solution Space and Neighbourhood Structure

We have introduced in the previous chapter the concept of solution and neighbours and how these are used to aid in the search towards finding a better strip length. Our solution is the sequence or the order list of pieces and their corresponding strip length once all the pieces in the sequence are packed one by one onto the layout. We have designed quite an extensive neigbourhood structure for our experiments as quite often this is the key to achieving good solutions. A good neighbourhood structure will enable the search to sample the infinite solution space most efficiently. Thus we have come up with a number of alternatives for our neighbourhood structure. Our basic idea is similar to the one we have described in the previous chapter, generating neighbours using swap distance. The bigger the neighbourhood size the more solutions will be sampled but will increase the computational overhead. We have introduced sampling into the neighborhood structure where only a small percentage from the entire set of possible solutions in the neighborhood will be sampled; this will be a random sampling of 10%. Thus we have defined three representations for our neighbourhood structure, the first one is based on the set of possible permutation within a given swap distance, the second is a subset of the first representation derived by applying a special restricting condition and the third is applying random sampling of 10% to our first representation. These representations are discussed in detail in the Chapter 3, Section 3.1.2.

Manipulating our three solution representations we then come up with a total of five different neighbourhood structures types to be used with our search engine. These are generated by applying a specific swap distance on the first, second and third representation of our neighbourhood structure. This will give us five different types of neighbourhood structures. We will call them *OptN1(5), OptN2(5), OptN2(10), OptN3(5)* and *OptN3(10)*, refer to Chapter3, Section 3.1.2. *OptN1(5)* is our biggest neighbourhood while *OptN2(5)* is our smallest.

4.2.5 Starting Solution

To generate the initial starting solution we test two different ranking criteria; i) sort the pieces to be packed by their decreasing area and ii) generate a random packing order. We then pack them one by one onto the layout using the bottom most placement heuristic, this will give us our initial strip length.

4.3 Searching Over Layout

4.3.1 Search Heuristic

We will also be using Tabu Search as our main search engine for SOL. We implemented two tabu representations. The first one uses the piece that was moved in the layout as the tabu candidate. The second uses the combination of the piece that was moved and the position of this piece in the layout. To avoid piece from being placed in close proximity to its' previous position, we imposed a tolerance of +/- one unit way from its previous position.

In the first representation our tabu list size is restricted by the quantity of shapes in a shapes data file. For example referring to Appendix A.1, data set 1 has 43 pieces so we could not have a tabu size length exceeding 43 since this would lead to all the pieces being on the tabu list and none can be moved again.

We decided to have three different tabu sizes and we will call them as type low, medium and high. A low tabu size length will be calculated as one quarter of the quantity of the shapes rounded down to the nearest integer, a medium type as half of the quantity of the shapes rounded down to the nearest integer and a high type as three quarter of the quantity of the shapes rounded down to the nearest integer. For example using shapes data 1, a low type tabu size length is 10, medium is 21 and high is 32. In our second representation we are not restricted by the quantity of the shapes so we will try out 5 different tabu list sizes like ICH; 5, 10, 20, 50 and 100.

4.3.2 Evaluation Function

As we have explained in the previous chapter the cost function will store the total amount of overlap for a given piece within the layout. The overlap measure (see Fig. 3.10, Chapter 3, Section 3.1.1) can be used to determine whether polygons overlap and how much they overlap. Thus to find the total amount of overlap for a given piece, we will calculate the overlap amount of this piece with one other piece on the layout, equation 3.1 in Section 3.1.1 of Chapter 3 describes this evaluation cost function.

If this piece is in an overlapping position with more than one piece then we will calculate all the overlaps and add them up to find the total amount of overlap for this piece. The evaluation cost function used for choosing the piece to be moved on the layout is based on maximum amount of overlap and to position this chosen piece within the layout the position that yields the least amount of overlap will be used. We keep moving the pieces around within the restricted strip length until the overlap is totally eliminated before proceeding to further reduce the strip length, refer to Chapter 3, Section 3.1.3.

4.3.3 Solution Space and Neighbourhood Structure

Unlike ICH the entire neighbourhood will be searched. A neighbour is defined as all the available placement positions on the layout. The available placement positions for the piece chosen to move on the layout are given by the set of points derived from the no-fit-polygon, inner-fit-rectangle (IFR) and intersections between the lines of these polygons for this chosen piece as explained in detail in the previous chapter. This idea of generating the placement points was originally implemented for ICH by Gomez and Oliveria (2002) but we are the first ever to apply it for SOL. The rationale for doing this is that search positions generated this way are position where this piece will be touching with at least one other piece on the layout.

4.3.4 Starting Solution

To generate the initial starting solution we tested two different criteria; i) sort the pieces to be packed in their decreasing area size and ii) generate a random packing order. We then pack them one by one onto the layout using the bottom most placement heuristic as we did with ICH. We calculate the initial strip length from this layout and we then decrease this length by one. Pieces which are no longer contained within the reduced strip length are translated horizontally back into the feasible packing region, which then creates the overlaps in the layout. These steps are explained clearly in the chapter3, section 3.3.3, where we also discuss which aspects of the work of Egeblad et al. (2007) we adapted to develop our SOL mechanism.

4.4 Summary

In this chapter we gave an overview of our implementation of ICH and SOL heuristics which consists of placement heuristics, tabu search heuristic, evaluation function, neighbourhood structure and their initial solutions. We also described the comparison test parameters that we have developed. We are now ready report on our experiments on both approaches and evaluate the effects of the different test parameters on the solution quality. In the next chapter we will discuss the results and draw some conclusions with respect to strengths and weakness of these approaches.

5 Experimentation and Results

We will break down this section into discussing results we obtained during our initial ICH and SOL algorithm design phase especially the search heuristic and the neighbourhood structure. These results greatly influenced our final experimentation strategy which we then used to compare solution from both ICH and SOL.

5.1 Iterative Constructive Heuristics (ICH)

We carried out extensive experiments using the placement and search heuristic strategy as explained in the implementation chapter. Recall that the available search points were derived from the no-fit-polygon and inner-fit-rectangle, we employed a bottom left most placements heuristic and our search strategy was based on tabu search. The motivation behind our neighbourhood structure design strategy was to develop a structure that will sample the solution space most effectively. We have implemented 5 types of neighbourhood structure which we *OptN1(5), OptN2(5), OptN2(10), OptN3(5)* and *OptN3(10)*, refer to Chapter3, Section 3.1.2. *OptN1(5)* is our biggest neighbourhood while *OptN2(5)* is our smallest. The impact of the neighbourhood size is, the bigger the size the longer it will take for the tabu search to accept a move while the smaller it is the faster it will accept a move. However with bigger sizes the tabu search is allowed to sample more neighbouring solution thus increasing the chances to reaching a better solution within a promising neighbourhood structure. This is what we are interested in finding out with our computational experiments. We have also implemented two types of ranking criteria test for the initial packing order, the first is based on area decreasing packing order and the next is a random packing order.

Initially we wanted to evaluate the effectiveness of the different types of neighbourhood structure that we have proposed using our ICH based on tabu search. Thus we limited our experiments to only five data sets to test the algorithm; these are Shape0, Shape2, Trousers, Shirt and Dagli.

The reason for choosing these data sets is because they are quite popular with researchers as the best benchmark data sets. However in our final experiments we will be considering 10 data sets in total. Our tabu search will be tested using five different tabu lists of varying lengths; which are 5, 10, 20, 50 and 100. We will be running the experiments for the duration of 30 minutes so that we can evaluate the effectiveness of the neighbourhood structure more accurately. We realize that during the course of the iterations there may exist more than one solution or packing order that produces the shortest strip length, so we break ties by selecting one of these arbitrarily. The experiments were replicated five times for each of the five tabu list given and best solution from this five runs will be reported. Using all this parameters we will generate a total of 50 experiments of 30 minutes each for a data set, to run the experiments for the one data sets will take us around 25 hours in total for a neighbourhood structure type, thus for the 5 types we need 125 hours. Thus to complete the entire experiment for the five data sets we need 625 hours or almost 4 weeks.

The table of results is given in Table 5.1 (decreasing area initial packing order) and Table 5.2 (random initial packing order) where the best results are shown in bold. Referring to table 5.1, it is not a surprise that we are getting consistent result with our *OptN1(5), OptN2(5)* and *OptN2(10)* neighbourhood structures where best result and the average from the five replications of the experiments appears to be the same. It is natural for the search algorithm initialized with the area sorted ordering to converge to better solutions faster and the systematic manner in which the neighbourhood was generated as explained in Chapter 4, Section 4.2.4 makes the search for better solution very deterministic. However our *OptN3(5)* and *OptN3(10)* has the element of randomness in their solution as we used a random sampling strategy to generate their neighbourhood. We also observed that *OptN3(5)* structure was consistently giving good quality solutions in all five data set tested. *OptN1(5)* structure did not do particularly well on Shape 0. This can be explained because these data sets contain large quantity of pieces making each neighbourhood highly computationally expensive and thus reducing numbers of neighbourhood searched

within the restricted run. The fact that Shape 0 is not allowed to rotate can also lead to some good positions being ignored by the algorithm. We drew this conclusion because *OptN1(5)* was performing better in data set Trousers and Shirt which have bigger neighbourhood sizes compared to Shape 0 but as these shapes are allowed to rotate they produced competitive results. When considering the tabu list length we found the best solution obtained from any of the structure types always falls within tabu list length of 10, 20 and 50. Only in some instances the best results obtained by tabu list length of 5 and 100 matched that of 10, 20 and 50. However it was never a better performer.

ICH Neighbourhood Structure Experiments

Sorted Order

Neighbour		Shape 0		Shape 2		Trousers		Shirt		Dighe1	
Structure	Tabu L	Best L	Avg L	Best L	Avg L	Best L	Avg L	Best L	Avg L	Best L	Avg L
O	5	66.00	66.00	27.16	27.38	251.31	251.61	63.88	64.69	125.41	125.41
P	10	66.00	66.00	27.16	27.38	251.22	251.39	**63.58**	63.78	119.30	119.30
T	20	66.00	66.00	**27.00**	27.36	249.75	250.17	**63.58**	63.78	**112.07**	112.07
N1	50	66.00	66.00	**27.00**	27.36	250.96	251.40	**63.58**	63.78	119.57	119.57
(5)	100	66.00	66.00	**27.00**	27.36	251.31	251.49	**63.58**	63.78	120.27	120.27
O	5	64.99	64.99	27.66	27.79	251.45	251.79	63.96	63.96	**112.07**	112.07
P	10	65.00	65.00	27.78	27.90	251.11	251.35	63.96	63.96	113.62	113.62
T	20	63.00	63.00	27.54	27.61	250.00	250.57	63.96	63.96	113.69	113.69
N2	50	65.00	65.00	27.86	27.89	249.59	251.05	63.96	63.96	122.79	122.79
(5)	100	65.00	65.00	27.76	27.98	251.71	252.14	63.96	63.96	126.94	126.94
O	5	64.00	64.00	27.48	27.49	249.61	251.67	63.65	63.81	112.82	112.82
P	10	**62.00**	62.00	**27.00**	27.34	251.36	252.27	63.65	63.65	**112.07**	112.07
T	20	63.00	63.00	27.40	27.47	250.96	251.11	63.65	63.66	112.82	112.82
N2	50	63.00	63.00	27.49	27.49	249.61	251.37	63.65	63.75	113.93	113.93
(10)	100	63.00	63.00	27.38	27.48	251.36	251.74	63.65	63.75	121.45	121.45
O	5	64.00	64.33	27.40	27.52	250.42	251.47	63.73	64.17	112.82	112.82
P	10	**62.00**	62.67	27.38	27.46	**248.08**	251.10	**63.58**	64.01	112.82	114.62
T	20	63.00	63.83	27.50	27.55	**248.08**	251.25	63.63	64.03	**112.07**	113.64
N3	50	62.49	63.16	**27.00**	27.31	**248.08**	251.25	63.63	64.13	119.75	122.45
(5)	100	**62.00**	62.69	27.81	27.89	**248.08**	251.25	63.63	64.13	121.43	120.18
O	5	**62.00**	62.67	27.34	27.52	251.24	251.48	63.97	64.20	112.82	112.82
P	10	63.00	63.83	27.26	27.48	250.00	250.00	63.64	63.82	112.82	112.82
T	20	62.99	64.00	27.20	27.57	250.00	250.00	63.62	63.81	112.82	113.19
N3	50	63.00	64.33	27.26	27.48	250.00	250.00	64.42	65.10	112.82	116.40
(10)	100	63.00	64.00	27.26	27.48	250.00	250.00	63.62	63.63	118.62	120.08
Best Know Solution		59.47		25.84		240.77		61.33		100.00	

Table 5.1 ICH Experimental Results with Area Decreasing Initial Order

ICH Neighbourhood Structure Experiments

Random Order

Neighbour		Shape 0		Shape 2		Trousers		Shirt		Dighe1	
Structure	Tabu L	Best L	Avg L	Best L	Avg L	Best L	Avg L	Best L	Avg L	Best L	Avg L
O	5	64.08	65.36	27.90	28.24	266.88	271.01	67.39	68.22	134.65	141.04
P	10	64.08	66.03	28.16	28.32	258.95	266.16	66.80	67.09	113.69	125.57
T	20	64.08	65.36	27.80	27.94	261.41	268.93	66.92	67.48	112.09	125.38
N1	50	64.08	65.36	27.80	28.01	267.06	271.07	67.24	67.32	113.69	117.60
(5)	100	64.08	65.36	27.80	28.01	258.95	264.14	67.37	69.65	125.44	126.04
O	5	65.00	65.00	27.86	28.06	268.20	273.46	67.37	69.04	**112.07**	112.32
P	10	64.83	65.11	27.81	28.03	265.51	269.41	**66.13**	67.40	112.09	115.02
T	20	65.16	65.72	27.66	28.13	**254.83**	267.09	66.95	67.31	112.82	114.19
N2	50	65.00	65.55	28.18	28.33	268.20	270.68	66.17	67.55	113.69	119.08
(5)	100	64.58	64.86	28.03	28.26	257.27	264.06	**66.13**	68.43	122.07	126.88
O	5	64.00	64.93	28.13	28.30	269.93	271.31	67.13	68.49	122.98	128.87
P	10	64.00	65.33	27.49	27.74	270.00	274.12	66.58	66.74	**112.07**	116.97
T	20	64.08	65.05	27.30	27.65	264.31	272.97	66.78	67.07	**112.07**	112.33
N2	50	64.00	65.33	27.50	27.76	264.71	269.55	66.89	67.01	112.82	115.94
(10)	100	64.00	65.33	27.49	27.84	261.25	267.09	66.58	69.09	117.77	121.89
O	5	63.00	63.72	27.40	27.61	267.05	270.19	66.86	68.55	112.82	113.60
P	10	64.00	65.00	27.34	27.46	267.05	270.09	66.86	68.55	112.82	113.10
T	20	64.00	65.17	**27.18**	27.68	267.05	270.27	66.86	68.55	112.82	116.56
N3	50	63.50	64.17	27.57	27.92	267.05	270.27	68.86	68.55	116.67	120.90
(5)	100	64.50	65.08	28.15	28.25	267.05	270.27	68.86	68.55	118.22	121.88
O	5	64.00	64.69	27.73	27.85	264.45	265.73	67.30	67.89	112.82	112.82
P	10	**62.99**	63.69	27.73	27.82	264.45	265.73	66.35	66.89	**112.07**	112.57
T	20	64.00	64.03	27.77	27.96	264.45	265.73	66.52	66.96	112.09	112.72
N3	50	64.00	64.03	27.80	28.01	264.45	265.73	66.28	66.79	113.93	117.87
(10)	100	64.00	64.03	27.80	28.01	264.45	265.73	67.42	69.53	119.90	126.84
Best Know Solution		59.47		25.84		240.77		61.33		100.00	

Table 5.2 ICH Experimental Results with Random Initial Order

The sorted packing order ranking criteria dominated the best results achieved as shown in bold in Table 5.3. However random packing order did show some promise and matched some of the best results, as shown in bold italic in Table 5.3. Based on our results we did not perform better than the current best results achieved in the literature but this was not our goal when we started our research. However our result did perform better than previous best in the literature. Shapes 0 data set, achieved a strip length of 62 and this was better than best known solution of 63 till 2006 (Dowsland et al., 1998). Shapes 2 data sets achieved a best of 27 and this was better than previous best of 27.2 (Burke et al., 2006), the authors reported having to extend their initial experiments to achieve this result. Overall we feel that we have developed a good implementation of the ICH algorithm and we can now proceed with our SOL performance testing.

Neighbourhood Type	Shape 0		Shape 2		Trousers		Shirt		Dighe1	
	Best Length		Best Length		Best Length		Best Length		Best Length	
	Sorted	Random	Sorted	Random	Sorted	Random	Sorted	Random	Sorted	Random
OptN1(5)	66.00	64.08	**27.00**	27.80	249.75	258.95	**63.58**	66.80	**112.07**	112.09
OptN2(5)	63.00	64.58	27.54	27.66	249.59	254.83	63.96	66.13	**112.07**	**112.07**
OptN2(10)	**62.00**	64.00	**27.00**	27.30	249.61	261.25	63.65	66.58	**112.07**	**112.07**
OptN3(5)	**62.00**	63.00	**27.00**	27.18	**248.08**	267.05	**63.58**	66.86	**112.07**	112.82
OptN3(10)	**62.00**	62.99	27.20	27.73	250.00	264.45	63.62	66.28	112.82	**112.07**
Best Known	59.47		25.84		240.77		61.33		100.00	

Table 5.3 ICH Best Result Comparison

5.2 Searching Over Layout Heuristics (SOL)

We carried out extensive experiments using the two SOL search strategies as explained in the implementation chapter 4, section 4.3. Recall that the available search points for the piece selected to move within the layout were derived from the no-fit-polygon and inner-fit-rectangle (IFR). Our initial solution as with ICH is generated using the two ranking criteria as explained in section 5.1. We then use the bottom left most placement heuristic to position these pieces in the layout to build our initial solution. Next we improve the solution by decreasing the initial strip length and resolving overlaps as was explained in chapter 4, section 4.3.4. Our first representation of tabu search stores the piece that was selected to move into the tabu list. This strategy will be tested using 3 different tabu list length test parameters, which was classified as low, high and medium as explained in the previous chapter 4, section 4.3.1. Our second representation of tabu search stores the combination of the piece selected to move and its current position into the tabu list and five tabu list length types are tested. In the first strategy once a piece has been moved within the layout this piece will not be moved again until it drops off from the tabu list and this is determined by the tabu list length, the shorter the length the faster it will be available for a move and the longer the length, pieces which may not be in overlap positions will be selected to be moved within the layout. This will potentially create big gaps in the layout and force the algorithm to diversify the search. This is an interesting feature to be tested out, that was the reason for choosing the three types of tabu list lengths as explained above. In the second representation of tabu search although the same piece may be selected again and again to be moved within the layout, the tabu list restricts the piece from occupying its' historical positions on the layout.

As with ICH we will be running the experiments for the duration of 30 minutes so that we can evaluate the effectiveness of the two proposed strategies fairly. We realize that during the course of the iterations there may exists more than one piece that produces the largest overlap cost and more than one position that produces the smallest overlap cost, so we break ties by selecting one of these arbitrarily. This element of randomness could have an impact on the final solution quality, thus we decided to replicate the experiments five times as with the ICH approach. Based on the parameters given above we will be using 5 data sets (Shape0, Shape2, Trousers, Shirt and Dagli) to test the algorithm. For the fist strategy this will generate a total of 30 experiments of 30 minutes each for a data set, to run the experiments for the 5 data sets will take us around 75 hours in total. For our second strategy this will generate a total of 50 experiments of 30 minutes each for a data set, to run the experiments for the 5 data sets will take us around 125 hours in total. Unlike ICH where it was relatively easy to manipulate the neighborhood structure this was not the case with ICH where our sensible options are limited. Table 5.4 and Table 5.5 shows the results obtained using our first strategy and second strategy respectively and Table 5.6 shows the comparative results between these two tabu strategies for our SOL; best results are shown in bold.

Piece Tabu

Sorted Order

	Shape 0		Shape 2		Trousers		Shirt		Dighe1	
Tabu Length	Best L	Avg L	Best L	Avg L	Best L	Avg L	Best L	Avg L	Best L	Avg L
Low	**63.50**	65.70	**27.27**	28.59	**284.40**	286.60	**66.24**	67.44	**100.00**	129.97
Mid	65.50	66.50	27.40	27.85	286.00	286.82	**66.24**	67.74	**100.00**	105.66
Hi	65.50	67.30	27.90	28.15	**284.40**	286.76	**66.24**	67.54	**100.00**	112.87
Best Known	59.47		25.84		240.77		61.33		100.00	

Random Order

	Shape 0		Shape 2		Trousers		Shirt		Dighe1	
Tabu Length	Best L	Avg L	Best L	Avg L	Best L	Avg L	Best L	Avg L	Best L	Avg L
Low	64.00	66.30	**27.66**	28.16	**276.78**	286.26	**67.41**	70.23	127.86	139.21
Mid	65.00	67.30	28.16	28.51	279.00	288.17	68.31	70.33	120.21	126.99
Hi	**63.00**	67.10	28.50	28.86	282.00	288.27	68.41	70.63	**120.12**	126.39
Best Known	59.47		25.84		240.77		61.33		100.00	

Table 5.4 SOL Best Result using Tabu Strategy 1

Piece + Placement Tabu

Sorted Order

	Shape 0		Shape 2		Trousers		Shirt		Dighe1	
Tabu Length	Best L	Avg L	Best L	Avg L	Best L	Avg L	Best L	Avg L	Best L	Avg L
5	**66.50**	67.50	27.90	28.23	**286.00**	286.00	**68.24**	68.24	129.27	132.94
10	67.50	68.19	27.90	28.23	**286.00**	286.13	**68.24**	68.24	**117.98**	127.17
20	**66.50**	67.83	**27.40**	28.23	**286.00**	286.27	**68.24**	68.24	128.27	135.27
50	67.50	67.83	28.40	28.40	**286.00**	286.13	**68.24**	68.24	135.27	136.60
100	67.50	69.17	28.40	28.40	**286.00**	286.00	**68.24**	68.24	132.27	136.27
Best Known	59.47		25.84		240.77		61.33		100.00	

Random Order

	Shape 0		Shape 2		Trousers		Shirt		Dighe1	
Tabu Length	Best L	Avg L	Best L	Avg L	Best L	Avg L	Best L	Avg L	Best L	Avg L
5	67.00	68.00	29.00	29.19	280.76	283.43	68.17	69.76	**137.67**	149.02
10	**66.00**	67.67	**28.17**	29.01	**275.32**	288.43	**67.35**	69.92	145.21	149.40
20	67.16	68.32	28.40	28.78	280.08	283.22	67.77	69.01	151.63	155.80
50	67.00	67.67	29.16	29.40	278.77	282.32	69.17	69.51	141.67	151.78
100	67.50	68.33	**28.17**	28.81	278.75	286.95	70.98	71.42	139.86	147.47
Best Known	59.47		25.84		240.77		61.33		100.00	

Table 5.5 SOL Best Result using Tabu Strategy 2

Best Results

	Piece tabu		Piece + Placement tabu		Best Known Solution
	Best Length		Best Length		
	Sorted	Random	Sorted	Random	
Shapes0	63.5	**63.00**	66.5	66.00	**59.47**
Shape2	**27.27**	27.66	27.4	28.17	**25.84**
Trousers	284.4	276.78	286	**275.32**	**240.77**
Shirt	**66.24**	67.41	68.24	67.35	**61.33**
Dighe1	**100**	120.12	117.98	137.67	**100**

Table 5.6 SOL Tabu Strategy Comparison

From Table 5.6 we can see that our second strategy was not performing as well as our first strategy and it was worst in all instances. In some instances the result obtained were quite inferior in quality, example Shape 0 data. This can be the result of keeping allowing the same piece to be moved again and again in the layout thus slowing the algorithm from achieving good solution quickly. We can also derive from Table 5.4 and 5.5 that there are higher chances of obtaining good quality results in lower tabu list length, this maybe because longer tabu list lengths create greater diversification or disturbance in the behaviour of the algorithm and make it inefficient. Table 5.6 also shows the randomness of solution quality in SOL, unlike ICH where all the best results obtained were from sorted area ranking criteria; in SOL we seem to get best result from either ranking criteria. Based on our results we did not perform better than the current best results achieved in the literature except in Dighe1. In Dighe1 our SOL using the first tabu

strategy matched the current best result of 100 in the literature; this is the known optimum solution for this jigsaw puzzle problem. The result of Dighe1 was also better than what the 112.07 that we have achieved using ICH. Shapes 0 data set, achieved a strip length of 63 and this matched the best known solution of 63 till 2006 (Dowsland et al., 1998). Shapes2 data sets achieved a best of 27.27 and this was very close to the previous best of 27.2 (Burke et al., 2006), the authors reported of having to extend their initial experiments to achieve this result. Overall even we are satisfied with the performance of our SOL as we have stated that our research goal was not to produce better results that the current best in the literature but our algorithm should produce reasonably competitive result to ascertain its effectiveness. Thus we decided to use the first tabu representation as our basis to compare results with ICH.

5.3 ICH and SOL Comparison

Based on this set of result we decided to set the parameters for comparing the results from the two approaches, ICH and SOL. For ICH we decided to fix the neighbourhood structure to *OptN3(5)* as this was identified to be consistent in producing good quality results. The tabu lists to be tested was reduced from five to three, we will only be using 10, 20 and 50 as our tabu list lengths as from our experiments above this are the ones which produced good quality results. For SOL we will be using our first tabu representation with three tabu list lengths as described in section 5.2. For this final experimentation used to compare ICH with SOL we replicated the experiments 10 times for each shape data instances. This way we can be more confident in evaluating the quality of the solutions due to the element randomness as we have explained earlier. We will be evaluating a total of 10 datasets from the 16 datasets available on ESICUP to gain better insights into the effectiveness of both approaches. We left the six datasets out as these data sets allow the shapes to rotate in 4 different angles (0°, 90°, 180°, 270°) and this will require more computation time and will deteriorate the performance of the algorithm, so we decided to focus only on datasets with maximum two allowable rotations. Four out of these six datasets are of artificial type and the other two are garment shapes data.

The summary of results from our experiments is given in Figure 5.7 and 5.8 which shows comparison of result between area sorted and random ranking criteria for ICH and SOL; the best results are given in bold. For ICH no clear conclusion can be made as to which tabu list length was better in producing good results. Tabu list of size 10 produced the best result for all of the shapes expect Shape1, Shapes2, Shirt and Dighe1. Tabu list of size 20 produced good results in all except Shapes 0, Shapes 2, Albano and Dagli. Tabu list of size 50 produced good results in all except Shape1, Albano, Dagli and Dighe1.

ICH

	Sorted						Random					
Tabu Length	10		20		50		10		20		50	
	Best L	Avg L	Best L	Avg L	Best L	Avg L	Best L	Avg L	Best L	Avg L	Best L	Avg L
Shape 0	**62**	63.499	63	63.783	**62**	63.466	63	64.707	63	64.183	63	64.632
Shape 1	58	59.05	**57**	59	58	59.7	59	61.121	59	61.272	59	61.324
Shape 2	26.94	27.499	27.19	27.558	**26.8**	27.499	27.25	27.699	27.18	27.612	27.5	27.806
Albano	**10247.19**	10461.23	10248.44	10436.41	10354.07	10449.43	10304.4	10444.41	10341.27	10472.57	10418.07	10522.5
Trousers	**248.08**	251.129	**248.08**	251.132	**248.08**	251.132	267.05	269.814	267.05	270.143	267.05	270.143
Swim	**6216.88**	6441.959	**6216.88**	6431.873	**6216.88**	6431.873	6807.42	6956.731	6807.42	6978.349	6807.42	6978.349
Shirt	63.58	63.982	**63.47**	63.966	**63.47**	63.966	66.86	68.167	66.86	68.167	66.86	68.167
Dagli	**60.24**	61.122	60.31	61.185	60.91	61.563	60.71	62.213	60.59	62.558	61.66	62.886
Dighe1	112.82	113.525	**100**	116.523	119.57	122.763	112.82	113.231	112.82	115.674	116.67	123.079
Dighe2	**100**	100	**100**	100	**100**	101.632	**100**	100	**100**	100	**100**	101.171

Table 5.7 ICH Best Result Comparison

SOL

	Sorted						Random					
Tabu Length	Low		Mid		Hi		Low		Mid		Hi	
	Best L	Avg L	Best L	Avg L	Best L	Avg L	Best L	Avg L	Best L	Avg L	Best L	Avg L
Shape 0	63.5	65.7	65.5	66.5	65.5	67.3	64	66.296	65	67.296	**63**	67.096
Shape 1	58	59.3	59	61.2	61	62.3	**57**	58.525	58.75	60.825	61	62.325
Shape 2	**27.27**	28.587	27.4	27.85	27.9	28.15	27.66	28.156	28.16	28.506	28.5	28.856
Albano	**10399.93**	10639.93	10499.93	10609.93	10599.93	10709.93	10607.01	10743.1	10514.72	10726.79	10545.02	10826.59
Trousers	284.4	286.6	286	286.82	284.4	286.76	**276.78**	286.258	279	288.166	282	288.266
Swim	**6743.15**	6973.15	6843.15	6973.45	6943.15	7073.15	6932.94	7098.732	7045.14	7173.808	7080.56	7203.01
Shirt	**66.24**	67.44	**66.24**	67.74	**66.24**	67.54	67.41	70.232	68.31	70.332	68.41	70.632
Dagli	**61.2**	63.883	62.2	63.937	62.2	66.372	62.96	63.613	63.08	63.907	63.27	64.504
Dighe1	**100**	129.97	**100**	105.66	**100**	112.87	127.86	139.209	120.21	126.992	120.12	126.393
Dighe2	130.43	135.129	130.45	131.15	130.45	131.75	126.89	134.873	**100**	126.587	**100**	123.652

Table 5.8 SOL Best Result Comparison

Each tabu list length had an equal share in producing the best result so we can conclude that it is always better to experiment with a number of parameters of tabu list lengths in the packing algorithm. If we were to fix the tabu list length to only one length in some cases we would have produced inferior results given the variability of the input data. Again from the results in ICH we can see that random order ranking criteria was inferior in performance compared to the sorted by area initial packing order. In SOL we notice

that the low tabu list length more frequently gets the best result. This conforms to the result obtained in our earlier experiments and we have explained in section 5.1 why we think this was an expected scenario. Similar to the observation we made in section 5.1 in SOL best solution are spread between sorted area ranking criteria and random order ranking criteria, thus no clear conclusion can be made as to which ranking criteria works best with SOL.

When sorted area ranking criteria was used ICH performed better or equal to SOL in all data sets. When random ranking criteria was used, SOL matched the result in Shape0 and Dighe2 and was better than ICH in Shape1. However in rest of the shapes ICH performed better than SOL.

Figure 5.10 summarizes the best result of ICH and SOL and in the last column the current best result in the literature and the name of the author is added; Egebald et al., 2006; Gomes and Oliveira, 2006 and Bennell and Song, 2007. These are denoted as E, G and S respectively. Recall from our Chapter 2, Section 2.3 and 2.4 that E and G used the SOL approach while S used the ICH approach.

Data Type	ICH			SOL			Literature	
	Best Length	Std. Dev.	Util(%)	Length	Std. Dev	Util(%)	Util(%)	Author
Shapes0	**62**	0.76	64.35%	63	1.20	63.33%	67.09%	E
Shape1	**57**	0.90	70.00%	**57**	1.57	70.00%	73.84%	E
Shapes2	**26.8**	0.26	80.60%	27.27	1.96	79.21%	83.59%	G
Albano	**10247.19**	87.34	84.95%	10797.28	110.59	80.63%	87.88%	E
Trousers	**248.08**	1.34	87.80%	276.78	0.88	78.69%	90.46%	E
Swim	**6216.88**	94.09	71.16%	6743.15	124.64	65.60%	75.04%	S
Shirt	**63.47**	0.37	85.08%	66.24	0.88	81.52%	88.05%	S
Dagli	**60.24**	0.64	84.19%	61.2	2.71	82.87%	87.99%	S
Dighe1	**100**	4.57	100.00%	**100**	14.56	100.00%	100.00%	G
Dighe2	**100**	2.98	100.00%	**100**	3.02	100.00%	100.00%	G

Table 5.9 ICH and SOL Best Results Comparison

Utilization percentage is calculated as below:-

Utilization percentage

= (sum total area of all the pieces) / (best packing length X width of stock sheet)

From the results shown ICH clearly performs better than SOL in all problem instances tested except Shapes1 where SOL matched ICH's result and in Dighe1 where it was better than ICH and matched the current best result in the literature. ICH and SOL matched the current best result in literature in Dighe2. Overall ICH performs reasonably well compared to the best in literature. SOL produced worst results in three data sets; Trousers, Swim and Shirt. These three data sets are derived from the textile industry; these data sets can be found in figure 4.2. A common element of these data sets is the high irregularity and large quantities of shapes. This will cause a great problem when the strip length get shorter and shorter, the overlap becomes quite impossible to be resolved. This may be due to a number of factors, small pieces may be overlapping with larger pieces and it becomes difficult to separate these pieces. This might be the reason why researches quite often formulate special cost function and special penalty to overcome this problem. Bennell and Dowsland (1999) injected problem specific knowledge by observing the graphical animation of the algorithm in progress and made SOL more effective in removing the overlaps and reaching better solutions. To avoid heavily penalizing small pieces, the respective widths of pieces in overlap situation are also included in the overlap function and the minimum of these are then selected as the overlap cost. This will allow small pieces to be preferred to be moved. Gomes and Oliveira (2006) combined SOL with a separation and compaction routine using LP. The advantage of using ICH is that we do not need to worry about transforming the solution from infeasible to feasible because we do not permit overlap and we are guaranteed a feasible solution every time. The experiments have proven that with ICH we can expect reasonably good solution in a short span of computation run.

5.4 Summary

We introduced five alternative neighbourhood structures for ICH on 5 data sets. We used the learning outcome from these experiments to choose and fix our neighbourhood structure for the final comparison experiment between SOL and ICH. We investigated 2 possible tabu search design for ICH and compared their computational results. We established the test parameters of our experiments to compare ICH and SOL. We made the conclusion that ICH is an effective algorithm and is able to reach a reasonably good solution quickly. We believe that SOL needs tweaks and tuning on the cost function parameters to enable the overlaps within the layout to be removed more effectively. The literature suggested that if this done SOL is capable excellent results as shown in the literature but to get there is not an easy task. ICH is simple, deterministic and once implemented the final solution generated is always a feasible solution, we do not need to worry about removing overlap from the layout. Further with ICH it is relatively easy to experiments with different neighbourhood structure which could be generated by swapping or inserting pieces within a packing order.

6 Conclusions and Future Work

6.1 Background

We set out off with the research aim to compare the two common problem representations for 2D ODP; ICH and SOL. The literature reports both approaches as producing excellent results but if an OR practitioner were to choose either one as the preferred method of solution which one would it be. To do this we have stripped both approaches from any special features or sophistication and implemented them based on their basic principles. By doing this we have created a level playing field for comparison. We then used our computational experiments to analyze the success of both these approaches with regards to their success rate of producing good quality solutions over a range of different data sets, their strength and weakness with respect to the different data types and total number of pieces. In ICH we used a bottom left placement policy to position the pieces within the layout, we generated search positions based on NFP as explained in our methodology chapter and we used tabu search as our search engine to find the best packing solution within our predefined neighbourhood structure. In SOL we based our overlap measure on the sum of the extent of the horizontal and vertical overlap distance as explained in our methodology chapter, we used this measure as the penalty value in our cost function, we generated our placement positions based on NFP (similar to ICH) and also used tabu search as our search engine. The difference between the tabu search in ICH and SOL is that in the former the tabu list will contain the swap positions of the pieces that were swapped whereas in the latter the tabu list contains the piece identification and its historical position in the layout. This is because ICH seeks to find the best ordering of the pieces and SOL the best placement positions available in the layout to reach a near optimal packing solution.

We have demonstrated based on our extensive computational experiments that ICH has several advantages over SOL. One of which is the guarantee of producing a feasible packing solution, on the other hand SOL cannot guarantee this and will potentially get stuck if it cannot eliminate the overlaps in the layout within the restricted computational

time. In ICH regardless of the shapes profile we consistently achieved good quality solutions within a short span of time. Our experiments have indicated this is not the case with SOL, the solution quality deteriorated when the shapes profiles contained high irregularity, ranging from very small pieces to very large pieces and large quantity of pieces, typical of industrial data set. We believe this is because as the layout gets compacted many small pieces may overlap with large ones and it becomes extremely difficult for the algorithm to resolve this. This observation was also mentioned in Bennell and Dowsland (1999) and they used a special cost function to overcome this problem. Although quite a number of researchers (e.g., Bennell and Dowsland, 1999, Gomes and Oliveira, 2006, Egeblad, et al., 2006) have indicated SOL as producing excellent packing result, we believe that this may be because of specific knowledge injected into the algorithm based on their experimental learning. Further more the benchmark data sets used by researches are limited to only 17 data sets. As a result it is possible that enhancement to the algorithms generate better performance against specific data leading to better and better solutions rather than in general. We have also shown tabu search as an excellent search tool as all the result obtained are close to the best produced in the literature and some even matched the known solutions (e.g. dighe1, dighe2).

6.2 Future Work

As we have only experimented with only one local search technique we could not draw a conclusion as to which search technique is superior in terms of consistently achieving the best solution. Thus other search techniques like simulated annealing and genetic algorithm could be applied and tested. Implementation and comparison of different search techniques might alter the observations that we have made so far.

The calculation of overlap measure might have a critical influence on the effectiveness of the SOL. Thus we suggest that the different available approaches to be compared against each other and analyzed in terms of the solution quality produced. Combination of this different overlap measure with the alternative search technique would yield a wide set of comparison results. We can also broaden our neighbourhood structure by implementing

other types of moves to produce the neighbourhood of solutions; this will widen our comparative experimental data to test the effectiveness of both approaches. Finally the data sets could be broadened and deduction made from our observations above could be tested with this new data sets.

BIBLIOGRAPHY

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

Agarwal, P.K., Flato, E., Halperin, D. 2002. Polygon Decomposition for Efficient Construction of Minkowski Sums. Computational Geometry Theory and Applications, vol. 21, pp. 39-61.

Albano, A., Sappupo, G. 1980. Optimal Allocation of Two-Dimensional Irregular Shapes Using Heuristic Search Methods. IEEE Transactions on Systems, Man and Cybernetics, vol. 10, no. 5, pp. 242- 248.

Art, R.C. 1966. An Approach to the Two Dimensional, Irregular Cutting Stock Problem. Technical Report 36.Y08, IBM Cambridge Scientific Center.

Bennell, J.A, Dowsland, K.A. 2001. Hybridising Tabu Search with Optimisation Techniques for Irregular Stock Cutting. Management Science, vol. 47, no.8, pp. 1160-1172.

Bennell, J.A. 1998. Incorporating Problem Specific Knowledge into a Local Search Framework for the Irregular Shape Packing Problem. PhD Dissertation, EBMS. University of Wales, Swansea, UK.

Bennell, J.A., Dowsland, K.A. 1999. A Tabu Thresholding Implementation for the Irregular Stock Cutting Problem. International Journal of Production Research, vol. 37, pp. 4259-4275.

Bennell, J.A., Oliveira, J.F. 2008. The Geometry of Nesting Problems: A Tutorial. European Journal of Operational Research, vol. 184, no. 2, pp. 397-415.

Bibliography

Bennell, J.A., Song, X. 2007. A Beam Search Implementation for the Irregular Shape Packing Problem. University of Southampton, Discussion Paper Series - Centre for Operational Research, Management Science and Information Systems, CORMSIS-07-01.

Bennell, J.A., Song, X. 2008. A Comprehensive and Robust Procedure for Obtaining the Nofit Polygon Using Minkowski Sums. Computers and Operational Research.

Blazewicz, J., Hawryluk, P., Walkowiak, R. 1993. Using a Tabu Search Approach for Solving the Two-Dimensional Irregular Cutting Problem. Annals of Operations Research, vol. 41, pp. 313-325.

Blazewicz, J., Hawryluk, P., Walkowiak, R. 1993.Using Tabu Search Approach for Solving the Two-Dimensional Irregular Cutting Problem. Annals of Operations Research, vol. 41, pp. 313-325.

Blazewicz, J., Walkowiak, R. 1993. An Improved Version of Tabu Search for Irregular Cutting Problem. In: Karmann, A., Mosler, K., Schader, M., Uebe, G. (Eds.), Operations Research. Physica, Heidelberg, pp. 102–104.

Blazewicz, J., Walkowiak, R. 1995. A local Search Approach for Two-Dimensional Irregular Cutting Problem. OR Spektrum, vol. 17, pp. 93-98.

Blazewicz, J., Walkowiak, R. 1995. A Local Search Approach for Two-Dimensional Irregular Cutting. OR Spektrum, vol. 17, pp. 93-98.

Bounsaythip, C., Maouche, S. 1996. A Genetic Approach to a Nesting Problem. Proceedings of the Second Nordic Workshop of Genetics Algorithms, pp. 89-104.

Bremermann, H.J., 1958. The Evolution of Intelligence: The Nervous Systems as a Model of its Environment. Technical Report, Technical Report No. 1, Contract No. 477(17), Department of Mathematics, University of Washington Seattle.

Bibliography

Burke E.K., Hellier R.S.R., Kendall G., Whitwell G. 2007. Complete and Robust No-Fit Polygon Generation for the Irregular Stock Cutting Problem. European Journal of Operations Research, vol. 179, no. 1, pp. 27-49.

Burke, E.K. and Kendall, G. 1999. Applying Ant Algorithms and the No Fit Polygon to the Nesting Problem. Proceedings of the 12th Australian Joint Conference on Artificial Intelligence, Sydney, Australia, Lecture Notes in Artificial Intelligence, pp. 453-464.

Burke, E.K., Hellier, R., Kendall, G., Whitwell, G. 2007. Complete and robust no-fit polygon generation for the irregular stock cutting problem. European Journal of Operational Research, 179(1), pp. 27-49.

Burke, E.K., Hellier, R.S.R., Kendall, G., Whitwell, G. 2006. A New Bottom-left-Fill Heuristic Algorithm for the 2D Irregular Packing Problem. Operations Research, vol. 54, no. 3, pp. 587-601.

Burke, E.K., Kendall, G. 1999. Applying Evolutionary Algorithms and the No-Fit Polygon to the Nesting Problem. Proceedings of the 1999 International Conference on Articial Intelligence, vol. 1, pp. 51-57.

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

Burke, E.K., Kendall, G., Whitwell G. 2004. A New Placement Heuristic for the Orthogonal Stock-Cutting Problem. Operations Research, vol. 52, no. 4, pp. 655-671.

Bibliography

Cheng, S.K., Rao, K.P. 2000. Large-Scale Nesting of Irregular Patterns Using Compact Neighborhood Algorithm. Journal of Materials Processing Technology, vol. 103, no. 1, pp. 135-140.

Dagli, C.H. 1990. Neural Network in Manufacturing: Possible Impacts on Cutting Stock Problem. Proceedings of the 2nd International Conference on Computer Integrated Manufacturing, IEEE Computer Society Press, pp. 531-537.

Dagli, C.H., Hajakbari, A. 1990. Simulated Annealing Approach for Solving Stock Cutting Problem. Proc. IEEE Internat. Conf. Systems, Man, and Cybernetics, Los Angeles, CA, pp. 221-223.

Dighe, R., Jakiela, M.J. 1996. Solving Pattern Nesting Problems with Genetic Algorithms Employing Task Decomposition and Contact Detection. Evolutionary Computation, pp. 239-266.

Dowsland, K.A., Dowsland, W.B. 1992. Packing Problems. European Journal of Operational Research, vol. 56, pp. 2-14.

Dowsland, K.A., Dowsland, W.B. 1995. Solution Approaches to Irregular Nesting Problems. European Journal of Operational Research, vol. 84, pp. 506-521.

Dowsland, K.A., Dowsland, W.B., Bennell, J.A. 1998. Jostling for Position: Local Improvement for Irregular Cutting Patterns. Journal of the Operational Research Society, vol. 49, pp. 647-658.

Dowsland, K.A., Vaid, S., Dowsland, W.B. 2002. An Algorithm for Polygon Placement Using a Bottom-Left Strategy. European Journal of Operational Research, vol. 141, pp. 371-381.

Dyckhoff, H. 1990. Typology of Cutting and Packing Problems. European Journal of Operational Research, vol. 44, pp. 145-159.

Bibliography

Dyckhoff, H., Finke, U. 1992. Cutting and Packing in Production and Distribution. Springer Verlag, Berlin.

Egeblad J., Nielsen B.K., Odgaard A. 2007. Fast Neighborhood Search for the Nesting Problem. European Journal of Operational Research, vol. 183, no. 3, pp. 1249-1266.

Faina, L. 1999. Application of Simulated Annealing to the Cutting Stock Problem. European Journal of Operational Research, vol. 114, pp. 542-556.

Fraser, A.S., 1957. Simulation of genetic systems by automatic digital computers. Australia Journal of Biological Science, vol. 10, pp. 492-499.

Fujita, K., Akagji, S., Kirokawa, N. 1993. Hybrid Approach for Optimal Nesting Using a Genetic Algorithm and a Local Minimisation Algorithm. Proceedings of the 19th Annual ASME Design Automation Conference, Albuquerque, NM, USA, pp. 477-484.

Glover, F. 1986. Future Paths for Integer Programming and Links to Artificial Intelligence. Computers and Operations Research, vol. 13, no. 5, pp. 533-549.

Gomes, A.M., Oliveira, J.F. 2002. A 2-Exchange Heuristic for Nesting Problems. European Journal of Operations Research, vol. 141, pp. 359-370.

Gomes, A.M., Oliveira, J.F. 2006. Solving Irregular Strip Packing Problems by Hybridising Simulated Annealing and Linear Programming. European Journal of Operational Research, vol. 171, no. 3, pp. 811-829.

Han, G.C., Na, S.J. 1996. Two-Stage Approach for Nesting in Two-Dimensional Cutting Problems Using Neural Network and Simulated Annealing, Journal of Engineering Manufacture, vol. 210, pp. 509-519.

Bibliography

Heckmann, R., Lengauer, T. 1998. Computing Closely Matching Upper and Lower Bounds on Textile Nesting Problems. European Journal of Operational Research, vol. 108, pp. 473-489.

Heckmann, R., Lengauer, T. 1995. A Simulated Annealing Approach to the Nesting Problem in the Textile Manufacturing Industry. Annals of Operations Research, vol. 57, pp. 103-133.

Heistermann, J. and Lengauer, T. 1995. The Nesting Problem in the Leather Manufacturing Industry. Annals of Operations Research, vol. 57, pp. 147-173.

Hifi, M., Paschos, V.T. 2003. A Simulated Annealing Approach for the Circular Cutting Problem, European Journal of Operations Research.

Holland, John, H. 1975. Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor.

Hopper, E., Turton, B. 1999. A Genetic Algorithm for a 2D Industrial Packing Problem. Computers and Industrial Engineering, vol. 37, no. 1-2, pp. 375-378.

Hopper, E., Turton, B.C. 2001. A Review of the Application of Meta-Heuristic Algorithms to 2D Strip Packing Problems. Artificial Intelligence, vol. 16, no. 4, pp. 257-300.

Hopper, E., Turton, B.C.H. 1999. A genetic algorithm for a 2D industrial packing problem. Comput. Indust. Engrg. 37 375-378.

Imahori, S., Yagiura, M., Nagamochi, H. 2007. Practical Algorithms for Two-Dimensional Packing. Handbook of Approximation Algorithms and Metaheuristics, Gonzalez (ed.), Chapman & Hall/CRC in the Computer & Information Science Series.

Bibliography

Ismail, H.S., Hon, K.K.B. 1995. The Nesting of Two-Dimensional Shapes Using Genetic Algorithms. IMechE Part B: Journal of Engineering Manufacture, vol. 209, pp. 115-124.

Jain, P., Fenyes, P., Richter, R. 1992. Optimal Blank Nesting Using Simulated Annealing. Journal of Mechanical Design, vol. 114, pp. 160-165.

Jain, S., Chang, H.G. 1998. Two-Dimensional Packing Problems using Genetic Algorithms, Engineering Computing, vol. 14, pp. 206-213

Jakobs, S. 1996. On Genetic Algorithms for the Packing of Polygons. European Journal of Operational Research, pp. 165-181.

Kim, Y., Gotoh, K., Toyosada, M. 2003. Automatic Two-Dimensional Layout Using a Rule-Based Heuristic Algorithm. Journal of Marine Science and Technology, vol. 8, pp. 37–46.

Konopasek, M. 1981. Mathematical Treatments of Some Apparel Marking and Cutting Problems. US Department of Commerce Report, 99-26-90857-10.

Li, Z., Milenkovic, V.J., 1995. Compaction and Separation Algorithms for Non-Convex Polygons and their Applications. European Journal of Operational Research, vol. 84, pp. 539-561.

Lodi, A., Martello, S., Monaci, M. 2002. Two-Dimensional Packing Problems: A Survey. European Journal of Operational Research, vol. 141, pp. 241-252.

Lodi, A., Martello, S., Monaci, M. 2003. Two-Dimensional Packing Problems: A Survey. European Journal of Operations Research, vol. 141, pp. 241-252.

Bibliography

Lutfiyya, H., McMillin, B., Poshyanonda, P., Dagli, C. 1992. Composite Stock Cutting Through Simulated Annealing. Journal of Mathematical and Computer Modelling, vol. 16, no. 2, pp. 57-74.

Mahadevan, A. 1984. Optimization in Computer Aided Pattern Packing. PhD Dissertation, North Carolina State University, USA.

Marques, V.M.M., Bispo, C.F.G., Sentieiro, J.J.S. 1991. A System for the Compaction of Two-Dimensional Irregular Shapes based on Simulated Annealing. Proceedings of the 1991 International Conference on Industrial Electronics, Control and Instrumentation, Kobe, Japan, pp. 1911-1916.

Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E. 1953. Equation of State Calculation by Fast Computing Machines. J. of Chem. Phys., vol. 21, pp. 1087-1091.

Milenkovic, V.J. 2002. Densest Translational Lattice Packing of Non-Convex Polygons. Computational Geometry, vol. 22, pp. 205-222.

Milenkovic, V.J., Daniels K.M. Li, Z. 1991. Automatic Marker Making. Proceedings of the Third Canadian Conference on Computational Geometry, pp. 243-246.

Nye, T.J. 2001. Optimal Nesting of Irregular Convex Blank in Strips via an Exact Algorithm. International Journal of Machine Tools & Manufacture, vol. 41, pp. 991-1002.

Oliveira, J.F. and Ferreira, J.S. 1993. Algorithms for Nesting Problems. Applied Simulated Annealing, pp. 255-273.

Oliveira, J.F., Gomes, A.M., Ferreira, J.S. 2000. TOPOS - a New Constructive Algorithm for Nesting Problems. OR Spektrum, vol. 22, pp. 263-284.

Bibliography

O'Rourke, J. 1998. Computational Geometry in C (Second Edition), Cambridge University Press.

Poshyanonda, P., Bahrami, X. 1992. Artificial Neural Networks in Stock Cutting Problems. Neural Networks in Manufacturing and Robotics, ASME Press, New York, pp. 143-153.

Poshyanonda, P., Dagli, C.H. 1992. A Hybrid Approach to Composite Stock Cutting. Neural Network and Genetic Algorithms, Robotics and Manufacturing, Recent Trends in Research, Education and Applications, vol. 4, pp. 775-780.

Prasad, Y.K.D., Somasundaram, S. 1991. CASNS - A Heuristic Algorithm for the Nesting of Irregular Shaped Sheet Metal Blanks. Computer Aided Engineering Journal, pp. 69-73.

Prasad, Y.K.D., Somasundaram, S., Rao, K.P. 1995. A Sliding Algorithm for Optimal Nesting of Arbitrarily Shaped Sheet Metal Blanks. International Journal of Production Research, vol. 33, pp. 1505-1520.

Qu, W., Sanders, J.L. 1987. A Nesting Algorithm for Irregular Parts and Factors Affecting Trim Losses. International Journal of Operations Research, vol. 25, no. 3, pp. 381-397.

Ramesh Babu, A. , Ramesh Babu, N. 2001. A Generic Approach for Nesting of 2-D Parts in 2-D Sheets Using Genetic and Heuristic Algorithms. Computer-Aided Design vol. 33, no. 12, pp. 879-891.

Ramesh Babu, A., Ramesh Babu, N. 1998. A Genetic Approach for Nesting of Two-Dimensional Complex Parts. The 14th International Conference on CAD/CAM, Robotics and Factories of the Future.

Bibliography

Ramesh Babu, A., Ramesh Babu, N. 2001. A Generic Approach for Nesting of 2-D Parts in 2-D sheets Using Genetic and Heuristic Algorithms. Computer-Aided Design, vol. 33, no. 12, pp. 879-891

Reeves CR, 1993. Modern Heuristic Techniques for Combinatorial Problems. Blackwell Publishing, London.

Segenreich, S.A. and Braga, L.M.P.F. 1986. Optimal Nesting of General Plane Figures: A Monte Carlo Heuristical Approach. Computer and Graphics, vol. 10, pp. 229-237.

Stoyan, Y., Scheithauer, G., Gil, N., Romanova, T. 2004. Φ-Functions for Complex 2D-Objects. Quarterly Journal of the Belgian, French and Italian Operations Research Societies, vol. 2, no. 1, pp. 69–84.

Sweeney, P.E., Paternoster, E. 1992. Cutting and Packing Problems: A Categorised, Application-Orientated Research Bibliography. Journal of the Operational Research Society, vol. 43, pp. 691-706.

Tay, F.E.H., Chong, T.Y., Lee, F.C. 2002. Pattern Nesting on Irregular-Shaped Stock Using Genetic Algorithms. Engineering Applications of Artificial Intelligence, vol. 15, no. 6, pp. 551-558.

Theodoracatos, V.E., Grimsley, J.L. 1995. The Optimal Packing of Arbitrarily-Shaped Polygons Using Simulated Annealing and Polynomial-Time Cooling Schedules. Computer Methods in Applied Mechanics and Engineering, vol. 125, pp. 53-70.

Wäscher, G. Haußner, H., Schumann, H. 2007. An Improved Typology of Cutting and Packing Problems. European Journal of Operational Research, vol. 183, no. 3, pp. 1109-1130.

Bibliography

Watson, P.D., Tobias, A.M. 1999. An Efficient Algorithm for the Regular W1 Packing of Polygons in the Infinite Plane. Journal of the Operational Research Society, vol. 50, pp. 1054-1062.

Whelan, P. F., Batchelor, B.G. 1993. Automated Packing Systems: Review of Industrial Implementations. Machine Vision Architectures, Integration and Applications, vol. 2064, pp. 358-369.

Yagiura, M., Ibaraki, T. 2001. Metaheuristic Algorithms for Combinatorial Optimization Problems. Systems and Computers in Japan, vol. 32, no. 3, pp. 33-55.

Yeung, L.H.W., Tang, W.K.S. 2003. A Hybrid Genetic Approach for Garment Cutting in the Clothing Industry. IEEE Transactions on Industrial Electronics, vol. 50, no. 3, pp. 449-455.

APPENDIX A

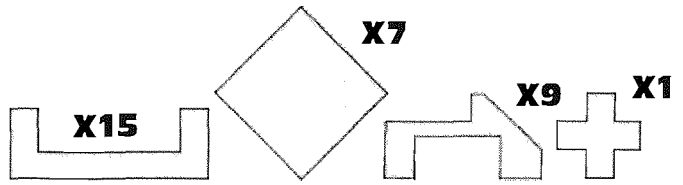
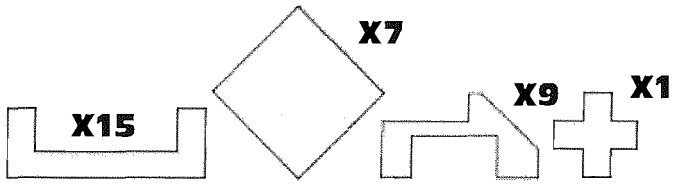
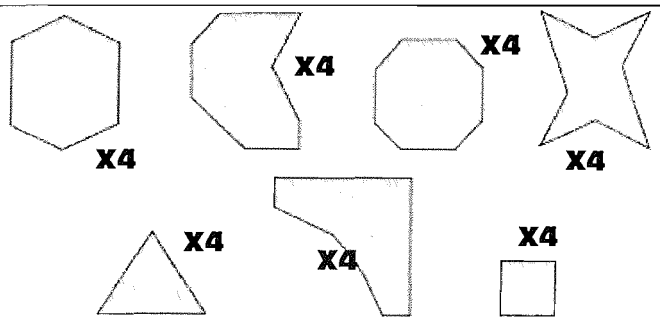
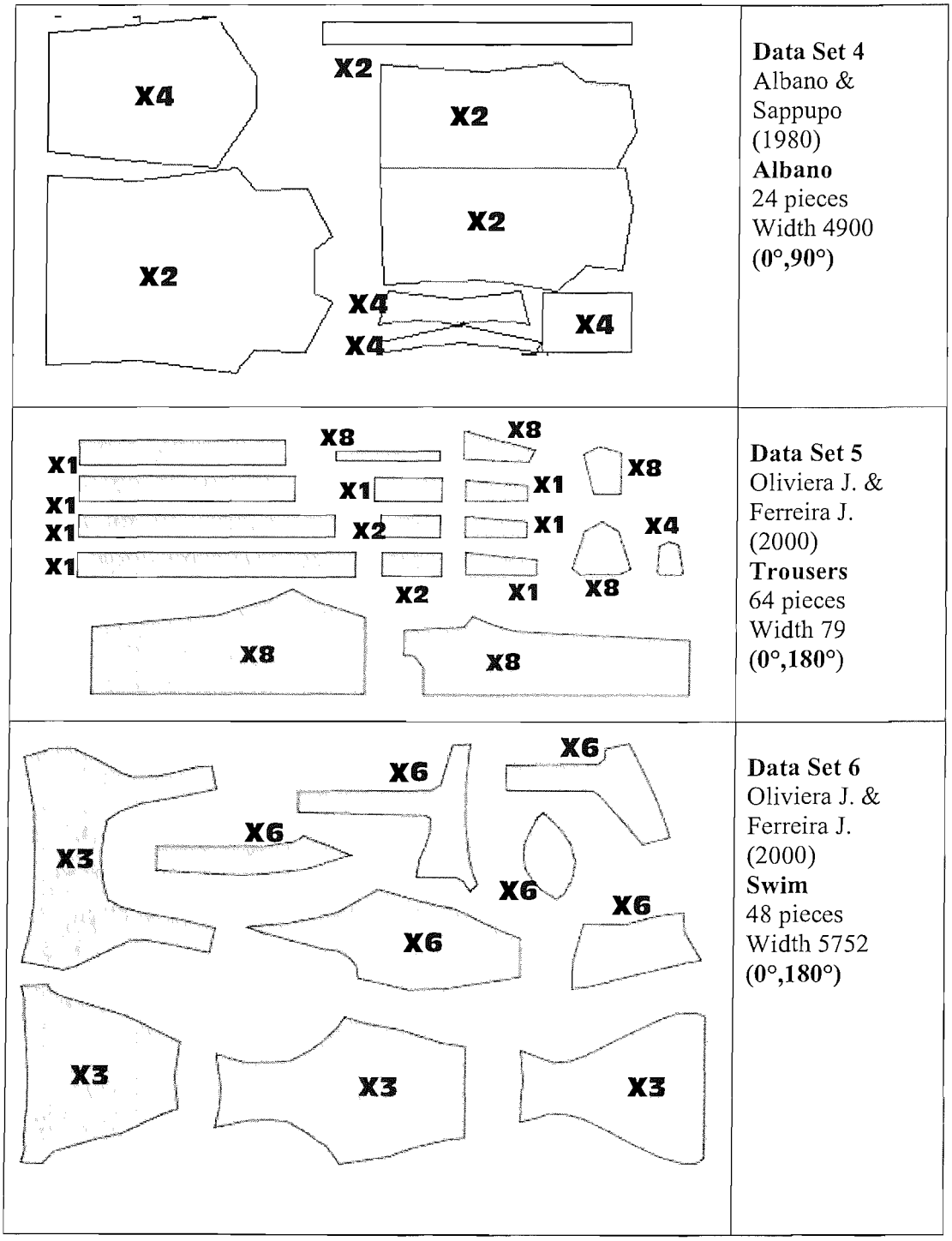
	**Data Set 1** Oliviera J. & Ferreira J. (1993) **Shape 0** 43 pieces Width 40 **(0°)**
	**Data Set 2** Oliviera J. & Ferreira J. (1993) **Shape 1** 43 pieces Width 40 **(0°,180°)**
	**Data Set 3** Blazewicz J. (1993) **Shape 2** 28 pieces Width 15 **(0°,180°)**

Figure A.1

Appendix A

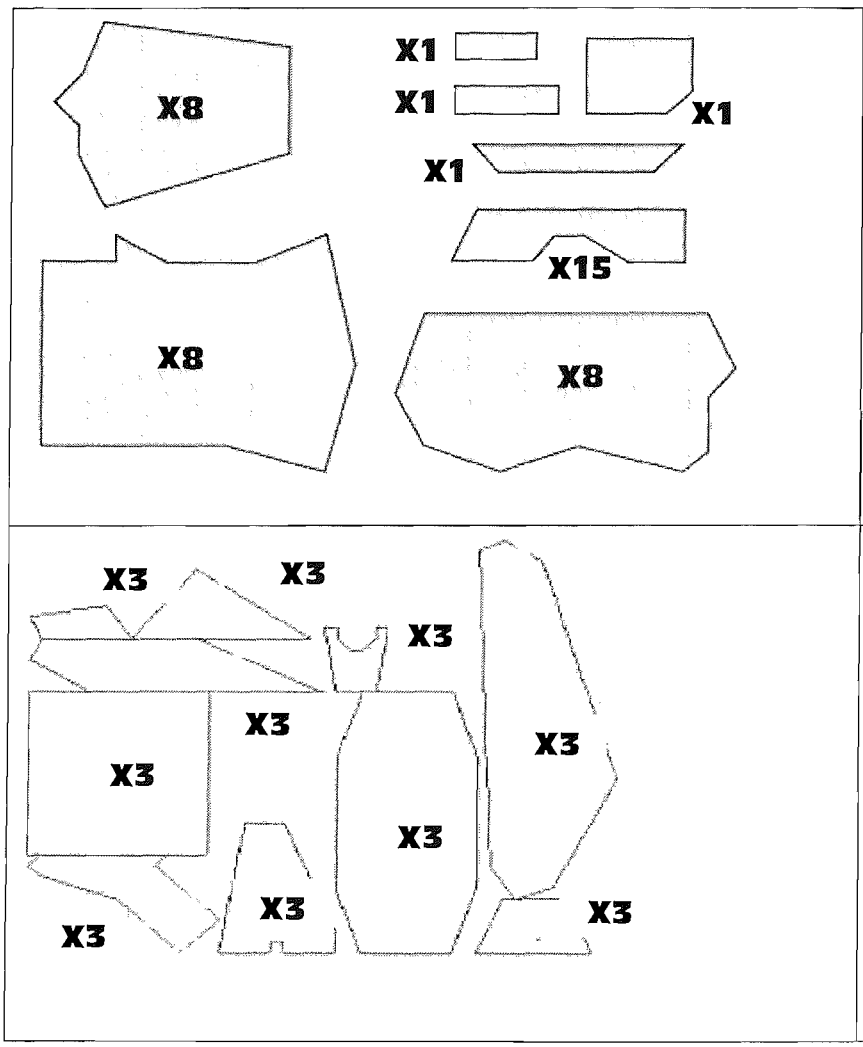


Data Set 4
Albano & Sappupo (1980)
Albano
24 pieces
Width 4900
(0°,90°)

Data Set 5
Oliviera J. & Ferreira J. (2000)
Trousers
64 pieces
Width 79
(0°,180°)

Data Set 6
Oliviera J. & Ferreira J. (2000)
Swim
48 pieces
Width 5752
(0°,180°)

Figure A.2

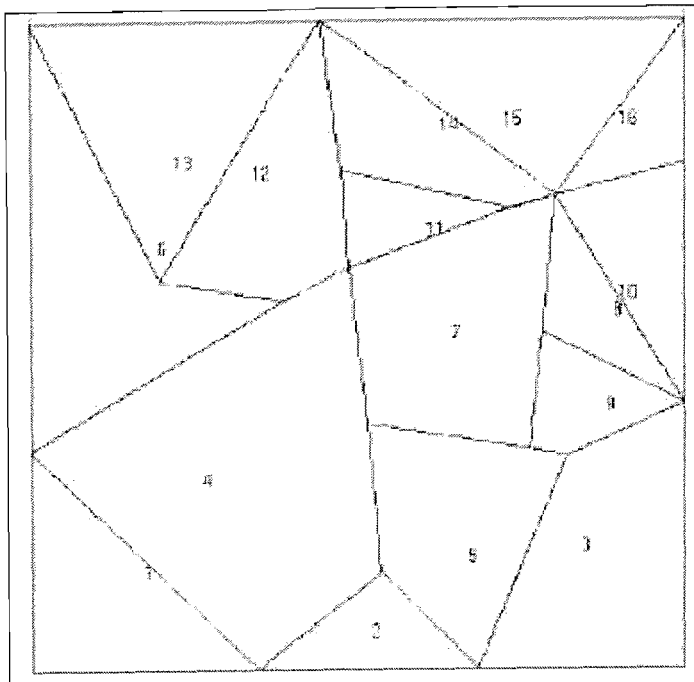


Data Set 7
Oliviera J. & Ferreira J. (1993)
Shirts
99 pieces
Width 40
(0°,180°)

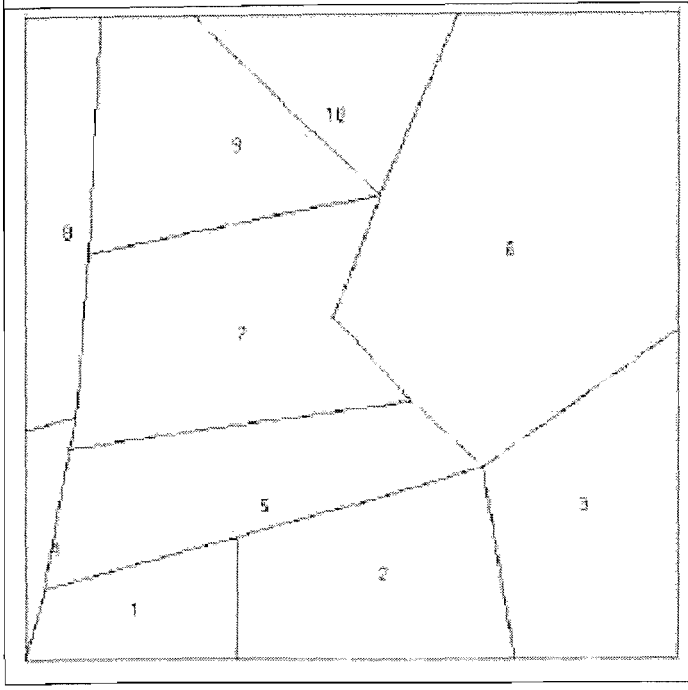
Data Set 8
Ratanapan & Dagli (1997)
Dagli
99 pieces
Width 60
(0°,180°)

Figure A.2 continued

Appendix A



Data Set 9
Dighe &
Jakiela
(1997)
Dighe
16 pieces
Width 100
(0°)



Data Set 10
Dighe &
Jakiela
(1997)
Dighe
10 pieces
Width 100
(0°)

Figure A.3