Applying Meta-Heuristic Algorithms to the Nesting Problem Utilising the No Fit Polygon

A Thesis submitted to the University of Nottingham for the Degree of Doctor of Philosophy

by

Graham Kendall, BSc (Hons)

University of Nottingham, Nottingham
School of Computer Science and Information Technology

October 2000

Contents

Figures

Tables

Abstract

This thesis develops and investigates meta-heuristic and evolutionary approaches to the stock cutting problem. In particular, we concentrate on hill climbing, tabu search, simulated annealing, genetic algorithms, ant algorithms and two types of memetic algorithm.

In carrying out this work, a number of diverse issues have had to be explored. It is often the case that the evaluation function is a bottleneck within a search algorithm. This thesis shows that by using a cache to store previously evaluated solutions, the evaluation function does not have to be called when the solution is already in the cache. This significantly improves the speed of the algorithm. However, under certain circumstances, it is beneficial to re-evaluate a solution even though it is stored in the cache. This thesis shows under which conditions this applies and shows that the re-evaluation only has to be done with low probability without affecting solution quality.

Ant algorithms are investigated for the first time with regard to stock cutting. This algorithm is relatively new but has been applied successfully to difficult search problems including the travelling salesman problem, vehicle routing and the quadratic assignment problem. In addition, we have implemented and evaluated hybrid ant algorithms with a local search operator.

This thesis also develops a no fit polygon algorithm, which is used to pack non-convex shapes.

Throughout this thesis both tabu search and a genetic based memetic algorithm consistently produce the best quality results. This is despite trying to find suitable parameters for the other search algorithms so that they carry out more effective searches.

The conclusion is that, although tabu search and a genetic based memetic algorithm produce the best quality results, it does not follow that they are the best search strategies for all problems. If there was one area of research that arose from this work it is a need for investigating methods of applying the *correct* search strategy to any given problem instance.

Acknowledgements

To my wife, Helen, for all her support throughout both my undergraduate and postgraduate studies.

Also to the rest of my family, for the support over the past six years.

To my supervisor, Professor Edmund Burke, for all his help, support and advice he has given me throughout the duration of my studies, both in relation to this thesis and also in the advancement of my academic career.

To all the academic and research staff within ASAP, past and present, for making my time spent at the University of Nottingham an enjoyable experience.

To Gerard Conroy, my third year project supervisor at UMIST, for first introducing me to genetic algorithms.

Finally, to my internal examiner (Peter Cowling) and external examiner (Kath Dowsland) for their thoughts and comments which have been incorporated into the final version of this work.

Chapter 1

"If we knew what it was we were doing, it would not be called research, would it?" Albert Einstein

1. Introduction

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

Cutting out the material in the most effective way usually has a financial incentive. If a company is able to minimise the amount of waste it produces then there is a quantifiable saving in the cost of raw material. This saving can either be passed on to the customer which, in turn, makes the company more competitive in the market place or the saving can be realised in increased profits for the company. In addition, the company may be able to reduce its stock holding which can make additional savings through improved cash flow. The company may also be able to reduce its warehousing capacity, resulting in further savings.

Although the normal motivation for effective stock cutting is financial, companies may have other objectives in implementing efficient stock cutting procedures. For example, there may be a requirement to meet certain orders within a given time.

Under these circumstances the order in which shapes are cut may be more important than the packing so as to meet the deadline.

Consideration may also be given to the quality of the material being used. This is particularly apparent in the leather industry where a given piece of leather is not only an irregular shape but also consists of a number of different areas which vary in quality. Different parts of the final garments may require leather of different qualities. For example, the back of a leather jacket will require the leather of the best quality, whereas leather used in the lining of the jacket can use material of an inferior quality.

1.1 Aims and Objectives of this Thesis

Over recent years researchers have been using various techniques in order to solve cutting and packing problems. However, it is difficult to find works which compares many different search techniques when applied to the same problems. This thesis addresses this issue by using a variety of search techniques to solve instances of cutting and packing problems and comparing the results. This work also introduces new algorithms to this problem domain. In particular, an ant algorithm is applied, for the first time, to cutting and packing problems, as are memetic algorithms. In addition, the problem of finding suitable parameters to the search algorithms is highlighted.

The author could have chosen problems from a variety of domains (for example, scheduling, timetabling etc.) but cutting and packing was chosen as it was of interest to the author, a funding opportunity had presented itself (in the form of

EPSRC sponsorship) and there was always the possibility of developing commercial links to develop the work even further.

In choosing this problem domain this work was able to contribute to the cutting and packing field in a number of ways, not least of all, by developing a revised algorithm to calculate the no fit polygon for two given polygons. It also applies ant and algorithms to this problem for the first time.

The author hopes it is recognised that this thesis contributes to both the cutting and packing field as well as the meta-heuristic and evolutionary computation field but the author feels that the main contribution is in the comparison and development of the search algorithms that have been employed.

1.2 What is the problem called?

One of the challenges in undertaking this work is being able to identify suitable references. Dyckhoff (Dyckhoff, 1990b) gives examples from computer science, operational research, engineering, manufacturing, and other disciplines, where the cutting and packing problem needs to be solved. This has led to the same problems being referred to in many ways (for example, bin packing, the trim loss problem and one-dimensional packing). It was not until Dyckhoff (Dyckhoff, 1990b) suggested a typology for the problem (see section 1.6) that it became apparent just how diverse this subject had become. Unfortunately, although Dyckhoff is frequently cited, many of the same multi-discipline problems remain to this day.

Below are just some of the terms that can be used to describe cutting and packing problems.

Assortment Problem:

This problem is concerned with deciding which stock pieces to hold (and then use) so that the objects being cut are done efficiently.

Bin Packing Problem:

An assignment of items to bins, minimising the height of the bins. Also see, knapsack problem.

Cutting Stock Problem:

This is normally used as a generic term for the entire class of cutting and packing problems. Strictly, the term should be used to refer to a problem where the parts to be cut and the stock sheets are both rectangular.

Layout Problem:

An arrangement of the stencils (shapes) on the surface is called a layout (compare with marker problem). Normally applies to the clothing industry.

Loading Problem:

Normally refers to three dimensional problems as these are often concerned with loading pallets or lorries.

Knapsack Problem:

Determine a combination of a set of objects, that each have a value associated with them, such that a function is minimised. This problem can be viewed as a bin packing problem with some additional constraint on the value assigned to each piece.

Marker Problem:

A legal placement of the stencils on the surface (compare with layout). Normally applies to the clothing industry.

N-Partition Problem:

Consider all possible combinations of a set of numbers which sum to a given *N*.

Nesting Problem:

Packing of irregular shapes. Often used in the ship building industry.

Orthogonal Packing:

The rectangles to be packed have to have their sides parallel to the bottom or vertical edges of the bin.

Stencil Problem:

A set of irregularly shaped two-dimensional objects which have to be placed on the surface.

Template Layout:

A stock cutting problem where there are no restrictions on the number of shapes to be cut. For example, it might be necessary to cut a number of shapes from a coil of sheet metal where the number of shapes to be cut is limited but it is important to cut as many as possible.

Trim Loss Problem:

The waste material after all the shapes have been cut is often referred to as the trim loss. This type of problem aims to minimise the trim loss.

1.3 Problem Dimensions

A stock cutting problem can be categorised in one of *n*-dimensions. The most usual values for *n* are 1, 2 and 3 and these type of problems are discussed below, followed by problems with higher values of *n*.

The one-dimensional (1D) problem can be seen as cutting pieces from a given length of material. It is often referred to as strip packing. In this problem we are only concerned with the length of the pieces that results from the cutting process. The width of the resultant pieces is either immaterial or (more usually) fixed. Typical applications for the 1D problem involves cutting lengths of steel from steel bars. Many 2D problems can be converted to 1D problems by ignoring the width of the pieces to be cut, thereby converting the problem to one in which only the height of the shapes are of concern.

In the two-dimensional (2D) cutting problem we are concerned with cutting two dimensional shapes from two dimensional stock sheets. The shapes can either be rectangular in shape or can be irregular shapes.

Typical applications include glass cutting, sheet metal cutting and cardboard cutting.

In the three-dimensional (3D) problem we are normally concerned with packing three dimensional shapes into a given area.

Typical applications for the 3D problem include pallet loading and loading delivery vehicles.

It might not seem possible that we can have an *n*-dimensional problem where *n* > 3, however, the *n*-dimensional stock cutting problem can be applied to applications which might not seem immediately obvious. For example, arranging programs in computer memory can be regarded as a 1D problem, where the width of the stock sheet is the *width* of the memory (maybe 8 bits) and the height of the stock sheet can be seen as the program length. Once we notice that many applications can be viewed as a stock cutting application we can add further dimensions; for example a temporal dimension. As an example, a 3D packing problem, such as loading a goods delivery vehicle, may have a temporal dimension added due to the order in which the goods will be unloaded. This results in attempting to pack the goods so that they are easily accessible at each delivery point.

1.4 Types of Cutting

Guillotine cuts only allow a cut from one side of the stock sheet to the other. Some problems only allow guillotine cuts and the shapes have to be laid out in such a way as to accommodate this constraint. Figure 1.1 shows an example of two layouts. One can be cut with guillotine cuts whilst the other cannot.

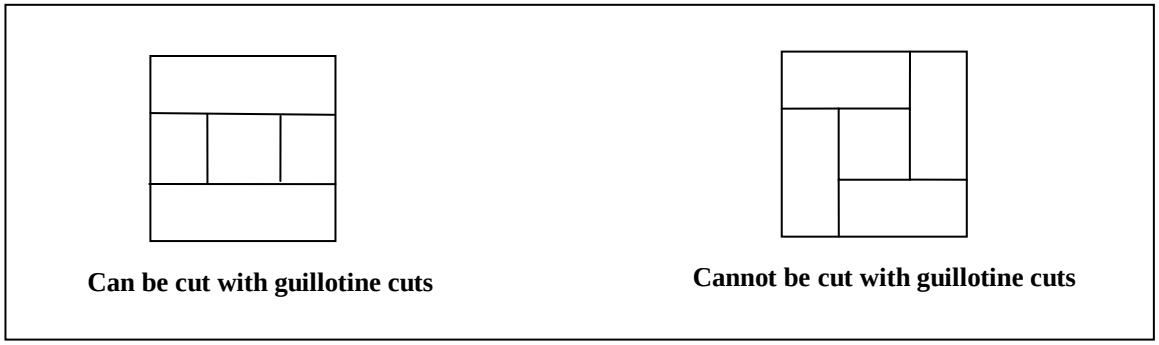


Figure 1.1 – Guillotine Cuts versus Non-Guillotine Cuts

Only guillotine cuts, for example, are allowed when cutting glass, due to the nature of the material. The type of cutting tool can also dictate the type of cutting that is allowed. If the company is cutting cardboard, a sharp blade (such as a Stanley blade) could be used. If the items to be cut are packed tightly (i.e. next to each other), they may be no scope to rotate the blade whilst within the cardboard. Therefore, only guillotine cuts are allowed, with the blade being rotated outside the confines of the material being cut. One company visited by the author (Molan UK) cuts polycarbonate shapes from stock sheets. The shapes are ultimately used in the manufacture of conservatories. To carry out the cutting operation Molan use either a cross cut saw or a router. The cross cut saw can only perform guillotine cuts. This is for the same reasons as described above with regard to cutting cardboard, with the added constraint that the polycarbonate is too rigid to allow the blade to rotate. Cutting using a router allows any pattern to be cut (depending upon the material). This is due to the fact that the router is, in effect, a drill bit which can move in any direction. The disadvantage is that the router is wider than the cross cut blade so that the shapes have to be increased in size to accommodate for

the waste produced by the router. Molan actually insist that their polycarbonate shapes are laid out so that they can all be cut with guillotine cuts. This allows them to use either the cross cut saw or the router for any job.

Another company visited during the course of this work (Esprit Automation Ltd.) produce cutting machines for various industries. Their machines offer the option of not having to be constrained by guillotine cutting but they need to deal with an additional complication. As the cutting tool turns a corner it can lead to *burring* which results in the corners of the shapes not being cleanly cut. To alleviate this problem Esprit use a cutting pattern which *leaves* the shape at a corner and performs a type of *butterfly* cut. This allows it to produce sharp corners. This cutting operation is shown in figure 1.2, with the broken line showing the path of the cutting tool.

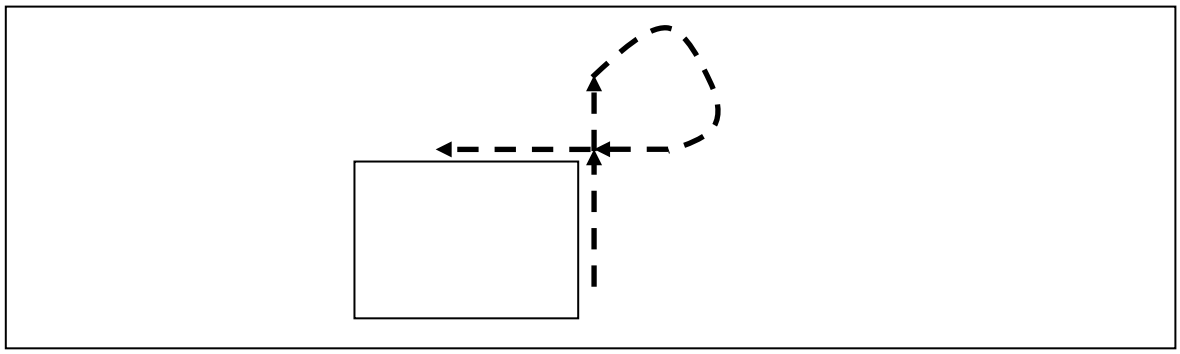


Figure 1.2 – A Butterfly Cut

1.5 Placement versus Cutting

The ultimate industrial aim is normally to cut out a number of shapes from a given stock sheet. However, for the purposes of research, the problem is often reduced to simply laying out the shapes and disregarding the cutting operation. To find the

best layout is often acceptable but in some situations simply finding a good layout may not be good enough to solve a particular problem. For example, an additional constraint might be to minimise the cutting time. This could involve solving an instance of a travelling salesman problem to find the shortest possible *tour* for the cutting tool.

1.6 Classification

In order to make information exchange across different disciplines Dyckhoff (Dyckhoff, 1990b) proposed a typology of packing problems. Packing problems are formed from an intersection of geometry and combinatorics. Problems can be divided between those involving some spatial dimension(s) and those involving non-spatial dimension(s). The former group consists of packing and loading problems in up to three dimensional Euclidean space. The latter group of problems consist of those not involving a spatial dimension such as a temporal dimension or an even more abstract dimension such as computer memory.

Those problems which operate in Euclidean space can be divided into cutting and packing problems. Cutting problems are those which involve a large object(s) (stock sheet, roll of material etc.) which must be divided into smaller objects, usually represented as a set of orders. Packing problems involve filling a bin(s) with a number of given shapes. From the above description it can be appreciated that cutting problems are usually two dimensional and packing problems are normally three dimensional.

These observations are shown in figure 1.3.

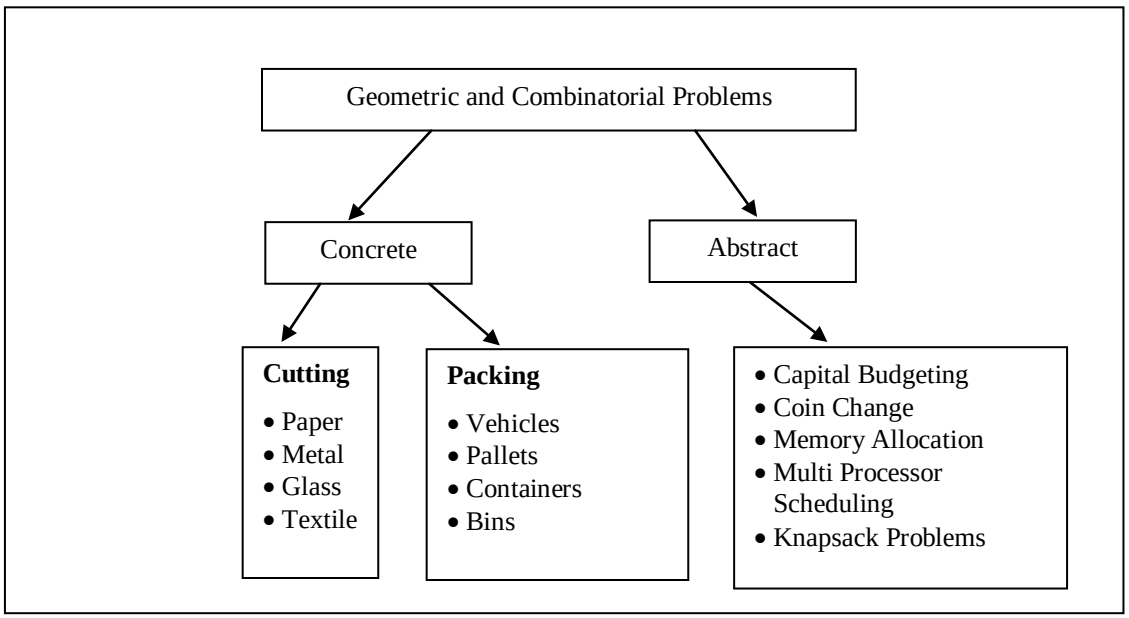


Figure 1.3 - Classification of Cutting and Packing Problems (Hopper 2000b and based on (Dyckhoff, 1990b))

Dyckhoff's classification can be viewed as classifying any problem using four characterics. This typology is shown in table 1.1. Using this proposed typology the problems tackled in this work can be classified as 2/V/O/, that is, two dimensional (2), a selection of large objects that must be packed with a number of smaller items, all of which must be packed (V), there is only one large object (O). Further examples of various problem types can be found in (Dyckhoff, 1992).

Characteristic	Symbol	Description
Dimension	1 2 3	One Dimension Two Dimension Three Dimensional
Kind Of Assignment	B V	All Objects and a Selection of Items A Selection of Objects and All Items
Assortment of Large Objects	O I D	One Object Identical Figures Different Figures
Assortment of Small Objects	F M R C	Few Items (of different figures) Many Items of Many Different Figures Many Items of Relatively Few Figures Congruent Figures

Table 1.1 - Typology of Packing and Cutting Problems (Hopper 2000b and based on (Dyckhoff, 1990b))

1.7 NP-Completeness

P problems are a class of problems that can be solved in polynomial time. They are often referred to as easy problems as P problems include problems with running times such O(log n) and O(n), but this class can also include such problems of order O(n1000). Therefore, easy, does not necessarily transform to short computation times.

Non-deterministic Polynomial problems (NP problems) are a class of decision problems that has some algorithm that can guess a solution and then verify whether or not the guess is correct in polynomial time.

One way to view these problems is to imagine an exponentially large number of processors so that you can try all the guesses at once. Alternatively, you could be very lucky and always guess right the first time. In this case NP problems become P problems. One open question in computer science is whether NP problems = P problems, assuming you do not have the luxury of an infinite number of processors or can guess right every time, which is a realistic assumption.

Most scientists are convinced that P≠NP, that NP problems are inherently hard and only have exponential time algorithms. But this yet to be proven.

If a problem is such that every problem in NP is polynomially transformable to it, the problem is said to be NP-Hard. If in addition the problem itself belongs to NP it is said to be NP-Complete. An example of a problem which is NP but not NP-Complete is finding the XOR of two binary digits. The problem is NP, as it can be verified quickly that the answer is correct. However, knowing how to XOR two binary digits does not help solve another NP-Complete problem such as finding the hamiltonian cycle in a graph. Therefore, this problem is NP but not NP-Complete.

As an example of an NP-Hard problem consider a decision problem to decide if there exists *n* star shaped polygons whose union is equal to a simple polygon, P_1. This problem is NP-Complete. The optimization problem, finding the minimum *n* of star-shaped polygons whose union is equal to P_1, is NP-hard.

The NP-Complete class is of interest as many important problems are known to be NP-Complete. An example is the satisfiability problem. Given a logical expression, is there an assignment of truth values to the variables of the expression that make it true?

Bin Packing has been shown to be NP-Hard in the strongest sense as there is little hope of finding a polynomial time optimisation algorithm for it (Garey, 1979).

Coffman (Coffman, 1988) shows how the one-dimensional bin packing problem can be reduced to a problem of deciding whether a list of numbers can be

partitioned into two sets with equal sums. This problem has been shown to be NP-Complete by (Karp, 1972).

Garey and Johnson (Garey, 1979) is recognised as the standard text on NP-Completeness.

1.8 Motivation for this work

The nesting problem (and variants) have been studied since the 1940's so it is reasonable to ask why is there still an interest in studying this problem? The answer lies in the fact that there are new techniques that can be applied in searching for good quality solutions to the problem.

The motivation behind this work is two-fold. Firstly, although the problem has been the subject of rigorous research there are many new areas for research. This can be seen, for example, in the commercial arena. Over the duration of this research the author has spoken to a number of companies who all need better ways to nest shapes so as to minimise their waste. The problems are compounded by additional constraints set by the companies. This only makes the problem even more challenging. To demonstrate that this area is still very much a research topic, one company the author is working with (Esprit Automation Ltd.) has committed about £50,000 over a three year period. This supports two EPSRC funded projects (a Teaching Company Scheme award and a CASE for New Academics award), which run from October 2000 for three years.

A second motivation for this research is to apply modern search techniques to this problem. This has been done in previous work but this is the first time a variety of

techniques have been applied and compared against one another. In particular, this is the first time ant algorithms have been applied to the nesting problem.

1.9 Outline of the Thesis

This work concentrates on the 2D packing problem. In particular, meta-heuristics and evolutionary algorithms are used to search for good quality solutions to these problems. In addition, the concept of the no fit polygon (NFP) is used to pack two shapes together in the most efficient way.

In chapter 2 a review of related work is presented. A detailed review of cutting and packing is presented, along with a review of various meta-heuristic and evolutionary algorithms.

Chapter 3 presents a simplified problem that was used in order to show that the search techniques applied in the rest of the work are able to effectively explore the search space. This work was published by Burke and Kendall (Burke, 1998).

In chapter 4 methods are presented that allow the search algorithms to be speeded up. Particular emphasis is placed on the evaluation function which is the most computationally expensive part of the algorithm. This work was published by Burke and Kendall (Burke, 1999d).

Convex problems are studied in chapter 5 which uses a convex version of the NFP. Five search techniques are used; Hill Climbing, Simulated Annealing, Tabu Search, Genetic Algorithms and Memetic Algorithms. This work was published by Burke and Kendall (Burke, 1999a), (Burke, 1999b).

In chapter 6 particular emphasis is placed on the ant algorithm search technique as this has never before been applied to cutting and packing problems. The algorithm

is described along with the way it has been applied to cutting and packing problems. The results are compared to those from chapter 5. This work was published by Burke and Kendall (Burke, 1999c).

In Chapter 7, the NFP is considered in some detail, paying particular regard to the non-convex algorithm. One particular contribution of this thesis is in presenting a revised algorithm that allows calculation of the no fit polygon for non-convex shapes. In addition, some recent work is discussed as, during the latter part of this work, other researchers Bennell, Dowsland and Dowsland (Bennell, 2001), independently, published a non-convex NFP algorithm.

Chapter 8 utilises the revised NFP algorithm so that non-convex problems can be tackled. The results are compared against those from chapter 5. In addition, further problems are tackled to demonstrate the effectiveness of the techniques used.

Summary and Conclusions, with suggestions for further work, complete the work.

Chapter 2

"Any change in structure and function which can be effected by small stages, is within the power of natural selection." Origin of Species

2. Related Work

This chapter presents the related work with regards to this thesis. Cutting and Packing is treated mainly chronologically. Following the discussion on cutting and packing, the other two areas of particular importance to this thesis are addressed; computational geometry and search methods.

2.1 Cutting and Packing

2.1.1 Introduction

The stock cutting problem has been a research area since the 1940's, although the main research started in the early 1960's. Over the past half-century many techniques have been developed in order to produce good cutting patterns for a variety of problems and industries.

Due to the problem being NP-complete, exact techniques, such as linear programming cannot be used for large instances of the problem. In recent years heuristic based techniques have been applied to the problem and this is likely to continue to be an active research area for many years. This is not only due to the

fact that heuristic based approaches have been shown to produce good quality solutions to the problem but also due to the fact that new heuristic approaches are being investigated all the time. As an example, this work considers ant algorithms for the first time in the context of cutting and packing.

In the future, as well as continuing to develop heuristic based approaches a new idea, named hyper-heuristics, is starting to attract academic interest. This approach develops heuristics that choose other heuristics in order to find a good solution for a particular *problem instance*. In fact, the author has just started working in this area and the results will, hopefully, be implemented within a commercial application. This work is being done with the support of two EPSRC grants (a CASE for new Academics award and a Teaching Company Scheme award) and with additional funding from a local company (Esprit Automation Ltd.).

It is interesting to note how interest in the subject has developed over the years. In (Sweeney, 1992) a summary of the papers published for the stock cutting problem between 1940 and 1990 is given. Sweeney lists over 350 papers, various dissertations and theses as well as work in progress and conference proceedings. In total over 400 references are given. A summary, by year, of the number of papers published is shown in figure 2.1.

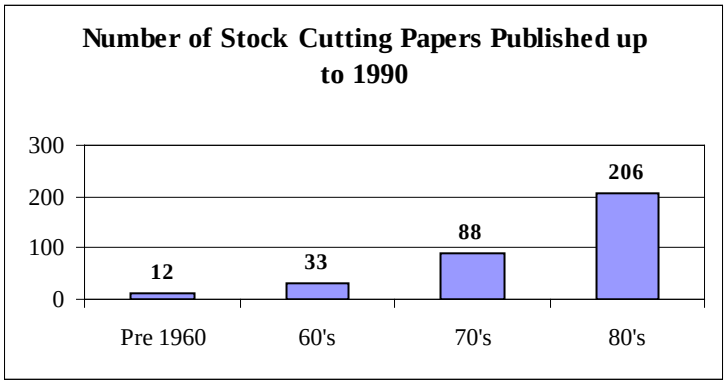


Figure 2.1 - No. of published papers between 1940 and 1989 (Sweeney, 1992)

It can be seen how interest has developed since the 1960's. It would not be practical to mention every paper that has been published in this literature review. Therefore, the approach taken is to highlight those papers which are thought to be the most important within the field and also those most applicable to the research area that this thesis addresses.

Over the years there have been several survey documents produced looking at work carried out to date. The main survey papers are listed in table 2.1.

Author(s)	Year	Author(s)	Year
Brown	1971	Martello and Toth	1987
Salkin and de Kluyver	1975	Rode and Rosenberg	1987
Golden	1976	Dyckhoff, Finke and Kruse	1988
Hinxman	1980	Dyckhoff and Wäscher	1990a
Garey and Johnson	1981	Coffman and Shor	1990
Israni and Sanders	1982	Dowsland	1991
Sarin	1983	Haessler and Sweeney	1991
Rayward Smith and Shing	1983	Dowsland and Dowsland	1992
Coffman, Garey and Johnson	1984a	Dyckhoff and Finke	1992
Berkey and Wang	1985	Sweeney and Paternoster	1992
Dowsland	1985	Dowsland and Dowsland	1995
Dyckhoff, Kruse, Abel and Gal	1985	Hopper and Turton	1997
Israni and Sanders	1985	Hopper and Turton	2000a
Dudzinski and Walukiewicz	1987		

Table 2.1 - Survey Papers for Cutting and Packing Problems

The review by Hopper and Turton (Hopper, 2000a) is particularly useful, being the most recent. The author has also found K. Dowsland and W Dowsland ((Dowsland, 1985), (Dowsland, 1991), (Dowsland, 1992) and (Dowsland, 1995)) to be particularly useful during the course of this research, due the number of survey papers they have published and due to the their depth of experience in this field.

2.1.2 The Stock Cutting Problem

In 1940 Brooks et.al. (Brooks, 1940) published a paper discussing how to dissect a rectangle into squares. Nothing more was published on stock cutting problems for over ten years, until Dantzig (Dantzig, 1951) and Kantorovich and Zalgaller (Kantorovich, 1951) highlighted the relationship between the stock cutting problem and linear programming (LP) theory.

(Paull, 1956), (Eisemann, 1957), and (Vajda, 1958) used linear programming techniques to solve a restricted version of the rectangular layout problem when cutting rolls of paper.

Most of the work done before 1960 was concerned with the paper manufacturing industry. In 1961 Gilmore and Gomory (Gilmore, 1961) presented a linear programming method that, for the first time, solved the one-dimensional cutting problem to optimality. In (Gilmore, 1963) this work is developed further and was applied to the problem of cutting paper. They give an example of the complexity of the problem by stating that faced with a standard roll of paper of 200in and a demand for forty different lengths ranging from 20in to 80in the number of possible cutting patterns can exceed 100 million. This means they are faced with an LP problem with up to 100 million columns. In their paper they outlined a column generation procedure that avoided this difficulty. The procedure involves solving a knapsack problem at every pivot step.

Many future papers would refer back to this work and develop it further.

In (Gilmore, 1965) Gilmore and Gomory presented a method for solving the two-dimensional stock cutting problem. They had to place some restrictions otherwise the number of columns in the LP model was too great. However, they argued, that many industries have such restrictions and that their work could still be used to solve practical problems. One restriction they placed on the problem was that the cutting had to be done in stages. In effect, this meant only allowing guillotine cuts.

In (Gilmore, 1966) they studied more closely the knapsack problem when it is applied to one and two-dimensional stock cutting. In this work they used a dynamic programming technique.

In (Wolfson, 1965) the trim loss problem was approached from a different angle. Instead of trying to minimise the waste using available stock lengths, Wolfson considered the best lengths to hold in stock so that trim loss was minimised. If, for example, you have two stock lengths of 22" and 17" and you need to cut two lengths of 12" then the trim loss is 15" ([22-12] + [17-12]). But if the stock lengths were 20" and 16" then the waste is only 9".

The problem is that as the number of stock lengths and required lengths increases, so does the scale of the problem. Wolfson states that if you are trying to cut ten lengths from one hundred stock lengths then there are $1.7x10^{12}$ combinations.

In order to reduce the problem to manageable proportions Wolfson points out that certain trim losses remain fixed no matter what stock lengths are being used. If, for example, you have stock lengths of 17" and 22" then any length greater than 17" must be cut from the 22" stock item. In this case the waste remains constant. This principle is used to reduce the size of the problem space.

This work was enhanced by (Cohen, 1966) who showed that the problem could be modeled using dynamic programming.

(Barnett, 1967) showed that a simplified version of the stock cutting problem could be solved using simpler algorithms that those previously presented. Their algorithm allows non-guillotine cuts which are often necessary in order to obtain an optimal solution. The ability to allow non-guillotine cuts was at the expense of

placing restrictions on the size of rectangles with regards to their relationship to one another.

The placement of the rectangles is based on the method of removing two opposite corners of a chess board and then filling the remainder of the board with dominoes (Golomb, 1966).

In 1968 (Hahn, 1968) presented an algorithm that optimally produced a layout for stock sheets that contained defects. In fact, the defects were marked as rectangles on the stock sheet and the other rectangles were fitted in around these. All the stock sheets were considered to be of the same size.

The algorithm was based on the dynamic programming technique of (Gilmore, 1966) but, as the cutting process was done in three stages, it only needed to solve one-dimensional problems.

The technique assigned a value to each rectangle. It is important that a large rectangle be assigned a higher value than two rectangles with the same combined area. This is because it is harder to place a large rectangle especially when the stock sheet contains defects. The values for the rectangles are calculated using a quadratic function, which allows the user to supply two coefficients which can be changed to introduce some priority rules. Other user-defined functions are also allowed to assign values to the stock sheets but the functions must be non-linear.

(Haims, 1970) considered the problem of packing irregular shapes into rectangles and then packing the rectangles using a dynamic programming approach. This was to be the first of many solutions that took this two-stage approach.

Haims presents the method he uses to pack the rectangles. The clustering method, which clusters a number of shapes into the smallest enclosing rectangle, is presented in (Haims, 1966).

Eilon and Christofides (Eilon, 1971) looked at the knapsack problem as a zero-one programming problem and also developed a heuristic to solve the same problem. They applied both programs to the same fifty problems. In all but two, the heuristic method found the optimal solution, using significantly less computational time than the zero-one method. The heuristic method worked by sorting the items to be packed in ascending order of value and also sorting the available space in each knapsack in ascending order of size. Then the algorithm searched through both lists and assigned items to the best available space. There was also a "re-shuffle" procedure which improved feasible solutions but it was found that this was not always needed.

(Haessler, 1971) pointed out that the trim loss was not the only factor that should be considered when trying to devise a suitable cutting pattern. It may be the case that an optimal cutting pattern requires that the operators have to frequently set up the machine in order to accommodate a new pattern. It might be preferable to have a less optimal cutting pattern (i.e. with higher trim loss) but which results in the operators not having to attend to the cutting machine as often.

Haessler formulated the problems as a mathematical programming model and used a heuristic procedure from a PhD dissertation (Haessler, 1968) to solve the problem.

In (Haessler, 1975) a stock cutting problem, where the trim loss is not the only consideration, is formulated such that it had a certain aspiration level. If the aspiration level was not reached in one iteration it was lowered and another iteration was done. This continued until a solution was found within the aspiration level.

(Herz, 1972) used a recursive search technique for multistage guillotine cutting problems. It was an improvement over exhaustive iterative-type approaches but was not efficient for medium sized problems.

Dyson et. al. (Dyson, 1974) also considered the problem where the trim loss is not the only factor to consider. They stated that, in practice, there were many factors that had to be taken into account when devising a cutting pattern. One example they give is that if an order is not completed in a continuous number of stock sheets then the partially competed order has to be taken from the production line and stored until the other parts of the order have been cut.

In order to overcome this they use Gilmore's linear programming model to create a cutting pattern and then they used a travelling salesman algorithm to order the cutting patterns so that there are no discontinuities in the orders. They also tried a heuristic approach but reported that it was only marginally better than the manual methods and could not be implemented for practical reasons (e.g. the computer being too far away from the shop floor).

In (Freeman, 1975) an algorithm is given that encloses a shape into a minimal rectangle which could then be packed using one of the known algorithms.

(Adamowicz, 1976a) presented a method for laying out rectangles using a dynamic programming method. The method mimics the way a human operator would lay out rectangles. That is, by laying the rectangles, that have at least one common dimension into strips. The solutions were not often optimal but they were generally good and the computation times were reasonable.

Adamowicz and Albano (Adamowicz, 1976b) considered how to cluster shapes into rectangles as it is easier to lay out rectangles on a stock sheet rather than a given set of polygons.

This work has been much cited mainly due to the use of the No Fit Polygon (NFP), which had originally been presented in (Art, 1966). The NFP allows you to find all the possible positions that one polygon can take with respect to another so that the polygons touch but do not overlap. Using this approach it allows you to find the minimal enclosing rectangle for two polygons.

The NFP forms a major theme throughout this thesis. There are two methods to calculate an NFP for non-convex polygons. One method uses the orbiting principle described in section 4.2 and which is investigated in more depth in chapter 7 of this work for non-convex pieces.

An alternative method is to use minkowski sums. Minkowki sums are widely used in robot motion planning [(Canny, 1987), (Ghosh, 1993), (Lozano-Pérez, 1979), (O'Rourke, 1998), (Ramkumar, 1996), and (Schwartz, 1990)] and in image analysis (Serra, 1982).

In chapter 3 of (Zhenyu, 1994) "*The Theory of Minkowski Sum and Difference*" is presented, with particular reference to compaction algorithms. In this work the author states

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

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

In (Bennell, 2001) the authors state

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

indicating that calculating the no fit polygon for non-convex pieces is still an active area of research. Bennell's paper presents a revised and simplified method for deriving the no fit polygon using minkowski sums and derives a set of simple rules that are easier to implement.

In this thesis the orbiting method of calculating the NFP is used (with modifications – see chapter 7) although we would like to have had access to (Bennell, 2001) earlier in this PhD programme.

Adamoiwicz and Albano (Adamowicz, 1976b) describe how to cluster two polygons together but the principle can be extended to cluster any number of

polygons together. The authors also present a number of problems such as finding a workable data structure for polygon manipulation. This is potentially computationally expensive and has high storage costs. Another problem they pose is trying to find suitable pairs of polygons to cluster from the given set of polygons. One suggestion as to how best to do this is to choose two shapes that are roughly the same size and rotate one through 180°.

The work in (Adamowicz, 1976b) was used in (Albano, 1977) where the techniques were used in an interactive system that allowed the operator to manipulate the layout proposed by the algorithm. As well as briefly describing the method again and presenting the data representation the paper also describes the functionality that is available to the operator.

The work from (Haessler, 1971) was developed by Coverdale and Wharton (Coverdale, 1976). Like Haessler's work their enhanced heuristic algorithm considers other factors other than simply trim loss (e.g. set up time for the machine operators).

Chambers and Dyson (Chambers, 1976) looked at the stock cutting problem from a different angle. In some ways it was similar to (Wolfson, 1965) in that they considered the best stock sizes to hold rather than trying to simply reduce the trim loss using the available stock sizes. Whereas Wolfson considered the one-dimensional problem, Chambers and Dyson applied the problem to a two-dimensional situation (glass cutting).

The paper uses two techniques (integer programming and a heuristic method). Both methods made real savings when applied to real-life problems.

Christofides and Whitlock (Christofides, 1977) suggested a tree search approach for the two-dimensional problem that had a restriction of only allowing guillotine cuts.

The search tree that is produced is reduced in size by ensuring that there are no duplicate nodes (for example, by the same cutting pattern being produced by a different sequence of cuts).

Good computational results were recorded for medium sized problems (for example, twenty rectangles taking an average of 130 seconds and generating a tree with 38,807 nodes).

In 1978, Albano and Orsini (Albano, 1978) also presented a tree based solution which was applied to generic knapsack problems and M-Partition problems.

The solution they present has a simple structure, linear storage requirements and, on average, lower computation times than other algorithms.

The work done by Gilmore and Gomory in 1961 and 1963 (Gilmore, 1961 and 1963) was improved by (Haessler, 1980) when he produced a modified algorithm that gave the same trim loss values but produced solutions with better characteristics. For example, the algorithm used fewer stock sheets. The improved algorithm was at the expense of computational speed but the increased execution time was negligible.

Albano and Osrini (Albano, 1980b) state that the two-dimensional stock cutting problem, even with a guillotine cutting restriction, when stated as a mathematical model, can only be solved to optimality for medium sized problems. This is due to

the time and space complexity. In their paper they present a tree search heuristic algorithm. It is an improved and extended version of (Adamowicz, 1976a). The heuristic is still based on laying out rectangles into strips using a bottom left placement policy but, in addition, the problems can be broken down into sub-problems. How this sub-problem generation is solved is defined by one of six strategies. Experimental results are shown, together with the data that produced those results.

In (Baker, 1980a) a heuristic was presented which packed rectangles at the bottom left of the stock sheet. Using this heuristic it can be shown the employment of an unordered list of rectangles can result in bad packing patterns. However, simply sorting the list of rectangles by decreasing order of widths guarantees that the total bin height is at most three times the optimal height.

This work was improved by Coffman et. al. (Coffman, 1980) when they presented two new algorithms (Next Fit Decreasing Height (NFDH) and First Fit Decreasing Height (FFDH)). Both algorithms take the rectangles to be nested, sort them into height order and place them in rows. For example, the FFDH algorithm tries to improve the nesting efficiency by placing rectangles at the end of rows that still have space available.

In (Albano, 1980b), for one of the first times (if not the first), the term Artificial Intelligence (AI) was used in connection with the stock cutting problem. Albano and Osrini formulated the search for an optimal solution as a heuristic search and used terms which are common in the domain of AI today. For example, they talk

about the search being in a certain state and applying operators to move from one state to another.

The approach they use is an A* algorithm (although they do not use the term). They expand the lowest cost search node which is given by the evaluation function $g(n)+h(n)$, where $g(n)$ is the cost of the search so far and $h(n)$ is a heuristic measure that estimates the amount of waste in the optimal solution should the current piece be included in the placement. If it were possible to find suitable values for $h(n)$ then an optimal solution could be found. However, this is not possible so only "good" solutions can be found. In addition, other restrictions are placed on the search so that the computational times are acceptable. For example, the size of the search tree is limited and the number of successor states is also limited.

Dyckhoff, in 1981, (Dyckhoff, 1981) stated that the LP model developed by Gilmore and Gomory in the early 60's (Gilmore, 1961, 1963, 1965) did not work in some situations. Dyckhoff presented another LP model that had advantages when there were many items to be cut or there were a large number of stock items. This model is also able to deal with cases where the trim loss is not valueless and can be used later in the cutting process.

Another heuristic solution was presented in (Bengtsson, 1982). This addresses the problem of both two-dimensional bin packing (where you can consider the bin as an open ended rectangle) and where the rectangles are fixed in size (i.e. several stock sheets of given sizes). The rectangles are allowed to rotate through 90°. The heuristic sorts the rectangles and then arranges them in piles. As the algorithm

progresses it checks to see if the current arrangement has any chance of beating the best arrangement known so far (by assuming that there is zero waste for the remaining rectangles). If the previous best arrangement is better than the current one then the current search terminates.

The algorithm is not looking for an optimal solution but it aims to find a near optimal solution in a reasonable amount of time. Experiments on randomly created rectangles produced trim loss of between 2% and 5%. The authors cannot compare their results with earlier work as nobody has tackled exactly the same problem.

A new LP solution to the knapsack problem was presented in (Akinc, 1983). This solution gave optimal values for many of the variables very quickly. A branch-and-bound technique was then applied to the reduced problem. The method worked efficiently for up to 5000 variables.

(Chazelle, 1983) developed this work. The best known implementations required $O(N^3)$ steps. The implementation of the algorithm in this paper required linear space, O(N), and quadratic time, $O(N^2)$. The paper also discusses the data structure required to implement the algorithm and how that data structure should be updated.

Otten (Otten, 1982) produced a data structure called the Slicing Tree Structure. This allowed a series of cuts to be represented as a tree, with each node representing one cut to the stock sheet. Although the idea was first proposed with floorplan design in mind the structure has been used in various stock cutting problems.

Wang (Wang, 1983) used a heuristic approach that, unlike (Christofides, 1977), did not attempt to find all the possible cuts that could be made and thus find the optimal cutting pattern. Wang's approach was to iteratively add rectangles together to form patterns that were suitable for guillotine cutting. Wang gives the data used for the testing and reports that the results were good.

Dori and Ben-Bassat (Dori, 1984) looked at the problem of placing irregular shapes (polygons) onto a plane (stock sheet). The paper considered the template-layout problem. The approach adopted was two phased. The first stage was to find a convex polygon that fitted around the shape. Next, a suitable polygon was found that could be used to enclose the shape and that was also suitable to tile the plane. Test data and results are given and various limitations of the algorithm are listed.

Roberts (Roberts, 1984) used a heuristic technique to tackle the problem of cutting worktops for a local company. The problem could not be formulated as an LP model due to the "L" shaped pieces. Four other reasons are also given as to why the LP model developed by Glimore could not be used.

The heuristic Roberts used is a two stage approach. The first stage effectively breaks down the problem into a series of one-dimensional problems by cutting all the straight pieces and pairing "L" shaped pieces so that they are similar to one straight piece.

Shapes which do not fit into this one-dimensional scheme are catered for on an ad-hoc basis by the heuristic.

Roberts system was actually developed for a customer and in the first twelve months it was judged to have produced less waste than the previous manual system. In addition there was no noticeable build up of offcuts.

In 1985 Beasley (Beasley, 1985a) presented an exact algorithm that solved the two-dimensional stock cutting problem that was not restricted to guillotine cuts. This was the first time an exact algorithm for this problem had been presented.

The method uses a zero-one integer model and allows moderately sized problems to be solved in realistic times.

Beasley published another paper in 1985 (Beasley, 1985b) in which he pointed out an error in (Glimore, 1966) and provided a correct dynamic programming model. This model is applied to large problems and the results are presented.

Also in 1985 a short paper (Smith, 1985) applied genetic algorithms (GA) to the bin packing problem. The problem was represented as a list of rectangles and two algorithms, SLIDE PACK and SKYLINE PACK, were used to process the list to fit the rectangles into the bin. SLIDE PACK places a rectangle in a corner of the bin and lets it fall to the farthest away corner, as if under gravity but which drags it in a diagonal direction. The effect is a zigzag motion as the rectangle falls into place. SKYLINE PACK is slower as its tries all possible positions and all orientations. However, it leads to better packings.

A modified crossover operator had to be used as standard crossover would have produced invalid solutions, in the same way that a standard one-point crossover used for solving an instance of the travelling salesman problem would lead to

invalid tours. Evaluation was carried out on the basis of how well the rectangles had been packed (i.e. trim loss)

The GA produced similar results to a technique based on dynamic programming and using heuristics but produced the results 300 times faster. If higher packing density is required the program can simply be left to run for a longer period.

Dagli (Dagli, 1987) used a heuristic approach to solve the two-dimensional cutting problem. A number of constraints are presented that can arise in this type of problem (e.g. sheet defects) but only three are used within the paper. These are; the rectangles must not overlap, the rectangles must lie within the boundary of the stock sheet and there must be a specified tolerance between the shapes to allow for cutting.

The heuristic is based on the idea of assigning a priority to each shape. This priority is used to select which shape to place next. Eleven priority rules are built into the system together with a user definable priority.

Two of the priorities include the maximum and minimum area. Even using these simple priorities the trim loss was reduced from 20% to 7.7% using a real life example.

Qu and Sanders, in 1987, (Qu, 1987) tackled the problem of nesting irregular shapes. The approach either enclosed the shape in a rectangular module or broke the shape down into a number of rectangles. Qu states that little work has been done in the area of irregular shape nesting but recognises the work of (Haims, 1970), (Adamowicz, 1976b) and (Roberts, 1984). Qu discarded the idea of developing a data structure to store a complete shape representation. This decision

was made for two reasons. Firstly it uses valuable computer memory. Secondly the computational requirements to manipulate such a representation would be too great. Instead, if shape representation was required to greater accuracy, this could be achieved by using smaller rectangles to mimic the contours of the shape.

The heuristic that Qu used was based on placing rectangles as near to the bottom left hand corner as possible.

(Berkey, 1987) looks at the two-dimensional bin-packing problem by adapting the bottom-left bin packing heuristics. The authors present implementation details and their results indicate that their heuristics are suitable for practical use.

(Sarker, 1988) formulated a dynamic programming model that solved the one-dimensional slitting problem. The model takes into account defective areas so that the cuts can be made so as to maximise the value of the resultant pieces.

Towards the end of the 80's a simulated annealing approach was applied to the bin packing problem (Kampe, 1988). This is an optimisation technique first introduced by Kirkpatrick et al. (Kirkpatrick, 1983).

In 1991 (Dietrich, 1991) presented a rule based approach to the trim loss problem. It was based on the principle of the rectangular stock sheet consisting of a number of holes (initially one hole covered the entire stock sheet). The next best stock piece was chosen in order to fill one of the holes. This has the effect of reducing the size of one of the holes, maybe creating another hole and possibly allowing another hole to expand.

It was shown that it was more preferable to select a stock piece that had a "two-degree" fit (that is, it fits a hole exactly in both its dimensions). Following that it

was preferable to choose a “one-degree” fit rectangle (fits a hole exactly in one of its dimensions). Lastly, a “zero-degree” fit was chosen. That is, a stock piece that fits into a hole but one of its dimensions is not the same as any of the dimensions of the hole.

When there was a choice of two stock pieces various heuristics were tried. For example, longest piece, largest area etc. Area heuristics were found to out-perform length based heuristics.

The algorithm was tested using previously published data and results shows that it performs well.

(Yeong, 1991) reports the experiences of a Singaporean company. After outlining research to date the authors describe their particular problem. As none of the published research is able to solve their problem directly they develop their own three stage heuristic, with each stage being optional depending on the outcome of the previous stage. The result of the heuristic is a list of stock items to order. That is, the heuristic solves the assortment problem as opposed to the trim-loss problem.

(Prasad, 1991) uses a heuristic approach to nest sheet metal blanks. The heuristic sorts the blanks in descending order of area. It takes each shape in turn and tries to fit it in various positions in relation to those shapes already placed.

Jain et. al. (Jain, 1992) used simulated annealing to produce an optimal solution for nesting blanks. Their approach places no restriction on the shape of the pieces to be nested but only allows two or three shapes. This is done on the basis that producing blanks from, say, a coil of sheet metal only requires two or three shapes to be nested and then the pattern is repeated.

Due to the fact that there is no restriction on the shapes that can be nested there are various geometry problems that need to be addressed. These are more fully discussed in the geometry section of this literature review but the problem is essentially how to detect if two shapes overlap.

When compared to a multi-start algorithm (e.g. hill climbing and returning the best solution from a number of runs) it was found that the simulated annealing approach found a solution with about half the trim loss of a multi-start approach.

(Arbel, 1993) used a column generation procedure to pack irregular shapes. But, due to the potential number of columns a pruning strategy was used to limit the number of columns that were actually generated.

The time complexity of the algorithm was further reduced by estimating the packing area that a particular column would produce rather than actually doing the packing.

In 1993 (Dowsland, 1993) developed a "jostle" algorithm. Pieces are initially laid out using a leftmost placement policy, using a random ordering of the pieces. Unlike (Albano, 1980a) the placement of the pieces is not limited by the profile of those already placed. Pieces are allowed to jump over one another and fill holes. The pieces are then "jostled" to try and produce a more compact packing.

In 1993 (Oliveira, 1993) used simulated annealing for the nesting problem. Although this optimisation technique had been used for bin packing (Kampe, 1988) and nesting a small number of repeating blanks (Jain, 1992), this was the first time it has been used for a general nesting problem. The simulated annealing algorithm which laid out the pieces randomly (allowing overlaps). The

neighbourhood was defined as the movement of one piece to an adjacent position. The objective function minimised the total length of the plate and penalised overlap.

Another meta-heuristic approach, tabu search, was used by (Blazewicz, 1993). They employed a similar neighbourhood structure to (Oliveira, 1993) except that the piece being moved was not allowed to overlap other pieces.

(Cagan, 1994) used shape annealing which is a development of the simulated annealing idea. Shapes are defined by grammars which dictate permissible orientations of the pieces. The shape annealing algorithm is used to determine whether a randomly selected shape rule should be applied to the current configuration. The example presented in the paper is to pack as many half hexagons as possible into a predefined area. The results are acceptable but not provably optimal.

This extended simulated annealing method is then applied to the knapsack problem and the author reports good, though not optimal solutions.

(Vasko, 1994) considers the assortment problem where a two-stage cutting approach is used that only allows guillotine cuts with the first series of cuts being made across the width of the material and the second series of cuts being made parallel to the width.

The algorithms used are based on a modified version of the authors program from 1991 (Vasko, 1991) and a facility location algorithm developed by Erlenkotter (Erlenkotter, 1978)

Heckmann and Lengauer (Heckmann, 1995) uses simulated annealing, with a dynamic cooling schedule, for the nesting problem (specifically the textile industry which has a range of specific constraints).

As well as describing the cooling schedule and the type of moves allowed the paper also discusses two different ways that shapes can be represented. Both models were tested and the polygonal model was preferred to the raster model because of the improved accuracy given by this representation.

The approach works by initially laying out the shapes so that they fit loosely on to a stock material. Simulated annealing is then used to move the shapes around. During this process overlaps are allowed but the final configuration must not have any overlapping shapes.

(Vassilos, 1995) also used simulated annealing to pack arbitrarily shaped polygons. Much of their work was based on (Jain, 1992). Unlike Jain the number of shapes to be packed is not limited to just two or three. The approach used by Jain to detect overlapping shapes (based on detecting intersecting lines) is not suitable for Vassilos's algorithm as it does not allow for one polygon being completely enclosed by another. Therefore another method is used based on (Sedgewick, 1992).

Vassilos et. al. also show that a simple decrement cooling schedule is not suitable for the larger polygon packing problem. Instead they suggest a polynomial cooling schedule and show that it is effective by using it to pack circles in a square which they compare against previous results using other methods to perform this type of packing.

Heistermann and Lengauer (Heistermann, 1995) looked at the nesting problem within the leather industry. Like the textile industry they are typically working with non-rectangular shapes. In fact, the paper deals with polygonal shapes with pre-processing of any shapes that are not already in this form.

The leather industry has particular constraints which need to be considered. For example, different areas of the hide are of different quality and have to be used for different parts of the finished garment. Also, some parts of the hide may be unusable due to, for example, the hide having marks from a barb wire fence.

To tackle these difficulties the hide is split into different quality zones.

The method used by Heistermann and Lengauer is a heuristic greedy algorithm. They report waste of between 20% and 40%. This compares with average waste of 30% for human nesting. The algorithm takes between two and three minutes to run.

The selection of a greedy heuristic was chosen as this was the only method able to produce a nest in the given timescales (an iterative approach, due to the NP-completeness of the problem, would take too long). The heuristic works by choosing the best place on the hide to place the next piece and then choosing the most suitable piece to place at that location. No iteration or backtracking is carried out.

(Fayard, 1995) used a linear programming model and a heuristic to produce solutions to the stock cutting problem. The paper reports that optimal solutions are found 91% of the time.

The algorithm produces optimal strips of rectangles by solving one-dimensional knapsack problems. Another one-dimensional problem is solved to place the optimal strips onto the stock sheet.

(Li, 1995) considers the problem of compacting layouts that have already been placed. As well as compacting, the authors also use separation so that shapes can be moved apart in order to fit in another shape. This is the first time that this method has been tried in order to improve layouts. Although human operators can produce highly efficient layouts they find it difficult to compact and separate the layouts in order to make the layout more efficient. This is due to the fact that the human operator can only move one piece at a time whereas a computer algorithm is able to move all the pieces at the same time (relatively speaking). A representative from a textile firm stated that a 0.1% reduction in wastage could save the company $2 million a year. Two algorithms are presented. The first, using a velocity-based optimisation model, shows that it is not suitable for this purpose.

The second algorithm uses a linear programming model.

(Kröger, 1995) presents a genetic algorithm (GA) approach to the nesting problem. The representation the author uses is based on a slicing tree structure. This structure can be converted to a string of characters which can be manipulated by the GA and can also be parsed to create the cutting pattern.

The idea of meta-rectangles is used. This groups rectangles together so that the complexity of the problem is reduced.

Various operators (both crossover and mutation) are described and the author states that the work is empirically better than methods such as random search or simulated annealing.

In May 1995 Karen Daniels (Daniels, 1995) published her PhD thesis, undertaken at Harvard University, entitled *Containment Algorithms for Nonconvex Polygons with Applications to Layout*. This was the second student (the other being (Zhenyu, 1994), discussed above) supervised by Victor Milenkovic to publish their PhD thesis in this area in a twelve month period. Daniels focussed on two-dimensional containment problems where both the shapes and the container can be irregular (nonconvex) shaped polygons. Daniels also used compaction techniques to accelerate the search as well as using a pre-packing strategy for multi-stage pattern layout for the apparel industry.

(Bounsaythip, 1996) also used a genetic algorithm as a method of solving the nesting problem. The shapes that were nested were irregular and were represented by a "comb" (this representation can be likened to pressing a comb with moveable teeth into the sides of the shape). Like (Kröger, 1995), a tree based structure is used, which the GA manipulates.

The crossover and mutation operators are described along with experimental results. The results are compared against a simulated annealing implementation and the authors report favourable results.

Also in 1996 (Falkenauer, 1996) presented another genetic algorithm implementation. Much of the paper considers the encoding scheme for the problem

and gives examples of why obvious coding schemes are inefficient both in terms of the crossover operator and in the way that two different genes can represent the same packing solution. A representation is proposed which the author claims is more efficient that the classic 'Holland-style' GA. This representation ensures that the GA keeps groups of good genes together.

Daza's paper of 1995 (Daza, 1995) also used a genetic algorithm to solve the two-dimensional guillotine cutting problem.

Their approach was to use a string representation of a binary tree which was then manipulated by the GA. The latter part of the paper extends the notation so that the rectangles can be rotated. Their results demonstrate that the approach is superior to (Oliveira, 1990), which itself was an improvement on (Wang, 1983).

(Hower, 1996) compares three evolutionary/meta-heuristic algorithms (Genetic Algorithm, Simulated Annealing and Evolution Strategy) when nesting small numbers (4 and 6) of triangles and rectangles. In addition, the three algorithms are also compared against a CSP (constraint satisfaction problem) model of the problem which finds all the optimal solutions. The user is able to interact with the system so that the (semi-) automatic layout can be changed by choosing an object and placing it in another position. The author reports that all three evolutionary strategies were able to find good solutions in very fast times (too fast to be measured).

Shpitalni and Manevich (Shpitalni, 1996) presents a new Integer Linear Programming (ILP) model for the two-dimensional stock cutting problem. The importance of their model was the ability to formulate the problem as an ILP

problem. This had not been done before. The focus of this paper is to build a model; not to accelerate the solution which it admits would take large amounts of computer time.

(Mileham, 1996) presented a new algorithm (the total fit algorithm) for packing rectangles onto stock sheets. The algorithm is designed for low volume, high variety manufacturing environments. The total fit algorithm contains three sub-algorithms (strip fit, area fit and strip squeeze). They are applied in the order given with strip squeeze only applicable if non-guillotine cuts are allowed.

The results from their work are compared against (Baker, 1980a) and (Coffman, 1980) and it is reported that it significantly out-performs these two methods.

(Dagli, 1997) proposed two approaches to the stock cutting problem, both based on artificial neural networks (ANN).

The first approach uses just an ANN, which is trained using back-propagation. The second method combines an ANN with a genetic algorithm (GA).

The results show that the average waste for the ANN approach was 7.88%. The combined ANN/GA approach produced packings between 94% and 97%.

In (Oliveira, 1998) a new algorithm was presented (TOPOS). Pieces are placed on the plate one by one and the location of the next piece is ascertained by calculating the No Fit Polygon using the partial pieces already placed and the piece about to be placed. Once the best placement has been found the piece is added to the partial solution and the next piece is placed. The results of this work are compared (directly or indirectly) with the results of (Dowsland, 1993), (Oliveira, 1993) and (Blazewicz, 1993). The algorithm is run against five data sets, which are available

in the literature and also given in the paper, and computational results are presented.

Bennell and Dowsland (Bennell, 1999) use tabu search to produce solutions for irregular packing problems. In addition, problem specific knowledge is injected into the problem domain and this is shown to produce better and better solutions as more changes are made. The results show that their approach is competitive with other approaches reported in the literature.

2.2 Computational Geometry

A complete literature review of computational geometry is not presented here as most of it is not relevant to the stock cutting problem. For example, much of the computational geometry literature, considers computer graphics which is outside the scope of this work.

One of the fundamental data structures in computational geometry is the representation of a point. All of the textbooks implement a point in some way; maybe as a C++ struct (Sedgewick, 1992), as a typedef of an integer (O'Rourke, 1998) or as a C++ class (Laszlo, 1996). A point is generally represented as an integer because of the increased speed and accuracy over a floating point representation. It is usual to use a Cartesian co-ordinate representation for a point.

One of the primitive operations needed for a point is to rotate it. Hearn (Hearn, 1994) gives the equation that allows a point to be rotated about any rotation position given just its *x*,*y* co-ordinates.

To rotate around the origin the formulae are

$new_x = x * \cos(\theta) - y * \sin(\theta);$

$new_y = x * \sin(\theta) + y * \cos(\theta);$

where θ is a the degree of rotation in radians.

To rotate around a given point, *xp* and *yp*, the following can be used.

$new_x = xp + (x - xp) * \cos(\theta) - (y - yp) * \sin(\theta);$

$new_y = yp + (x - xp) * \sin(\theta) + (y - yp) * \cos(\theta);$

These equations are well known and are given, for example, in (Adamowicz, 1976b).

If a point can be rotated then rotating a polygon is reduced to rotating a series of points. The most common requirement is to rotate a polygon around one of its vertices, *pv*. Each point, with the exception of *pv*, has to be rotated around *pv*.

The literature is in general agreement that the best way to represent an arbitrary polygon is by using a list of points to represent each vertex (Laszlo, 1996, O'Rourke, 1998, Sedgewick, 1992) with the last vertex in the list assumed to be connected to the first. This can either be done by using a circular linked list. Or, alternatively, by an array or linear list and using the *mod* operator to implement the circular property.

It is also generally accepted that the points should be in counter clockwise order.

Once the representation of a polygon has been defined there are fundamental operations that we need to be able to perform on the polygon. Many of these operations call upon primitive functions.

One important primitive is to calculate the area of a triangle given its Cartesian co-ordinates. O'Rourke (O'Rourke, 1998) presents one such algorithm and describes how this can be used as a basis to calculate the area of a polygon. In essence an arbitrary vertex on the polygon is taken and triangles are formed by joining this vertex to all others. The polygon area is now reduced to summing the area of the triangles. This works for convex as well as non-convex polygons as, in non-convex cases, triangles are formed *outside* of the polygon but due to the orientation they will have negative area and will be deducted from the overall sum.

Another useful primitive is to decide the relationship between two points, p_1 and p_2, and another point, p_3. It is useful, for example, to decide if p_3 is co-linear with p_1, p_2 or if p_3 is to the left (or right) of p_1,p_2.

In (O'Rourke, 1998) this property is returned via a Left predicate (i.e. is p_3 on the left of the line represented by p_1, p_2?). The Left predicate calculates the area of a triangle formed by the three points. If the area is positive it indicates that p_3 is to the left of p_1 and p_2. If the area is zero, the points are collinear. If the area is negative it shows that p_3 is to the right of p_1 and p_2.

(Sedgewick, 1992) uses the same basic principle but instead of a Left predicate, the operation is implemented as a CCW (Counter Clockwise) function. This returns an integer depending on whether the points travel clockwise or counter clockwise. It is Sedgewick's algorithm that is most often quoted in the literature on Cutting and Packing. This is due to the fact that many of the algorithms only need to know how points are oriented with regard to one another and an area calculation is not required, which is the basis of O'Rourke's algorithm.

An alternative is to use D-Functions (Mahadevan, 1984). This is the primitive adopted in this work due to the fact that the modified no fit polygon algorithm presented in this work uses D-Functions. D-Functions are described in chapter 7.

A fundamental property in computational geometry is to decide if two lines (where a line is represented by two vertices) intersect. Sedgewick (Sedgewick, 1992) uses the CCW primitive to implement this algorithm. O'Rourke (O'Rourke, 1998) uses the Left predicate as the basis for his intersection algorithm.

Both of these algorithms simply return a boolean stating if the lines intersect. Sometimes it is necessary to know the point of intersection. O'Rourke (O'Rourke, 1998) describes an algorithm to do this and states that in this situation it is necessary to leave the *comfortable* world of integer co-ordinates and return floating point values that represent the x and y co-ordinates of the point of intersection.

D-Functions (Mahadevan, 1984) can also be used and their use is shown in chapter 7. Line intersection is obviously fundamental to many other algorithms (such as deciding if two polygons overlap).

Another useful primitive is to calculate the angle a given line (represented by two points) makes with the horizontal. This primitive is useful in many situations. For example, you can take an arbitrary number of points, sort them according to the angle they form with the horizontal and then create a polygon by assembling the points into a polygon data structure using a counter clockwise ordering based on their angle.

Sedgewick (Sedgewick, 1992) uses a theta function to carry out this operation

which, when given two points, returns a real number that is not the angle formed with the horizontal but it does have the same properties. The reason that the actual angle is not calculated is because this would require a call to the tangent function, which is computationally expensive.

Having defined the primitive operations for a polygon it is now possible to define higher level operations.

A fundamental operation for a polygon is being able to find its convex hull. This can be visualised as stretching an elastic band around a shape. The contour made by the band is the convex hull of the shape. Figure 2.2 shows this, although the convex hull, represented by the dashed line has been slightly enlarged so that it is visible.

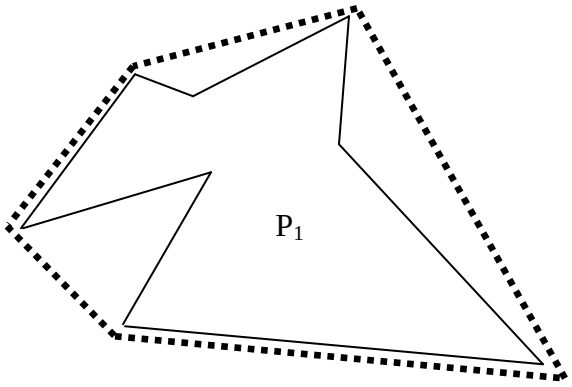


Figure 2.2 – Convex Hull of P_1, shown as polygon with dashed vertices

There are several algorithms that can be used to calculate the convex hull of an arbitrary polygon.

Package (or Gift) Wrapping is described in (O'Rourke, 1998), (Sedgewick, 1992)

and (Preparata, 1985). The original idea for this method was proposed by (Chand, 1970). The algorithm works by choosing a point that is obviously on the convex hull (for example, the vertex with the smallest y co-ordinate) and then sweeping a horizontal ray in the positive direction and moving it round until another point is hit. This point is guaranteed to be on the hull. Of course, in practice the algorithm needs to consider each point to see which one will be hit next by the sweep line. In (Sedgewick, 1992), the theta function is used to determine which point should be next on the hull.

The main disadvantage with this algorithm is that it runs in $O(n^2)$ in the worst case, which is when every point is on the hull.

Other convex hull algorithms are also available. For example, Quick Hull (Preparata, 1985) is an alternative to Package Wrapping but it is the Graham Scan (Graham, 1972) that is most often used with regards to the Cutting and Packing group of problems. The main advantage of this algorithm is that it runs in O(n log n) in the worst case. Most of the computational geometry textbooks show an implementation of the Graham Scan.

Another basic operation needed in cutting and packing is to decide if two polygons intersect. There are two ways in which we can represent the intersection. The first is just to return a boolean value indicating if the polygons intersect. A more complex, but potentially more useful, operation is to return a polygon that represents the intersecting area. In addition we need to consider both convex and non-convex polygons. The operation is a lot more complex for non-convex polygons.

With regard to convex polygons, the obvious method to decide if two polygons intersect is to check each line of one polygon against every line of the other polygon. If any lines intersect then the polygons intersect. However, one degenerate case has to be taken into account, i.e. if one polygon totally includes another. Assuming that no lines intersect, inclusion can be tested for by taking every point on one polygon and checking to see if it lies to the left (using the Left Predicate or the CCW primitive) of every line of the other polygon. In doing this, the normal assumption must hold that the vertices are stored in a counter-clockwise direction so that the edges are directed edges. Figure 2.3 shows this

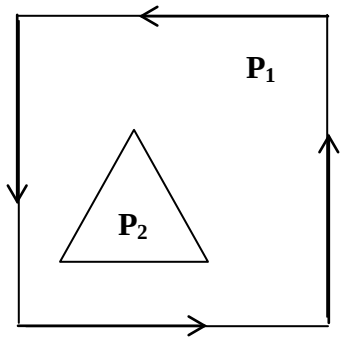


Figure 2.3 – All vertices on P_2 are to the 'left' of all edges of P_1 indicating total inclusion

Sedgewick (Sedgewick, 1992), although not presenting an algorithm to detect if two polygons intersect, does provide all the necessary algorithms to implement the method described above. It also states that such a naive approach to check if any lines intersect runs in time proportional to n^2, where n is the number of edges.

Therefore, an algorithm is presented that can detect if any two lines intersect that runs in time proportional to n log n (but still n^2 in the worst case). This method,

based on maintaining a scan line as it passes across a plane containing the points, was originally proposed by (Shamos, 1976).

In 1978 Shamos (Shamos, 1978) developed the first algorithm that ran in linear time; O(n + m), where *n* and *m* are the number of edges.

In (O'Rourke, 1998) an algorithm is presented that also achieves O(n + m) time. This is based on an algorithm developed by O'Rourke and his students (O'Rourke, 1982). The algorithm involves two lines "chasing" one another around the edges of the polygons and plotting points as the head of the lines are advanced. At termination the algorithm returns a polygon representing the overlap area. This method is also described in (Preparata, 1985).

As in many areas of computational geometry, implementation of algorithms (such as testing for intersection) is delicate. For example, is one polygon touching another returned as intersection? What happens should the polygons only intersect at the vertices (so there are no edge intersections)? These questions, in relation to this work, are considered in chapter 7.

With regard to the stock cutting problem, another useful algorithm is one that generates the No Fit Polygon (NFP). This algorithm determines all arrangements that two arbitrary polygons may assume such that the shapes do not overlap. The concept of the NFP was originally proposed by (Art, 1966) but was used by (Adamowicz, 1972 and 1976b).

This work relies heavily on the NFP, for both convex and non-convex polygons. Chapter 7 is devoted to this topic although the NFP is also discussed in other chapters where necessary.

An alternative to implementing computational geometry algorithms is to use the libraries of routines that are available. Two of the better known are LEDA (http://www.mpi-sb.mpg.de/LEDA/leda.html) and CGAL (Computational Geometry Algorithms) (http://www.cs.uu.nl/CGAL/). It was decided early in this research project that these libraries would not be used. This decision was made for a number of reasons.

- Some algorithms are not provided by these libraries; most noticeably a no fit polygon algorithm could not be found.
- Although the primitives supplied by the library algorithms could be used to implement a no fit polygon it was thought beneficial, certainly in the early stages, to implement these algorithms so that a better understanding could be gained.
- It is recognised that these algorithms are high in space/time complexity. The quote below is from the LEDA FAQ.

  27. What is the run time/space overhead compared to programs written in C?

  Dr. Ulrich Lauther from the Siemens AG made a couple of experiments and compared LEDA graph algorithms with his own hand-coded and well-tuned C-programs. The running time of LEDA programs are typically slower by a factor between 2 and 5. The space requirement of the LEDA graph data structures is larger by a factor of about 2.

2.3 Search Methods

This section presents a review of the search algorithms that were considered during the course of this work.

2.3.1 Heuristic Search Methods

A heuristic algorithm uses domain knowledge in order to search for a solution. The greedy algorithm (so called as it takes the biggest bite towards the solution at any given time) is one example. Applying a greedy algorithm to the travelling salesman problem (TSP) results in taking the shortest route to the next city at any given point. A greedy algorithm cannot guarantee to find an optimal solution but it can give a reasonable solution, although it can suffer towards the end of the search as it may be forced to take more expensive options that were ignored earlier.

The A* algorithm (Hart, 1968, 1972), uses domain knowledge to decide which step to take next. The cost of the solution so far (an exact measure) is added to an estimated cost (using a suitable heuristic) to give an estimated cost of the final solution. A* can be proved to produce an optimal solution if the heuristic never under estimates the cost to the goal. In this case the heuristic is said to be admissible.

The problem with these approaches is two-fold. Firstly, they can lead to poor quality solutions. This can be seen with the greedy approach having to take bad steps towards the end of the algorithm, which degrades the overall quality. Secondly, although a heuristic approach such as the A* algorithm, can be proven to give an optimal solution (provided that the heuristic is admissible), the number of nodes that have to searched rises exponentially for most problems.

2.3.2 Neighbourhood (or Local Search) Methods

To address some of the problems with heuristic algorithms, strategies have been developed which allow the search space to be explored using the concept of a *neighbourhood*. These methods maintain a single solution and move to a neighbouring state in an attempt to find a better quality solution. A set of states is considered at each move and one is chosen depending on the algorithm being used. The way in which the set of neighbourhood states is defined is problem specific. Unlike heuristic approaches, the search does not necessarily need any domain knowledge, only the ability to evaluate a solution so that it can be compared against other solutions.

These search methods are also referred to as *local search* as they confine their moves to a point in the search space that is next to (a neighbour) of the current state.

Various types of neighbourhood search are described below.

Hill Climbing

Hill climbing starts with a random point in the search space and considers *some* of its neighbourhood states. If a state is found that is better than the current state, it becomes the current state. This process continues until a state has been reached which contains no better states in the neighbourhood.

A modification to this algorithm is steepest ascent hill climbing. This considers *all* the states in the neighbourhood before choosing which one will replace the current state. If two neighbourhood states return the same evaluation then one can be

chosen at random. For large neighbourhoods, a neighbourhood size can be specified. In fact, standard hill climbing is simply steepest ascent hill climbing with *neighbourhood size* = 1.

A further refinement is random restart hill climbing. The algorithm is run a number of times, each time starting at a random position. The best solution from all the hill climbing algorithms is returned.

The problem with hill climbing is that it can get stuck in local optima, where the current solution cannot be improved upon but there is a better solution elsewhere in the search space. Rich (Rich, 1993) contains a description of hill climbing, as do the majority of AI textbooks. The algorithm shown below is typical and is taken from (Russell, 1995).

```
Function HILL-CLIMBING(Problem) returns a solution state
    Inputs:          Problem, problem
    Local variables: Current, a node
                     Next, a node
Current = MAKE-NODE(INITIAL-STATE[Problem])
Loop do
    Next = a highest-valued successor of Current
    If VALUE[Next] < VALUE[Current] then return Current
    Current = Next
End
```

Simulated Annealing (SA)

The ideas that form the basis of simulated annealing were first published in 1953 (Metropolis, 1953). The motivation for the algorithm comes from the physical annealing process in which a solid is heated until it melts and is then slowly cooled to a low energy state so that large, uniform crystals can develop. However, if the material is cooled too quickly then imperfections will occur. The Metropolis

algorithm (Metropolis, 1953) regards the material as a system of particles and simulates the change in energy of the system when it is cooled. The algorithm cools the system until it converges to a steady, frozen state.

Thirty years later simulated annealing was introduced by (Kirkpatrick, 1983) as a strategy to search for feasible solutions in combinatorial optimisation problems.

Simulated annealing is similar to hill climbing in that it always accepts better moves but it can also accept worse moves with some probability. This allows it to jump out of local minima. The probability, P, of accepting a worse move is given by (2.1)

$$P = \exp(-\Delta E/kT) \tag{2.1}$$

Where ΔE is the positive change in the energy level (the difference between the current evaluation and the evaluation of the neighbourhood state)

k is Boltzmann's constant

T is the current temperature of the system

It is usual to drop k, giving the revised acceptance criteria (2.2)

$$P = \exp(-\Delta E/T) \tag{2.2}$$

The temperature is started at a high value and is slowly decreased. At each temperature a number of neighbourhood states are considered. At high

temperatures the system is more likely to accept worse states than it is at lower temperatures. In addition, states which have a larger energy (evaluation) change have less chance of being accepted.

The main issue that arises in developing a simulated annealing algorithm is defining a cooling schedule, {*i*, *t*, *d*, *iter*}, where

- *i*, is the initial temperature
- *t*, is the terminating temperature
- *d*, is a formula used to decrement the temperature
- *iter*, defines how many iterations to carry out at each temperature

Although, under certain mild conditions simulated annealing can be proven to converge to the optimal solution it has been shown that this may require an exponential number of steps which makes it infeasible in practise. (Aarts, 1997) contains what is widely regarded as the best reference discussing the theoretical issues surrounding simulated annealing.

It is usual to calculate the figures for the cooling schedule for a given problem using empirical evidence from experimental runs of the algorithm. To find an initial temperature Rayward-Smith et al (Rayward-Smith, 1996) suggests running the algorithm, initially using a very high temperature, which is reduced quickly. When the temperature reaches the point where between forty and sixty percent of worse solutions are being accepted, this is used as the starting temperature for the algorithm in future runs. A similar idea, suggested in (Dowsland, 1995a), is to rapidly heat the system until a certain proportion of worse solutions are accepted

and then slow cooling can start. This can be seen to be similar to how physical annealing works in that the material is heated until it is liquid and then cooling begins.

Dowsland (Dowsland 1995a) offers many suggestions as to how the cooling schedule can be determined as well as discussing other improvements. For example, it is possible to speed up the algorithm by replacing the acceptance function with one that estimates the exponential, thus removing the need to perform the computationally expensive call to the exponential function.

As well as the references in the stock cutting literature review there are many general references to Simulated Annealing. Most AI text books, for example, (Rich, 1993 and Russell, 1995), describe the algorithm. Other books referred to during the course of this work, include (Aarts, 1997) and (Press, 1996). The algorithm shown below is taken from (Russell, 1995).

Function SIMULATED-ANNEALING(*Problem*, *Schedule*) **returns** a solution state

Inputs : *Problem*, a problem
Schedule, a mapping from time to temperature
Local Variables : *Current*, a node
Next, a node
T, a "temperature" controlling the probability of downward steps

Current = MAKE-NODE(INITIAL-STATE[*Problem*])
For t = 1 **to** ∞ **do**
T = *Schedule*[t]
If T = 0 **then return** C*urrent*
Next = a randomly selected successor of *Current*
ΛE = VALUE[*Next*] – VALUE[*Current*]
if ΛE > 0 **then** *Current* = *Next*
else *Current* = *Next* only with probability exp(-ΛE/T)

The *schedule* parameter represents the cooling schedule described above.

Tabu Search (TS)

Although originally proposed in (Glover, 1977) it took a long time for tabu search to become a popular technique for solving combinatorial optimisation problems. For this reason the seminal papers for TS are normally considered to be (Glover, 1989 and 1990). A book by Fred Glover and Manuel Laguna (Glover, 1998) is likely to be the main general point of reference for tabu search practitioners for the foreseeable future.

TS can be likened to hill climbing with the additional property that it maintains a memory of where it has been in the search space. It does not allow the search to return to a state in its memory for a certain number of iterations. This forces the search to explore states that it has not searched before in the hope that it will lead to better quality solutions.

In addition, unlike hill climbing, tabu search considers its neighbourhood states (or a subset of them) and moves to the best one (taking into account any tabu states), *even if it is worse than the current state*. This allows it to escape local minima.

The memory of previous states is held in a tabu list. A state is tabu, and thus cannot be re-visited, if it exists in the tabu list.

Glover and Laguna (Glover, 1998) addresses many implementation issues with regard to tabu search, such as problem representation and directing the search.

The algorithm shown here is written using a similar style to that used in (Russell, 1995) so that it is consistent with the hill climbing and simulated annealing algorithms, shown previously.

```
Function TABU_SEARCH(Problem) returns a solution state
    Inputs          : Problem, a problem
    Local Variables : Current, a state
                      Next, a state
                      BestSolutionSeen, a state
                      H, a history of visited states

    Current = MAKE-NODE(INITIAL-STATE[Problem])
    While not terminte
        Next = a highest-valued successor of Current
        If  (not Move_Tabu(H,Next) or
             Aspiration(Next)) then
            Current = Next
            Update BestSolutionSeen
            H = Recency(H + Current)
        End-If
    End-While
    Return BestSolutionSeen
```

2.3.3 Evolutionary (or Population) Based Approaches

Unlike the neighbourhood methods discussed above, population based approaches maintain a population of solutions.

Ant Algorithms

Ant algorithms are based on the real world phenomena that ants, despite being almost blind, are able to find their way to a food source and back to their nest, using the shortest route. Chapter 6 investigates this approach for packing problems. As this is the first time ant algorithms have been applied to these problems the basic algorithm is discussed in some detail below. In doing so, the travelling salesman problem (TSP) is used as a model application as the approach adopted in this thesis is based on the TSP model of Dorigo (Dorigo, 1996). This work, tested on the Oliver30 data sets, out performed general purpose heuristics

(tabu search and simulated annealing) and is also competitive with specialised TSP methods (2-opt and Lin-Kernigan (Lin, 1973)). However, in these cases the ant algorithm required much longer run times.

In (Dorigo, 1996) the phenomena that ants are able to find their way to and from food is discussed by initially considering ants that only have to walk in a straight line. However, if an obstacle is placed in the way then the ants have to decide which route to take around the obstacle. Initially, there is a 0.5 probability as to which way they will turn when they come across the obstacle. If we assume that one route around the obstacle is shorter than the alternative route then the ants taking the shorter route will arrive at a point on the other side of the obstacle before the ants which take the longer route.

If we now consider other ants coming in the opposite direction, when they come across the same obstacle they are also faced with the same fifty-fifty decision. However, as ants walk they deposit a pheromone trail. The ants that have already taken the shorter route will have laid a trail on this route so ants arriving at the obstacle from the other direction are more likely to follow that route as it has a deposit of pheromone.

Over a period of time, the shortest route will have high levels of pheromone so that all ants are more likely to follow this route. Dorigo (Dorigo, 1996) states that this collective behaviour of ants forms an autocatalytic behaviour. That is, there is positive feedback which reinforces that behaviour so that the more ants that follow a particular route, the more desirable it becomes.

To convert this idea to a search mechanism for the Travelling Salesman Problem (TSP) there are a number of factors to consider. These are outlined below and are based on Dorigo's seminal paper (Dorigo, 1996).

At the start of the algorithm one ant is placed in each city. Variations have also been tested but Dorigo found that these gave inferior results. Therefore, the number of ants, *n*, in the system is equal to the number of cities (which can also be represented by *n* in this model) and that ant starts in a separate city.

Time, *t*, is discrete. *t*(0) marks the start of the algorithm. At *t*+1 every ant will have moved to a new city and the parameters controlling the algorithm will have been updated. At *t*+*n* each ant will have completed a tour.

Assuming that the TSP is being represented as a fully connected graph, each edge has an *intensity of trail* on it. This represents the pheromone trail laid by the ants. Let $T_{i,j}(t)$ represent the intensity of trail edge (*i*,*j*) at time *t*.

When an ant decides which town to move to next, it does so with a probability that is based on the distance to that city and the amount of trail intensity on the connecting edge. The distance to the next town, is known as the *visibility*, n_{ij}, and is defined as

$$n_{ij} = 1/d_{ij} \tag{2.3}$$

where, *d*, is the distance between cities *i* and *j*.

At each time unit *evaporation* takes place. This (which also models the real world) is to stop the intensity trails building up unbounded. The amount of evaporation, *p*, is a value between 0 and 1.

In order to stop ants visiting the same city in the same tour a data structure, *Tabu*, is maintained. This stops ants visiting cities they have previously visited. $Tabu_k$ is defined as the list for the k^{th} ant and it holds the cities that have already been visited.

After each ant tour the trail intensity on each edge is updated using the following formula (2.4)

$$T_{ij}(t+n) = p \cdot T_{ij}(t) + \Delta T_{ij} \tag{2.4}$$

where

$$\Delta T_{ij}^{k} = \begin{cases} \frac{Q}{L_k} & \text{if the kth ant uses edge}(i, j) \text{ in its tour} \\ & (\text{between time } t \text{ and } t+n) \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

This represents the trail substance laid on edge (i, j) by the k^{th} ant between time t and $t+n$.

Q is a constant and L_k is the tour length of the k^{th} ant.

Finally, we define the transition probability (2.6) that the k^{th} ant will move from city i to city j.

$$p_{ij}^{k}(t) = \begin{cases} \frac{[T_{ij}(t)]^{\alpha} \cdot [n_{ij}]^{\beta}}{\sum k \in allowed_k \quad [T_{ik}(t)]^{\alpha} \cdot [n_{ik}]^{\beta}} & \text{if } j \in allowed_k \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

where α and β are control parameters that control the relative importance of trail versus visibility.

As well as the travelling salesman problem, ant algorithms have also been applied to a number of other areas. Bullnheimer et. al. have applied the technique to Vehicle Routing Problems (VRP) (Bullnheimer, 1999). They showed that the positive results given by applying ant algorithms to the TSP could be repeated for another problem type. They could not improve on published results but claim it is a viable alternative when tackling VRP's as, for practical purposes, deviations of up to 5% are acceptable due to uncertainty about travel costs, service times etc. make perfect planning impossible.

More recently Maniezzo and Colorni (Maniezzo, 1999) have applied ant algorithms to the quadratic assignment problem (QAP). They run their algorithm against test cases from the literature as well as a real world problem. They report competitive results in all cases.

Scheduling (Colorni, 1994 and Forsyth, 1997), Graph Colouring (Costa, 1997), Partitioning Problems (Kuntz, 1997) and Telecommunication Networks (Di Caro, 1998) have also been addressed by ant algorithms.

Additional introductions to ant algorithms can be found in (Dorigo, 1999, Dorigo 1999a and Bonabeau, 1999). Probably, the best resource for ant algorithm literature is the web site maintained by Marco Dorigo (http://iridia.ulb.ac.be/~mdorigo/ACO/ACO.html).

Recently, a new optimization algorithm based on the foraging behaviour of a population of primitive ants has been suggested (Monmarché, 2000). This approach is based on the way a particular species (*Pachycondyla apicalis*) search for prey by partitioning a given surface into many hunting sites.

The author is not aware of any published work that applies ant algorithms to the nesting problem except a presentation at Optimization 98, University of Coimbra, Portugal (10-22 July 1998), which used an ant algorithm to place shapes. Unfortunately the conference did not publish proceedings.

Genetic Algorithms

Genetic Algorithms (GA) were first proposed by (Fraser, 1957 and 1960) and (Bremermann, 1958). These algorithms simulated genetic systems. Despite the age of this early work it still has relevance today as it has recently been revisited by Fogel (Fogel 2000b and Fogel 2000c). John Holland, his students and colleagues at the University of Michigan in the 1960's and 1970's are also credited with carrying out much of the pioneering work in GA's. Holland's book of 1975 (Holland, 1992 - reprinted) is recognised as one of the seminal work's for GA's.

Genetic Algorithms are computer programs that are based on the principle of natural evolution. Holland's *Adaptation in Natural and Artificial Systems* (Holland, 1992 - reprinted) presented the GA as an abstraction of biological evolution.

A GA holds a population of solutions (often known as individuals or chromosomes). The separate parts of an individual are known as genes and the value that is stored in a gene is called an allele. The way in which each solution is represented is largely down to the designer of the GA. Historically, bit strings have been used but these are by no means the only representation that can be used, as shown later in this thesis.

Each individual is assigned a fitness value which indicates the quality of the solution the chromosome represents.

During the execution of a GA algorithm the population is continually replaced by new populations. The new populations are created by applying operators (crossover and mutation) to members of the existing population.

Crossover is seen as the most important operator. It takes two individuals (the parents) and transfers genetic material from parents to produce new individuals (children). An individual's chance of being chosen as a parent is proportional to its fitness. This is done so that the principle of natural selection is mimicked; that is the fittest members of the population are allowed more opportunity to breed in the hope that they will pass their good genetic material to the next population. If this happens enough the population should gradually improve as fitter and fitter individuals are created.

An outline algorithm is shown below

1. Initialise a population of chromosomes
2. Evaluate each chromosome (individual) in the population
    2.1. Create new chromosomes by mating chromosomes in the current population (using crossover and mutation)
    2.2. Delete members of the existing population to make way for the new members
    2.3. Evaluate the new members and insert them into the population

3. Repeat stage 2 until some termination condition is reached (normally based on time or number of populations produced)
4. Return the best chromosome as the solution.

In implementing a genetic algorithm there are a number of parameters to consider.

Population Size : How many individuals should be present at each generation?

Generations : How many generations should be produced before the algorithm terminates (other stopping criteria can also be used such as a measure of convergence)?

Crossover Operator : How should the chromosomes be mated? *N*-point crossover is often used (see any GA reference, below, for example (Goldberg, 1989)). Alternatively, operators which give legal solutions to problems such as the travelling salesman problem, are available. A commonly used operator that falls into this category is the PMX (Partially Matched Crossover) (Goldberg, 1989).

Crossover Probability : Chromosomes only mate with some probability. De Jong's PhD thesis of 1975 (De Jong, 1975) studied a series of five problems and concluded that a high crossover rate should be used, for example 0.6, and a low mutation probability, for example 0.05.

Evaluation method : The fitness associated with a chromosome can simply be the same as the value returned from the evaluation function (fitness is evaluation). However, this can lead to premature convergence so

a linear scaling (linear normalization) is often applied, based on the evaluation value for a given chromosome.

Elitism : Using elitism ensures that the top *n*% of chromosomes, based on their fitness, are carried forward to the next generation.

Introductions to GA's can be found in (Beasley, 1993), (Coley, 1999), (Davis, 1987, 1991), (Forrest, 1993), (Goldberg, 1989), (Man, 1999), (Michalewicz, 1996, 2000), (Mitchell, 1996) and (Reeves, 1995).

Evolutionary Strategies

Evolutionary strategies (ES) are closely related to genetic algorithms. Originally they used only mutation, only used a population of one individual and were used to optimise real valued variables. More recently, ES's have used a population size greater then one, they have used crossover and have also been applied to discrete variables (Bäck, 1991) and (Herdy, 1991). However, their main use is still in finding values for real variables by a process of mutation, rather than crossover.

Work on evolutionary systems began with (Rechenberg, 1965, 1973) when they were used to optimize real valued parameters for airfoils. These ideas were later developed by Schwefel in his PhD thesis (Schwefel, 1975) and in a later paper (Schwefel, 1977).

An individual in an ES is represented as a pair of real vectors, $\boldsymbol{v} = (\boldsymbol{x},\boldsymbol{\sigma})$.

The first vector, $\boldsymbol{x}$, represents a point in the search space and consists of a number of real valued variables. The second vector, $\boldsymbol{\sigma}$, represents a vector of standard deviations.

Mutation is performed by replacing $\boldsymbol{x}$ by

$$\boldsymbol{x}^{t+1} = \boldsymbol{x}^{t} + N(0, \boldsymbol{\sigma}) \qquad (2.7)$$

where $N(0, \boldsymbol{\sigma})$ is a random Gaussian number with a mean of zero and a standard deviation of $\boldsymbol{\sigma}$. This mimics the evolutionary process that small changes occur more often than larger ones.

In the earliest ES's (where only a single solution was maintained), the new individual replaced its parent if (and only if) it had a higher fitness.

Even though this "single solution" scheme only maintains a single solution at any one time, it is sometimes referred to as a "two-numbered evolution strategy." This is because, there is competition between two individuals (the parent and the offspring) to see which one survives to become the new parent.

In addition, these early ES's, maintained the same value for $\boldsymbol{\sigma}$ throughout the duration of the algorithm.

$\boldsymbol{\sigma}$ stays constant throughout the run as it has been proven that the algorithm converges to the optimal solution (Bäck, 1991). Although the global optimum can be proven to be found with a probability of one, the theorem also states that it only holds for a *sufficiently* long search time. The theorem says nothing about how long

that search time might be. To try and speed up convergence rate Rechenberg has proposed the "1/5 success rule." It can be stated as follows

> *The ratio, φ, of successful mutations to all mutations should be 1/5. Increase the variance of the mutation operator if φ is greater than 1/5; otherwise, decrease it.*

The motivation behind this rule is that if many successful moves are being found then larger steps should be attempted in order to try and improve the efficiency of the search. If successful moves are not being found then the search should proceed in smaller steps.

The 1/5 rule is applied as follows

$$\text{if } \varphi(k) < 1/5 \text{ then } \boldsymbol{\sigma} = \boldsymbol{\sigma} c_d$$

$$\text{if } \varphi(k) > 1/5 \text{ then } \boldsymbol{\sigma} = \boldsymbol{\sigma} c_i$$

$$\text{if } \varphi(k) = 1/5 \text{ then } \boldsymbol{\sigma} = \boldsymbol{\sigma}$$

The variable, *k*, which is a parameter to the algorithm, dictates how many generations should elapse before the rule is applied. c_d and c_i determine the rate of increase or decrease for $\boldsymbol{\sigma}$. c_i must be greater than one and c_d must be less than one. Schwefel (Schewel, 1981) used $c_d = 0.82$ and $c_i = 1.22$ (=1/0.82).

The problem with the 1/5 rule is that it may lead to premature convergence for some problems. One answer to this is to increase the population size, which now turns evolutionary strategies into a population based search mechanism. By increasing the population size the following observations can be made.

1. The population size is now (obviously) > 1.
2. All members of the population have an equal probability of mating (compare this to chromosomes in a genetic algorithm which mate in proportion to their fitness).
3. The possibility of crossover can be introduced.
4. As there is more than one individual there is the opportunity to alter σ independently for each member.
5. There are more options with regard to how the population can be controlled. These are discussed below.

In evolutionary computation there are two variations with regard to how the new generation is formed. The first, termed $(\mu + \lambda)$, uses μ parents and creates λ offspring. Therefore, after mutation, there will be $\mu + \lambda$ members in the population. All these solutions compete for survival, with the μ best selected as parents for the next generation.

An alternative scheme, termed (μ, λ), works by the μ parents producing λ offspring (where $\lambda > \mu$). Only the λ compete for survival. Thus, the parents are

completely replaced at each new generation. Or, to put it another way, a single solution only has a life span of a single generation.

The original work on evolution strategies (Schwefel, 1965) used a (1 + 1) strategy. This took a single parent and produced a single offspring. Both these solutions competed to survive to the next generation.

Good introductions to evolutionary strategies can be found in (Bäck, 1997), (Fogel, 1998), (Fogel, 2000a), (Michalewicz, 1996) and (Michalewicz, 2000).

Memetic Algorithms

The Selfish Gene by Richard Dawkins (Dawkins, 1976) introduces the idea of a meme. In 1989 Moscato (Moscato, 1989) coined the term *memetic algorithms*.

> *"Examples of memes are tunes, ideas, catch-phrases, clothes, fashions, ways of making pots or of building arches. Just as genes propagate themselves in the gene pool by leaping from body to body via sperm eggs, so memes propagate themselves in the meme pool by leaping from brain to brain via a process which, in the broad sense, can be called imitation."*
>
> (Dawkins, 1976)

A meme can be considered to be a unit of information, for example an idea which is passed from generation to generation. Unlike genetic material, which cannot be altered by its recipient, a meme can be amended to suit the receiver.

The idea behind a memetic algorithm, with regard to optimisation problems, is to combine evolutionary algorithms with a local search mechanism in the hope that

an individual in the population can be improved. An outline algorithm, based on (Corne, 1999) is shown below.

InitialisePopulation, *Pop*
Foreach individual, $i \in Pop$ do $\leftarrow$ Local-Search-Engine(i)
Foreach individual, $i \in Pop$ do $\leftarrow$ Evaluate(i)
Repeat
    for j = 1 to #*recombinations* do
        SelectToRecombine a set $S_{par} \subseteq$ Pop ($|S_{par}| \geq 2$)
        offspring $\leftarrow$ Recombine(S_{par})
        offspring $\leftarrow$ Local-Search-Engine(*offspring*)
        add *offspring* to Pop
    endfor
    for j = 1 to #*mutations* do
        SelectToMutate an individual $i \in Pop$
        $i_m \leftarrow$ Mutate(i_m)
        $i_m \leftarrow$ Local-Search-Engine(i_m)
        Evaluate(i_m)
        add i_m to *Pop*
    endfor
until termination-condition = true

A good introduction to memetic algorithms can be found in (Corne, 1999) in which Pablo Moscato introduces the topic and other practitioners discuss the area in more depth.

2.3.4 Meta-heuristics

The literature and academic community often use the phrase meta-heuristic. As far as the author is aware there is no definitive statement as to what this term actually means.

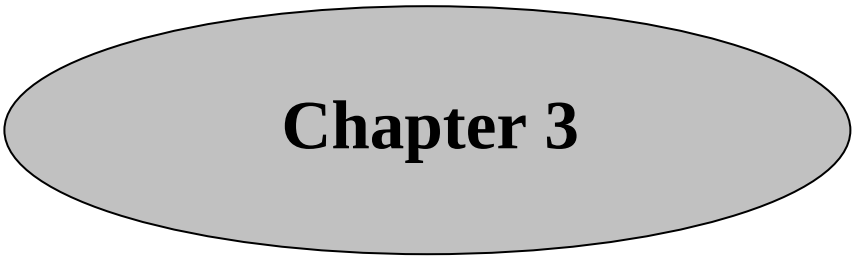
If you look up the term meta in a dictionary (see, for example www.dictionary.com) the term is defined as "*Beyond; transcending; more comprehensive*", suggesting that a meta-heuristic takes us beyond a heuristic algorithm. An alternative, computer science, definition from http://webopedia.internet.com/Computer_Science/meta.html gives the following

> "*In computer science, a common prefix that means "about". So, for example, metadata is data that describes other data (data about data). A metalanguage is a language used to describe other languages. A metafile is a file that contains other files. The HTML META tag is used to describe the contents of a web page.*"

This would suggest that a meta-heuristic is somehow a *heuristic about heuristics*.

Of the two definitions given above, for the purposes of this thesis the first definition is preferred. The algorithms that we use go beyond the usual heuristic approaches.

Therefore, for this work the term meta-heuristic can be used to describe both neighbourhood approaches and population based approaches.



"I have called this principle, by which each slight variation, if useful, is preserved, by the term Natural Selection." Origin of Species.

3. A Simplified Problem

3.1 Introduction

In this, and subsequent, chapters we consider various types of cutting and packing problems and apply various search techniques to find good quality solutions.

In particular, this chapter looks at a simplified version of the problem in order to give an initial comparison of three search algorithms. It is hoped that these initial results will be similar, with regard as to how they compare relatively to one another, when the same search algorithms are applied to more complex problems. In chapter five, convex cutting and packing problems are addressed and in chapter six two new algorithms (ant algorithms and memetic algorithms) are tested on the same problems. Chapter eight compares the same search algorithms as used in chapters five and six on non-convex problems. This incremental approach to the problems and search algorithms is taken so that the results can gradually be built up and more easily compared.

Due to the fact that many types of search algorithms are being applied to different problems, choice of representation was crucial in being able to use different search algorithms on the same problem data. The representation chosen was a list of shapes (rectangles in this chapter, convex polygons in chapters five and six and non-convex polygons in chapter eight) which were packed based on their position in the list. For example, the shape at the start of the list was packed first and the shape at the end of the list was packed last. Using a list allowed the various search algorithms to operate directly on that data structure. Using algorithms such as tabu search, simulated annealing and hill climbing the neighbourhood function could be implemented as a swap operator which simply swaps shapes, for example, at random.

Using a genetic algorithm, the list represents a chromosome, which can use recognised crossover and mutation operators.

Ant algorithms are also able to use the list representation. Each list represents the order in which the ants "visit" the shapes, with a separate data structure being used to maintain the pheromone levels between each shape.

Therefore, we chose this data structure as it provides a convenient representation for all the search algorithms we employ during this research. However, it is recognised that this is not the only representation that could be used. For example, Otten (Otten, 1982) uses a tree representation (called a slicing tree structure). Whilst we could have used a similar representation it would be more difficult to define the various operators for all the searches and, in addition, Otten used this structure for rectangular pieces and it has never been applied to more complex

shapes and certainly not non-convex polygons. Daza et. al. (Daza, 1995) also used a tree representation (a binary search tree) but this suffers from the same limitations as described above for (Otten, 1982).

Dietrich et. al. (Dietrich, 1991) represented the larger sheet, in the first instance, as a single hole that allowed other pieces to be placed within it. Using various heuristics (such as decreasing area first fit, increasing length first fit) shapes were selected and placed into the holes. This had the effect of creating more holes (in some cases holes were combined and enlarged) so that other pieces could be selected and fitted into the newly created holes. Similar to the tree representation, this approach was only applied to rectangular shapes. Feasibly, it could be applied to other shapes but the various operators, specifically for genetic algorithms and ant algorithms, would be difficult to define.

An approach, adopted by Bennell and Dowsland (Bennell, 1999), is to represent the larger shape as a grid. A neighbourhood function can then be defined as moving the smaller shapes a certain number of grid positions, and in a given direction. The shape to be moved, the distance and direction it is moved and the resolution of the grid are discussed within the paper, with experimental results being presented. Bennel and Dowsland employ a variation on tabu search (simple tabu thresholding) to search for good quality solutions and a similar representation was considered for this work but it was not apparent how genetic algorithms and ant algorithms could be mapped onto this representation.

The representation chosen for this work is the same representation as used by Oliveira et. al. (Oliveira, 1998). In their work a series of polygons is packed utilising the no fit polygon at each stage. As a similar approach is used in PhD thesis (that is packing shapes one at a time and using the no fit polygon to decide on the next placement), it seems appropriate to adopt a similar representation as used in previous work.

However, it is recognised that that is not the only representation possible. Despite those mentioned above, and the reasons for rejecting them, others are possible. One of the approaches we are considering, in conjunction with a commercial partner, is to use a bitmap representation and try to fit shapes into the holes on the larger shape (holes being represented by the bits in the bitmap being set to, say, one). At present, only heuristic based methods place the shapes but we hope to extend the method so that we can employ meta-heuristic and evolutionary algorithms. But we already recognise that it is a challenge to define the various operators.

This chapter does not directly address the nesting problem. Instead, it takes a simplified version and investigates three techniques (Tabu Search, Simulated Annealing and Genetic Algorithms) to solve it. The aim of solving a simplified version of the problem is to ascertain whether or not these techniques offer a sensible approach for solving this type of problem. (Blazewick, 1993), (Jain, 1992), (Kröger, 1995), (Parada, 1998) have had success with these techniques but the three methods have not been rigorously compared on the particular problem of

interest in this PhD project. If these approaches show promise on a simplified version of the problem this would indicate that it is worthwhile investigating the same techniques for more difficult problems.

3.2 The Problem

Presented with two rectangles (R_1, R_2) we can make the following observations. By placing R_1 and R_2 in contact with each other we can place a bounding box, B_{box}, around them. The area of B_{box}, Area(B_{box}), depends on how the rectangles are placed in relation to one another. The minimum for Area(B_{box}) = Area(R_1) + Area(R_2) (see figure 3.1). If the rectangles are placed such that the area of B_{box} > Area(R_1) + Area(R_2) the placement is sub-optimal (see figure 3.2). Finally, if the rectangles are of different sizes in both the dimensions then Area(B_{box}) ≠ Area(R_1) + Area(R_2) (see figure 3.3).

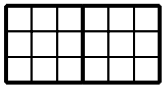


Fig. 3.1 - Placement of two rectangles which has a minimal bounding box

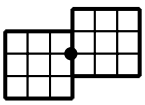


Fig. 3.2 – Placement of two rectangles such that the area of the bounding box is not minimised

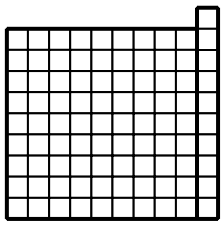


Fig. 3.3 – Placement of two rectangles, R_1 and R_2, which has a minimal bounding box, B_{box}, but Area(B_{box}) > Area(R_1)+Area(R_2)

Figure 3.1 shows two rectangles of size 3x3. The sum of their areas is 18, as is the area of their bounding box. Figure 3.2 shows the same two rectangles but now placed in relation to each other so as to give the bounding box an area of 24. The two rectangles (9x9 and 1x10) in figure 3.3 are positioned in an optimal way so that the area of the bounding box is minimized, but the area of the bounding box is still greater than the area sum of the two rectangles. In fact, the area of this bounding box will always be greater than the area sum of the rectangles, no matter how they are placed in relation to one another. Our simplified version of the stock cutting problem is to take pairs of rectangles from a given set and combine them with the objective of minimising the sum of their bounding boxes.

3.3 Representation of the Problem

Let *x* and *y* be the width and height of a rectangle respectively, expressed as an integer. Let *Bottom* define a point along the bottom of the rectangle where *Bottom* can take value between 0 and *x*-1. Similarly, let *Top* define a point along the top of the rectangle.

Left and *Right*, are used to define a point on the left and right of the rectangle. These values are constrained between 0 and *y*-1.

Finally, one further variable is defined, *Place*, which dictates where one rectangle is placed in relation to another. This variable is discussed in section 3.3.1. These variables are summarised below.

x	:	The width of the rectangle
y	:	The height of the rectangle
Bottom	:	0 >= Bottom <= x-1. Defines a point on the bottom of the rectangle
Left	:	0 >= Left <= y-1. Defines a point on the left of the rectangle
Right	:	0 >= Right <= y-1. Defines a point on the right of the rectangle
Top	:	0 >= Top <= x-1. Defines a point on the top of the rectangle
Place	:	A variable indicating if another rectangle should be placed on top, or to the right, of this one.

3.3.1 Interpretation

Given the rectangles $R_{1..}R_{6,}$ we can pair them as follows; R_1 & R_2, R_3 & R_4 and R_5 & R_6 (and in any other permutation). Given these pairings we can decide how any two rectangles can be placed in relation to one another. Assume R_1 and R_2 have the following values.

R_1 = {x=3, y=3, Bottom=0, Left=2, Right=2, Top=1, Place=On Right}

R_2 = {x=3, y=3, Bottom=1, Left=1, Right=0, Top=3, Place=On Top}

We can place these two rectangles in relation to one another using the following reasoning. R_2 (being the second rectangle of the pair) will be placed in relation to R_1. Using the Place variable of R_1 we note that R_2 should be placed to the right of R_1. We use the Right variable of R_1 to define where R_2 should *joined*. We also need a point on R_2 so that the *join* can be completed. This is given by the Left variable of R_2. We use the Left variable as it is the complement of Right. If we had used the Top variable of R_1 we would use the Bottom variable of R_2. The *joined* rectangle is shown in figure 3.2.

The filled circle, in figure 3.2, shows this interpretation.

3.4 Comparison of Algorithms

To compare the algorithms (simulated annealing, tabu search and genetic algorithm) three rectangle pairing problems were considered. The aim was to observe the quality of the solutions produced when the algorithms attempted to minimise the sum of the bounding boxes of the rectangle pairs in the given set. The first problem consisted of twelve rectangles (2 of 9x9, 2 of 8x8, 4 of 4x4 and 4 of 3x3). The optimum solution for this problem is 390. All the algorithms were able to find this optimum. The second problem consisted of 50 rectangles where the optimum was known to be 20000. A third problem consisted of 100 random rectangles where the optimum is unknown. The results for the 50 and 100 rectangle problem are shown below (table 3.1), with comments following. All results are averaged over ten runs.

	GA		TS		SA	
	Best	Avg	Best	Avg	Best	Avg
50 Rectangles	22053	22149	20024	20099	20047	20089
100 Rectangles	79586	80356	75269	75949	73845	74003

Table 3.1 - Pairing Rectangles using Meta-Heuristic (GA, TS, SA) Algorithms

3.4.1 Crossover, Neighbour and Mutation

In testing the algorithms we used three types of crossover operator for the GA. One was a generic order based crossover (Davis, 1987). The other two crossover operators were developed specifically for this problem. One (named rectangle crossover) ensured that promising rectangle pairs were carried forward to the next generation. A promising rectangle pair is defined as a pair of rectangles, in a chromosome, that satisfies $MIN(Area(B_{BOX})(_{i, i+1}) - (Area(R_i) + Area(R_{i+1}))$. Using the rectangle operator, the most promising pair in a parent is copied to one of the children before the other positions in the child are filled. The other parent and child are treated similarly. An extended version of this operator, the enhanced rectangle operator, finds the most promising rectangle pair in each parent and copies these rectangles to *both* children. In the event that this would lead to duplication in the child normal rectangle crossover is used.

The neighbour for both SA and TS was a random swap of two rectangles (making allowances for the tabu list in TS). Mutation, for all the algorithms, changed one of the variables in a randomly selected rectangle to a random, allowed, value.

3.4.2 Genetic Algorithm (GA)

Initially we carried out GA runs by testing various parameter combinations. The parameters we varied were the population size (50 and 100), the three types of crossover operator, two types of evaluation (fitness is evaluation and linear normalization) and elitism (0.05 or 0.00). The permutation of these parameters led to 24 runs being necessary. Each run was carried out fifteen times using three different starting populations (360 tests in all). Some parameters to the GA algorithm remained constant. These were the crossover probability (0.6), the mutation probability (0.05) and the number of generations (100).

The order based crossover did not yield good results. Using the problem specific operators yielded better quality solutions. Linear normalization produced better results than fitness is evaluation. As might be expected, a population size of 100 produced better results than a population size of 50. Better results were produced when using elitism. The results shown above (table 3.1) used the best parameter combinations we found. However, even with these combinations, TS and SA outperform the GA.

3.4.3 Tabu Search (TS)

To produce the results shown above, TS was run with a list size of 50% of its population size and a neighbourhood size equal to its population size (that is, when looking for a better neighbour at each iteration it considered *n* neighbours). 5000 iterations were performed. To confirm that the TS list was helping find good quality solutions we ran a test, for the 50 rectangle problem, with the list size set to

zero. This yielded a result of 20828. We also tried a list size of 48, which gave a result of 20506. A smaller list size (10) produced an average result of 20632 and increasing the neighbourhood size to 100 (with the list size still at 25) produced similar results to a neighbourhood size of 50.

3.4.4 Simulated Annealing (SA)

To produce the results a starting temperature of 10 was used, a decrement of 0.001 and 100 iterations were carried out at each temperature.

3.5 Conclusions

All three meta-heuristic techniques were able to find the optimum solution to a small problem. When presented with larger problems, tabu search and simulated annealing produced good quality solutions whereas a genetic algorithm produced lower quality solutions despite many tests during the small problem to try and ascertain the best combination of parameters. In addition, the GA runs took longer due to the overheads involved in processing a population of chromosomes, rather than a single individual. In conclusion, for this problem the SA and TS algorithms produce similar solutions and both outperform the GA.

Chapter 4

"As many more individuals of each species are born than can possibly survive; and as consequently, there is a frequently recurring struggle for existence, it follows that any being, if it vary however slightly in any manner profitable to itself under the complex and sometimes varying conditions of life, will have a better chance of surviving and thus be naturally selected." Origin of the Species

4. Evaluation of the Two Dimensional Bin Packing Problem using the No Fit Polygon

4.1 Introduction

When employing meta-heuristic and evolutionary algorithms it is often the case that the evaluation function is the most computationally expensive part of the algorithm. The evaluation function employed in this work (see section 4.2) calculates the no fit polygon (NFP) for two polygons and then calculates the smallest convex hull for the two polygons. This process is repeated for each polygon. As the manipulation of polygons is computationally expensive, the process creates a bottleneck at this stage. However, many of the evaluations are simply re-evaluating solutions that have already been evaluated. Before solving problems using meta-heuristic and evolutionary algorithms some work was carried out in an attempt to speed up the evaluation function.

In order to do this a cache was utilised which stores previous evaluations (the cache was developed using the CMap class available with Microsoft Visual C++

version 6.0). Increasing the size of the cache significantly increases the speed of the algorithm. In addition the concept of a polygon type was also introduced which allows much better use to be made of the cache.

It can also be shown that in some circumstances, it may not be beneficial to use a cached evaluation. Therefore, a re-evaluation parameter is introduced which forces a complete re-evaluation of a solution, even though it might be held in the cache. In this chapter it is shown that this parameter can be set to a small value so that the advantages of the cache are not lost.

The approaches adopted here are intuitive but are not often implemented. However, techniques such as these, will become increasingly important as met-heuristics and evolutionary algorithms are more widely used.

Other researchers have used various methods in order to reduce the computational load of their evaluation function on their algorithm. In (Rana, 1996) a warehouse scheduling problem is solved using a genetic algorithm. The evaluation function is a list based simulation of orders progressing through the warehouse. An internal (detailed) simulator is used to verify solutions. This takes about three minutes. An external (coarse) simulator runs in about one tenth of a second and is used to identify potential solutions. Ross et al (Ross, 1994) uses delta evaluation on the timetabling problem. Instead of evaluating every timetable they show that, as only small changes are being made between one timetable and the next, it is possible to evaluate just the changes and update the previous cost function using the result of that calculation.

In this chapter only convex polygons are considered but the same techniques are used later in this thesis (chapter 8) when working with non-convex polygons.

4.2 No Fit Polygon

The No Fit Polygon (NFP) determines all the arrangements that two arbitrary polygons may take so that the shapes do not overlap but so that they cannot be moved closer together without intersecting. To show how the NFP is constructed consider two polygons; P_1 and P_2. The aim is to find an arrangement such that the two polygons touch but do not overlap. If this can be achieved then we know that we cannot move the polygons closer together in order to obtain a tighter packing. In order to find the various placements the procedure can be described as follows (see figure 4.1). One of the polygons (P_1) remains stationary. P_2 moves around P_1 and stays in contact with it but never intersects it. P_1 and P_2 retain their original orientation. That is, they never rotate. As P_2 moves around P_1 one of its vertices (the filled circle) traces a line.

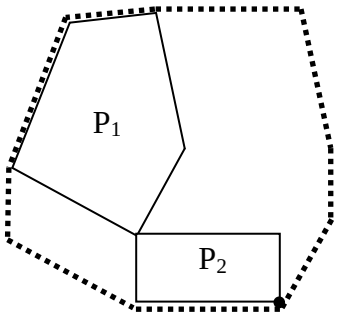


Figure 4.1 – The No Fit Polygon

Figure 4.1 shows the starting (and finishing) positions of P_1 and P_2. The NFP is shown as a dashed line. It is slightly enlarged so that it is visible. In fact, some of the edges would be identical to P_1 and P_2. Once the NFP has been calculated for a given pair of polygons the reference point (the filled circle) of P_2 can be placed anywhere on an edge of the NFP in the knowledge that it will touch, but not intersect, P_1. In order to implement a NFP algorithm it is not necessary to emulate one polygon orbiting another. Cunningham-Green (Cunningham-Green, 1992) presents the algorithm that is used as part of this work.

The no fit polygon is considered in more depth in chapter 7.

4.3. Evaluation

4.3.1 The Basic Method

In order to fill the bin we proceed as follows. The first polygon is chosen and this becomes the stationery polygon (P_1 in figure 4.1). The next polygon (P_2 in figure 4.1) becomes the orbiting polygon. Using these two polygons the NFP is constructed. The reference point of P_2 is now placed on various points on the NFP. For each position the convex hull for the two polygons is calculated. Once all placements have been considered the convex hull that has the minimum area is returned as the best packing of the two polygons. This larger polygon now becomes the stationery polygon and the next polygon is used as the orbiting polygon. This process is repeated until all polygons have been processed. As each large polygon is created, its width is checked. If this exceeds the width of the bin, then a new row within the bin is started. In this case, the polygon which forced the

width of the bin to be exceeded becomes the stationery polygon. That is, the large polygon built thus far forms one row and the next row is constructed using a single polygon as a starting point.

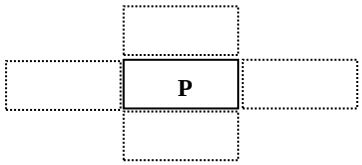


Figure 4.2 – Possible Placements

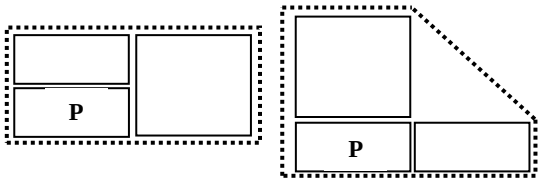


Figure 4.3 – Same Polygons, Different Solutions

There are two problems that need to be addressed in order to make the evaluation strategy feasible. The number of placements the reference point can take on the NFP is infinite. In order to reduce the problem to manageable proportions, we only place the reference point on the vertices of the NFP. The second problem is that there could be more than one optimal placement for two given polygons. Consider two rectangles of the same dimensions. There are four optimal placements, as shown in figure 4.2. No matter which placement is chosen the four evaluations return the same value as the convex hulls all have the same area. Therefore, it is immaterial which one we choose. However, it may make a difference when later

polygons are added. Consider, for example, if the placement is chosen where one polygon is placed on top of P. Depending on the characteristics of the next polygon it could effect the quality of the solution that is being built. Figure 4.3, illustrates this. It shows that by choosing one placement over another different solutions can be obtained later on in the packing.

The cost function that measures a complete (or partial) solution is taken from (Falkenauer, 1998). Rather than simply measuring the bin height (which is the intuitive evaluation function), the cost function measures how efficiently the bin has been packed. This gives a much better search space for the algorithm to explore.

Section 5.2 of this work contains a fuller description of the evaluation function.

4.3.2 Caching of Evaluations

Manipulating polygons is computationally expensive and, due to the nature of evolutionary algorithms, the evaluation function has to be called many times. It is likely that the same evaluation will be performed many times. This makes this part of the algorithm a serious bottleneck. In order not to evaluate previously seen solutions again a cache has been implemented. Each polygon is assigned a unique identifier. In this way a solution (complete or partial) can be recognised by considering a concatenation of the identifiers (a key), which is used to access the cache. The cache either returns a null value, meaning that the solution is not present, or it returns the previous evaluation which means the evaluation function does not have to be called. In addition a polygon data structure is held in the cache so that this can be returned and paired with the next polygon. In fact, the cache can

hold more than one polygon for each hash key. Figure 4.2 shows four optimal placements. These placements would all be held in the cache but only one (randomly selected) is returned.

4.3.3 Polygon Types

Another potential bottleneck is evaluating polygon permutations that have already been evaluated. Consider polygons with identifiers ABCDE and assume the polygons DE are identical. After evaluating the permutation ABCDE, {AB, ABC, ABCD, ABCDE} will be stored in the cache. Later in the algorithm ABCE might need to be evaluated. This key will not be in the cache and the evaluation function is called. In fact, there is no need to do this as polygons D and E have the same dimensions and evaluating ABCD will yield the same result as ABCE. In order to cater for this, the notion of a polygon type was introduced. This gives each polygon a type identifier which acts as a pointer to the polygon description. This can significantly reduce the size of the search space and allows for much more effective use of the cache.

4.3.4 Forcing Re-evaluations

One problem in using a cache is that it might be holding an evaluation which is not the optimal for a particular permutation of polygons. An example of this is shown in figures 4.2 and 4.3. If the first two (smaller) polygons are evaluated, it will be found they can be positioned in four ways (see figure 4.2). The placement chosen is random, so assume one polygon is placed next to the other. When the third polygon is evaluated, the best configuration that can be found is that shown on the

right of figure 4.3. It is the result from this evaluation that will be stored in the cache and whenever these three polygons are evaluated, it will be this value that is returned from the cache. We will never have access to the better solution (left hand side of figure 4.3) where the first two polygons are placed on top of one another. The problem is that once a configuration is stored in the cache, the only way we can arrive at another placement is for the cache to exceed its limit and for the (possibly) inferior solution to be discarded. However, there is no guarantee that this will happen. In fact, the larger the cache size (in the hope of improved performance) the less chance there is of items being discarded from the cache. In an attempt to alleviate this problem a re-evaluation parameter is introduced. This is set to a value between 0 and 1 and determines the probability of the solution being re-evaluated, regardless of whether or not it is in the cache. A value of zero means that the value in the cache, if it exists, should always be used. A value of one means that the solution is always re-evaluated, effectively ignoring the cache. Higher values of the re-evaluation parameter will slow the algorithm down as it is not making as much use of the cache. However, re-evaluating solutions should mean that better quality solutions are found as more of the search space is explored.

4.4 Testing, Results and Comparisons

4.4.1 Test Data

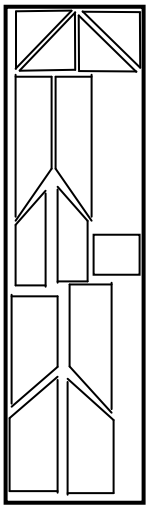


Figure 4.4 – Test Data 3

Testing was carried out on three sets of data. The first test consisted of fourteen pieces which were defined using three different polygon types. The second test used the same pieces but this time the polygons were defined as (fourteen) different types. The third test used the pieces shown in figure 4.4. This problem is taken from a company that cuts polycarbonate pieces for the manufacture of conservatories. This data is defined using different types.

4.4.2 Testing the Cache

The upper bound for the cache size was found by starting with a high value and recording the maximum number of elements held in the cache at the completion of the run.

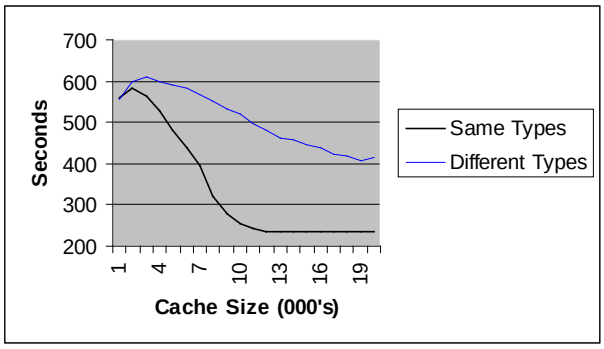


Figure 4.5 – Cache benefits (Test Data 1 & 2)

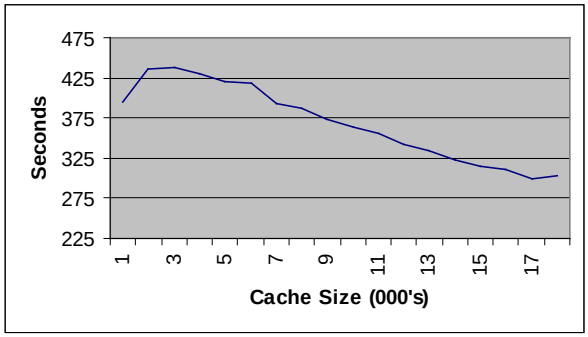


Figure 4.6 – Cache Benefits (Test Data 3)

The cache size was then varied, in increments of 1000, between zero and the upper bound. It can be seen (figure 4.5 and 4.6) that the cache size has a significant effect on the running time of the algorithm. In addition, when using polygons of the same type (figure 4.5, which shows the first two sets of test data) the program not only runs faster but also requires a smaller cache (due to the reduced size of the search space). The third test problem (figure 4.6) shows similar, confirmatory results. It is interesting to note that with a cache size of zero, the algorithm is slightly slower than when the cache is set to only hold a small number of elements. We attribute this to the fact that there is an overhead in maintaining the cache, so

that when the cache size is set to a small value the overheads in maintaining the cache are greater, in terms of time, than the benefits gained by using a cache. However, this is not a major issue as it is always better to have the cache set to as large a value as possible and a small value would never be used, only in this case to show the effect of varying its size.

4.3 Re-evaluation Parameter

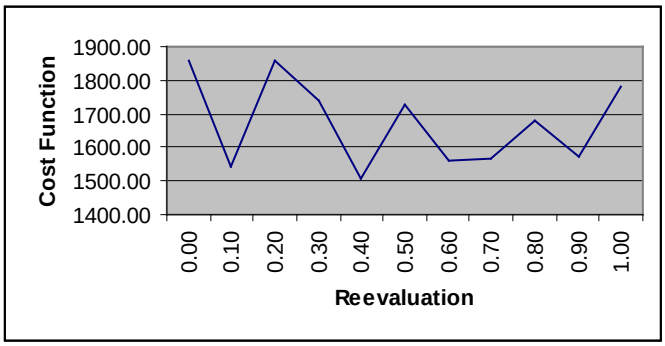


Figure 4.7 – Effect of Re-evaluation

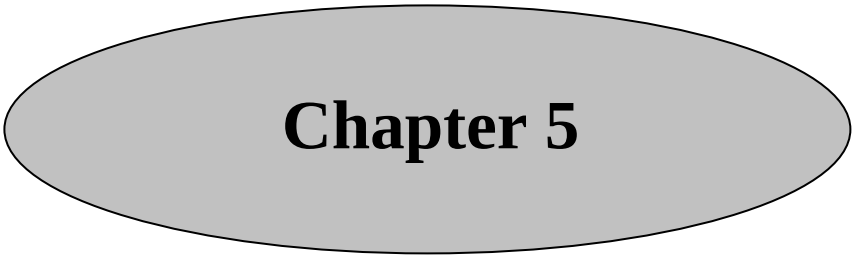
The effect of increasing the re-evaluation parameter is to increase the run time of the algorithm because cached data is not used as often. The worst case comes when the re-evaluation parameter is set to 1.0 as this is the same as setting the cache size to zero and means that every solution has to be evaluated. The three sets of test data were tested using values between 0.0 and 1.0 for the re-evaluation parameter, incrementing by 0.1 on each test. We might expect to see, with higher values for the re-evaluation parameter, better quality solutions appearing. In fact, this was not the case as can be seen from figure 4.7, which shows the results from the third set of data but is representative of all three tests. It is difficult to draw any

concrete conclusions from this graph but it can be seen that when the re-evaluation parameter is set to zero the result is a high cost function. From this observation it would appear sensible to not set the parameter to zero but to a low value.

4.5 Conclusions

The evaluation function is a bottleneck in some systems, which is the case in this research due to the computational geometry aspects of the algorithm. Three ways have been presented which increase the speed of the algorithm. Although these methods are intuitive, techniques such as these, to the authors knowledge, have never been reported in the literature, especially with regards to the nesting problem in conjunction with the no fit polygon. The first improvement stores previous evaluations in a cache so that the evaluation function can be bypassed if the same solution is seen again. By varying the cache size it has been shown that the speed of the algorithm is significantly faster. The concept of polygon types has also been introduced. Whilst most researchers will use this method of representing their data we have demonstrated that this approach does lead to an improvement in run time, especially when used in conjunction with the cache. Finally, a re-evaluation parameter was used so that we can force a solution to be re-evaluated even if it is stored in the cache. This is required as it is possible that the cache will hold an inferior solution for a given permutation of polygons.

Without the techniques outlined above we do not believe that an evolutionary approach to the nesting problem, using our evaluation method, would be feasible. This work provides a sound foundation for the remainder of the research in this thesis.



"Descent with modification must be the answer not catastrophic destruction followed by fresh creation." Descent of Man

5. Comparing Meta-Heuristics and Evolutionary Algorithms when Applied to the Convex Nesting Problem

5.1 Introduction

In this chapter a number of meta-heuristic and evolutionary algorithms are used to search for good quality solutions to the nesting problem. These search methods rely on a new method for packing polygons. The method utilises the no fit polygon (see chapter 4, section 4.2).

This chapter also shows how the parameters for the various search strategies were selected.

The reported results are encouraging and provide a set of comparative test results for later work in this thesis.

The main aim of this chapter is to show that the proposed approach produces good quality results for two problems. More problems could have been used but, due to the convex nature of the problems, there is limited scope to develop this work.

However, if it can be shown that the proposed method produces good quality solutions then the same techniques can be applied to non-convex problems with the hope of achieving even better results.

To produce a solution to the two dimensional nesting problem it is necessary to place a number of shapes onto a larger shape. In doing so the shapes must not overlap one another and must stay within the confines of the larger shape. The usual objective is to minimise the waste of the larger shape. Only two dimensions, height and width, are considered and the larger piece is sometimes considered to be of infinite height so that only the width of the placements needs to be checked. This is a realistic assumption for the real world as the larger shapes are sometimes rolls of material which can be considered as being of infinite length for the purposes of the placement procedure.

In this chapter a number of assumptions are made. The height of the bin (the larger piece) is considered infinite, although it remains the aim of the evaluation function to minimise this height. Only one bin is used (that is, there is no concept of filling a bin and having to start another). Only guillotine cuts are allowed (that is, a cut must be made from one edge to the other).

5.2 Evaluation Method

Falkenauer (Falkenauer, 1998), although not specifically addressing the nesting problem, considered grouping problems, using genetic algorithms (GA's). One of the problems addressed is bin packing. The nesting problem can be considered to be a bin packing problem where the cost associated with each piece is equal to its area.

The cost function used by Falkenauer is

Maximize

$$f_{BPP} = \frac{\sum_{i=1...N}(F_i / C)^k}{N} \qquad (5.1)$$

Where N is the number of bins used

F_i is the sum of the sizes of objects in bin i

C is the bin capacity

And k is a constant, $k > 1$

This cost function measures the average 'bin efficiency' to the k^{th} power. Falkenauer achieved good results with k=2. The research presented in this chapter has used both this cost function (suitably amended and described below) as well as a cost function that simply minimises the bin height.

Our cost function is based on that used in (Falkenauer, 1998). It can be stated as follows

Minimise

$$(\sum_{1}^{n}((1-(UsedRowArea/TotalRowArea))^2)*k)/n \qquad (5.2)$$

Where *UsedRowArea* is the total area of the polygons placed in that row

TotalRowArea is the total area of the bin occupied by that row

k is a factor simply to scale the result (we used 100 but 1 could be used)

n is the number of rows in the bin

In essence, we are trying to minimise the area used by each row. This evaluation function is preferable to the more obvious method of simply measuring the bin height as, using the bin height, many solutions will map to the same evaluation value. This makes it much more difficult to effectively explore the search space.

The method used to evaluate a solution, given a permutation of polygons, is the method described in the previous chapter (section 4.3).

5.3 Test Data

Two test problems were used in this part of the research. The packing shown in figure 5.1 is from (Christofides, 1977), see appendix B and C for the problem definition. The reason that this data was chosen is because it consists of convex polygons and the optimum is known. The only change made is to multiply the measurements in the original paper by a factor of two. This assists us when displaying results.

Our algorithms will not be able to find the optimum. The first two rows (A, B, C and D, E, F) can be constructed without problems. However, to find the optimum for the third row, the polygons would need to be presented in the order of G, H, I, J, K, L and M. The optimal solution could be built until the last polygon (M) came to be placed. At this time, due to the convex nature of the large polygon that had been built, the final polygon will not be placed in the position shown. In fact, the final polygon would be placed on a new row. Under these circumstances the total bin height would be 188 (as opposed to the optimal height of 140).

Therefore, using the permutation ABCDEFGHIJKLM produces a bin height of 188. We will never achieve a bin height of 140, due to the convex nature of the problem. However, bin heights below 180 indicate good quality, convex solutions which were maybe not immediately obvious.

The second problem (figure 5.2) is taken from the real world. The objective is to cut polycarbonate shapes from larger stock sheets. In reality, the company receiving the orders has to cut many shapes from many stocks sheets whilst minimising the waste. The solution shown (figure 5.2) represents one stock sheet on a given day. However, it is a worthwhile exercise to see how our algorithm packs these shapes. Similar to the first problem, it is constrained by the convex properties of the algorithm. For example, the bottom four shapes cannot be packed in the way they are shown as once three of the shapes have been packed, the fourth shape cannot be placed in the position shown. The same is true for the four shapes above (excluding the rectangle).

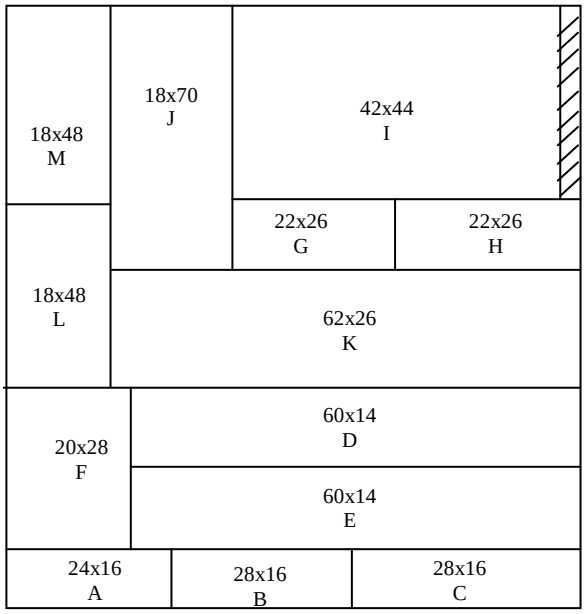


Figure 5.1 – Test Data 1

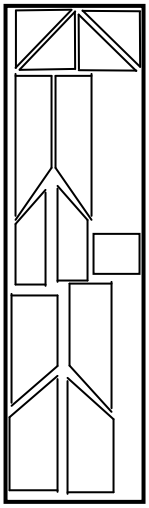


Figure 5.2 – Test Data 2

The height of the stock sheet is 23940 units. Its width is 6240 units. Due to the convex properties of the algorithm it is not possible to achieve this solution but our main concern is to see if using evolutionary and meta-heuristics algorithms coupled with the no fit polygon does produce good quality solutions.

All the results are averaged over ten runs. The number of evaluations performed were equivalent in all cases (SA, TS and GA). This leads to runs of approximately the same time (about 300 seconds on a Cyrix 166 processor) so that the results are compared fairly.

5.4 Parameter Selection for Search Methods

Selecting the most suitable neighbourhood operators and parameters for a meta-heuristic or evolutionary algorithm is a difficult task. There is little theoretical work published in this area and it is still more of an art than a science.

5.4.1 Simulated Annealing

To find suitable values for the simulated annealing cooling schedule the ideas outlined in section 2.3.2 of this thesis were used.

Several neighbourhood functions have been implemented to allow us to explore the search space

Collect : Randomly selects a polygon and then scans through the remaining polygons and moves all polygons of the same type so that they are next to each

other. The idea behind this function is that polygons of the same type will fit well together when packed.

NextDoor : Picks a polygon at random and swaps it with its next door neighbour. The motivation behind this function is to make small changes in the neighborhood in the hope that the search space will be systematically explored, leading to a good quality solution.

Random : Selects two polygons at random and swaps them.

For both problems a number of experiments were conducted, in the first instance to show that simulated annealing performs better than hill climbing. Therefore, the following experiments were run.

- A hill climbing algorithm.
- A simulated annealing algorithm using a linear cooling schedule of {*i*, 0, *NewTemperature = OldTemperature – n*, *iter*}
- A simulated annealing algorithm using a geometric cooling schedule of {*i*, 0, *NewTemperature = OldTemperature* * 0.9, *iter*}
- The neighbourhood functions described above were tested.
- The cost function described in 5.2 was used, along with a cost function that simply attempts to minimise the bin height. Where the cost function from 5.2 is used, the bin height is still shown in the results as it is an important measure of the quality of the solution.

Initially, the Random and NextDoor neighbourhood functions were tested against one another. The first set of test data and the modified Falkenauer cost function were used and the following results were obtained (table 5.1).

Function	Evaluation	Bin Height
Random	655.21	188.10
NextDoor	772.76	193.00

Table 5.1 – Random vs NextDoor Neighbourhood

Based on this result the Random neighbourhood function was used in preference to the NextDoor function.

A further test, comparing hill climbing against simulated annealing, was also run. A very slow linear cooling schedule, {990, 0, temp = temp –15, 250}, and a hill climbing algorithm were run for the same number of iterations (16750) (and thus the same amount of time). This initial test was run on the second set of test data using the modified cost function of Falkenauer. The results were as follows.

Method	Evaluation	Bin Height
Simulated Annealing	1385.06	30312.40
Hill Climbing	1489.74	31605.60

Table 5.2 – Hill Climbing vs Simulated Annealing

This gave an initial indication that simulated annealing out performs hill climbing, although further results are presented below.

Following these tests the separate problems were considered. Table 5.3 shows the results when the first set of test data was used. As well as comparing hill climbing

against simulated annealing, comparisons are also being made against a linear and geometric cooling schedule, the two evaluation functions and two of the neighbourhood functions.

Search Algorithm	Neighbour Function	Cost Function	Cooling Schedule	Eval	Bin Height	Time (Secs)
SA	Random	Modified Falkenauer	{40,0,t=t–1,50}	643.95	188.40	297
SA	Random	Bin Height	{40,0,t=t–1,50}		191.60	298
SA	Random	Modified Falkenauer	{40,0,t=t* 0.9,50}	705.70	189.20	643
HC	Random	Modified Falkenauer		752.64	196.40	300
SA	Collect	Modified Falkenauer	{40,0,t=t–1,50}	397.01	170.80	220
SA	Collect	Bin Height	{40,0,t=t–1,50}		169.07	221
SA	Collect	Modified Falkenauer	{40,0,t=t* 0.9,50}	390.05	168.13	431
HC	Collect	Modified Falkenauer		457.93	173.07	223

Table 5.3 – Hill Climbing vs Simulated Annealing over a variety of parameters

From these results the following observations can be made

- In a similar manner to the earlier tests, simulated annealing performs better than hill climbing.
- Using a geometric cooling schedule does lead to slightly better solutions but this is at the expense of run times that are about twice as long.
- The modified falkenauer cost function produces better solutions than using the bin height cost function although the difference is not as much as had been expected.

- The most significant result is that using the collect neighbourhood function produces better quality solutions than the random neighbourhood function, with reduced run times.

In conclusion we can say that the best quality results use the collect neighbourhood function and use simulated annealing with a cost function based on Falkenaur's. In addition, a linear cooling schedule can be used in order to produce faster results than a geometric cooling schedule.

Figure 5.3 shows three of the best solutions found. The first figure shows a bin height of 158. The other two figures have a bin height of 162.

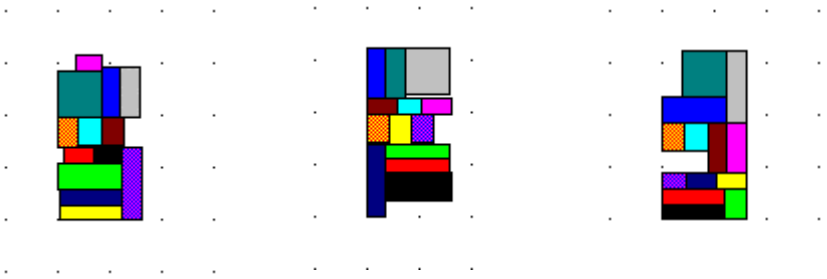


Figure 5.3 – Best Solutions Found (SA)

For the second set of test data it was not possible to use the collect neighbourhood function as all the polygons have different dimensions (some may look the same but they are, in fact, all different). Therefore, the random operator was used. The results produced are shown in table 5.4.

The best quality solutions were found using simulated annealing with a geometric cooling schedule but at the expense of a run time almost four times that of the

other methods. All simulated annealing algorithms again performed better than hill climbing but this time using the bin height cost function performed better than the modified falkenauer cost function.

Search Algorithm	Neighbour Function	Cost Function	Cooling Schedule	Eval	Bin Height	Time (Secs)
SA	Random	Modified Falkenauer	{1000,0,t=t–25,50}	1661.77	33052.20	314
SA	Random	Bin Height	{1000,0,t=t–25,50}		31799.20	320
SA	Random	Modified Falkenauer	{1000,0,t=t*0.9,50}	1520.58	30999.60	1101
HC	Random	Modified Falkenauer		1683.53	33532.80	330

Table 5.4 – Results using Test Data 2

5.4.2 Tabu Search

In Tabu Search (TS) the neighbourhood function has been implemented as a swap of two random polygons to give a new permutation. Experiments were conducted with various neighbourhood sizes and a value of twenty gave good results.

The data stored in the tabu list can have an effect on the search. We tested two different methods. Initially just the polygon identifier was stored, which effectively stopped that polygon being moved again until it was removed from the tabu list. This did not perform very well as it restricts the search too much. This is borne out by the fact that the tabu list can, at maximum, store *number_of_polygons* entries, which constrains the search too much.

Another option is to hold the complete solution in the tabu list. It was found that this increased the run time of the algorithm (due to checking if a move is tabu) but we could achieve equally good results by making moves tabu based on the first *n*

polygons in a solution. We achieved the results presented below with *n*=3. That is, if the first three polygons are the same then a move to that state is tabu.

The results achieved on both sets of test data are shown in table 5.5.

	Tabu List Size	Evaluation	Bin Height
Test Data 1			
	0	368.03	171.40
	25	342.69	169.00
	50	323.77	165.80
	75	386.68	172.20
	100	386.91	171.00
Test Data 2			
	0	1089.89	28228.50
	25	1101.31	28362.60
	50	1006.75	27566.40
	75	1066.30	28142.40
	100	1200.09	28856.40

Table 5.5 – Tabu Search Test Results

5.4.3 Genetic Algorithms

For this research the PMX crossover operator (Goldberg, 1989) has been employed. Other operators were tested but it was found that PMX gave the best results. This is not surprising as PMX is designed for ordering problems. As this problem relies on the ordering of the polygons it falls into this category.

The mutation operator is applied to a single chromosome. It is applied to each child with some (low) probability and aims to stop the population converging to a single solution by adding diversity to the population. We have tested two types of mutation which we call heavy and light mutation. Heavy mutation is applied to a child solution. A gene is picked at random and is swapped with another randomly selected gene, from the same chromosome. This is done as many times as there are

genes in the chromosome. Light mutation carries out the same random swap but only does a single exchange.

As an initial test for the GA the crossover probability was set to 0.0 and the mutation probability to 1.00. This effectively turns the GA into a random search and the result can be used to check that the GA was actually directing the search.

Two normalization methods were tested. Evaluation is Fitness makes the fitness of each chromosome the same as the value returned from the evaluation function. Linear Normalization adjusts the fitness of each individual so that the fitness is related to its evaluation but the fitness values are linear throughout the population. Using linear evaluation with roulette wheel selection ensures that the fitter members of the population do not dominate, leading to premature convergence. Better results were achieved with linear normalization.

The two types of mutation described above were tested. Various values for the crossover and mutation probability were tried. It was found that values of 0.8 and 0.1 gave the best results. A dynamic mutation probability was also tested, varying the mutation value between 0.0 and 0.5 during the course of the run. This was found to produce similar results, but no better, than a static value of 0.1.

The results achieved for the genetic algorithm are shown in table 5.6.

Test Data 1

Normalization Method	Crossover Probability	Mutation Probability	Mutation Type	Eval	Bin Height
Eval is Fitness	0.8	0.1	Heavy	684.41	192.60
Eval is Fitness	0.8	0.1	Light	700.10	190.40
Linear	0.8	0.1	Heavy	610.43	185.20
Linear	0.8	0.1	Light	516.41	179.00

Test Data 2

N/A	0.0	1.0	Heavy	1655.78	32972.40
Eval is Fitness	0.8	0.1	Heavy	1727.51	32695.50
Eval is Fitness	0.8	0.1	Light	1607.38	32799.30
Linear	0.8	0.1	Heavy	1528.06	31353.00
Linear	0.8	0.1	Light	1377.99	30478.50

Table 5.6 – Genetic Algorithm Test Results

5.5 Testing, Results and Comparisons

5.5.2 Results

From the above results, we can make the following observations.

Of the three methods (SA, TS, GA), GA performed the worst, although a linear normalization function and light mutation produces the best results on both sets of test data.

Tabu search produced better results than simulated annealing for both sets of test data.

The size of the tabu list does not need to be large. Once the list gets too large the quality of the solution starts to deteriorate. We believe this is because the search becomes too constrained. Intuitively, the size of the tabu list should be less than the number of iterations otherwise the search can never visit previously visited states. These results back up that view.

Two of the better solutions we found when using tabu search against test data 1 are presented in figure 5.4. The nesting on the left has a bin height of 158. The right

hand figure has a bin height of 162. Figure 5.5 is a sample solution, using tabu search on test data 2. This nesting has a bin height of 26430

Figure 5.4 – Sample Solutions (Test Data 1)

Figure 5.5 – Sample Solution (Test Data 2)

A summary of all the results are shown in table 5.7.

	Test Data 1	**Test Data 2**
Tabu Search	323.77	1006.75
Simulated Annealing	397.01	1661.77
Hill Climbing	457.93	1683.53
Genetic Algorithm	516.41	1528.06

Table 5.7 – Summary of Results

5.6 Conclusions

In this chapter, several search algorithms have been tested in order to produce good quality solutions to a nesting problem. The problems use convex shapes and, in addition, any polygons that are created during the algorithm are also convex as it is transformed using a convex hull algorithm.

At this stage of the work we are still working with convex shapes as we are still trying to ascertain if the search methods being used are effective. It was thought beneficial to use convex polygons in the first instance as the algorithm for the no fit polygon is well known for these polygons and it is relatively efficient. In chapters 7 and 8 of this thesis non-convex problems are considered.

	GA		**TS**		**SA**	
	Best	Avg	Best	Avg	Best	Avg
50 Rectangles	22053	22149	20024	20099	20047	20089
100 Rectangles	79586	80356	75269	75949	73845	74003

Table 5.8 – Results from Simplified Problem (copy of table 3.1)

In chapter 3 (the results of which are re-produced in table 5.8), it was noted that tabu search and simulated annealing performed similarly with respect to the quality of solutions that were produced. It was noticeable that genetic algorithms did not perform well. The results in table 5.7 confirm these earlier findings in that genetic algorithms, again, failed to perform as well as tabu search and simulated annealing. In fact, genetic algorithms failed to do better than hill climbing on one of the problems. From this, I do not think that we can conclude that genetic

algorithms are not an effective search technique but, I feel, it highlights the problems in trying to find effective parameters for a given search technique. In chapter 3 (see section 3.4.2) we carried out 24 genetic algorithm tests, trying to find a good combination of parameters and, even then, only a small subset of the total number of permutation of parameters were tested. I believe this highlights one of the greatest problems facing designers of meta-heuristic and evolutionary algorithms today; that is finding a suitable parameter set for a given problem (or even a particular instance of a problem).

This is further demonstrated in that for these problems tabu search is able to out perform simulated annealing (see table 5.7). The problem with simulated annealing, like genetic algortithms, is that a suitable cooling schedule has to be selected. As mentioned previously this is still an art more than a science.

Tabu search benefits from the fact that there are very few parameters and, this works suggests that the parameter values are not too critical to the success of the algorithm in finding good quality solutions.

In the next chapter a new search technique is explored (ant algorithms). It will be interesting to see how this algorithm compares to those studied so far as it has a number of parameters that have to be set. In fact, like chapter 3 and genetic algorithms, we will be forced to carry out a number of experiments to find suitable values for those parameters.

Chapter 6

"Ants pass on food to one another by 'kissing'. During the kiss, they also pass on messages which tell them what job to do" Small World: Ants

6. Applying Ant and Memetic Algorithms with the No Fit Polygon to the Nesting Problem

6.1 Introduction

In this chapter ant algorithms are investigated and the results compared against the results from chapter 5. In addition, memetic algorithms are also compared against the results from the previous chapter and also those acquired using ant algorithms. The evaluation method used for the work in this chapter is exactly the same as that from the previous chapter, as are the two sets of test data.

Ant algorithms have already been introduced in chapter 2 (section 2.3). Therefore, this chapter starts by considering how ant algorithms can be adapted so that they can be used to solve the nesting problem using the approach adopted for other search algorithms, and described in the previous chapter.

6.2 Ant Algorithms and the Nesting Problem

Using the Travelling Salesman Problem (TSP) ant system as a model (see chapter 2, section 2.2.3), an ant system has been developed for the nesting problem using

the no fit polygon and the evaluation method described in the previous chapter (section 5.2). Each polygon can be viewed as a city in the TSP and these are fully connected so that there is an edge between each polygon and every other one. An ant is placed at each city (polygon) and using the trail and visibility values (formulae 2.6, from chapter 2) the ant decides which polygon should be visited (placed) next. Once an ant has placed all the polygons the edge trail values are updated. The edge trail values are calculated using the value returned from evaluating the nesting. This is equivalent to using the tour length in the TSP (L_k in formula 2.5, from chapter 2).

Using the method described above we can use very similar formulae to those described in chapter 2 (formulae 2.3 and 2.4), with some minor amendments. Visibility is now defined as how the polygon, just placed, fits with the piece about to be placed. For example, two rectangles of the same dimensions would fit together with no waste so the visibility would be high. Two irregular shapes, when placed together, may result in high wastage. This would result in a low visibility value. In order to calculate the visibility, n_{ij} (where i is the polygon just placed and and j is the polygon just about to be placed), the combined area of the two pieces is divided by the best placement of the two shapes (using the NFP). That is

$$n_{ij} = \text{TotalArea}_{ij} / \text{BestPlacement}_{ij} \qquad (5.1)$$

(Note, in order to maintain consistency in the literature we are using similar notation to Dorigo and thus we use n_{ij}, rather than, say $visib_{ij}$).

This returns a value between 0 and 1. In order to improve the speed of the algorithm all the visibility values are calculated at the start of the algorithm and held in a cache. The transition probability is defined as the probability of a polygon being placed next taking into account the polygons placed so far and the visibility of the next polygon. The same formula as 2.6 from chapter 2, can be used to calculate the transition probability.

6.3 Memetic Algorithms

A memetic algorithm can be considered to be a population based approach that incorporates a local search element. Each time a member of the population changes (after crossover or mutation in the case of a genetic algorithm and after a tour in the case of the ant algorithm) a local search is applied. We have used both genetic algorithms and ant algorithms as the main search mechanism and have tried other search methods (such as tabu search, hill climbing etc.) as a local search operator. It was found that a simple hill climbing search produces the best results. Other search operators produced inferior results and took longer to run. With tabu search, for example, the overheads in maintaining the tabu list, combined with the additional problems introduced in having to select suitable parameters (list size, neighbourhood size etc.), meant that the results were inferior to using hill climbing.

6.4 Testing and Results

6.4.1 Ant Algorithm Parameters

Initially we attempted to find suitable values for the parameters that control the ant algorithm. In order to find suitable values we carried out several hundred runs simply setting the parameter values at random. We used these results, along with the best parameters found by (Dorigo, 1996), to conduct more selective testing. Dorigo reported that the value of Q (the constant in formula 2.3) had little effect on the algorithm. We experimented with various values, {1, 10, 100, 1000}, and reached a similar conclusion. Therefore in the remainder of our tests Q = 100. In order to find a good value for the evaporation parameter, *p*, it was set to the values {0.1, 0.5, 0.9}, using a trail importance, $\alpha = 1$ and a visibility importance, β, of {0, 1, 2, …, 30}. These tests were carried out on test data 1. The results from these tests are shown in figure 6.1.

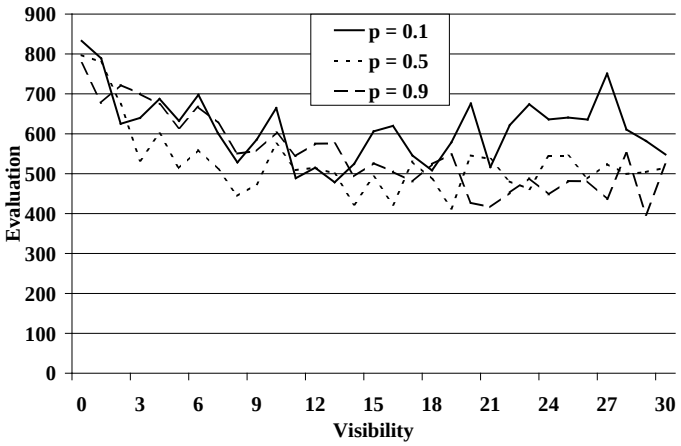


Figure 6.1 – Test Data 1, $\alpha = 1$, p = {0.1, 0.5, 0.9}, β = {0, 1, …, 30}

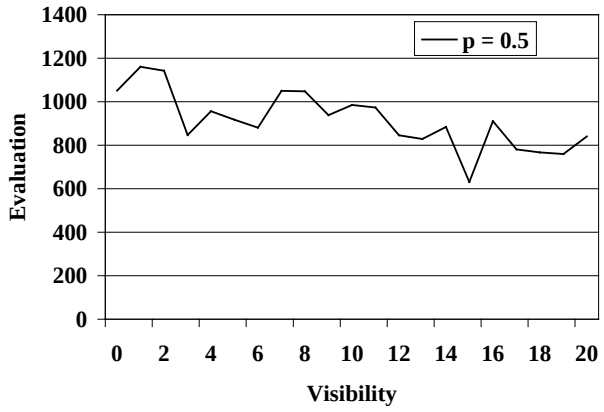


Figure 6.2 – Test Data 1, $\alpha = 5$, $p = 0.5$, $\beta = \{0, 1, \ldots, 20\}$

All tests in figure 6.1 show the highest evaluation when $\beta = 0$. This is expected as when $\beta = 0$ the search is effectively transformed into randomised greedy search with multiple starting points. All three runs also show, in the early stages, a downward trend as the visibility parameter increases. With $p = 0.1$ the evaluation values are generally higher than when p is 0.5 or 0.9. The algorithm performs better when $p = 0.5$ rather than when $p = 0.9$, at least in the early stages (until $\beta = 19$). In the latter stages, when the visibility is high, the graph has either flattened or is showing an upward trend for all values of p. Again, this would be expected as having too high a visibility starts returning the algorithm to a greedy search. This is due to the effect of the intensity trail becoming diminished. Taking $p = 0.5$ as a good value agrees with the results in (Dorigo, 1996) in which it was reported that this was the best value found for p. In (Dorigo, 1996) the best value for α was found to be 1 (which is the value used above). In order to see if our algorithm

agrees with this a test was carried that set p = 0.5, α = 5 and β = {0, 1, …, 20}. Figure 6.2 shows that this set of parameters produces worse results than when α = 1. None of the tests produce an evaluation below 600, which was consistently done when α = 1 (figure 6.1)

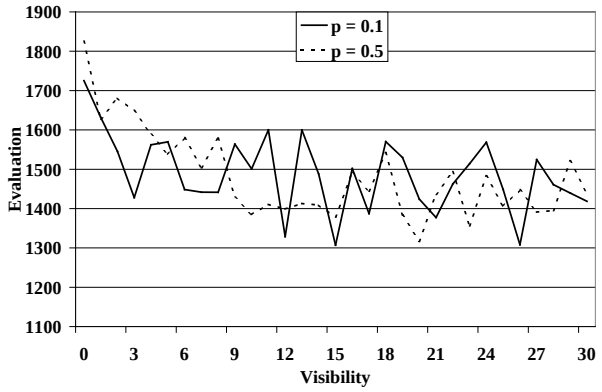


Figure 6.3 – Test Data 2, α = 1, p = {0.1, 0.5}, β = {0, 1, …, 30}

Having established a good parameter set we used test data 2 to confirm these values. Figure 6.3 shows two runs that compare the effect of p (evaporation) on test data 2. In fact the two runs mimic each other closely but it is interesting to note that the lowest evaluation for p = 0.5 is when visibility is around 20. This matches the result from the first set of test data. This test appears to confirm that p = 0.5 is a good choice and we used this in the remaining tests.

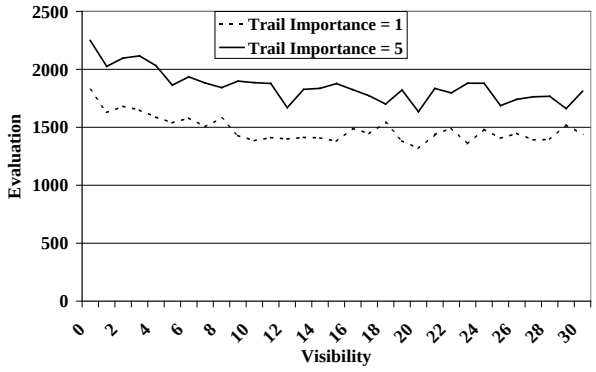


Figure 6.4 – Test Data 2, $\alpha = \{1, 5\}$, $p = 0.5$, $\beta = \{0, 1, \ldots, 30\}$

Figure 6.4 shows the effect of trail importance, $\alpha = \{1, 5\}$ using test data 2. It shows that a higher value of α leads to inferior solutions. Again, this confirms the results from the first set of test data. In summary, the best results were achieved on both sets of test data when $\alpha = 1$, $p = 0.5$, $\beta = 20$.

6.4.2 Results

In the previous chapter, results were presented for the various search algorithms when applied to the above problems. Table 6.1 shows modified results using the same test data and the same search algorithms. These modified tests were carried out for the following reasons.

- Some of the results below have had the parameters amended from the previous work in order to improve the results even further.
- By necessity, the previous work was carried out over several months and on different computers. Although we tried to take this into account during this

time it was felt beneficial to run all the algorithms on the same computer in order to guarantee that the results could be fairly compared, when presented together.

- Running the algorithms on the same computer also gave the opportunity to ensure that the algorithms all ran for a similar amount of time. This is not always easy due to the stochastic element of the algorithms, but this was generally achieved.

It is worth noting that the algorithms performed similarly, relative to one another, for both sets of test data (only simulated annealing and hill climbing differ). It is also interesting, and encouraging, that the ant algorithm out performs the other population based approach (genetic algorithm).

	Test Data 1	Test Data 2
Tabu Search	326.97	1056.96
Ant Algorithm	371.67	1188.60
Simulated Annealing	513.33	1269.63
Hill Climbing	506.47	1364.77
Genetic Algorithm	615.57	1412.82

Table 6.1 – Summary of Results

The parameters used for each algorithm are shown in table 6.2.

Algorithm	Test Data 1	Test Data 2
Tabu Search	Iterations = 400, List Size = 100, Neighbourhood Size = 20	Same as test data 1
Ant Algorithm	Iterations = 615, Trail = 1, Visibility = 20, Evaporation = 0.5, Q = 100	Same as test data 1
Simulated Annealing	Start = 200, Stop – 0, Decrement = 0.5, Iterations = 66	Start = 1000, Stop – 0, Decrement = 10, Iterations = 80
Hill Climbing	Iterations = 8000, Neighbourhood Size = 1	Iterations = 400, Neighbourhood Size = 20
Genetic Algorithm	Population Size = 30, Iterations = 640	Same as test data 1

Table 6.2 – Algorithm Parameters

Table 6.3 and 6.4 shows the results from the memetic algorithm experiments. A number of runs were carried out, in order to find the best combination of parameters.

Main Search Method	Main Search Parameters	Hill Climbing Parameters	Eval
AA	Iterations = 70	Iterations = 3, Neighbourhood Size = 3	283.07
GA	Pop Size = 50, Generations = 40	Iterations = 1, Neighbourhood Size = 5	293.65
AA	Iterations = 25	Iterations = 5, Neighbourhood Size = 5	309.81
GA	Pop Size = 20, Generations = 10	Iterations = 10, Neighbourhood Size = 5	358.35
AA	Iterations = 13	Iterations = 17, Neighbourhood Size = 3	389.24
GA	Pop Size = 50, Generations = 40	Iterations = 5, Neighbourhood Size = 1	397.27
GA	Pop Size = 20, Generations = 20	Iterations = 5, Neighbourhood Size = 5	407.18
AA	Iterations = 13	Iterations = 10, Neighbourhood Size = 5	407.32
GA	Pop Size = 30, Generations = 20	Iterations = 15, Neighbourhood Size = 1	449.73
GA	Pop Size = 20, Generations = 10	Iterations = 50, Neighbourhood Size = 1	482.31
GA	Pop Size = 20, Generations = 20	Iterations = 25, Neighbourhood Size = 1	487.09

Table 6.3 - Memetic Algorithm for Test Data 1

Main Search Method	Main Search Parameters	Hill Climbing Parameters	Eval
GA	Pop Size = 20, Generations = 20	Iterations = 5, Neighbourhood Size = 5	890.51
GA	Pop Size = 20, Generations = 10	Iterations = 10, Neighbourhood Size = 5	981.95
GA	Pop Size = 50, Generations = 40	Iterations = 1, Neighbourhood Size = 5	994.46
AA	Iterations = 13	Iterations = 10, Neighbourhood Size = 5	1014.83
AA	Iterations = 70	Iterations = 3, Neighbourhood Size = 3	1116.65
AA	Iterations = 13	Iterations = 17, Neighbourhood Size = 3	1157.31
AA	Iterations = 25	Iterations = 5, Neighbourhood Size = 5	1158.31
GA	Pop Size = 50, Generations = 40	Iterations = 5, Neighbourhood Size = 1	1178.24
GA	Pop Size = 20, Generations = 20	Iterations = 25, Neighbourhood Size = 1	1196.74
GA	Pop Size = 30, Generations = 20	Iterations = 15, Neighbourhood Size = 1	1221.18
GA	Pop Size = 20, Generations = 10	Iterations = 50, Neighbourhood Size = 1	1369.72

Table 6.4 - Memetic Algorithm for Test Data 2

Although the parameters that lead to the best results are not identical for both sets of test data, it is the case that using a neighbourhood size of one, within the local search, does lead to inferior results. This suggests that the local search hill climber should be some form of steepest ascent.

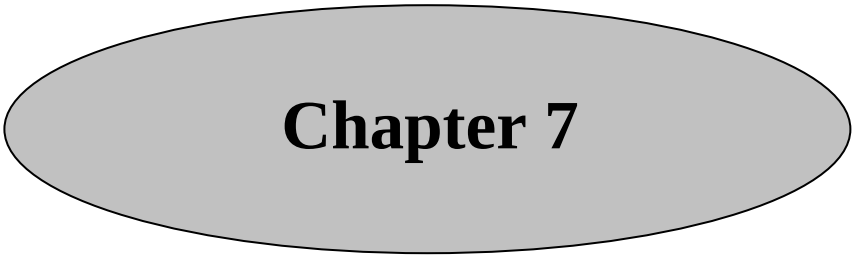
6.5 Conclusions and Discussion

This is the first time that ant algorithms have been applied to the nesting problem and the results are encouraging. Ant algorithms out perform genetic algorithms and provide a viable alternative to simulated annealing, although more work is required for it to compete with tabu search.

By comparing the results from table 6.3 and table 6.4 with table 6.1 it can be seen that the memetic algorithm, for both sets of test data, produced better results than had been achieved using the other algorithms in isolation. This might be surprising as we are combining two search methods (genetic algorithms and hill climbing) that, on their own, do not produce good results. Yet, when they are combined, the results are superior to any single search method. Looking more closely at the results in tables 6.3 and 6.4 shows that, although using a genetic algorithm as the main search method dominates the top of table 6.4 it is ant algorithms that appears at the top of table 6.3. This suggests that, to a certain extent, it is not important which evolutionary search method is used. It is the combination of a population based approach combined with a local search that is leading to good quality solutions.

This observation is further enhanced by the fact that the local search method (hill climbing) does not perform well on its own but when it is allowed to take a given solution to a local optimum it greatly assists the search process.

In the next chapter an algorithm is developed to allow the no fit polygon for non-convex polygons to be calculated. In chapter 8 we are then able to compare the search algorithms using non-convex polygons and judge if the observations we have made so far still hold true.



"There is no royal road to geometry." Euclid(said to Ptolemy I), quoted in Proclus, Commentary on Euclid

7. Determining the No Fit Polygon for Non-Convex Polygons and Combining Non-Convex Polygons

7.1 Introduction

The No Fit Polygon (NFP) is a method to find all the possible arrangements of one polygon, in relation to another, so that the two polygons touch but do not intersect. Once the NFP has been calculated for the given polygons, a reference point from one of the polygons can be placed on any edge or vertex of the NFP in the knowledge that the polygon will touch but not intersect the other polygon. The NFP can be applied to areas such as robot motion planning (where it is often referred to as the Configuration Space Obstacle (CSO)). It can also be applied to cutting and packing problems as has been shown in the previous chapters for convex polygons.

It is relatively easy to determine the NFP for convex polygons (see 7.2), using the algorithm presented in (Cunninghame-Green, 1992).

However, the Cunninghame-Green algorithm is only applicable to convex polygons and the construction of algorithms that determine no fit polygons for non-convex polygons is still very much an active research area. For example, (Bennell, 2001) has recently presented a revised algorithm based on the concept of Minkowski sums. Although the methods to determine the NFP are well known the implementation is delicate due to the number of degenerate cases that can occur.

In this chapter, an algorithm is developed that enables us to calculate the NFP for non-convex polygons. The work is based on previous work which uses D-Functions to define a set of primitive operations. We develop the use of D-Functions so that vertex intersection can be reported correctly and we also develop the no fit polygon algorithm so that it caters for degenerate cases. The algorithm has been tested on a variety of cases and, so far, it has been found to be robust.

Finally, we show, using D-Functions again, how two polygons can be combined to form a single polygon.

7.2 The No Fit Polygon – In Outline

The No Fit Polygon (NFP) determines all arrangements that two arbitrary polygons may assume such that the shapes touch but so that they cannot be moved closer together without intersection. The example below shows how to calculate the NFP for two polygons.

Consider the two polygons in figure 7.1. The aim is to find all arrangements such that the two polygons touch but do not intersect. If this can be achieved then we know that we cannot move the polygons closer together in order to obtain a tighter

packing. A description of how to calculate the NFP for convex polygons was shown in section 4.2.

In order to implement an NFP algorithm for ***convex*** polygons it is not necessary to have one polygon orbiting another. The algorithm in (Cunninghame-Green, 1992) works on the assumption that (for convex polygons only) the NFP has its number of edges equal to the number of edges of P_1 and P_2. In addition, the edges of the NFP are simply copies of the edges of P_1 and P_2, suitably ordered. To build the NFP it is a matter of taking the edges of P_1 and P_2, sorting them and building the NFP using the ordered edges. This algorithm is also presented in (Cunninghame-Green, 1989), although here it is presented as a Configuration Space Obstacle (CSO).

Calculating the NFP for non-convex polygons is more difficult. The idea is the same in that one polygon orbits another, tracing a reference point. However, due to the conditions that can arise, the algorithm is more complex and has to deal with a number of degenerate cases. A full description of these issues and their resolution is given later in the chapter. For now, we simply show an example of an NFP for non-convex polygons and describe one of the cases that have to be dealt with, which does not arise in convex polygons. Consider the two polygons shown in figure 7.1a

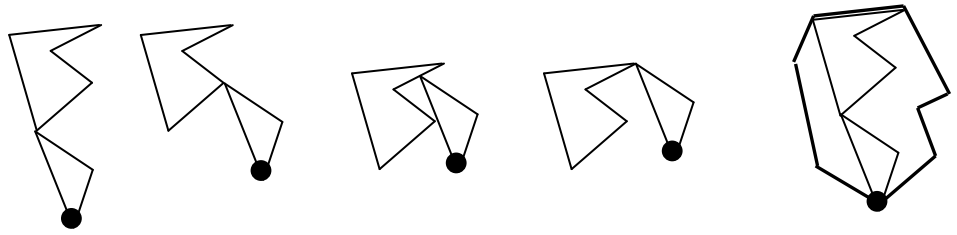


Figure 7.1 – Calculating the NFP for Non-Convex Polygons

The top polygon is to be the stationery polygon and the bottom polygon will orbit it, tracing its reference point (the filled circle) as it does so. Figure 7.1a shows the initial position. Figure 7.1b shows the state after the orbiting polygon has made its first move. Figure 7.1c shows the state after the next move. Notice that the orbiting polygon cannot slide the full distance else it would pierce the stationery polygon. Therefore, it has to stop prematurely and change direction (7.1d). If the orbiting polygon is allowed to continue the NFP polygon is constructed as shown in figure 7.1e. Notice that the NFP edges are no longer simply copies of the edges from the two polygons that formed the NFP. In addition, it is no longer the case that the number of edges are equal to sum of the number of the edges from the original polygons. Additional problems will be described as the algorithm is presented.

7.3 D-Functions

In order to build higher level operations that we can apply to polygons it is beneficial to define low level primitives. For example, it is useful to be able to detect if points are co-linear, if lines intersect or if lines overlap. The literature provides many ways to implement such primitives. For example, in (O'Rourke,

1998) a LeftOn predicate is defined which can be used to decide if a given point is to the left of a directed line. This predicate can be developed to decide if the point is to the right of the line or if the point is co-linear with the line. O'Rourke develops the LeftOn primitive further so that line segment intersection can be detected. An alternative set of primitives is defined in (Mahadevan, 1984). These give a larger range of primitives which can be directly applied to the NFP algorithm and so, are employed in this work.

D-Functions are defined in (Mahadevan, 1984) but we repeat the main information here as they perform a central role in the remainder of the algorithm.

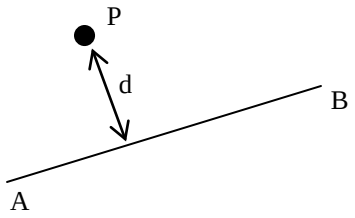


Figure 7.2 – Distance of Point from a Line

Given a vector, AB, and a point, P, the distance, d, from AB to P (see figure 7.2) is calculated as follows

$$d = \frac{(X_A - X_B)(Y_A - Y_P) - (Y_A - Y_B)(X_A - X_P)}{\alpha} \qquad (7.1)$$

where

X_N and Y_N are the *x* and *y* co-ordinates for point *N* and α is the length of AB, and is defined as

$$\alpha = \sqrt{(X_A - X_B)^2 + (Y_A - Y_B)^2} \tag{7.2}$$

Given a vector, AB, and point, P, a D-Function can be defined as $D_{ABP} = \text{sign}[d\alpha]$ or

$$D_{ABP} = \text{sign}[(XA - XB)\,(YA - YP) - (YA - YB)\,(XA - XP)] \tag{7.3}$$

where sign is used to return a value of +1, -1 or 0, depending on whether the result is positive, negative or equal to zero.

Using D-Functions we can determine the position of P in relation to AB as follows (figure 7.3)

If $D_{ABP} = 0$ then P is either on vector AB or on an extended line segment of AB (or BA).

If $D_{ABP} > 0$ then P is to the left of AB.

If $D_{ABP} < 0$ then P is to the right of AB.

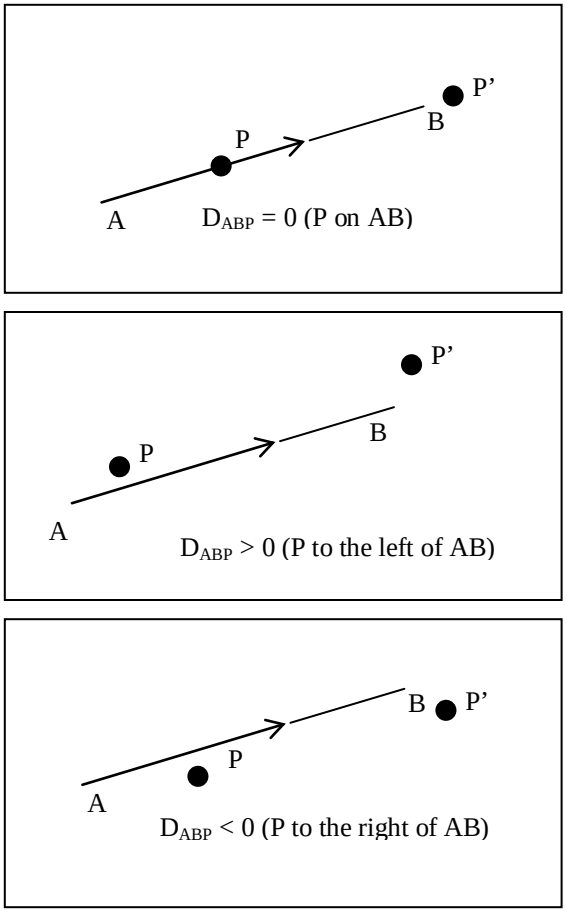
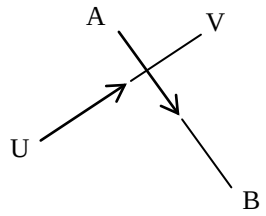


Figure 7.3 – Using D-Functions to determine the relationship between AB and P

D-Functions can be further developed to define the relationship between two lines. Note, that the definitions given here differ slightly from those in (Mahadevan, 1984). These are the minimum conditions that must hold. Mahadevan presents the

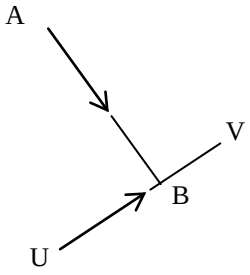
conditions in such a way that they mirror the algorithm given in his thesis. Figures 7.4a through to 7.4i show these conditions graphically and also describes them using D-Functions.



AB intersects UV if

U is to the left (right) of AB ($D_{ABU} \neq 0$)
V is to the right (left) of AB ($D_{ABV} \neq 0$)
A is to the right (left) of UV ($D_{UVA} \neq 0$)
B is to the left (right) of UV ($D_{UVB} \neq 0$)
A and B must be on opposite sides of UV ($D_{UVA} \neq D_{UVB}$)

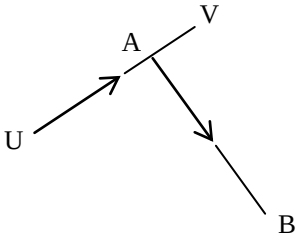
Figure 7.4a – AB intersects UV



B touches UV if

B must be linear with UV ($D_{UVB} = 0$)
U cannot be linear with AB (else U would be touching B) ($D_{ABU} \neq 0$)
V cannot be linear with AB (else V would be touching B) ($D_{ABV} \neq 0$)
U and V must be on opposite sides of AB ($D_{ABV} \neq D_{ABU}$)

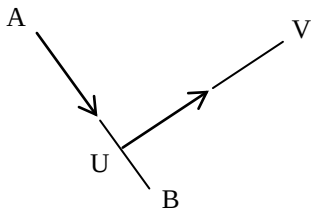
Figure 7.4b – B touches UV



A touches UV if

A must be linear with UV ($D_{UVA} = 0$)
U cannot be linear with AB (else U would be touching A) ($D_{ABU} \neq 0$)
V cannot be linear with AB (else V would be touching A) ($D_{ABV} \neq 0$)
U and V must be on opposite sides of AB ($D_{ABV} \neq D_{ABU}$)

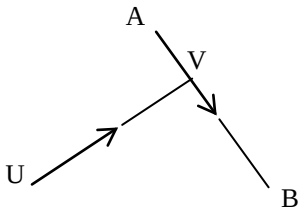
Figure 7.4c – A touches UV



U touches AB if

U must be linear with AB ($D_{ABU} = 0$)
A cannot be linear with UV (else A would be touching U) ($D_{UVA} \neq 0$)
B cannot be linear with UV (else B would be touching U) ($D_{UVB} \neq 0$)
A and B must be on opposite sides of UV ($D_{UVA} \neq D_{UVB}$)

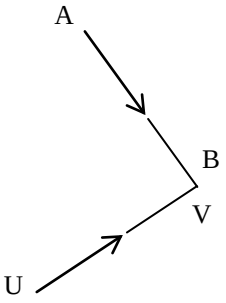
Figure 7.4d – U touches AB



V touches AB if

V must be linear with AB ($D_{ABV} = 0$)
A cannot be linear with UV (else A would be touching V) ($D_{UVA} \neq 0$)
B cannot be linear with UV (else B would be touching V) ($D_{UVB} \neq 0$)
A and B must be on opposite sides of UV ($D_{UVA} \neq D_{UVB}$)

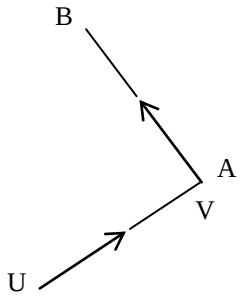
Figure 7.4e – V touches AB



B and V touch if

V must be linear with AB ($D_{ABV} = 0$)
B must be linear with UV ($D_{UVB} = 0$)
U must not be linear with AB ($D_{ABU} \neq 0$)

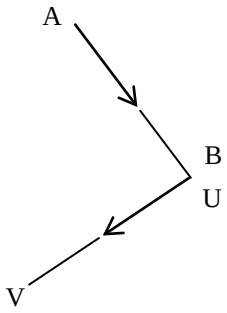
Figure 7.4f – B and V touch



A and V touch if

V must be linear with AB ($D_{ABV} = 0$)
A must be linear with UV ($D_{UVA} = 0$)
U must not be linear with AB ($D_{ABU} \neq 0$)

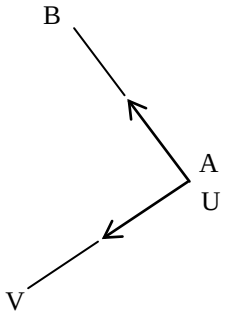
Figure 7.4g – A and V touch



B and U touch if

U must be linear with AB ($D_{ABU} = 0$)
B must be linear with UV ($D_{UVB} = 0$)
V must not be linear with AB ($D_{ABV} \neq 0$)

Figure 7.4h – B and U touch



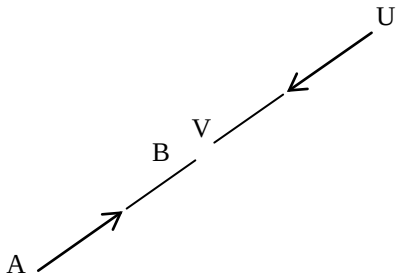
A and U touch if

U must be linear with AB ($D_{ABU} = 0$)
A must be linear with UV ($D_{UVA} = 0$)
V must not be linear with AB ($D_{ABV} \neq 0$)

Figure 7.4i – A and U touch

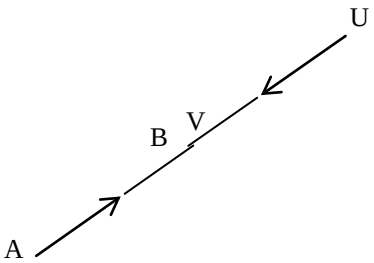
It is also useful to be able to detect if AB and UV overlap. This is a special condition where AB and UV are co-linear ($D_{ABU} = 0$ and $D_{ABV} = 0$). When this

conditions exists it indicates that either AB/UV do not intersect, AB/UV are touching (but still co-linear) or AB, UV overlap. These conditions are shown below (see figures 7.4j to 7.4l).



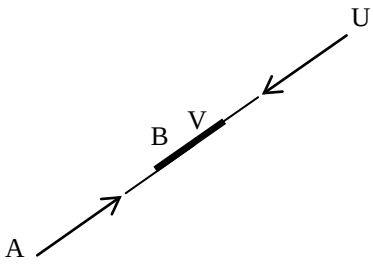
AB and UV do not overlap or intersect but are co-linear

Figure 7.4j – AB and UV do not overlap or intersect but are co-linear



AB and UV touch and are co-linear

Figure 7.4k – AB and UV touch and are co-linear



AB and UV overlap

Figure 7.4l – AB and UV overlap

Any other conditions, other than those described above mean that that AB, UV do not intersect.

The algorithms that implement D-Functions for the conditions shown in figure 7.4 are shown in (Mahadevan, 1984).

7.4 Checking for Intersection

When constructing the no fit polygon it is necessary to determine whether two polygons intersect (for reasons explained below). The usual description as to how this can be achieved is to check every edge on one polygon against every edge on the other polygon and, if any edges intersect, it means that the polygons intersect. This can be seen in figure 7.5a.

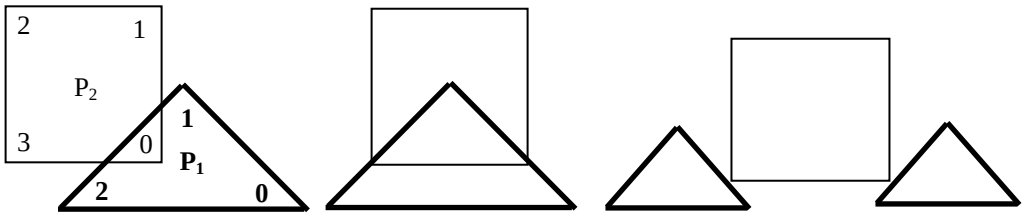


Figure 7.5 – Checking for Intersection

Mahadevan does not offer any additional advice in this area as the algorithm simply states "*check for intersection.*" However, implementing a simple 'check line' approach, does not work in all cases. Consider figure 7.5b. These polygons

obviously intersect yet none of their edges intersect as all the intersections are through vertices. It has been suggested that vertex intersections should be treated in the same way as edge intersections. Whilst this would resolve the problem in figure 7.5b, in that the polygons would be reported as intersecting, it would now incorrectly report that the polygons in figure 7.5c intersect, when in fact they do not (this depends on the definition of intersection – for our purposes, touching is not an intersection). Therefore, there is a requirement that reports intersection correctly, ideally using D-Functions so that the primitives already developed can be used.

Analysis of the nature of intersections leads to the following observations.

It is assumed that the vertices are ordered counter-clockwise and that the first vertex (zero) is the bottom right hand vertex. These indices are shown in figure 7.5a but have been omitted from some of the other figures for reasons of clarity.

In figure 7.5a, when checking edge(1,2) of P_1 against edge(0,1) of P_2 then intersection will be returned by the D-Function.

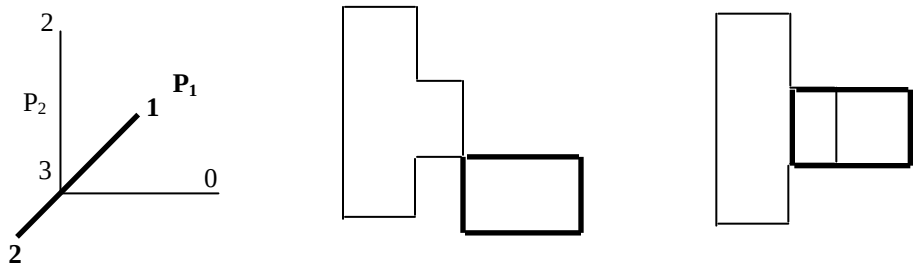


Figure 7.6 – Nature of Intersections

Consider figure 7.6a, which is a simplified version of figure 7.5b. Assume that we are checking edge(1,2) of P_1 against the edges of P_2. When we compare edge(1,2) of P_1 against edge(2,3) of P_2 the D-Function will return "VtouchesAB" (assuming P_1 takes on the edges AB and P_2 takes on the edges UV). Simply using edge intersection as a means for determining polygon intersection, would return a result indicating that the polygons do not intersect. However, if the situation is analysed further it can be seen that vertex 3 of P_2 is touching AB. The preceding, *previous*, vertex (2) is to the right of AB and the edge following it (0), *next*, is to the left. As the previous and next vertices are on different sides of line AB it indicates that the polygons intersect. If we apply this same principle to figure 7.5c the previous and next points will be on the same side of the line indicating that the polygons do not intersect; which is indeed the case.

A further condition that can arise is where one (or both) of the points (either *previous* or *next*) is linear with the point that intersects AB. This situation will arise with the two shapes shown in figure 7.6b. If, for example, the rectangle is placed on top of the protruding part of the other polygon (figure 7.6c) then all intersections will be through vertices but when we apply D-functions to this configuration they will report linear conditions. Therefore, simply checking if *previous* and *next* are on opposite sides of the line is not enough to capture all intersecting cases. Further analysis leads to the following conditions, which check for intersection (see table 7.1).

previous	Next	
Right	Right	NO Intersection
Right	Left	**Intersection**
Right	Linear	NO Intersection
Left	Right	**Intersection**
Left	Left	NO Intersection
Left	Linear	**Intersection**
Linear	Right	NO Intersection
Linear	Left	**Intersection**
Linear	Linear	NO Intersection

Table 7.1 – Detecting Intersections

This table can be realised into an implementation using the following algorithm.

```
p1, p2 are polygons, represented by vertices in ccw order
p1v, p2v is the number of vertices in p1 and p2
D_fn_Analysis is a function that returns the relationship of two lines using D-
Functions
All arithmetic on polygon vertices is assumed to be modular

For i = 0 until i = p2v-1
   For j = 0 until j = p1v-1
      ContactType = D_fn_Analysis(p2[i], p2.[i + 1], p1.[j], p1.[j+1]]
         switch(ContactType)
            case intersect
               return interection

            case UtouchesAB
               BeforeIdx = j-1
               AfterIdx = j+1
               If VtxI( p2.[i], p2.[i+1], p1.[BeforeIdx], p1.[AfterIdx])
                  return intersection

            case VtouchesAB
               BeforeIdx = j
               AfterIdx = j+2
               If VtxI(p2.[i], p2.[i+1], p2.[BeforeIdx], p2.[AfterIdx]
                  return intersection

            case AtouchesUV
               BeforeIdx = i-1
               AfterIdx = i+1
               If VtxI(p1.[j], p1.[j+1], p2.[BeforeIdx], p2.[AfterIdx])
                  return intersection

            case BtouchesUV
               BeforeIdx = i
               AfterIdx = i+2
               If VtxI(p1.[j], p1.[j+1], p2.[BeforeIdx], p2.[AfterIdx])
                  return intersection
   j = j+1
i = i+1
return no_interesct
```

The function VtxI (Vertex Intersection) is passed four points as parameters. The first two parameters represent a line. The third and fourth points are the *previous* and *next* points. They need to be checked against the line and whether the polygons intersect is dictated by referring to table 7.1.

7.5 The No Fit Polygon – Modified Algorithm

7.5.1 Introduction

In 1984 Mahadevan (Mahadevan, 1984) described a method that calculates the no fit polygon (NFP) for two non-convex polygons. In implementing the algorithm some degenerate cases were found. In this section Mahadevan's method will be described, followed by a description of the degenerate cases and how they have been overcome.

Section 7.2 of this work described the orbiting principle that forms the basis of Mahadevan's algorithm. That is, one polygon orbits another, and at each position the reference point is stored and these points eventually make up the vertices of the NFP.

The starting position is given by taking the largest y-coordinate of the orbiting polygon and the smallest y-coordinate of the stationery polygon. The polygons are aligned on these vertices. In this position the polygons are guaranteed not to intersect.

7.5.2 Sliding Edge and Sliding Vertex

In order to move the orbiting polygon to the next position a vertex (the sliding vertex) must slide along an edge (the sliding edge). The sliding edge can be on either the stationery polygon or the orbiting polygon. Similarly, the sliding vertex

can be on either polygon. However, the sliding edge and sliding vertex cannot be on the same polygon. The sliding edge and sliding vertex can be determined using D-Functions as follows.

In each of the figures in figure 7.7, vertex B is on the stationery polygon, S. Vertex E is on the orbiting polygon, O. B and E are in contact with one another.

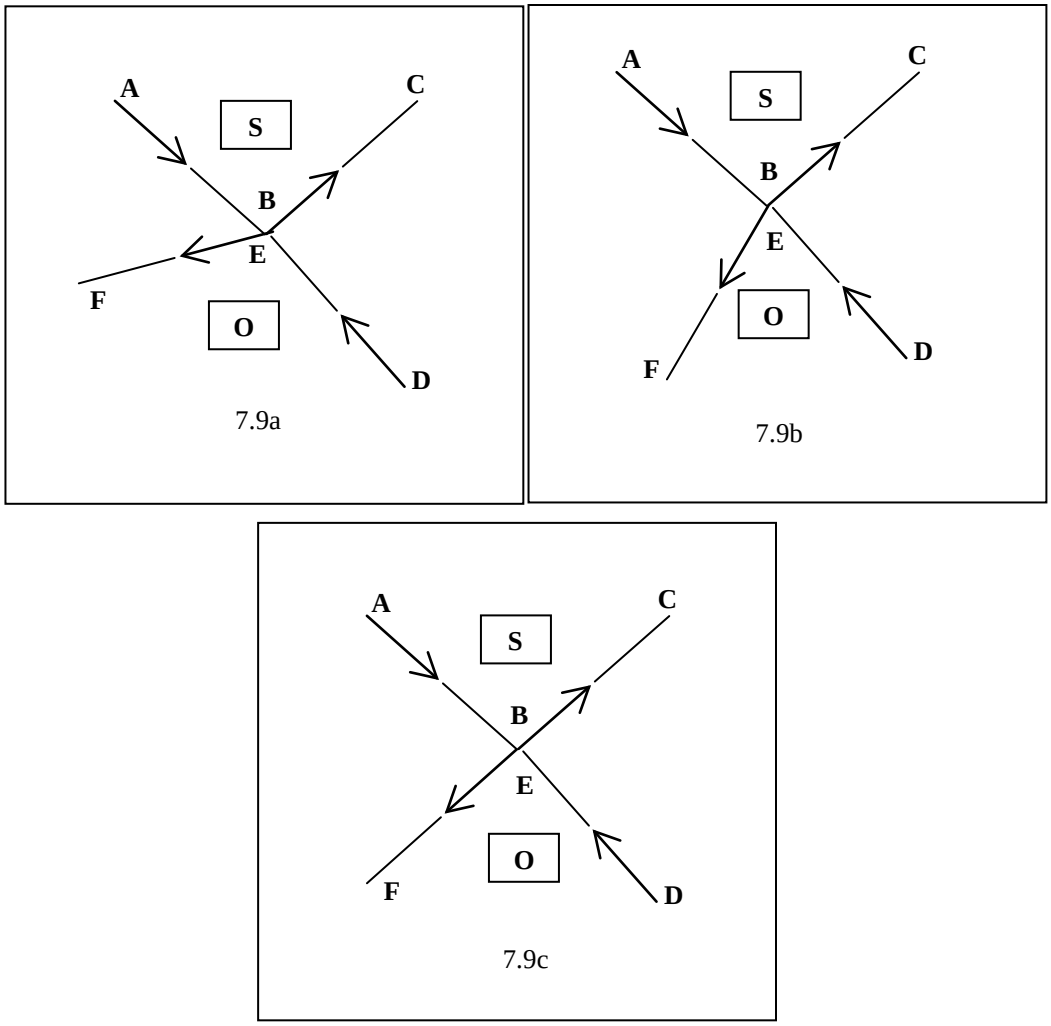


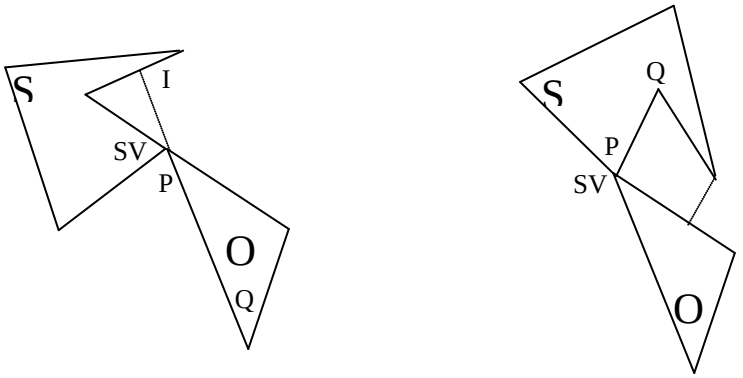
Figure 7.7 – Determining Sliding Edge and Sliding Vertex

- If F is to the left of BC ($D_{ABC} > 0$) then the sliding edge is EF and the sliding vertex is B (see figure 7.7a)

- If F is to the right of BC ($D_{ABC} < 0$) then the sliding edge is BC and the sliding vertex is E (see figure 7.7b)
- If F is linear with BC ($D_{ABC} = 0$) then the sliding edge is EF and the sliding vertex is B (see figure 7.7c)

7.5.3 Determination of Sliding Distance

When using the orbiting method on ***convex*** polygons it is ***always*** possible to move the orbiting polygon the full extent of the sliding edge. When the polygons are non-convex, this may not be possible (see figure 7.1b and 7.1c). Therefore, before moving the orbiting polygon, a check must be made as to how far the polygon can be moved. This is done by extending every vertex on the orbiting polygon, in the direction of motion, by the length of the sliding edge. The extended sliding edge is checked to see if it intersects any edge on the stationery polygon.



a **b**

Figure 7.8 – Projecting Vertices

In figure 7.8a the sliding vertex, SV, is on the stationery polygon, S, and the sliding edge, PQ, is on the orbiting polygon, O. The orbiting polygon should move

the full distance of PQ but when P is extended it intersects S at I. Therefore, the orbiting polygon can only be moved by the distance PI. Under these circumstances the new sliding edge is the edge that was intersected and the new sliding vertex is the vertex that intersected S (in this case P). Alternatively, the new sliding edge and sliding vertex can be determined using D-Functions, as described above.

Each vertex on S must also be extended in the ***opposite*** direction of motion by the distance of the sliding edge in order to check it does not intersect O (see figure 7.8b).

If no intersections are found then the orbiting polygon can be moved the full distance of the sliding edge.

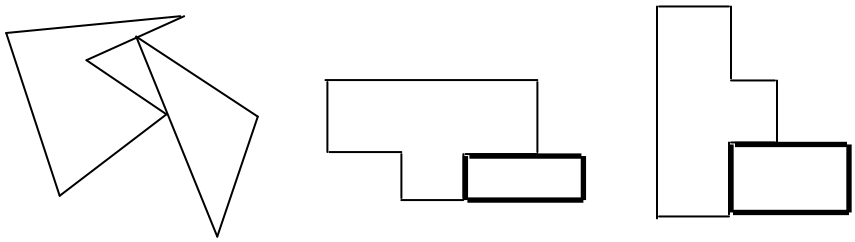
Equations 7.4 and 7.5 give a method to extend a point in a given direction. This can be used to extend point R, by a distance of PQ, in the direction of PQ, where $N.x$ and $N.y$ are the x and y co-ordinates for point N.

$$Extend.x = R.x + (Q.x - P.x) \tag{7.4}$$

$$Extend.y = R.y + (Q.y - P.y) \tag{7.5}$$

7.5.4 Multiple Points of Contact

In figure 7.8 it is relatively easy to determine the sliding edge and vertex as there is only one point of contact between the two polygons. However, it is possible to have multiple points of contact (see figure 7.9).



a **b** **c**

Figure 7.9 – Multiple Points of Contact

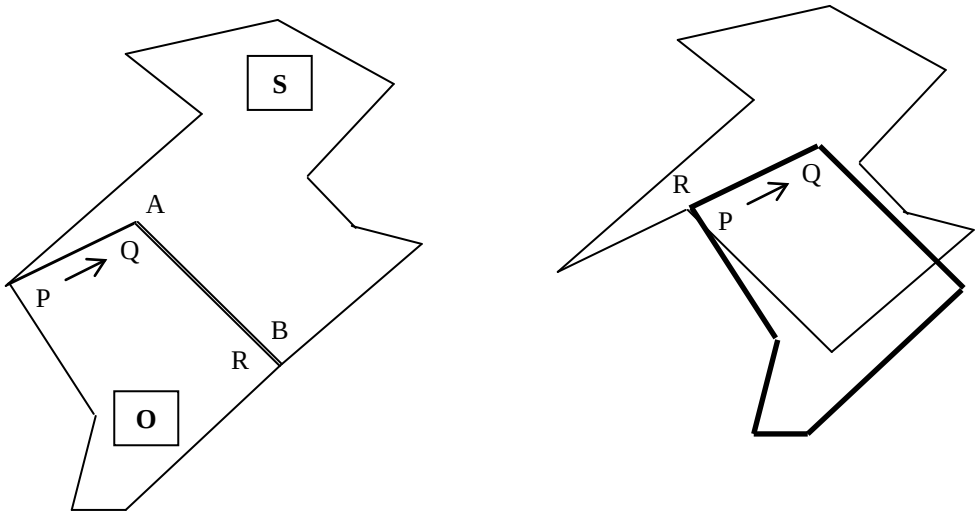
Figure 7.9a shows two points of contact, each of which are candidates for being a sliding edge and sliding vertex. Figures 7.9b and 7.9c show numerous points of contact (the larger shape in figure 7.9b is a T shape, which has been rotated in figure 7.9c). Mahadevan mentions the possibilities of multiple contact points but gives no method for determining them. Our algorithm checks each edge of one polygon against each edge on the other polygon. Using D-Functions it is possible to determine when a contact is found and thus identify the potential sliding edge and vertex.

Once a sliding edge/vertex has been identified the orbiting polygon can be moved, using the checks described above to ensure that a vertex on one polygon does not pierce the edge of the other polygon. However, it is possible, not to detect an intersection before the orbiting polygon has moved but moving the polygon still leads to an intersection (an example is shown below). Due to this a data structure is maintained so a search for valid sliding edge and vertex can be continued, should the current candidates lead to an invalid (intersecting) move. This data structure simply holds the vertex index of the stationery and orbiting polygon so

that the search can be continued from that point. The data structure is reset after each valid move so that the search starts from the zero'th vertex on each polygon.

7.5.5 Check for Intersection After Potential Move

Once the orbiting polygon has been moved it is necessary to check if the two polygons intersect. This is needed as the projection of the vertices described above does not capture all cases where intersection will occur. This is due to the fact that only the vertices have been extended but intersection could occur on any edge. The example given in (Mahadevan, 1984) is shown in figure 7.10. In figure 7.10a the sliding edge has been identified as PQ and the sliding vertex as A. The direction of travel is shown by the arrow. If every vertex on the orbiting polygon, O, is extended by length PQ in the direction of travel, no intersection is detected. Similarly, if every vertex on S is extended by length PQ in the opposite direction of travel, no intersection is detected. Therefore the move is made by moving P so that it aligns with A (figure 7.10b). It is now obvious that the polygons intersect. By continuing the search for a sliding edge and vertex, using the configuration of 7.10a, another point of contact would identify QR as the sliding edge and B as the sliding vertex. This move will not result in intersection and thus the move can be made permanent. At this point the search for a sliding edge and vertex will be reset so that the next sliding edge/sliding vertex search will start afresh (that is, start searching at the zero'th vertices).



a **b**

Figure 7.10 – Polygons may intersect after extending vertices

7.5.6 Additions to the Algorithm

Apart from the additions to the intersection procedure (section 7.4), the algorithms described above are the same as that described by Mahadevan (Mahadevan, 1984). However, there are some degenerate cases where the algorithm does not work. Our work has corrected these areas so that the algorithm is more robust.

Checking for Cycles

Consider figure 7.11a, which is part way through the NFP algorithm. Depending on the order in which the next contacts are determined the next position could either be figure 7.11b or figure 7.11c.

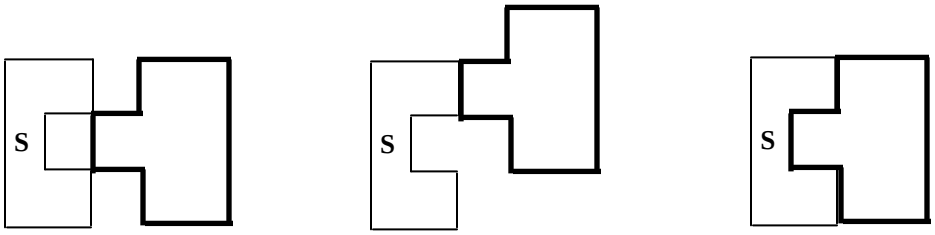


Figure 7.11 – Possibility of algorithm not terminating

Arriving at figure 7.11b presents no problem in that the orbiting polygon, O, can continue its journey around the stationery polygon, S. However, by moving immediately to the configuration of figure 7.11b the more compact packing of figure 7.11c will be missed. Therefore, depending on the eventual use to which the no fit polygon will be put it may be more beneficial to ensure that every contact is explored so that tighter packings are not passed over. Now consider figure 7.11c. This results in a tighter packing of the two polygons but presents two additional problems. Firstly, when O tries to move again, the only viable option is to move back to the configuration shown in figure 7.11a. From here the next move could be back to the configuration of figure 7.11c. In order to stop the algorithm entering an infinite loop there needs to be a mechanism to identify these situations and recover from them.

In the modified algorithm this is achieved by using a stack that stores valid positions. Each time a valid placement is found an item is pushed onto the stack. Each stack item contains three elements.

1. The current vertex index of the stationery polygon that is being processed as part of the next contact algorithm
2. The current vertex index, plus 1, of the orbiting polygon that is being processed as part of the next contact algorithm.
3. The polygon description of the orbiting polygon that reflects its position, before it was moved based on the sliding edge and vertex that are now stored (1 and 2, above).

If an item is popped off the stack the vertex indices are passed to the next contact algorithm. These are used as starting points for establishing the next contact between the orbiting and stationary polygons. By adding one to the orbiting polygon index before pushing it onto the stack it effectively moves the checking procedure on one stage from the position that existed when the item was pushed onto the stack. The polygon is pushed onto the stack so that the current position is stored and can be restored.

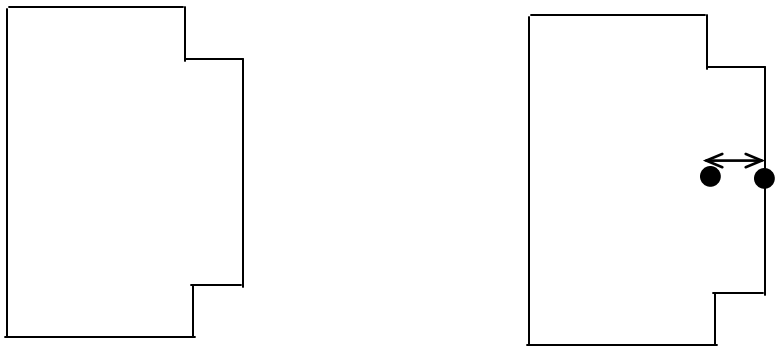
Another check the algorithm needs to make is to ensure that repeated positions are not revisited. For example, the original algorithm could find itself in the configuration shown in figure 7.11c. The next valid configuration will be that shown in fig 7.11a. However, this position will already have been seen, therefore, it is regarded as an invalid position.

When the position in figure 7.11c is reached the algorithm will search for the next valid position. It will arrive at the position shown in figure 7.11a and discover, by peeking at the polygon at the top of the stack and comparing it against the current

position, that this position has already been seen. Therefore, it will ignore this move and continue to look for another valid position. Not being able to find one it will pop the top item from the stack and will set the orbiting polygon to the polygon just popped (i.e. the position shown in figure 7.11a). The next contact vertices will be set to those that were stored on the stack. These will have been the indices in effect when the position in fig 7.11a was originally reached, except the orbiting index will have been incremented by one. The result is that the algorithm will continue and find the position shown in fig 7.11b.

Valid Polygons

A secondary problem is that the vertices returned from the algorithm are not guaranteed to generate a polygon when they are navigated in a counter clockwise order.



a **b**

Figure 7.12 – Algorithm does not have to produce a valid polygon

Figure 7.12a shows the no fit polygon that is produced by the two polygons in figure 7.11, under the assumption that the algorithm progresses from figure 7.11a to figure 7.11b and does not find the configuration shown in figure 7.11c. If the configuration in figure 7.11c is reached this results in an additional point on the no fit polygon, as shown in figure 7.12b. It is no longer possible to draw a polygon, at least not a simple polygon, as an edge would need to be drawn from the right hand point in figure 7.12b to the left hand point in this figure. Once the algorithm backtracked, an edge would then be drawn in the opposite direction before the no fit polygon was completed as shown in figure 7.12a.

A potential solution is to not store the left hand point in figure 7.12b. This is easily achieved by removing the point from the NFP polygon at the time there is a need to remove an item from the stack. However, this gives a potential problem in that the packing shown in figure 7.11c is effectively lost. In the application that is developed in this thesis we want to find the tightest possible packings. Therefore it is beneficial not to discard any potential solutions due to backtracking. Due to the nature of our problem we are only interested in the vertices of the NFP. Therefore, our NFP algorithm only needs to return a list of vertices never has to construct a polygon.

Different applications will obviously have different requirements. Therefore, the way in which the algorithm is implemented by other researchers is dependent upon the particular use of the resultant NFP, but we have highlighted several aspects which need to be considered.

Finally, as the NFP is built it is possible for three (or more) points on the NFP to be co-linear. Whilst this does not affect the properties of the resultant polygon it can lead to inefficiencies as the number of vertices on the NFP is greater than it needs to be to represent the same polygon. Therefore, we have introduced a check into our algorithm that checks for co-linear points as the NFP is built. This check is easily performed using D-Functions.

Sliding Distance

In section 7.5.3 a discussion was given as to how to determine the sliding distance for a given orbiting polygon. The amount that the polygon is allowed to slide is determined by recognising edge intersections. However, consider figure 7.13.

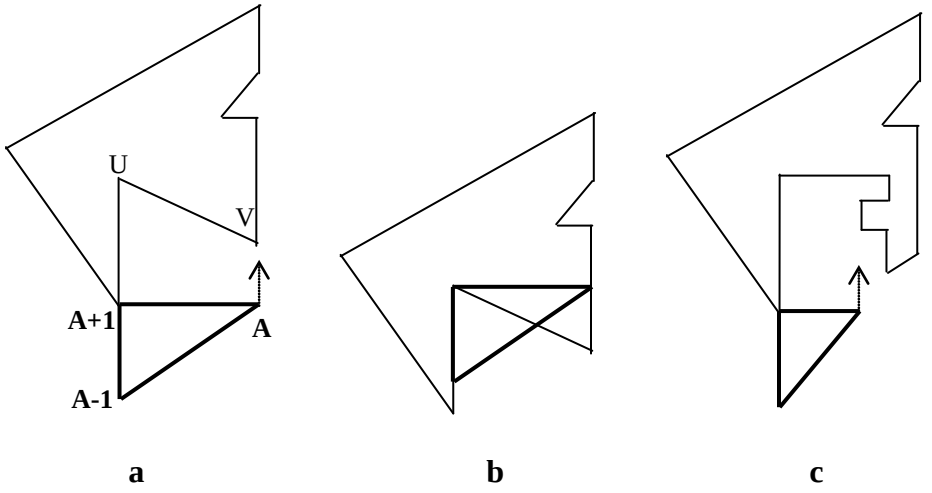


Figure 7.13 – Determining Sliding Distance when there is No Edge Intersection

In figure 7.13a, the sliding vertex will be identified as vertex A+1. The sliding edge will be the vertex leading towards U. Therefore, the direction of travel is

shown by the arrow leading from A. Vertex projections (as discussed in 7.5.3) will detect no edge intersections, therefore the polygon will be moved the full length of the sliding edge (7.13b). The usual check for intersection will now be carried out and it will be found that the polygons intersect. Therefore, the state will revert to that shown in 7.13a. The algorithm now tries to determine a new sliding edge and vertex. Not finding any suitable candidates the algorithm will either cycle or terminate.

The problem is due to the fact that the projection of vertex A (7.13a) is not able to detect that it can only move as far as V. The only indication that there is a potential problem is when the vertex A is extended (call this edge A, B) and, using D-functions, it is found that "VtouchesAB". However, the situation is complicated in that a configuration such as that shown in 7.13c could also arise. Extending the same edge as before, D-functions will return "VtouchesAB" but, in this instance the polygon can move the full distance. In fact, it is desirable if we want the concavity to be explored.

Therefore, it is necessary to identify some condition that allows the configurations of 7.13a and 7.13c to be differentiated.

Similar analysis to that carried out for checking for intersection through vertices (section 7.4) leads to the following observations.

Let the orbiting vertex being extended be called AB.

Call the vertices *before* and *after* A, A-1 and A+1 respectively.

The stationery polygon has vertices U and V. At a given point in the NFP algorithm, the D-functions will return "VtouchesAB" (see figure 7.13a).

7.14a shows a simplified version of 7.13a, at the point when the D-functions return VtouchesAB. Figure 7.14b is the state of 7.13c when VtouchesAB.

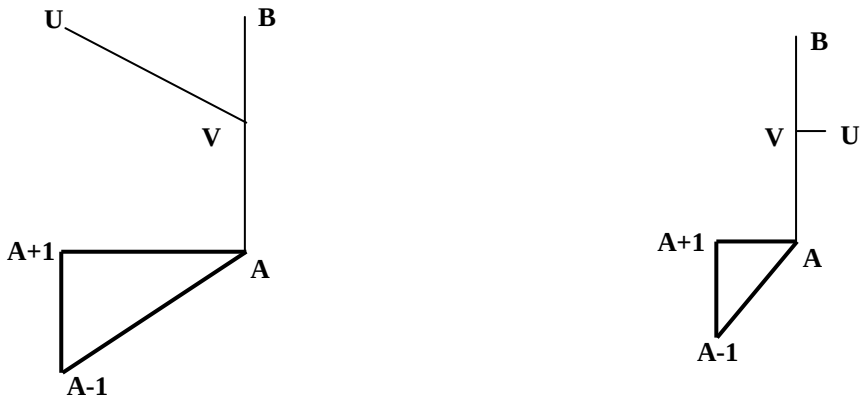


Figure 7.14 – Simplification of Figure 7.13 for D-function Analysis

Looking at figure 7.14 it is apparent that if U is linear with AB (which implies it is also linear with V), then A can slide the full distance as neither A,A-1 or A,A+1 will intersect UV.

If U is not linear with AB then two other conditions need to be checked to ascertain if A can slide the full distance. If U (in relation to AB) and A-1 (again in relation to AB) have different values, this indicates that A can slide the full distance without A,A-1 or A,A+1 intersecting UV. To put it another way, if A-1 and U are on opposite sides of AB, then A can slide the full distance. In addition the same must hold for U (in relation to AB) and A+1 (in relation to AB).

Of course, D-Functions can be used to check these conditions and this can be expressed as follows.

```
IF (D_fn(A, B, U) ≠ D_fn(A, B, A-1)) AND (D_fn(A, B, U) ≠ D_fn(A, B, A+1))
THEN
Move Full Distance
ELSE
Move as far as V
```

These observations can be captured in the following table. The final column gives a statement which captures whether A can be moved the full distance or if A can only be moved as far as V. The two highlighted rows shows the conditions in force for figure 7.13a (higher row) and figure 7.13c (lower row).

Relationship with AB			Conditions to Check			Move Full Distance or Not
U	**A-1**	**A+1**	**U linear AB?**	**U & A-1 different?**	**U & A+1 different?**	**(U is linear) ∨ ((U & A-1 different) ∧ (U & A+1 different))**
left	***left***	***left***	***FALSE***	***FALSE***	***FALSE***	***Move to V***
left	left	right	FALSE	FALSE	TRUE	Move to V
left	left	linear	FALSE	FALSE	TRUE	Move to V
left	right	left	FALSE	TRUE	FALSE	Move to V
left	right	right	FALSE	TRUE	TRUE	Move Full Distance
left	right	linear	FALSE	TRUE	TRUE	Move Full Distance
left	linear	left	FALSE	TRUE	FALSE	Move to V
left	linear	right	FALSE	TRUE	TRUE	Move Full Distance
left	linear	linear	FALSE	TRUE	TRUE	Move Full Distance
right	***left***	***left***	***FALSE***	***TRUE***	***TRUE***	***Move Full Distance***
right	left	right	FALSE	TRUE	FALSE	Move to V
right	left	linear	FALSE	TRUE	TRUE	Move Full Distance
right	right	left	FALSE	FALSE	TRUE	Move to V
right	right	right	FALSE	FALSE	FALSE	Move to V
right	right	linear	FALSE	FALSE	TRUE	Move to V
right	linear	left	FALSE	TRUE	TRUE	Move Full Distance
right	linear	right	FALSE	TRUE	FALSE	Move to V
right	linear	linear	FALSE	TRUE	TRUE	Move Full Distance
linear	left	left	TRUE	TRUE	TRUE	Move Full Distance
linear	left	right	TRUE	TRUE	TRUE	Move Full Distance
linear	left	linear	TRUE	TRUE	FALSE	Move Full Distance
linear	right	left	TRUE	TRUE	TRUE	Move Full Distance
linear	right	right	TRUE	TRUE	TRUE	Move Full Distance
linear	right	linear	TRUE	TRUE	FALSE	Move Full Distance
linear	linear	left	TRUE	FALSE	TRUE	Move Full Distance
linear	linear	right	TRUE	FALSE	TRUE	Move Full Distance
linear	linear	linear	TRUE	FALSE	FALSE	Move Full Distance

Table 7.2 – Detecting if Orbiting Polygon Can Move the Full Distance

7.6 Combining Polygons

Once the NFP has been determined for two given polygons, similar to the non-convex algorithm, the best placement needs to be found that minimises the area for the two polygons when they are placed together. This is achieved by placing the reference point of the orbiting polygon on each vertex of the NFP and returning the placement which returns the best fit. To decide which is the best fit, similar to

the non-convex case, the convex hull is calculated. The convex hull is used as a measure, rather than *combining* the polygons, as *combined* polygons will be identical, with regards to their area, and thus there will be no way of deciding between one placement and another. If there is more than one placement that returns the same minimum area, one placement is used at random, although all the placements are stored in the cache for future use. This part of the algorithm is exactly the same as for the non-convex case that was explored earlier in this thesis. Once the best placement has been found the two polygons are *combined* to form one polygon which is used as the next stationery polygon.

Combining two polygons is an operation which is seemingly simple but the implementation is delicate. Initially, the basic idea will be described, followed by the implementation issues that had to be overcome before the algorithm could be realised.

a **b**

Figure 7.15 – Combining Two Polygons

Figure 7.15a and figure 7.15b show a simple case of combining two polygons. There are libraries of algorithms that allow these types of operation to be done (e.g. LEDA and CGAL) but this work is implemented using our own algorithms which draw heavily on D-Functions.

The basic idea behind the algorithm is to identify a vertex on one of the polygons which is guaranteed to be on the combined polygon. In the case of figures 7.15a and 7.15b this could be the vertex identified by the filled circle. The algorithm proceeds to *move* around this polygon, at each stage, checking (using D-Functions) if there is any contact with the other polygon. If there is a contact then a decision is made whether to switch to the touching polygon, and continue to move around that one. Before describing the *combine* algorithm some support algorithms must be defined.

Given an edge, AB, from polygon, P_1, it is possible to establish the relationship of AB with another polygon P_2. Each edge on P_2 is checked against AB to check this relationship. As each edge of P_2 is processed it is called UV. Once the relationship between the P_1 and P_2 has been established the algorithm can return an index which represents U and the type of contact between the two polygons.

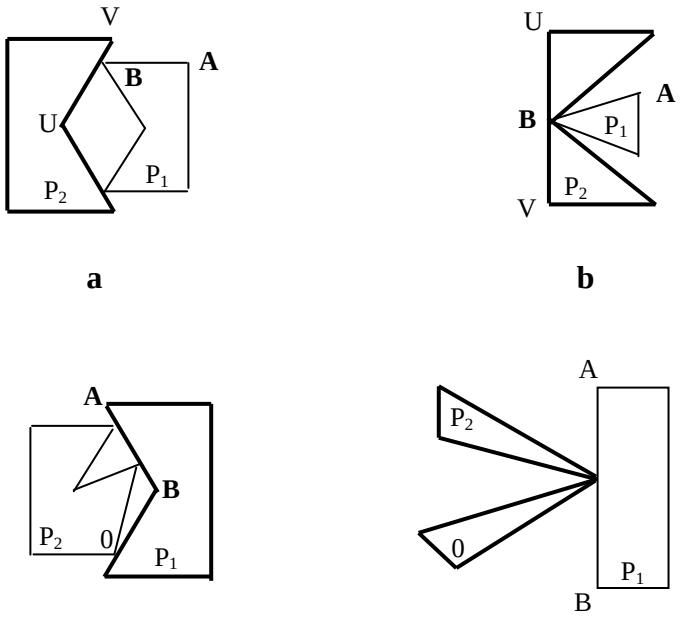


Figure 7.16 – Polygon Relationships to Consider

Figure 7.16 shows several configurations that demonstrate some of the cases that have to be considered. In figure 7.16a, AB is the edge from P_1 and UV, on P_2, is being considered. D-Functions will return that "BtouchesUV". As there are no other contacts with AB the function will return the index of U and a contact type of "BtouchesUV". However, one other check is required, as shown in figure 7.16b. Two of the vertices of P_2 lie on one of the edges, UV, that is currently being checked. When P_1 is checked against P_2 it will return a contact type of "BtouchesUV". However, in this case, the contact type is not valid as it is not valid to switch from AB on P_1 to UV on P_2. This is a simple check to make. When a contact type of "BtouchesUV" is returned if V is to the left of AB, then it is not valid to swap polygons. This is easily checked with D-Functions.

In figure 7.16c, there are two occurrences of "UtouchesAB" (there are also occurrences of "VtouchesAB" but these can be ignored). The next stage will be to move onto P_2, so that the combined polygon can continue to be formed. However, a decision needs to be made as to which edge on P_2 is employed. In this case it is the edge that is closest to A on AB. Therefore, in this case, the function can return a contact type of "UtouchesAB" and U as the index nearest A.

In figure 7.16d, the situation is similar to figure 7.16c in that there are two occurrences of "UtouchesAB" but they cannot be differentiated by their distance from A as they are effectively the same point. In this case, the edge to be used should be the one that forms the smallest angle with AB.

We now have the necessary conditions to return the relationship between P_1 and P_2. The function, which we have called CombineReln, returns an index which represents U and the type of contact found and is a key component in the main combine algorithm. The combine algorithm can be shown formally as follows.

```
Variable types
Integer  Crnt_Vertex : Vertex index being processed
Integer  Uidx : Vertex index for the "other" polygon
Polygon  Crnt_P : Polygon which is currently being considered
Polygon  Other_P : The "other" polygon
Point    A : Current vertex on Crnt_P
Point    B : Vertex_A +1


1. Crnt_Vertex ← Vertex guaranteed to be on resultant polygon
2. Crnt_P ← Polygon on which Start_vertex is placed
3.
4. A ← Crnt_P[Crnt_Vertex]
5. B ← Crnt_P[Crnt_Vertex +1]
6. DO
```

```
7. CombineReln(A, B) RETURNS Uidx & Contact
8. SWITCH(Contact)
9. CASE : BtouchesUV
10.       IF(D_fn(A, B, Crnt_P[Crnt_Vertex +2] = right
11.         Crnt_Vertex ← Crnt_Vertex +1
12.         BuildNewPolygon(Crnt_P[Crnt_Vertex])
13.         A ← Crnt P[Crnt_Vertex]
14.         B ← Crnt_P[Crnt_Vertex +1]
15.       ELSE
16.         BuildNewPolygon(Crnt_P[Crnt_Vertex +1])
17.         A ← Crnt_P[Crnt_Vertex +1]
18.         B ← Other_P[Uidx +1]
19.         Crnt_Vertex = Uidx
20.         SwapPolygons()
21.       ENDIF
22.       CASE : UtouchesAB
23.       IF(D_fn(A, B, Other_P[Uidx +1] = left
24.         Crnt_Vertex ← Crnt_Vertex +1
25.         BuildNewPolygon(Crnt_P[Crnt_Vertex])
26.         A ← Crnt P[Crnt_Vertex]
27.         B ← Crnt_P[Crnt_Vertex +1]
28.       ELSE
29.         BuildNewPolygon(Other_P[Uidx])
30.         A ← Other_P[Uidx]
31.         B ← Other_P[Uidx +1]
32.         Crnt_Vertex = Uidx
33.         SwapPolygons()
34.       ENDIF
35.       CASE : BandUtouch
36.       IF(theta(A, B, Crnt_Vertex, newA)
37.         SwapPolygons()
38.       ENDIF
39.       BuildNewPolygon(B)
40.       Crnt_Vertex = newA
41.       A ← Crnt P[Crnt_Vertex]
42.       B ← Crnt_P[Crnt_Vertex +1]
43.       CASE : AandUtouch OR nointersect
44.       Crnt_Vertex ← Crnt_Vertex +1
45.       BuildNewPolygon(Crnt_P[Crnt_Vertex])
46.       A ← Crnt P[Crnt_Vertex]
47.       B ← Crnt_P[Crnt_Vertex +1]
48.     END_SWITCH
49.  UNTIL finished processing polygons
```

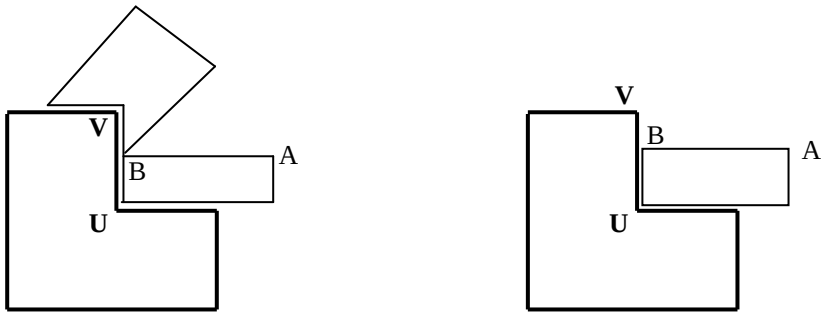
There are a number of support routines.

- D_fn(…) is the D-Function primitive (see figure 7.3).
- BuildNewPolygon(…) takes a vertex as a parameter and adds this vertex to the polygon being built (that is, the combined polygon).

- SwapPolygons(…) controls the process of swapping between one polygon and the other, when the necessary condition arises.
- theta(…) is used to determine, which edge, from those available should be moved to next. This operation is discussed further below.
- CombineReln(…) has been discussed above.

Using CombineReln, the relationship between P_1 and P_2 is established. Each of the contact types of interest is discussed below. These discussions should be read in conjunction with the algorithm shown above. Notice, the complications for the combine algorithm come from the fact that as polygons are combined self intersecting polygons can be formed (see 7.16b, 7.16c and 7.16d). If this were not the case, the combine algorithm would be considerably simpler to implement.

BtouchesUV



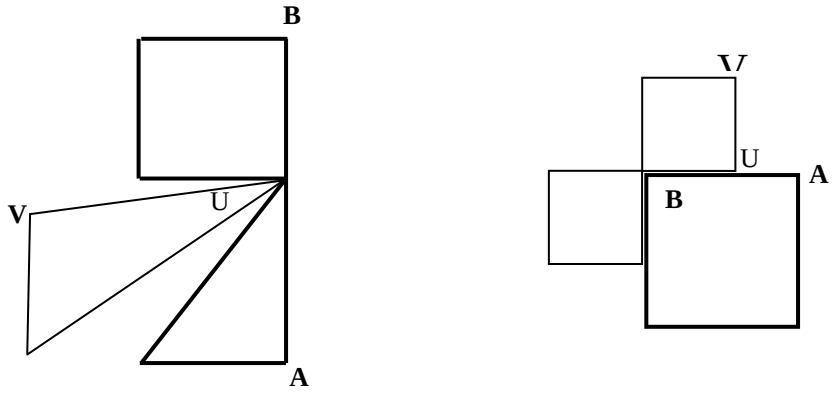
a **b**

Figure 7.17 – Whether to swap polygons or not when "BtouchesUV"

Given the configuration is figure 7.17a, if "BtouchesUV" the vertex following B is to the right of AB. This indicates that navigation should continue on the same polygon. Given the configuration shown in 7.17b, navigation should change to the other polygon.

UtouchesAB

If "UtouchesAB" is returned, given the configuration shown in figure 7.18a, then the vertex following U (that is V) is to the left of AB. This indicates that navigation should continue on the same polygon. Given the configuration shown in 7.18b, navigation should change to the other polygon as V is to the right of AB.



a **b**

Figure 7.18 – Whether to swap polygons or not when "UtouchesAB"

BandUtouch

If B and U touch, it is necessary to check if a polygon switch should take place by checking the angles of various edges with edge AB. In doing this, the angle of B, B+1 is calculated and this is compared with the angle of every edge from the other polygon. The edge which forms the smallest angle is the next edge on the polygon being formed. This is the operation carried out by the theta function. It returns a boolean stating if a swap should take place. It also returns an index for the new A vertex.

In figure 7.19a, the smallest angle with AB is B,B+1. Therefore, no polygon swap will take place and the new A index will be the current B index.

Later, the situation will be as shown in figure 7.19b. In this case, the smallest angle is found to be with the other polygon. This indicates that a polygon swap should take place. Also notice, that the new AB edge will not be the UV edge. Instead, it is the dashed edge. This is the reason why all edges have to be checked on the other polygon. Whereas, only one edge has to be checked on the current polygon as if navigation is to continue around the current polygon, is has to be on edge B,B+1.

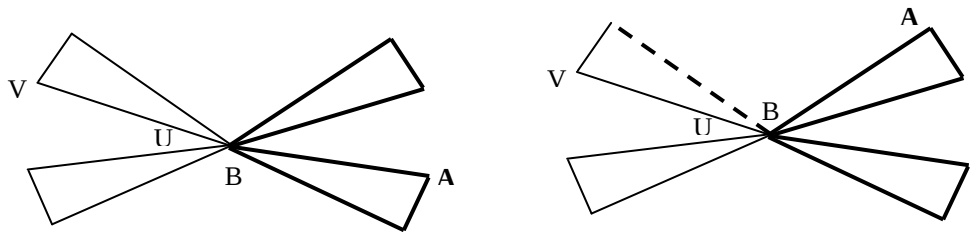


Figure 7.19 – Whether to swap polygons or not when AandUtouch

NoIntersect and AandUtouch

If either of these conditions arise it indicates that naviagtion around the current polygon should be continued.

7.7 Summary

We have presented an algorithm to produce a no fit polygon using non-convex polygons. We believe this algorithm is more robust than the algorithm it is based on (Mahadevan, 1984) as it deals with more degenerate cases. Our algorithm allows polygon intersections to be more accurately reported. This method of intersection uses D-Functions which are already integral to the NFP algorithm.

The revised algorithm also deals with cases where the algorithm has to backtrack, in order to stop it entering an infinite loop.

Depending on the polygons, certain configurations may be passed over. However, it is quite simple to ensure that the algorithm considers all possible moves before deciding where the orbiting polygon should move to. It may also be the case that the backtracking has to remove a vertex from the NFP that is being constructed. In

our implementation we prefer not to do this. Instead, we return an array of vertices because this is all our application requires.

In addition, this chapter has presented an algorithm to combine two polygons that relies heavily on D-Functions. Whilst this area is not new (see LEDA (http://www.mpi-sb.mpg.de/LEDA/leda.html), for example), the author has found no literature that uses D-Function to perform this operation. In the context of this work, it is beneficial to use D-Functions as they are already used for the NFP algorithm, they are well understood and it means we can make use of primitive functions which we already have available.

The algorithm is complicated by the fact that after several combine operations have been carried out, it is possible to have polygons that is self-intersecting. The algorithm presented here is able to deal with these cases.

Chapter 8

"The most extensive computation known has been conducted over the last billion years on a planet-wide scale: it is the evolution of life. The power of this computation is illustrated by the complexity and beauty of its crowning achievement, the human brain." David Rogers, Weather Prediction Using a Genetic Memory

8. Comparing Meta-Heuristics and Evolutionary Algorithms when Applied to the Non-Convex Nesting Problem

8.1 Introduction

Previous chapters of this thesis (chapters 5 and 6) have shown that, using evolutionary and meta-heuristic approaches can give good quality solutions to the nesting problem. In those chapters only convex pieces were used as these are easier to manipulate and it was beneficial to show that the methods employed did produce better quality solutions than straight forward approaches such as hill climbing.

In chapter 7, a no fit polygon algorithm was developed that allowed manipulation of non-convex polygons. Having this algorithm available allows us to continue the investigation using non-convex shapes. Intuitively, we would expect even better solutions to be produced. To ascertain if this is the case we, initially, test the non-

convex algorithms against the two test problems we have used thus far and then data from the literature is tested.

8.2 Comparison of Test Problems with Convex Results

In chapter 6, ant algorithms and memetic algorithms were used as a search mechanism for two test problems. The best results were obtained using a memetic algorithm and are summarised in table 8.1.

Test Data	Best Evaluation
1	283.07
2	890.51

Table 8.1 – Best Results for Test Data 1 & 2 using Convex Algorithm

These results were obtained using experiments that carried out 8000 evaluations. On a Pentium PII (laptop) with 16MB main memory this took about ten minutes. Identical runs were carried out using the non-convex no fit polygon algorithm and the number of evaluations were adjusted so that the algorithms ran for about the same amount of time. To achieve this the searches were only allowed to carry out 300 evaluations. All the same searches were conducted and the results are shown in table 8.2 and table 8.3. All results are averaged over 10 runs. The results are sorted in ascending order of evaluation.

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

Table 8.2 – Results for Test Data 1 using Non-Convex No Fit Polygon Algorithm

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

Table 8.3 – Results for Test Data 2 using Non-Convex No Fit Polygon Algorithm

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

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

8.3 Approximating the No Fit Polygon

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

This algorithm can be shown more formally as follows.

```
S ← Stationery Polygon
O ← Orbiting Polygon
sv ← Random Vertex on Stationery Polygon
start ← sv
ov ← Random Vertex on Orbiting Polygon

do {
      Align ov on sv

      if no intersection
            C ← Combine(O, S)
            if ConvexHull(C) is smallest so far
                  P ← C
            end if

      end if

      sv = sv +1 [using modulus arithmetic]
} while(start ≠ sv)

return P
```

Notice that the starting vertices are chosen randomly. This is to ensure that if two placements (for the two given polygons) have more than one convex hull which has the minimum area then each has a chance of being selected. If placements always started at, say, the zero'th vertex then the algorithm would always return the same placement, which may not be the best placement later in the nesting procedure (see section 4.3).

If we wish to use the cache it is necessary to store all the minimum placements in the cache and return one of the placements at random. This approximated no fit polygon algorithm will now operate in the same way as described in chapter 4.

Another possible performance improvement is to stop the evaluation of a particular permutation of polygons, once the evaluation exceeds the current minimum. Intuitively this seems like a good idea as if the polygons being evaluated exceeds the minimum evaluation found thus far, it is pointless continuing as the current evaluation cannot improve on the best nesting found so far. However, some of the search algorithms require a full evaluation to be carried out due to the way they operate.

Simulated annealing needs to carry out a full evaluation as it needs the change in evaluation so that it can use this value in the acceptance function in deciding whether to move to the solution under consideration. Ant algorithms need to carry out a full evaluation so that each ant can decide how much pheromone to deposit on each edge. Finally, a genetic algorithm also requires a full evaluation as the parent chromosomes are selected for breeding based on the fitness. In order to assign a fitness value a full evaluation has to be carried out.

Experiments were run to test the two ideas presented above. Four initial experiments were run, combining all the possible combinations. They can be summarised as follows.

1. Use the ***no fit polygon algorithm*** and ***do not stop evaluations*** when the minimum is exceeded.
2. Use the ***no fit polygon algorithm*** but ***stop evaluations*** when the minimum is exceeded.

3. Use the ***approximated no fit polygon algorithm*** and ***do not stop evaluations*** when the minimum is exceeded.
4. Use the ***approximated no fit polygon algorithm*** but ***stop evaluations*** when the minimum is exceeded.

The test data was taken from (Hopper, 2000a), using the first set of test data in category 1. This data comprises 16 rectangles which are to be nested in a bin of width 20.

The results obtained are shown in table 8.4, with the results being averaged over ten runs. The test numbers refer to the numbers given above for the four proposed tests, with the description being given for ease of reference.

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

Table 8.4 – Approximating the NFP and Using Partial Evaluation

At first sight the results are not encouraging. Using test 1 as a baseline, this uses the NFP algorithm that has been developed earlier in this thesis (chapter 7) and carries out complete evaluations. When partial evaluations are used, with the NFP algorithm developed in chapter 7 (test 2), the results show that the algorithm runs faster with no significant difference in solution quality. However, if the

approximated NFP algorithm is used (test 3 and 4), there is a significant speed improvement but also a reduction in solution quality. This is due to the fact that the normal NFP algorithm allows vertices of the orbiting polygon to rest on edges on the stationery polygon. Using the approximation algorithm means that at least one vertex of the orbiting polygon must be aligned with a vertex of the stationery polygon.

Therefore, at first sight, it would seem advisable to implement partial evaluation but to continue to use the slower, but better quality, NFP algorithm developed in chapter 7. However, further investigation shows that this may not have to be the case.

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

However, there could be a trade off between these two extremes. It may be possible to run the algorithm for longer, using the faster methods, achieve good quality solutions but still save on the execution time of the algorithm.

Table 8.5 shows the results of these experiments

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

Table 8.5 – Using an Approximation of the NFP to Reduce Run Times

The first two rows of table 8.5 are the best and worst results from table 8.4, to allow easier comparison. Row three doubles the number of iterations from the original tests. The result is an improvement of solution quality but in significantly less time than using the original NFP algorithm with complete evaluation (row 1).

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

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

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

Therefore, the work in the remainder of this chapter uses both the approximated no fit polygon and partial evaluation.

8.4 Hopper & Turton Datasets

The TSP (travelling salesman problem) community have access to a set of test problems (see, for example http://www.crpc.rice.edu/softlib/tsplib/) that are available to the community to allow researchers to use a common set of problems to test new algorithms. The cutting and packing field is limited in this respect, although it is encouraging to see that Hopper and Turton (Hopper, 2000c) are trying to address this problem. In addition, Hopper and Turton (Hopper, 2000a) do present several datasets (appendix D and E) which are investigated here.

8.4.1 Comparison over 7000 iterations

As an initial test of the Hopper and Turton datasets, several of the search methods were selected and these were tested against the category 1 and 2 datasets in (Hopper, 2000a) and reproduced in appendix D and E of this thesis.

The algorithms selected were hill climbing, HC, (to provide a base case), a genetic algorithm, GA, (to provide a population based approach) and tabu search, TS, and a memetic algorithm, MA, as these had proved to be the most effective searches found so far.

The category 1 and 2 data from (Hopper, 2000a) provides six sets of data, EH001..EH006. Each data set was run against each algorithm. In addition, the runs for each data set/algorithm were run twice, once allowing the shapes to rotate through 90°, the other run allowing no rotation. The results was averaged over ten runs. Therefore, there are 480 separate runs (8 algorithms (HC, TS, GA, MA, each

one run with and without rotation)), 6 data sets with all results being averaged over 10 runs).

For the hill climbing algorithm 7000 iterations were made, with a neighbourhood size of one. When allowing rotation the rotation probability, *rp*, was set to 0.5. This was used to decide if the current iteration should swap two polygons or rotate a random polygon. The value of *rp* was set to 0.5 on the basis that for any given problem there is no knowledge as to what is the best orientation of any given polygon. Therefore, it seemed reasonable to have an equal chance of swapping or rotating polygons.

For the genetic algorithm the population size was set to twenty and the number of iterations set to 350. This makes a total number of iterations of 7000. However, this is slightly misleading as in a genetic algorithm crossover is only made with some probability. Therefore, it could be argued that the number of iterations is less than for some of the other algorithms. Indeed, in general, the GA took less time to run. This is something addressed more fully in section 8.5, where the algorithms are compared using time rather than the number of iterations.

The other parameters for the GA were set to the best parameters found so far (see section 5.4.3 of this thesis). When rotation was not allowed, mutation simply swapped two randomly chosen polygons. When rotation was allowed the mutation operator rotated a random polygon.

For tabu search, 350 iterations were carried out with a neighbourhood size of 20. This resulted in 7000 iterations. However, the tabu list size was set to 50. Due to this, the algorithm took longer than, say, hill climbing due to the overheads in checking and maintaining the tabu list. Again, this issue will be addressed in section 8.5.

The memetic algorithm is a genetic algorithm with a hill climbing local search operator. In order to find a suitable mix of a global search strategy (GA) and a local search strategy (HC), several test runs were carried out which varied the parameters of the two algorithms, but maintained the overall number of iterations at about 7000. The results of these tests are shown in table 8.6

Parameters	Avg. Eval.
GA : PopSize = 10, Gens = 20 , HC : Iters = 7, NH Size = 5	112.07
GA : PopSize = 20, Gens = 20 , HC : Iters = 5, NH Size = 4	128.89
GA : PopSize = 20, Gens = 20 , HC : Iters = 20, NH Size = 1	149.76
GA : PopSize = 20, Gens = 20 , HC : Iters = 20, NH Size = 1	157.64
GA : PopSize = 8, Gens = 16 , HC : Iters = 15, NH Size = 4	166.11
GA : PopSize = 8, Gens = 16 , HC : Iters = 60, NH Size = 1	186.46

Table 8.6 – Searching for a Good Set of Parameters for the Memetic Algorithm

This was an arbitrary set of tests run on the second set of test data from category 1, therefore, one additional test was carried out to provide some form of justification for using the parameters in the top row of table 8.6. Using the first set of test data from category 1, two more tests were run. The results are shown in table 8.7

Parameters	Avg. Eval.
GA : PopSize = 10, Gens = 20 , HC : Iters = 7, NH Size = 5	47.53
GA : PopSize = 20, Gens = 20 , HC : Iters = 5, NH Size = 4	85.15

Table 8.7 – Confirming the test results from table 8.6

These results indicate that the parameters shown to be the best from table 8.6 are a reasonable selection. They will be used for the tests carried out below.

When running the memetic algorithm, the rotation (or not) decision is the same as for the individual algorithms, that is the mutation operator, if applied rotates a randomly selected polygon through 90°.

The results from the tests, for category 1 data, using the four search algorithms described above are shown in table 8.8. The notation for the problem is as follows

Eh*nnn* is the problem type within the category where problems EH001, EH002 and EH003 are the three problems from category 1 (16 or 17 polygons) and EH004, EH005 and EH006 are the three problems from category 2 (25 polygons). HC, GA, TS, MA describe the search algorithm being used (Hill Climbing, Genetic Algorithm, Tabu Search and Memetic Algorithm) If '(R)' appears at the end of the problem description it indicates that rotation is allowed.

For example, "EH003 TS (R)" is the third problem from category one. Tabu search is the search method being used and rotation is allowed.

"EH005 MA" indicates the second problem from category 2, a memetic algorithm is being used a the search mechanism and no rotation is allowed.

This notation is used throughout this chapter to describe the various problem types.

Problem	Avg. Eval.
EH001 HC	82.64
EH001 GA	111.83
EH001 TS	64.10
EH001 MA	47.53
EH001 (R) HC	141.83
EH001 (R) GA	188.87
EH001 (R) TS	127.98
EH001 (R) MA	123.27
EH002 HC	143.98
EH002 GA	156.71
EH002 TS	125.06
EH002 MA	112.07
EH002 (R) HC	136.42
EH002 (R) GA	134.09
EH002 (R) TS	128.03
EH002 (R) MA	115.20
EH003 HC	153.57
EH003 GA	152.45
EH003 TS	99.18
EH003 MA	97.93
EH003 (R) HC	160.39
EH003 (R) GA	193.87
EH003 (R) TS	122.65
EH003 (R) MA	116.58

Table 8.8 – Testing Hopper & Turton Data, Category 1

A graphical representation of these results is shown in figure 8.1.

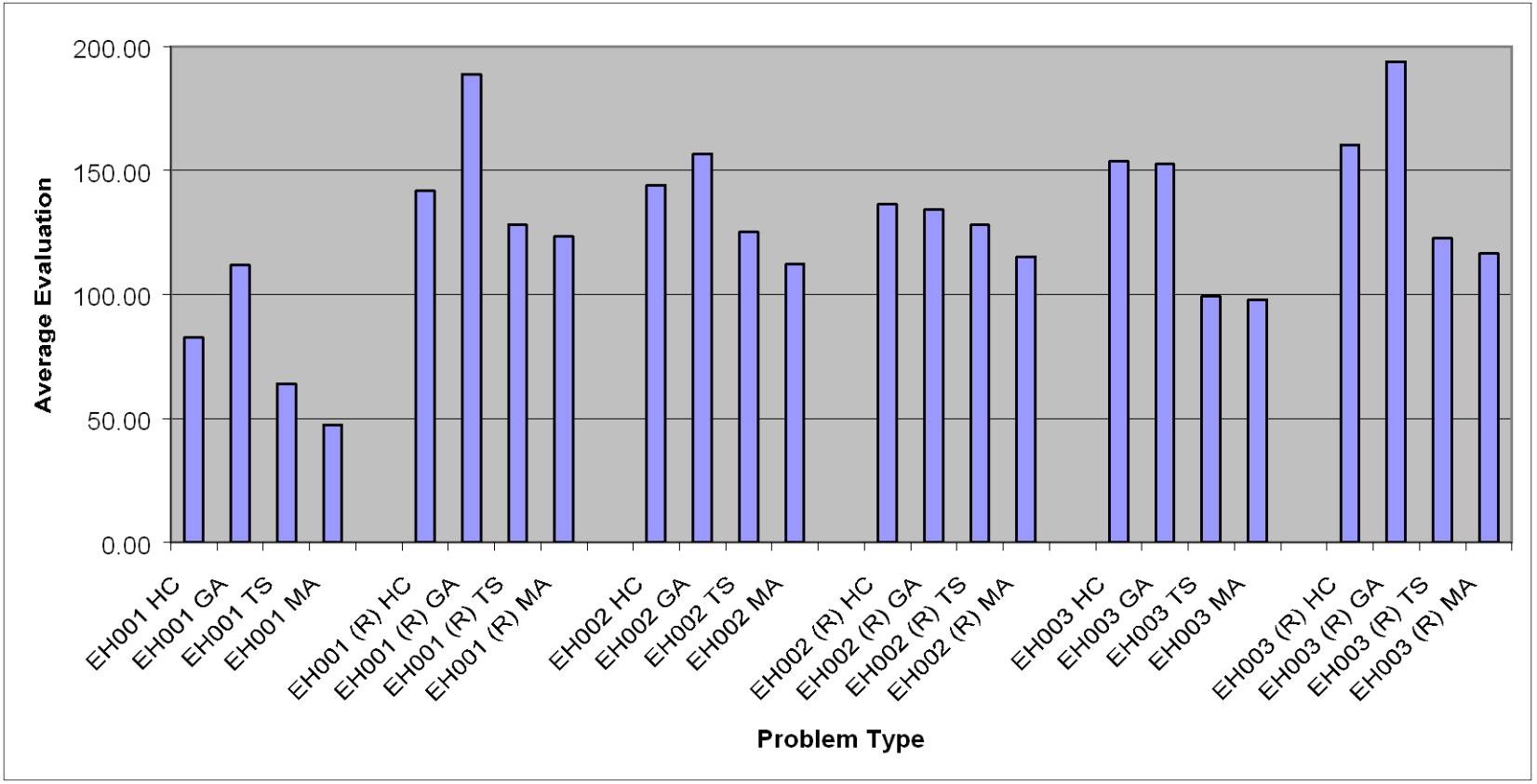


Figure 8.1 – Graphical Representation of Table 8.8

Each group of four bars in figure 8.1 represents one particular problem type. By looking at figure 8.1 or table 8.8 several observations can be made.

Firstly, memetic algorithms perform better in all cases. This is encouraging as it agrees with the results obtained previously in this thesis. Secondly, tabu search performs better than both genetic algorithms and hill climbing. Again, this is in agreement with previous results in this thesis.

It is disappointing to note that the genetic algorithm was frequently out performed by the hill climbing algorithm. This is both disappointing and unexpected. This is not necessarily due to the fact that a genetic algorithm is a poor search strategy. Instead it is a further indication that this algorithm is sensitive to the parameter values, as has already been indicated by the work presented in section 3.4.2 to find good value for GA parameters and in the conclusions drawn in section 5.6 of this thesis.

The same testing was carried out on the category 2 data of Hopper and Turton. The results are shown in table 8.9 and figure 8.2.

Problem	Avg. Eval.
EH004 HC	277.78
EH004 GA	261.53
EH004 TS	221.44
EH005 MA	220.76
EH004 (R) HC	277.78
EH004 (R) GA	268.76
EH004 (R) TS	244.91
EH005 (R) MA	174.00
EH005 HC	180.22
EH005 GA	221.28
EH005 TS	129.58
EH005 MA	149.90
EH005 (R) HC	240.27
EH005 (R) GA	185.60
EH005 (R) TS	166.28
EH005 (R) MA	196.58
EH006 HC	274.44
EH006 GA	130.89
EH006 TS	83.36
EH006 MA	63.45
EH006 (R) HC	222.03
EH006 (R) GA	240.92
EH006 (R) TS	194.16
EH006 (R) MA	180.01

Table 8.9 – Testing Hopper & Turton Data, Category 2

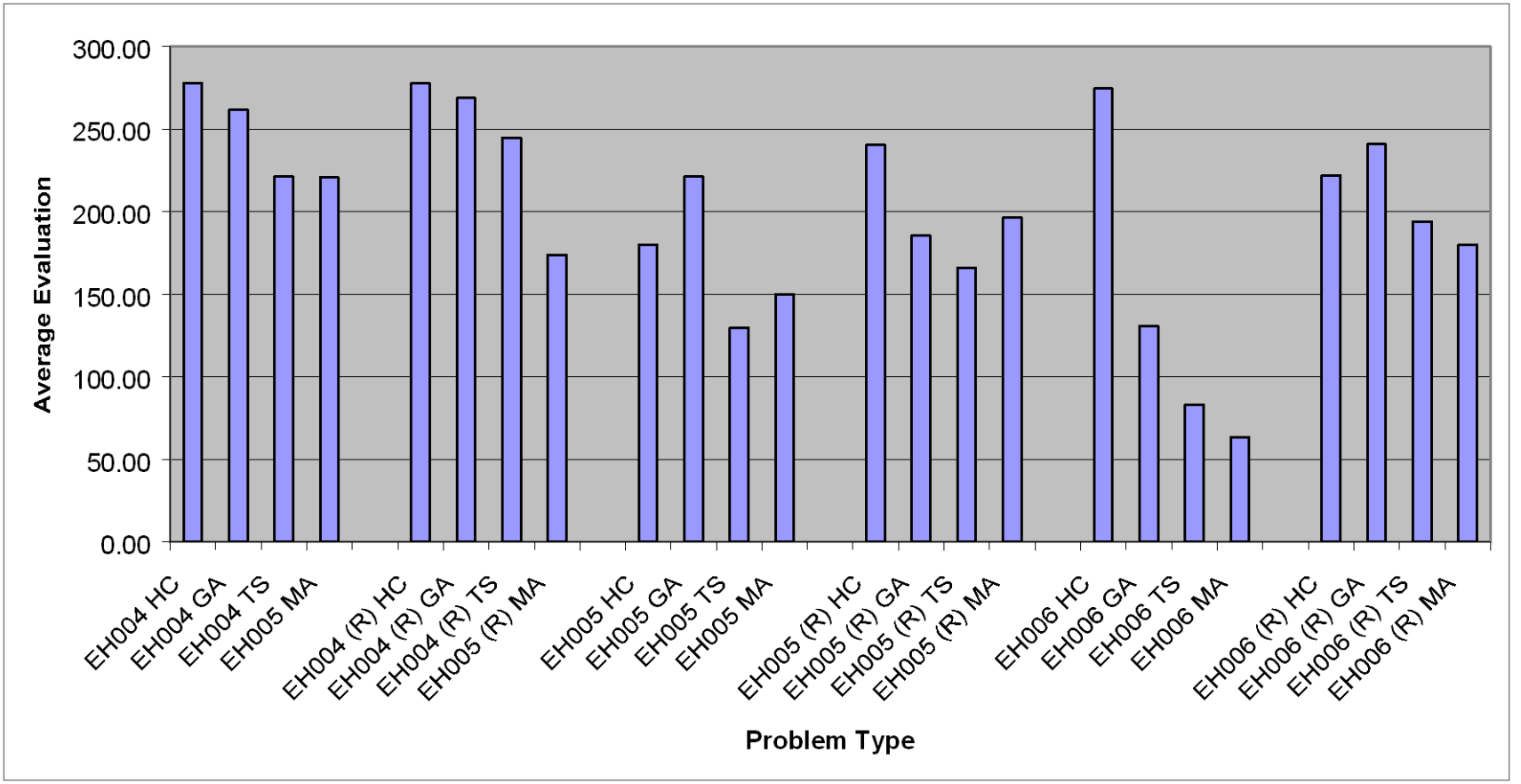


Figure 8.2 – Graphical Representation of Table 8.9

Again, tabu search and the memetic algorithm performed better than the other search algorithms. In fact, the memetic algorithm produced the best quality solutions in all but two of the problems (EH005 and EH005 (R)).

8.4.2 Comparison over fixed time interval

In the experiments of 8.4.1 the number of iterations was fixed. This, it could be argued, leads to an unfair comparison as each algorithm runs for a different amount of time. In fact, this is the case, for a number of reasons. A genetic algorithm (GA), for example, only has to call the evaluation function if the probability of crossover or mutation is exceeded. If not, then no action is required. If the GA does not call the evaluation function, is this regarded as an iteration or not? There are arguments for and against. Assuming the evaluation function is not called, it should not be ignored as an iteration. The GA is designed with this principle in mind. The same parent survives to the next generation, which could, in itself, be a good thing. However, it could also be argued that an iteration should only be counted if the evaluation function is called. This argument has some substance but there is still an overhead to the algorithm (e.g. calling the random number generator) which unjustly penalises the algorithm.

Another factor which gives support for not measuring the algorithms using the number of iterations is that the cache can have a significant effect as to whether the evaluation function should be called. If many items are found in the cache, then more iterations will be carried out.

Given these arguments a better comparison would be to allow the algorithms to execute for a specified time before terminating. For the results we show below each algorithm was allowed to run for 300 seconds. This figure was chosen as it is a good compromise between allowing the algorithms to find a good solution but not allowing it to run for so long, so that a real world user would find the search time unacceptable. These tests were run on an AMD 650 and, as before, all results are averaged over ten runs.

The parameters for the algorithms were the same as for those in 8.4.1. In addition, simulated annealing, ant algorithms and a memetic algorithm with an ant algorithm as the primary search mechanism were also tested.

The parameters for the ant algorithm were the same as those used in chapter 6 (see section 6.4.1). For the ant based memetic algorithm, the same ant algorithm parameters were used and the same hill climbing parameters were used as for the genetic based memetic algorithm.

The simulated annealing cooling schedule was a little more difficult to determine. The problem arises as, ideally, you need the algorithm to terminate just as it reaches its 300 second limit. If, for example, the cooling schedule was only half way through completion, we would expect a sub-optimal solution as the algorithm had not yet started to converge. To ascertain a starting temperature 500 iterations of the algorithm were run with an extremely high temperature (so that the majority, if not all, the solutions were accepted – in effect a random search). Over the 500 iterations the average change in any increase in the evaluation function was calculated. This was used to set the starting temperature so that, given the

average increase in the evaluation function, approximately 60% of worse solutions would be accepted. This is based on the advice of Dowsland (Dowsland, 1995a) and Rayward-Smith (Rayward-Smith, 1996). The cooling schedule was then set so that it would terminate when the temperature was as near to zero as possible. This resulted in the following cooling schedules.

Category Test Data 1

Starting Temperature	=	500
Final Temperature	=	0
Iterations at Each Temperature	=	500
Temperature Decrement	=	20

Category Test Data 2

Starting Temperature	=	600
Final Temperature	=	0
Iterations at Each Temperature	=	250
Temperature Decrement	=	25

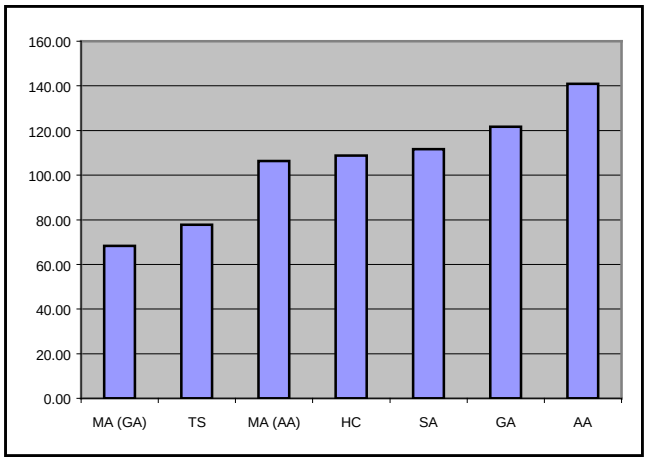
The reason that the different categories of problem require a different number of iterations is due to the fact that category 2 problems contain more shapes than category 1 problems, so more time is spent in the evaluation function.

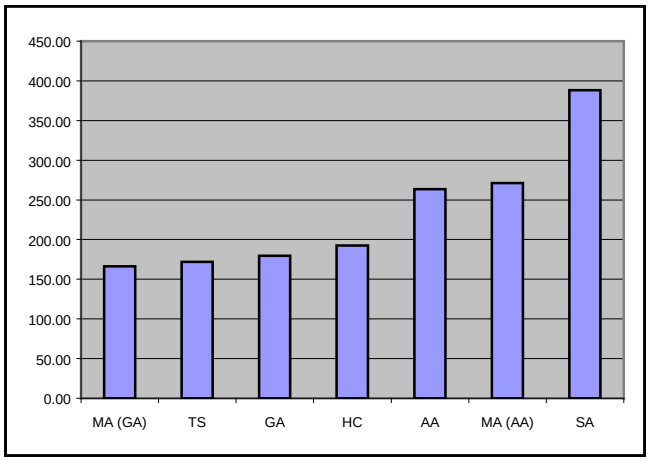
The algorithms are run against each set of data twice, once fixing the orientation of the shapes and once allowing rotation. Only the ant algorithm does not allow the shapes to rotate. This is due to the fact that rotation is performed by a mutation operator and the ant algorithm, as implemented, has no such operator. The ant based memetic algorithm does allow rotation as this can be performed through the mutation operator within the local search (hill climbing) operator.

The results for category 1 data are shown in table 8.10 and figures 8.3 and 8.4. The problem type represents the six problems in category 1 as defined by Hopper and Turton. The search methods are AA (ant algorithm), GA (Genetic Algorithm), HC (Hill Climbing), MA(AA) (ant based memetic algorithm, MA(GA) (genetic based memetic algorithm), SA (simulated annealing) and TS (tabu search). Eval. is the averaged evaluation over 10 runs of the algorithm. The results are sorted in ascending order of the evaluation function with each problem type.

Problem Type	Search Method	Eval.	Problem Type	Search Method	Eval.
EH001	MA (GA)	68.26	**EH001 (R)**	MA (GA)	143.11
	TS	77.81		MA (AA)	146.69
	MA (AA)	106.32		TS	158.54
	HC	108.72		GA	165.11
	SA	111.63		SA	167.21
	GA	121.73		HC	171.95
	AA	140.92		AA	N/A
EH002	MA (GA)	166.46	**EH002 (R)**	TS	184.08
	TS	171.75		MA (GA)	188.47
	GA	179.64		HC	202.85
	HC	192.56		MA (AA)	215.32
	AA	263.55		GA	216.64
	MA (AA)	271.40		SA	255.36
	SA	388.13		AA	N/A
EH003	MA (GA)	106.44	**EH003 (R)**	MA (AA)	159.26
	TS	108.14		MA (GA)	160.01
	GA	131.89		TS	166.28
	SA	154.65		HC	177.11
	AA	161.89		GA	218.06
	MA (AA)	172.11		SA	218.55
	HC	179.45		AA	N/A

Table 8.10 – Testing Hopper & Turton Data, Category 1 for 300 seconds





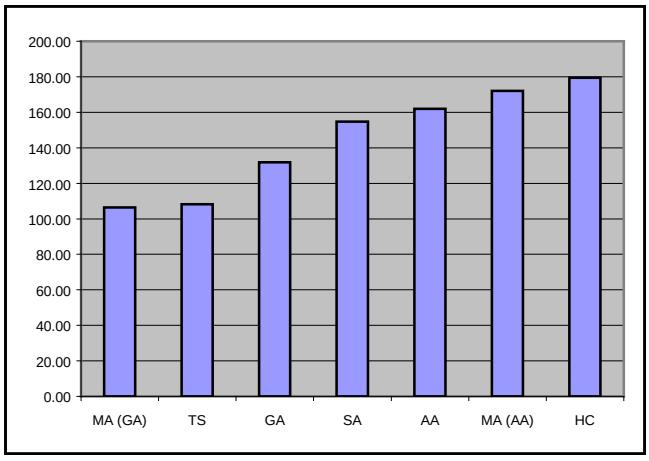
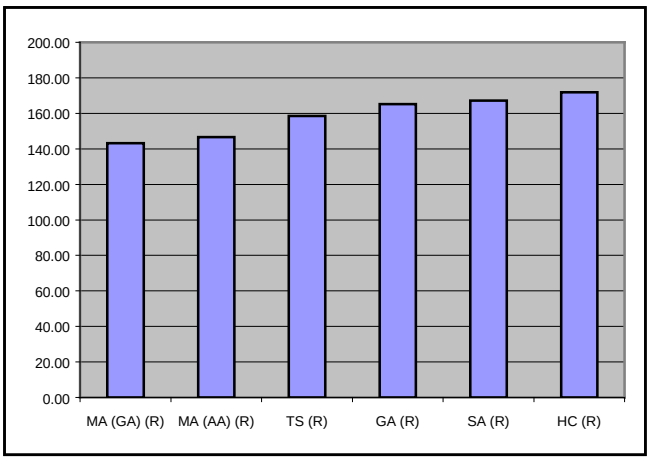
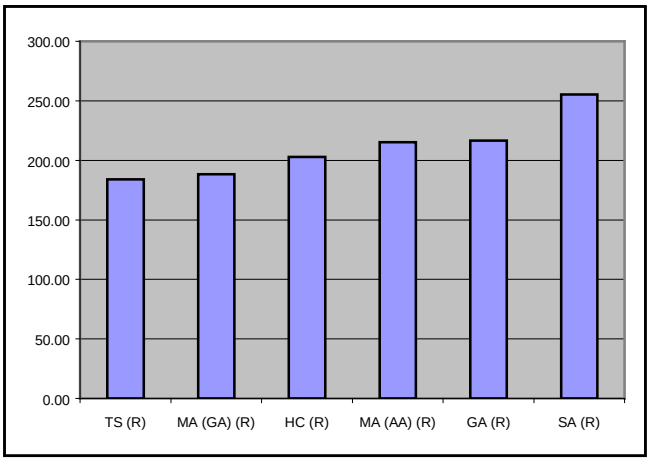


Figure 8.3 – Category 1 Problems – EH001/002/003 No Rotation





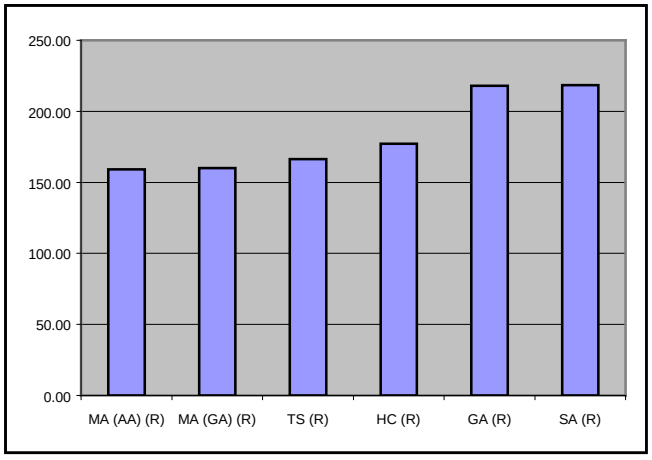


Figure 8.4 – Category 1 Problems – EH001/002/003 Rotation

Figure 8.2 and 8.3 presents the same data as table 8.10 but in a graphical format. It shows that in the majority of cases either the genetic based memetic algorithm or tabu search found the best quality solutions. In fact, with only two exceptions the genetic based memetic algorithm and tabu search were the search strategies which found the two best quality solutions.

An ant based memetic algorithm found the second best solution in EH001 (R), but with the genetic based memetic algorithm and tabu search still performing well.

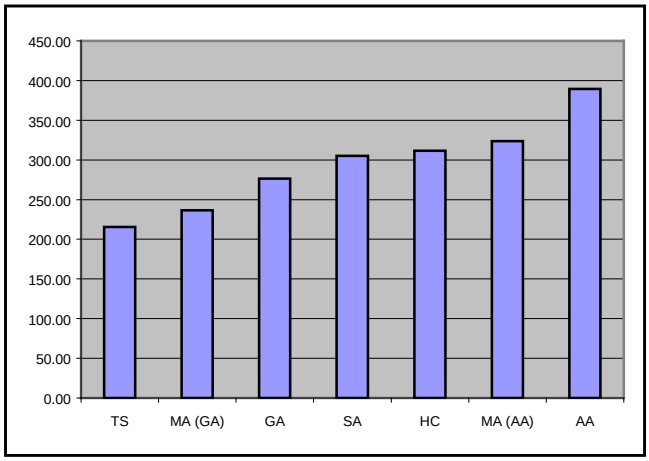
For EH003 (R) (bottom of figure 8.4) The ant based memetic algorithm found the best quality solution, but with the genetic based ant algorithm and tabu search having similar evaluation values.

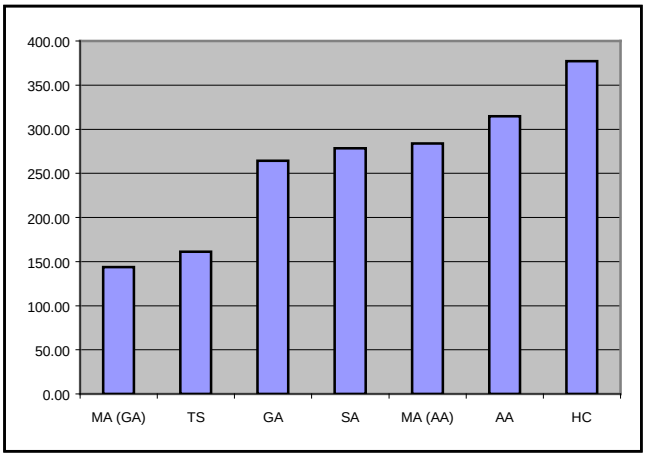
Apart from the ant based memetic algorithm performing well in EH001 (R) and EH003 (R), it did not perform particularly well for the other problems. In fact, the other search algorithms also performed poorly across all the problems.

Table 8.10 and figures 8.5 and 8.6 show the test results from the second category of test data. Similar conclusions can be drawn from these results in that the genetic based memetic algorithm and the tabu search, without exception, produce the best quality solutions. It is difficult to draw conclusions from the results for the other search algorithms. It is disappointing to note that ant algorithms always seem to perform badly, as does simulated annealing. The remainder sometimes perform badly and other times perform reasonably well. Yet, none of the search methods can beat the genetic based memetic ant algorithm or tabu search.

Problem Type	Search Method	Eval.	Problem Type	Search Method	Eval.
EH004	TS	215.38	**EH004 (R)**	TS	194.16
	MA (GA)	236.44		MA (GA)	222.03
	GA	276.42		MA (AA)	373.19
	SA	305.07		HC	413.40
	HC	311.44		GA	449.89
	MA (AA)	323.48		SA	571.42
	AA	389.60		AA	N/A
EH005	MA (GA)	143.88	**EH005 (R)**	MA (GA)	219.03
	TS	161.04		TS	244.37
	GA	264.11		GA	376.44
	SA	278.29		HC	377.04
	MA (AA)	283.74		MA (AA)	379.70
	AA	314.74		SA	554.22
	HC	377.04		AA	N/A
EH006	TS	108.31	**EH006 (R)**	TS	196.72
	MA (GA)	123.05		MA (GA)	243.07
	MA (AA)	206.73		MA (AA)	342.36
	GA	255.92		HC	381.28
	HC	261.90		GA	460.12
	SA	281.92		SA	535.06
	AA	352.63		AA	N/A

Table 8.10 – Testing Hopper & Turton Data, Category 2 data for 300 seconds





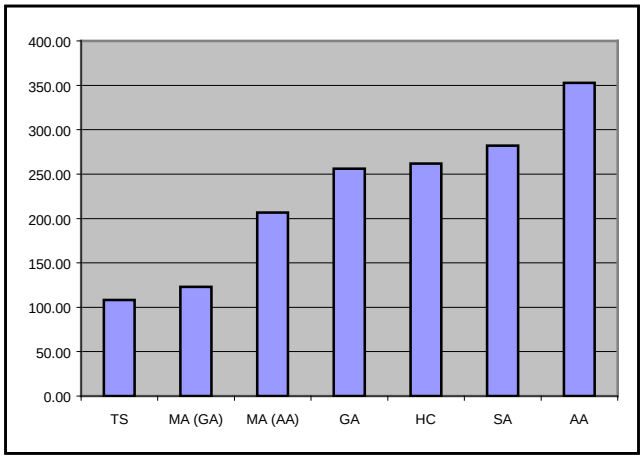
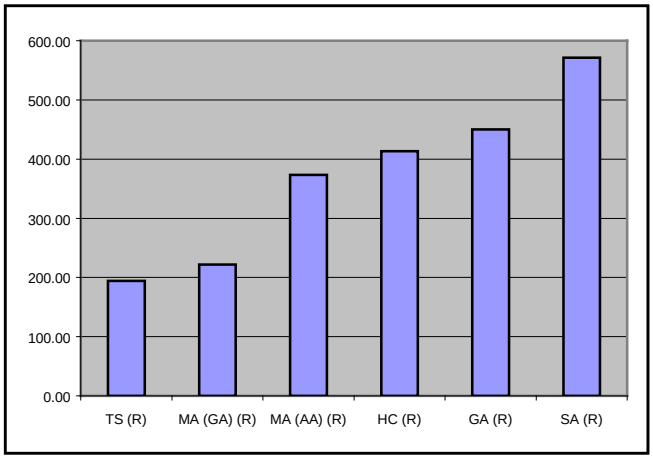
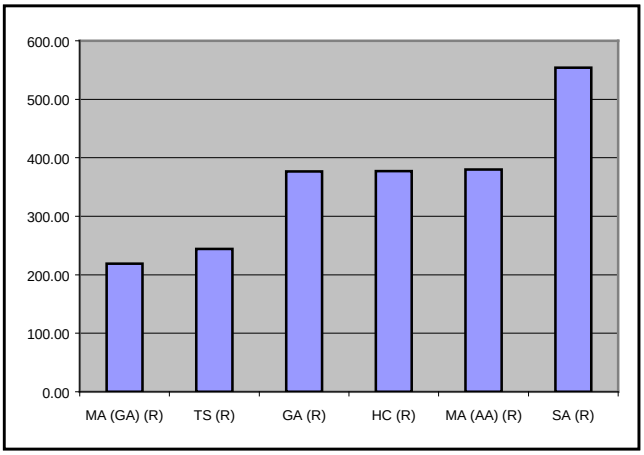


Figure 8.5 – Category 2 Problems – EH004/005/006 No Rotation





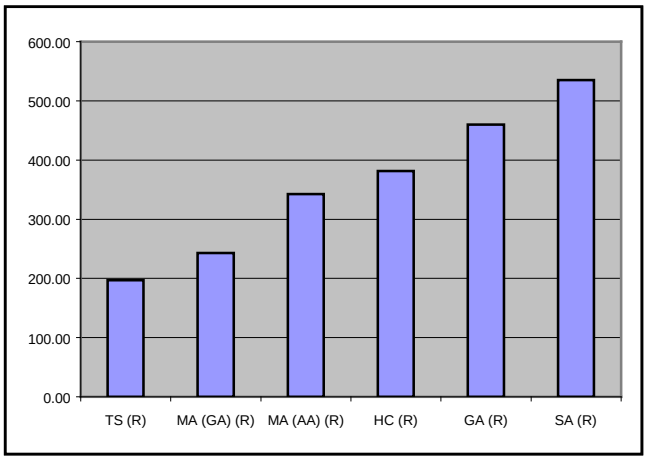


Figure 8.6 – Category 2 Problems – EH004/005/006 Rotation

In Hopper and Turton (Hopper, 2000a) the results from various search algorithms, both heuristic and meta-heuristic are presented. The figures are shown as the best solution expressed as a percentage above the optimal solution. These figures are re-produced in table 8.11. The top part of the table shows the results for a purely heuristic approach where BL is a bottom left heuristic and BLF is also a bottom left heuristic but with the ability to fill holes. DH and DW assumes the shapes have been pre-sorted by height and width respectively.

The middle part of the table shows the results from the hybridisation of the bottom left heuristic with various meta-heuristic search algorithms (genetic algorithm (GA), naïve evolution (NE), simulated annealing (SA), hill climbing (HC) and random search(RS)).

The final part of the table shows the results from using the bottom left with fill heuristic with the same meta-heuristic search algorithms.

	Category 1 %	Category 2 %
BL	25	39
BL-DH	17	68
BL-DW	18	31
BLF	14	20
BLF-DH	11	42
BLF-DW	11	12
GA+BL	6	10
NE+BL	6	8
SA+BL	4	7
HC+BL	9	18
RS+BL	6	14
GA+BLF	4	7
NE+BLF	5	7
SA+BLF	4	6
HC+BLF	7	10
RS+BLF	5	8

Table 8.11 – Results from (Hopper, 2000a), Expressed as % Over Optimal

Table 8.12 shows the best results we achieved for category 1 problems.

	% Over Optimal
EH001	0
EH002	10
EH003	0
EH001 (R)	5
EH002 (R)	5
EH003 (R)	10
Average	**5.00**

Table 8.12 – Best Solutions from the work conducted during this thesis for category 1 problems

It can be seen that we are competitive with the results from table 8.11. Two of our optimal results are presented in figure 8.7.

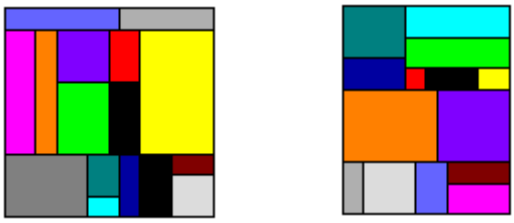


Figure 8.7 – Optimal Solutions for problems EH001 and EH003

For the larger problem (25 rectangles) our solutions average 11% over optimal (see table 8.13). Again, this is competitive with the solutions shown by Hopper.

	% Over Optimal
EH004	13
EH005	7
EH006	7
EH004 (R)	13
EH005 (R)	13
EH006 (R)	13
Average	**11.00**

Table 8.13 – Best Solutions from the work conducted during this thesis for category 2 problems

8.5 Blazewicz, 1993 Data

Oliveira (Oliveira, 1998) presents a new algorithm (TOPOS) for nesting problems. One of the test problems they consider comes from (Blazewicz, 1993). This problem has 28 shapes of seven different types. The shapes are to be packed into a

bin of width 15. The aim, as usual, is to minimise the bin height. The data is shown in appendix E.

The algorithm used by Oliveira (TOPOS) utilises the no fit polygon. It also has a local search/heuristic algorithm and various measures (waste, overlap and distance) that are used in the evaluation function. In addition they have some pre-processing procedures that sorts the shapes in various ways (e.g. decreasing length, decreasing area, decreasing concavity etc.).

Due to the domain knowledge, heuristics and pre-processing built into their system we cannot hope to compete with their solution (which has a bin height of 28.9, which improves on the previous best by 3.5%). In addition, the authors of this paper have conducted many experiments (126 variants of the algorithm) to help find this solution and state that there are still open questions which could make the algorithm produce even better results.

It is interesting to see how our algorithm performs on this problem, using one of the best search strategies found so far. We decided to use tabu search as this has minimal parameters and the values for those parameters have shown not to be too sensitive to the problem instance. The parameters were set to the same values we have used throughout this chapter (list size = 50, neighbourhood size = 20). Rotation, via the mutation operator was allowed with a probability of 0.5. The algorithm was allowed to run for 300 seconds.

Two of the best solutions are shown in figure 8.8. These both have bin heights of 38. Although this is much greater than the Oliviera solution it is encouraging to note that tabu search finds solutions that look reasonable. Even more encouraging is the fact tabu search can be applied with little regard for the parameter values. If we tried to use some of the other algorithms it would take many test runs to ascertain the values for the parameters. Even then there would be no guarantee that the optimum set of parameters had been found.

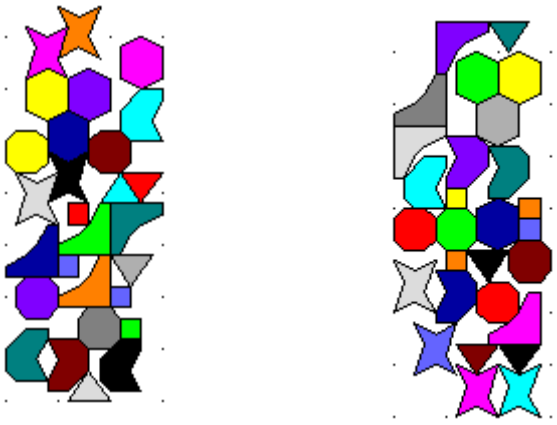


Figure 8.8 – Two Solutions from Blazewicz, 1993 Data

Although, tabu search did does not compete with the solution found by Oliviera the authors are encouraged that a relatively simply search strategy can produce a reasonable solution, using no domain knowledge or pre-processing.

8.6 Summary and Discussion

This chapter of the thesis has shown that the ideas we have presented throughout this thesis produce competitive results when applied to test data from the literature. Using the no fit polygon would lead to excessive execution times. However, we have shown that it is possible to approximate the no fit polygon without loss of solution quality, In fact, better solutions can be found as more of the search space can be explored.

Although not tackled in this thesis the majority of packing problems are likely to benefit from some form of pre-processing. Taking the problem shown in section 8.5, there are seven different shapes. Obviously, if some of these shapes are packed prior to employing the search strategy, then there will be less shapes to pack. For example, rectangles (and squares) with a common dimension could be pre-packed. Similarly, triangles with identical dimensions can be tightly packed if one of them is first rotated through 180°. The author believes that pre-processing of shapes will lead to even better quality solutions and in shorter times. However, deciding which shapes to pack together, and to what extent the shapes should be pre-packed, is a difficult problem in itself but the author would like to make this one of the streams of the future research that is planned (see section 9.3).

9. Conclusions and Discussion

9.1 Contribution

This thesis has contributed in four main areas which are summarised in the following four sections.

9.1.1 Speeding up Evaluations

When using meta-heuristic and evolutionary search strategies, the evaluation function is called at each iteration of the algorithm. If the evaluation function is computationally expensive (which is often the case) it can lead to a bottleneck in the algorithm. In this thesis, we have shown methods that speed up the algorithm. The proposed enhancements are both intuitive and may have been implemented in other work, but they have not (to the author's knowledge) been reported in the literature.

Our algorithm is speeded up by using a cache to store previously seen, partial and complete, solutions. If, when a solution is being evaluated, it is found in the cache then the evaluation function is not called. Instead, the value from the cache is retrieved and used. In chapter 4 of this thesis we have shown that by varying the size of the cache (and thus the number of solutions that can be stored) the execution time of the algorithm can be varied. However, due to the nature of the problem under consideration, there are circumstances where a partial solution stored in the cache will lead to a sub-optimal solution later in the search. Examples of this occurring are given in chapter 4.

To resolve this problem a re-evaluation parameter has been introduced. This parameter takes a value between zero and one, which is used to calculate a probability of using the data stored in the cache, if it exists. If this parameter is set to zero the cached data will always be used. If the parameter is set to one it is equivalent to setting the cache size to zero as the cached data will never be accessed. The higher the value of this parameter the less use of the cache will be made. However, if the value is too low then a potentially sub-optimal solution could be retrieved from the cache, which could lead to a reduction in the quality of the solutions found.

The results in chapter 4 show that the re-evaluation parameter can be set to a low value (0.1), without affecting solution quality.

These techniques, in the opinion of the author, are an important aspect in developing meta-heuristics and evolutionary algorithms. As well as being reported in this thesis, this work was also presented at the CIE 26 (Burke, 1999d) and we hope that this will persuade other researchers to use and report on these techniques.

9.1.2 Development of the No Fit Polygon Algorithm for Non-Convex Polygons

A large part of this thesis has been the investigation and development of the no fit polygon (NFP). The NFP for convex shapes is simple to understand and the algorithm is well known (Cunninghame-Green, 1992). However, the non-convex case, although simple to understand, is difficult to implement. In chapter 7 of this thesis the work of Mahadevan (Mahadevan, 1984) is developed so that the NFP

polygon algorithm presented in his PhD thesis is more robust in that it deals with more degenerate cases. In resolving these degenerate cases, algorithms have been presented which, for example, deal with intersection when it occurs through vertices rather than through edges. All the enhancements made during for this thesis have made use of D-functions. These are primitive operations which allow the algorithm designer to build many more complex operations.

The changes made to the NFP algorithm appear robust and allowed us (in chapter 8) to work with non-convex pieces.

It is well known that there are two algorithms for calculating the NFP for non-convex polygons. The approach used in this thesis is based on the *orbiting* principle. Another approach uses the concept of minkowski sums.

During the course of this work it was discovered that another researcher, Julia Bennell et. al. (Bennell, 2001) was also working on a new NFP algorithm based on minkowski sums.

Although (in one sense) it is beneficial to have researchers working independently on a given problem, for this research, it would have been helpful to have access to a no fit polygon algorithm at that start of this research. In many ways, it is unfortunate that both Bennell and Kendall were unaware of each others work until both pieces were almost completed.

9.1.3 Applying Ant Algorithms to the Stock Cutting Problem

In 1996, Dorigo et. al. (Dorigo, 1996) applied a new type of algorithm to the travelling salesman problem (TSP). The so called ant algorithms have since been

applied to various problems including vehicle routing and the quadratic assignment problem.

This thesis (for the first time) has investigated ant algorithms for the nesting problem. Using Dorigo's ant system as a model the nesting problem is described as a set of polygons that are nested in the order presented. The ants, in deciding the order in which to *visit* the polygons use a measure as to how well the polygons fit together as a *visibility* measure. Apart from this one change the algorithm is based on the standard ant algorithm approach presented in Dorigo's seminal paper.

The experiments conducted in this thesis found the same values for the control variables for the algorithm as those found by Dorigo in his 1996 work. It is these values that were used in the remainder of this thesis.

As well as implementing ant algorithms, we have also applied a local search operator to each permutation of polygons found by the ants in the hope that moving to a local optimum each time will lead to better quality results overall. This gives rise to a type of memetic algorithm (i.e. an evolutionary algorithm with a local search operator).

Initial results from this thesis show that ant algorithms, whilst not competing with tabu search or a genetic based memetic algorithm, are competitive with a genetic algorithm, which is another population based approach. Later results are not so conclusive and the memetic based ant algorithm does not show promising results, something that is commented upon below (see 9.2).

9.1.4 Search Strategies

In completing this work we have attempted to investigate and develop the latest meta-heuristic algorithms to ascertain which ones produce the best quality solutions. To allow this it was necessary to choose a representation that all the search algorithms were able to use. The choice of representation was discussed at the start of chapter three of this work. It is recognised that other representations are available but the author still feels that the list based method that was used was the most suitable for this work.

One of the major problems in implementing these algorithms is choosing suitable values for the various control parameters for the given algorithm. A large amount of time was spent trying to find suitable parameters for some of the algorithms (see chapters 3 and 6). However, this aspect of implementing meta-heuristic algorithms is an up and coming research area and, at present, the best we can do is to make educated guesses at the parameters and run a number of experiments to see how they perform. The author believes that he has acquired suitable parameters for the problems presented in this thesis, but these values may not perform well across a wide range of problems.

9.2 Discussion

This work has applied a range of meta-heuristic algorithms to a series of stock cutting problems. The obvious conclusion from the results is that a genetic based memetic algorithm and tabu search are the search strategies of choice for this problem domain. However, it would be a mistake to draw this conclusion. The

conclusion that should be drawn is that it is still extremely difficult to derive parameters for algorithms so that those algorithms work well across a whole range of problems. Take, for example, ant algorithms. When they were applied to a problem in chapter 6 they appeared to show a certain degree of promise. However, a large part of chapter 6 was devoted to finding suitable parameters for the algorithm and, eventually, the algorithm was able to compete favourably with another population based approach, a genetic algorithm.

Chapter 8 was approached with a certain degree of optimism for ant algorithms but they simply failed to perform, using the parameter values that had pain stakingly been found in chapter 6. The author has no doubts that he could have searched for a set of suitable parameters once again that would have made ant algorithms an effective search strategy. That is, until another problem instance was presented.

Simulated annealing and genetic algorithms have similar problems in that the number of parameters that need addressing is not only critical to the successful operation of the algorithm, but the values must change with each problem instance. Hill climbing suffers from the obvious downside in that it gets stuck in local optima.

Tabu search stands alone as the only algorithm that is capable of escaping from local optima and has the benefit of having very few parameters that have to be adjusted (only the list size and the neighbourhood size). It is also noticeable that the parameter values supplied to tabu search do not seem too critical to the successful running of the algorithm. Throughout this thesis, tabu search has used

the same parameter set. Despite being used to solve different instances of stock cutting problems the algorithm has always performed well.

It is a little surprising that a genetic based memetic algorithm performs as well as it does, whereas a genetic algorithm and hill climbing do not perform well in isolation. The success of the memetic algorithm is probably due to the fact that the genetic aspect of the algorithm is providing a broad search across the search landscape and the hill climbing operator allows any one area to be explored more fully. Therefore, the algorithm is able to explore large areas of the landscape with the GA and then exploit those areas with the hill climbing operator.

Why, then, does an ant based memetic algorithm not perform as well as the genetic based memetic algorithm? If you consider the interaction between the two search processes that comprise a memetic algorithm it may provide an insight as to why the ant based version fails to live up to initial expectations. The genetic based version uses information contained in the chromosome itself (i.e. the ordering of the genes) to produce a new generation. The local search operator has the task of improving this permutation, which is passed back to the genetic algorithm as a (potentially) fitter chromosome than existed before.

An ant algorithm works very differently. The knowledge as to whether the current solution is a good one is contained, not only in the ordering of the polygons, but also in the pheromone levels that are held independently and not passed to the local search operator. When the solution is passed back to the ant algorithm by the

local search operator, it may represent a good solution, but the pheromone levels will not have been updated to reflect this. Therefore, although the ordering is now better the ant cannot exploit this information as it still uses the pheromone levels from before the local search was executed.

For the ant based memetic algorithm there is no feedback between the ant algorithm and the local search operator. For the genetic based version this feedback happens automatically due to the fact that both search strategies operate directly on the chromosome. With the ant version there is additional information, by way of pheromone trails, which the local search operator has no knowledge about.

In summary, this work has shown that tabu search appears to be an effective search strategy for stock cutting problems. However, it does not mean that the other algorithms should be discounted. Rather, future work needs to concentrate on setting suitable parameter values. This is discussed further below, in section 9.3.

9.3 Future Work

9.3.1 Commercial Exploitation

As a result of this work the author has been able to secure two funded awards to allow this research programme to be taken forward.

A teaching company scheme started on the 1st October 2000 and will run for three years. The aim of this project is to exploit stock cutting research and incorporate these ideas into a commercial package. The company, Esprit Automation Ltd.,

already sells a nesting package but realise that to maintain their competitive advantage they must produce more efficient nestings, in less time.

A teaching company associate will be employed by the University of Nottingham for three years and will work at our partner company, Esprit Automation Ltd. The TCS is being funded by both the DTI (Department of Trade and Industry) and Esprit Automation Ltd. The total funding amounts to £126,268.

The author has also secured a CASE for New Academics award. This, again funded by the EPSRC and Esprit Automation Ltd., will allow the work in this thesis to be further developed by a three year PhD studentship. This funding amounts to £42,120. One of the conditions of a CNA award is that the student spends at least three months every year working at the sponsoring company. It is fortunate that the TCS and CNA award are being funded by the same company so the TCS associate and the CNA student will be able to work closely together.

This work has highlighted that particular search strategies work well for certain instances of a problem. However, we are unable to state that a genetic based memetic algorithm and tabu search will work well across all problems, even if the problems were restricted to the domain of stock cutting. In fact, the “no free lunch theorem” (Wolpert, 1997) proves that no one search technique can perform better than any other search technique across all problem domains. Therefore, the

ultimate goal is to be able to find suitable search strategies for the problem domain and problem instance under consideration at that time.

This has recently been recognised in the academic community and the research group to which the author is associated has recently secured an EPSRC grant to look at the concept of *hyper-heuristics*. The author is a co-investigator on this grant.

This research takes the concept of a heuristic to a higher level, in that heuristics are developed which search for other heuristics. Although this is currently "blue sky" research it is hoped that given a particular problem instance the hyper-heuristic will identify the most suitable search strategy to explore the search space for that problem. If the problem changes (even if that is only a small change, for example, reducing the amount of search time available), the hyper-heuristic should be able to adapt to this situation and apply a different search strategy.

As part of this research project, it is hoped to apply the idea of hyper-heuristics to the stock cutting problem. As such, the CNA student will become involved in this project which will ultimately lead to the TCS associate becoming involved. This could lead to the idea of a hyper-heuristic being incorporated into commercial stock cutting software within the next three years.

References

1. Aarts, E., Lenstra, J.K., 1997. Local Search in Combinatorial Optimization. John Wiley & Sons Ltd.
2. Adamowicz, M., Albano, A. 1972. A Two-Stage solution of the cutting-stock problem. Information Processing, Vol. 71, pp 1086-1091.
3. Adamowicz, M., Albano, A. 1976a. A Solution of the Rectangular Cutting-Stock Problem. IEEE Trans. Syst., Man and Cybernetics, SMC-6, pp 302-310
4. Adamowicz, M., Albano, A. 1976b. Nesting Two-Dimensional Shapes in Rectangular Modules. Computer Aided Design, Vol 8, pp 27-33
5. Akinc, U. 1983. An Algorithm for the Knapsack Problem. IIE Trans., Vol 15, pp 31-36
6. Albano, A., Sappupo, G. 1980a. Optimal Allocation of Two-Dimensional Irregular Shapes Using Heuristic Search Methods. IEEE Trans. Syst., Man and Cybernetics, SMC-10, pp 242-248
7. Albano, A., Orsini, R. 1980b. A Heuristic Solution of the Rectangular Cutting Stock Problem. The Computer Journal, Vol 23, pp 338-343
8. Albano, A., Osrini, O. 1978. A Tree Based Search Approach to the M-Partition and Knapsack Problems. The Computer Journal, Vol 23, pp 256-161
9. Albano, A. 1977. A Method to Improve Two-Dimensional Layout. Computer Aided Design, Vol 9, pp 48-52
10. Arbel, A. 1993. Large Scale Optimisation Methods Applied to the Cutting Stock Problem of Irregular Shapes. International Journal of Production Research, Vol 31, Iss 2, pp 483-500
11. Art, R.C. 1966. An Approach to the Two-Dimensional Irregular Cutting Stock Problem. Technical Report 36.008, IBM Cambridge Centre.
12. Bäck, T., Hoffmeister, F. and Schwefel, H-P. (1991). A Survey of Evolution Strategies. Proceedings of the Fourth Conference on Genetic Algorithms (eds Belew, R. and Booker, L.), Morgan Kaufmann Publishers, San Mateo, CA, pp 2-9.
13. Bäck, T., Fogel, D. B. and Michalewicz. 1997. Handbook of Evolutionary Computation. Oxford University Press. ISBN 0 7503 0392 1
14. Baker, B.S., Coffmann, E.G., Rivest, R.L. 1980a. Orthogonal Packing In Two Dimensions. SIAM J. Computing, Vol 9, pp 846-855
15. Baker, B.S., Brown, D.J., Katseff, H.P. 1980b. A 5/4 Algorithm for Two-Dimensional Packing. Journal of Algorithms, Vol.2, No.4, pp 348-368

16. Baker, B.S., Brown, D.J., Katseff, H.P. 1982. Lower Bounds for the Two-Dimensional Packing Algorithm. Acta Informatica, Vol.18, pp 207-225

17. Baker, B.S., Schwarsz, J.S., 1983. Shelf Algorithms for Two-Dimensional Packing Problems. SIAM J. Computing, Vol 12, pp 508-525

18. Barnett, S., Kynch, G.J. 1967. Exact Solution of a Simple Stock Problem. Operations Research, Vol 15, 1051-1056

19. Bartholdi, J.J., Vate, J.H, Zhang, J. 1989. Expected Performance of the Shelf Heuristic for Two Dimensional Packing. Operations Research Letters ORSA Journal, Vol 8, pp 11-16

20. Beasley, J.E. 1985a. An Exact Two-Dimensional Non-Guillotine Cutting Tree Search Procedure. Operations Research, Vol 33, pp 49-64

21. Beasley, J.E. 1985b. Algorithms for Unconstrained Two-Dimensional Guillotine Cutting. Journal of the Operational Research Society, Vol 36, pp 297-306

22. Beasley, D., Bull, D.R. and Martin, R.R. 1993. An Overview of Genetic Algorithms: Part 1, Fundamentals. University Computing. Vol 15, No. 2, pp 58-69

23. Bengtsson, B-E. 1982. Packing Rectangular Pieces – A Heuristic Approach. The Computer Journal, Vol 25, pp 353-357

24. Bennell, J., Dowsland, K. A. 1999. A Tabu Thresholding Implementation for the Irregular Stock Cutting Problem. Int. J. Prod. Res, Vol. 37, No. 8, pp 4259-4275

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

26. Berkey, J.O., Wang, P.Y. 1985 Two-Dimensional Finite Bin-Packing Algorithms. Journal of the Operational Society, Vol 38, pp 423-429

27. Berkey, J.O., Wang, P.Y. 1987. Two_Dimensional Finite Bin-Packing Algorithms. Journal of the Operational Research Society, Vol 38, No. 5, pp 423-429

28. Blazewicz, J., Hawryluk, P., Walkowiak, R. 1993. Using Tabu Search Approach for Solving the Two-Dimensional Irregular Cutting Problem. Tabu Search (eds. Glover, F., Laguna, M., Taillard, E., Werra, D.), Vol. 41 of Annals of Operations Research, Baltzer, J. C. AG.

29. Bonabeau, E., Dorigo, M. and Theraulaz, G. 1999. Swarm Intelligence : From Natural to Artificial Systems. Oxford University Press. ISBN 9 780195 131598

30. Bounsaythip, C., Maouche, S. 1996. A Genetic Approach to the Nesting Problem. Proceedings of the Second Nordic Workshop of Genetic Algorithms, 19-23 Aug, pp 89-104

31. Bremermann, H.J. 1958. The Evolution of Intelligence. The Nervous System as a Model of its Environment. Technical Report No. 1, Contract No. 477(17), Dept. of Mathematics, Univ. of Washington, Seattle.

32. Brooks, R.L., Smith, C.A.B., Stone, A.H., Tutte, W.T. 1940. The Dissection of Rectangles into Squares. Duke Math. J. Vol 7, pp 312-340

33. Brown, A.R. 1971. Optimum Packing and Depletion: The Computer in Space and Resource Usage Problems, New York, London

34. Bullnheimer B., R.F. Hartl and C. Strauss (1999). An Improved Ant system Algorithm for the Vehicle Routing Problem. The Sixth Viennese workshop on Optimal Control, Dynamic Games, Nonlinear Dynamics and Adaptive Systems, Vienna (Austria), May 21-23, 1997, to appear in: Annals of Operations Research (Dawid, Feichtinger and Hartl (eds.): Nonlinear Economic Dynamics and Control, 1999.

35. Burke, K. E. and Kendall, G. (1998). Comparison of Meta-Heuristic Algorithms for Clustering Rectangles", Computers and Industrial Engineering, Vol. 37, Iss. 1-2, pp 383-386 (Proccedings of the 24th International Conference on Computers and Industrial Engineering, Brunel University, September, 1998)

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

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

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

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

40. Cagan, J. 1994. Shape Annealing to the Constrained Geometric Knapsack Problem. Computer-Aided Design, Vol 26, Iss 10, pp 763-770

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

42. Chambers, M.L., Dyson, R.G. 1976. The Cutting Stock Problem in the Flat Glass Industry – Selection of Stock Sizes. Operational Research Quarterly, Vol 27, pp 949-957

43. Chand, D.R., Kapur, S.S. 1970. An algorithm for convex polytopes. JACM vol. 17, iss. 1, pp 78-86

44. Chazelle, B. 1983. The Bottom-Left Bin-Packing Heuristic : An Efficient Algorithm. IEEE Trans. Computers, C32, pp 697-707

45. Christofides, N., Whitlock, C. 1977. An Algorithm for Two_Dimensional Cutting Problems. Operations Research, Vol 25, pp30-44

46. Chung, F., Garey, M., Johnson, D. 1982. On Packing Two-Dimensional Bins. SIAM Journal on Algebraic and Discrete Methods, Vol 3, pp 66-76

47. Coffman, E, Shor, P. 1993. Packing in Two Dimensions : Asymptotic Average-Case Analysis of Algorithms. Algorithmica, Vol 9, pp 253-277

48. Coffman, E, Shor, P. 1990. Average-Case Analysis of Cutting and Packing in Two Dimensions. European Journal of Operations Research, Vol 44, pp 134-144

49. Coffman, Jr, E.G., Lueker, G.S., Rinnooy, A.H.G. 1988. Asympototic Methods in the Probabilistic Analysis of Sequencing and Packing Heuristics. Management Science, Vol 34, pp 266-290

50. Coffman, Jr, E.G., Garey, M.R., Johnson, D.S. 1984a. Approximation Algorithms for Bin-Packing – An Updated Survey. Approximation Algorithms for Computer System Design, Wien, pp 49-106

51. Coffman, E. Gilbert, E. 1984b. Dynamic, First-Fit Packings in Two or more Dimensions. Information and Control, Vol 61, pp 1-14

52. Coffman, Jr, E.G., Garey, M.R., Johnson, D.S., Tarjan, R.E. 1980. Performance Bounds for Level-Oriented Two-Dimensional Packing Algorithms. SIAM J. Comput., Vol 9, pp 808-826

53. Cohen, G.D. 1966. Comments on a Paper by M.L. Wolfson: Selecting the Best Lengths to Stock. Operations Research, Vol 14, p 341.

54. Coley, D.A. 1999. An Introduction to Genetic Algorithms for Scientists and Engineers. World Scientific Publishing Co. Pte. Ltd. ISBN 981-02-3602-6

55. Colorni A., M. Dorigo, V. Maniezzo and M. Trubian (1994). Ant system for Job-shop Scheduling. JORBEL - Belgian Journal of Operations Research, Statistics and Computer Science, 34(1):39-53.

56. Corne, D, Dorigo, M and Glover, F. (eds) 1999. New Ideas in Optimization. McGraw-Hill Publishing Company.

57. Costa D. and A. Hertz (1997). Ants Can Colour Graphs. Journal of the Operational Research Society, 48, 295-305.

58. Coverdale, I.L., Wharton, F. 1976. An Improved Heuristic Procedure for a Nonlinear Cutting Stock Problem. Management Science, Vol 23, 78-86

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

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

61. Dagli, C.H., Poshyanonda, P. 1997. New Approaches to Nesting Rectangular Patterns. Journal of Intelligent Manufacturing, Vol 8, Iss 3, pp 177-190

62. Dagli, C.H., Tatoglu, M.Y. 1987. An Approach to two-dimensional Cutting Stock Problem. International Journal of Production Research, Vol 25, pp175-190

63. Daniels, K. 1995. Containment Algorithms for Nonconvex Polygons with Applications to Layout. PhD Thesis, Center for Research in Computing Technology, Harvard University, Cambridge, Massachusetts.

64. Dantzig, G.B. 1951. Maximization of a Linear Function of Variables Subject to Linear Inequalities. Activity Analysis of Production Allocation. T.C. Koopmans, ed. Cowles Commission Monograph, 13. Wiley, New York, pp339-347

65. Davis, L.D. ed. 1987 Genetic Algorithms and Simulated Annealing. Pitman, London

66. Davis, L.D., ed. 1991. Handbook of Genetic Algorithms. Van Nostrand Reinhold

67. Dawkins, R. 1976. The Selfish Gene, Oxford University Press

68. Daza, V.D., Muñoz, R., Gómes de Alvarenga, A. 1995. A Hybrid Genetic Algorithm for the Two-Dimensional Guillotine Cutting Problem. Evolutionary Algorithms in Management Applications, Springer-Verlag., (Bietahn, J., Nissen, V., eds), pp 183-196

69. De Jong, K. A. 1975. An Analysis of the Behaviour of a Class of Genetic Adaptive Systems (Doctoral Dissertation, University of Michigan). Dissertation Abstracts International 36(10), 5140B (University Microfilms No 76-9381)

70. Di Caro G. & Dorigo M. (1998). AntNet: Distributed Stigmergetic Control for Communications Networks. Journal of Artificial Intelligence Research (JAIR), 9:317-365.

71. Dietrich, R.D., Yakowitz, S.J. 1991. A Rule Based Approach to the Trim-Loss Problem. International Journal of Production Research, Vol 29, No. 2, pp 410-415

72. Dori, D., Ben-Bassat, M. 1984. Efficient Nesting of Congruent Convex Figures. Comm. ACM, Vol 27, pp 228-235

73. Dorigo, M., Vittorio, M., Alberto, C. 1996. Ant System: Optimization by a Colony of Cooperating Agents. IEEE Transactions on Systems, Man and Cybernetics – Part B : Cybernetics, Vol. 26, No.1, February 1996, pp 29-41

74. Dorigo M. and G. Di Caro (1999). The Ant Colony Optimization Meta-Heuristic. In D. Corne, M. Dorigo and F. Glover, editors, New Ideas in Optimization, McGraw-Hill.

75. Dorigo M., G. Di Caro & L. M. Gambardella (1999a). Ant Algorithms for Discrete Optimization. Artificial Life, 5(2), in press.

76. Dowsland, W.B. 1985. Two and Three Dimensional Packing Problems and Solution Methods. New Zealand Journal of Operational Research, Vol 13, pp 1-18

77. Dowsland W.B. 1991. Three-Dimensional Packing Problems and Solution Methods. International Journal of Production Research, 13, 1-18

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

79. Dowsland, K., Dowsland, W. 1993. Heuristic Approaches to Irregular Cutting Problems. Working Paper EBMS/1993/13, European Business Management School, UC Swansea, UK

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

81. Dowsland, K.A. 1995a. Simulated Annealing. In Modern Heuristic Techniques for Combinatorial Problems (ed. Reeves, C.R.), McGraw-Hill, 1995

82. Dudzinski, K., Walukiewicz, S. 1987. Exact Methods for the Knapsack Problem and its Generalizations. European Journal of Operational Research, Vol 28, pp 3-21

83. Dutâ, L., Fabian, C. 1984. Solving Cutting-Stock Problems Through the Monte-Carlo Method. Econ. Comp. Econ. Cybern. Studies Res., Vol 19, pp35-54

84. Dyckhoff, H. 1981. A New Linear Programming Approach to the Cutting Stock Problem. Operations Research, Vol 29, pp 1092-1104

85. Dyckhoff, H., Kruse, H.J., Abel, D., Gal, T. 1985. Trim Loss and Related Problems. Omega, Vol 13, pp 59-72

86. Dyckhoff, H., Finke, U., Kruse, H.J. 1988. Standard Software for Cutting Stock Management. Essays on Production Theory and Planning, Berlin, pp 209-221

87. Dyckhoff, H., Wäscher, G., (eds). 1990a. Special Issue on Cutting and Packing. European Journal of Operational Research, Vol 44, No. 2

88. Dyckhoff, H. 1990b. A Typology of Cutting and Packing Problems. Euro. Jour. Of Operational research, Vol 44, pp 145-159

89. Dyckhoff, H, Finke, U. 1992. Cutting and Packing in Production and Distribution. Physica-Verlag, Heidelberg, Germany

90. Dyson, R.G., Gregory, A.S. 1974. The Cutting Stock Problem in the Flat Glass Industry. Operational Research Quarterly, Vol 25, pp 41-53

91. Eilon, S., Christofides, N. 1971. The Loading Problem. Management Science, Vol 17, pp259-268

92. Eisemann, K. 1957. The Trim Problem. Management Science, Vol 3, pp 279-284

93. Erlenkotter, D. 1978. A Dual_Based Procedure for Uncapacitated Facility Location, Operations Research, Vol. 26, pp 992-1009

94. Falkenauer, E. 1996. A Hybrid Grouping Genetic Algorithm for Bin Packing. Journal of Heuristics, Vol. 2, No. 1, Kluwer Academic Publishers, pp 5-30

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

96. Fayard, D., Zissimopoulos, V. 1995. An Approximation Algorithm for Solving Unconstrained Two-Dimensional Knapsack Problems. European Journal of Operational Research (Special Issue), Vol 84, pp 618-632

97. Fogel, D.B. (1998) Evolutionary Computation The Fossil Record, IEEE Press, ISBN 0-7803-3481-7

98. Fogel, D.B. (2000a) Evolutionary Computation : Toward a New Philosphy of Machine Intelligence, 2nd Ed., IEEE Press Marketing, ISBN 0-7803-5379-X

99. Fogel, D.B., Anderson, R. W. (2000b). Revisiting Bremermann's Genetic Algorithm: I. Simultaneous Mutation of All Parameters. In proceedings of Congress on Evolutionary Computation 2000 (CEC 2000), La Jolla Marriot Hotel, La Jolla, California, USA, 16-19 July 2000, pp 1204-1209

100. Fogel, D.B., Fraser, A. S. (2000c) Running Races with Fraser's Recombination. In proceedings of Congress on Evolutionary Computation 2000 (CEC 2000), La Jolla Marriot Hotel, La Jolla, California, USA, 16-19 July 2000, pp 1217-1222

101. Forrest, S. 1993. Genetic Algorithms: Principles of Natural Selection Applied to Computation. Science, vol 261, 872-878

102. Forsyth P. and A. Wren (1997). An Ant System for Bus Driver Scheduling. Presented at the 7th International Workshop on Computer-Aided Scheduling of Public Transport, Boston, August 1997.

103. Fowler, R.J., Paterson, M.S., Tanimoto, S.L. 1981. Optimal Packing and Covering in the Plane are NP-Complete. Information Processing Letters, Vol 12, pp 133-137

104. Fraser, A.S. 1957. Simulation of genetic systems by automatic digital computers. II. Effects of linkage on rates under selection. Australian J. of Biol Sci, vol 10, pp 492-499

105. Fraser, A.S. 1960. Simulation of genetic systems by automatic digital computers. IV. Epistatis. Australian J. of Biol Sci, vol 13, pp 329-346

106. Frederickson, G.N. 1980. Probabalistic Analysis for Simple One and Two-Dimensional Bin Packing. Inf. Proc. Lett., Vol 11, pp 156-161

107. Freeman, H., Shapira, R. 1975. Determining the Minimum-Area Encasing Rectangle for an Arbitrary Closed Curve. Communications of the ACM, Vol 18, pp 409-415

108. Garey, M.R. and Johnson. 1979. Computers and Interactability : A guide to the Theory of NP-Completeness. W.H. Freeman and Company, San Francisco

109. Garey, M.R., Johnson, D.S. 1981. Approximation Algorithms for Bin Packing Problems: A Survey. Analysis and Design of Algorithms in Combinatorial Optimization, CISM Courses and Lectures 266, Wien, pp 147-172

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

111. Gilmore, P.C., Gomory, R.E. 1961. A Linear Programming Approach to the Cutting-Stock Problem. Operations Research, vol 9, pp 849-859

112. Gilmore, P.C., Gomory, R.E. 1963. A Linear Programming Approach to the Cutting-Stock Problem, Part II. Operations Research, vol 11, pp 863-888

113. Gilmore, P.C., Gomory, R.E. 1965. Multistage Cutting Stock Problems of Two and More Dimensions. Operations Research, vol 13, pp 94-120

114. Gilmore, P.C., Gomory, R.E. 1966. The Theory and Computation of Knapsack Functions. Operations Research, vol 14, pp 1045-1074

115. Glover, F. 1977. Heuristics for Integer Programming using Surrogate Constraints. Decisions Science, Vol. 8, pp156-166

116. Glover, F. 1989. Tabu Search – Part I. ORSA Journal on Computing, Vol 1, No. 3, pp 190-206

117. Glover, F. 1990. Tabu Search – Part II. ORSA Journal on Computing, Vol 2, No. 1, pp 4-32

118. Glover, F., Laguna, M. 1998. Tabu Search. Kluwer Academic Publishers

119. Golan, I. 1981. Performance Bounds for Orthogonal Oriented Two-Dimensional Packing Algorithms. SIAM Journal of Computing, Vol 10, pp 571-582

120. Goldberg, D.E. 1989. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley

121. Golden, B.L. 1976. Approaches to the Cutting Stock Problem. AIIE Transactions, Vol 8, pp 265-274

122. Golomb, S.W. 1966. Polyominoes. George Allen and Unwin, London.

123. Graham, R.L. 1972. An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set. Information Processing Letters, 1

124. Hart, P.E., Nilsson, N.J., Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on SSC, Vol 4, pp100-107

125. Hart, P.E., Nilsson, N.J., Raphael, B. 1972. Correction to 'A Formal Basis for the Heuristic Determination of Minimum Cost Paths'. SIGART Newsletter, Vol 37, pp28-29

126. Haessler, R.W. 1968. An Application of Heuristic Programming to a Nonlinear Cutting-Stock Problem Occurring in the Paper Industry. Unpublished Doctoral Dissertation, The University of Michigan, Ann Arbor, No. 69-12, 118

127. Haessler, R.W. 1971. A Heuristic Solution to a Nonlinear Cutting Stock Problem. Management Science, Vol 17, B793-B803

128. Haessler, R.W. 1975. Controlling Cutting Pattern Changes in One-Dimensional Trim Problems. Operations Research, Vol 23, pp 483-493

129. Haessler, R.W. 1980. A Note on Some Computational Modifications to the Gilmore-Gomory Cutting Stock Algorithm. Operations Research, vol 28, pp1001-1005

130. Haessler, R.W. and Sweeney, P.E. 1991 Cutting Stock Problemsand Solution Procedures. EJOR, 54, 141-150

131. Hahn, S.G. 1968. On the Optimal Cutting of Defective Sheets. Operations Research, Vol 16, pp 1100-1114

132. Haims, M.J. 1966. On the Optimum Two-Dimensional Allocation Problem. PhD Dissertation, Dept of Elec. Eng., New York Uni., Bronx, Tech. Rept., 400-136

133. Haims, M.J., Freeman, H. 1970. A Multistage Solution of the Template Layout Problem. IEEE Transactions on System, Science and Cybernetics. SSC-6, pp145-151

134. Hearn, D, Baker, P., M. 1994. Computer Graphics. Prentice Hall, New Jersey

135. 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

136. Heistermann, J., Lengauer, T. 1995. The nesting Problem in the Leather Manufacturing Industry. Annals of Operations Research, Vol 57, pp 103-133 (check page numbers – and ref as a whole)

137. Herdy, M. (1991). Application of the Evolution Strategy to Discrete Optimization Problems. Proceedings of the First International Conference on Parallel Problem Solving from Nature (PPSN), Lecture Notes in Computer Science (eds Schwefel, H-P and Männer, R), Springer-Verlag, Vol. 496, pp 188-192

138. Herz, J.C. 1972. A Recursive Computing Procedure for Two-Dimensional Stock Cutting. IBM J. Res. Develop., Vol 16, pp 462-469

139. Hinxman, A.I. 1980. The Trim Loss and Assortment Problems: A Survey. European Journal of Operational Research, Vol 5, pp 8-18

140. Hofri, M. 1980. Two-Dimensional Packing : Expected Performance of Simple Level Algorithms. Information and Control, Vol 45, pp 1-17

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

142. Holland, J.H. 1992. Adaptation in Natural and Artificial Systems. University of Michigan Press (Second Edition: MIT Press, 1992).

143. Hopper E. and Turton B. C. H., 1997. Application of Genetic Algorithms to Packing Problems - A Review. In: Chawdry, P. K., Roy, R. and Kant, R. K. (eds.), Proceedings of the 2nd On-line World Conference on Soft Computing in Engineering Design and Manufacturing, Springer Verlag, London, pp. 279-288

144. Hopper E. and Turton B.C.H. 2000a. An Empirical Investigation of Meta-Heuristics and Heuristic Algorithms or 2D Packing Problem. Accepted for EJOR in June 1999, publication date not yet known.

145. Hopper, E. 2000b. Two-Dimensional Packing utilising Evolutionary Algorithms and other Meta-Heuristic Methods PhD Thesis, Cardiff University, UK

146. Hopper E. and Turton B. C. H. 2000c. A Suite of Benchmark Problems and Problem Generators for 2D Rectangular and Irregular Strip Packing Problems, submitted in April 2000 to the special issue on Cutting and Packing in EJOR

147. Hower, W., Rosendahl, M., Köstner, D. 1996. Evolutionary Algorithm Design. Artificial Intelligence in Design '96, Kluwer Academic Publishers, Netherlands (eds Gero, S., Sudweeks, F.), pp 663-680

148. Israni, S., Sanders, J.L. 1982. Two-Dimensional Cutting Stock Problem Research: A Review and a New Rectangular Layout Algorithm. Journal of Manufacturing Systems, Vol 1, pp 169-182

149. Israni, S., Sanders, J.L. 1985. Performance Testing of Rectangular Parts-Nesting Heuristics. International Journal of Production Research, Vol 23, pp 437-456

150. Jain, P., Fenyes, P., Richter, R. 1992. Optimal Blank Nesting Using Simulated Annealing. Journal of Mechanical Design (Transactions of the ASME), March 1992, Vol 114, pp 160-165

151. Kampe, T. 1988. Simulated Annealing: Use of a New Tool in Bin Packing. Annals of Operations Research, Vol 16, pp 327-332

152. Kantorovich, L.V., Zalgaller, V.A. 1951. Optimal Calculation for Subdivision in the Material Industry. Leningrad. Lenizdat. p 198

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

154. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P. 1983. Optimization by Simulated Annealing. Science, Vol 220, pp 671-680

155. Kröger, B. 1995. Guillotineable Bin Packing : A Genetic Approach. European Journal of Operational Research (Special Issue), Vol 84, pp 645-661

156. Kuntz P., P. Layzell and D. Snyers (1997). A Colony of Ant-like Agents for Partitioning in VLSI Technology. Proceedings of the Fourth European Conference on Artificial Life, P. Husbands and I. Harvey, (Eds.), 417-424, MIT Press.

157. Laszlo, M.J. 1996. Computational Geometry and Computer Graphics in C++. Prentice Hall.

158. Li, Z., Milenkovic, V. 1995. Compaction and Separation Algorithms for Non-Convex Polygons and Their Applications. EJOR (Special Issue) Cutting and Packing, 84, 3, 95, pp 539-561

159. Lin, S. and Kernighan, B. W. 1973. An Effective Heuristic Algorithm for the TSP. Oper. Res. Vol. 21, 498-516.

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

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

162. Man, K.F., Tang, K.S. and Kwong, S. 1999. Genetic Algorithms: Concepts and Design, Springer-Verlag, London.

163. Maniezzo V., Colorni, A. 1999. The Ant System Applied to the Quadratic Assignment Problem. IEEE Transactions on Knowledge and Data Engineering, to appear

164. Martello, S., Toth, P. 1987. Algorithms for Knapsack Problems. Annals of Discrete Mathematics, Vol 31, 213-258

165. Metropolis N., Rosenbluth A. W., Teller A. H., Teller E. 1953. Equation of State Calculation by Fast Computing Machines. J. of Chem Phys. Vol 21, pp1087-1091

166. Michalewicz, Z. 1996. Genetic Algorithms + Data Structures = Evolution Programs (3rd rev. and extended ed.). Springer-Verlag, Berlin

167. Michalewicz, Z and Fogel, D.B. 2000. How To Solve It. Springer-Verlag. ISBN 3-540-66061-5

168. Mileham, A.R., Scott, A.J. 1996. A New Algorithm for Nesting Sheet Metal Components. Proceedings of the twelth conference on CAD/CAM Robotics and Factories of the Future, 14-16 Aug '96, 1048-1053

169. Mitchell, M. 1996. An Introduction to Genetic Algorithms. Massachusetts Institute of Technology

170. Monmarché, N., Venturini, G., Slimane, M. 2000. On how Pachycondyla apicalis ants suggest a new search algorithm. Future Generation Computer Systems, pp 937-946

171. Moscato, P. 1989. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms, Report 826, Caltech Concurrent Computation Program, California Institute of Technology, Pasadena, California, USA

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

173. Oliveira, J. F., Ferreira, J. S. 1990. An Improved Version of Wang's Algorithm fot Two-Dimensional Stock Cutting Problems. European Journal of Operations Research, vol. 44, pp 256-266.

174. Oliveira, J.F., Ferreira, J.S. 1993. Algorithms for Nesting Problems. Applied Simulating Annealing, Lecture Notes in Economics and Mathematical Systems (ed. Vidal, V. V.), Springer-Verlag, pp 255-273

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

176. Otten, R.H.J.M. 1982. Automatic Floorplan Design. ACM-IEEE 19th Design Automation Conference, pp 261-267

177. Parada, V., Sepúlveda, M., Solar, M. (1998). Solution for the Constrained Guillotine Cutting Problem by Simulated Annealing. Computer Ops Res, 25, 37-47

178. Paull, A.E. 1956. Linear Programming: A Key to Optimum Newsprint Production. Pulp Paper Magazine. Canada. Vol 57

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

180. Preparata, F.P., Shamos, M.I., 1985. Computational Geometry: An Introduction. Springer-Verlag

181. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P. 1995. Numerical Recipes in C. Cambridge University Press.

182. Qu, W., Sanders, J.L. 1987. A Nesting Algorithm for Irregular Parts and Factors Affecting Trim Losses. International Journal of Production Research, Vol 25, pp 381-397

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

184. Rana, A., Howe, A.E., Whitley, L.D, Mathias, K. 1996. Comparing Heuristic, Evolutionary and Local Search Approaches to Scheduling. Third Artificial Intelligence Plannings Systems Conference (AIPS-96).

185. Rayward-Smith, V.J., Shing, M.T., 1983. Bin Packing. Bulletin of the IMA, Vol 19, pp 142-146

186. Rayward-Smith V.J., Osman I.H., Reeves C.R., Smith G.D. 1996. Modern Heuristic Search Methods. John Wiley and Sons.

187. Rechenberg, I. 1965. Cybernetic Solution Path of an Experimental Problem. Ministry of Aviation, Royal Aircraft Establishment (UK).

188. Rechenberg, I. 1973. Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution. Frommann-Holzboog (Stuttgart)

189. Reeves, C.R. 1995. Genetic Algorithms. In Modern Heuristic Techniques for Combinatorial Problems (ed. Reeves, C.R.), McGraw-Hill, 1995

190. Rich, E., Knight, K. 1993. Artificial Intelligence. Mc-Graw Hill.

191. Roberst, S.A. 1984. Application of Heuristic Techniques to the Cutting-Stock Problem for Worktops. Journal of the Operational Research Society, Vol 35, pp 369-377

192. Rode, M., Rosenberg, O. 1987. An Analysis of Heuristic Trim Loss Algorithms. Engineering Cost and Production Economics, Vol 12, pp 71-78

193. Ross, P. Corne, D., Hsiao-Lan, F. 1994. Improving Evolutionary Timetabling with Delta Evaluation and Directed Mutation. In Y. Davidor, H-P Schwefel and R. Manner (eds) Parallel Problem Solving in Nature, Vol 3, Springer-Verlag, Berlin

194. Russell, S., Norvig, P. 1995. Artificial Intelligence A Modern Approach. Prentice-Hall

195. Salkin, H.M., de Kluyver, C.A. 1975. The Knapsack Problem: A Survey. Naval Research Logistics Quarterly, Vol 22, pp 127-144

196. Sarin, S.C. 1983. Two Dimensional Stock Cutting Problems and Solution Methodologies. Journal of Engineering for Industry – Trans. of the ASME, Vol 105, pp155-160

197. Sarker, B.R. 1988. An Optimum Solution for One-Dimensional Slitting Problem: A Dynamic Programming Approach. Journal of Operational Research Society, Vol 39, pp749-755

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

199. Schwefel, H-P. 1975. Evolutionsstrategie und numerische Optimierung. Ph.D. thesis, Technische Universität Berlin

200. Schwefel, H-P. 1977. Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie. Basel: Birkhäuser

201. Schwefel, H. P. (1981) Numerical Optimization for Computer Models. John Wiley, Chichester, UK.

202. Sedgewick, R. 1992. Algorithms in C++. Addison-Wesley, Reading, Massachusetts

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

204. Shamos, M. I., Hoey, D. 1976. Geometric Intersection Problems. Seventeenth Annual IEEE Symposium on Foundations of Computer Science, pp 208-215.

205. Shamos, M.I. 1978. Computational Geometry. PhD Thesis, UMI #7819047, Yale University, New Haven, CT.

206. Shpitalni, M, Manevich, V. 1996. Optimal Orthogonal Subdivision of Rectangular Sheets. Journal of Manufacturing Science and Engineering, Vol 118, pp 281-288

207. Sleator, D. 1980. A 2.5 Times Optimal Algorithm for Packing in Two Dimensions. Information Processing Letters, Vol 10, pp 37-40

208. Smith, D. 1985. Bin Packing with Adaptive Search. International Conference on Genetic Algorithms and their Applications, Pittsburgh, Erlbaum, L. (ed), pp 202-207

209. Sweeney, E., Paternoster, R.E. 1992. Cutting and Packing Problems: A Categorized, Application-Orientated Research Bibliography. Journal of the Operational Research Society, Vol 43, No. 7, pp 691-706

210. Vajda, S. 1958. Trim Loss Reduction. Readings in Linear Programming. New York, Wiley, pp78-84

211. Vasko, F.J., Wolf, F.E., Pflugrad, J.A. 1991. An Efficient Heuristic for Planning Mother Place Requirements at Bethlehem Steel. Interfaces, Vol 21, No.2, pp 1-7

212. Vasko, F.J., Floyd, E.W. 1994. Journal of the Operational Research Society, Vol 45, No.3, pp 281-286

213. Vassilios, E., Theodoracatos, Grimsley, J. 1995. The Optimal Packing of Arbitarily-Shaped Polygons using Simulated Annealing and Polynolmial-Time Cooling Schedules. Computer Methods in Applied Mechanics and Engineering, Vol 25, Iss 1-4, pp 53-70

214. Wang, P.Y. 1983. Two Algorithms for Constrained Cutting Stock Problem. Operations Research, Vol 31, pp 573-586

215. Wolfson, M.L. 1965. Selecting the Best Lengths to Stock. Operations Research, Vol 13, pp 570-585

216. Wolpert, D. Macready, W. 1997. No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation, 67-82

217. Yeong, W.Y., Yue, T.M., 1991. Cutting Down Trim-Loss for Competitive Advantage – A Singaporean Company Experience. Journal of the Operational Research Society, Vol 42, Iss 8, pp 649-654

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

Appendix A – Papers produced as a result of this research

Journal Papers

E.K. Burke and G. Kendall, "*Comparison of Meta-Heuristic Algorithms for Clustering Rectangles*", Computers and Industrial Engineering, Vol. 37, Iss. 1-2, pp 383-386 (Proccedings of the 24th International Conference on Computers and Industrial Engineering, Brunel University, September, 1998)

Conference Papers

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

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

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

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

Appendix B – Data for Test Problem 1

Number of Items : 13
BinWidth : 80
Optimal Bin Height : 140

Test Data 1 (no. of items : width x height)
2 : 60 x 14
2 : 22 x 26
1 : 42 x 44
1 : 62 x 26
1 : 18 x 70
2 : 18 x 48
1 : 20 x 28
2 : 28 x 16
1 : 24 x 16

Appendix C – Data for Test Problem 2

Number of Items : 13
BinWidth : 6240
Optimal Bin Height : unknown

Test Data 2 (no. of vertices : *x*, *y* coordinates)
3 : 3024,0 : 0,3357 : 0,0
5 : 2025,0 : 2130,0 : 2130,7785 : 0,7785 : 0,2235
4: 2475,2760 : 2475,6060 : 0,6060 : 0,0
4 : 2127,0 : 2127,5328 : 0,2871 : 0,0
5 : 1050,0 : 2130,2235 : 2130,7785 : 0,7785 : 0,0
3 : 3024,0 : 3024,3357 : 0,0
4 : 2850,0 : 2850,5850 : 0,2730 : 0,0
4 : 1785,0 : 1785,2550 : 0,2550 : 0,0
4 : 2220,0 : 2220,5940 : 0,5940 : 0,2310
4 : 2550,0 : 2550,3015 : 0,6060 : 0,0
3 : 3054,0 : 3054,3390 : 0,3390
4 : 2127,0 : 2127,2871 : 0,5328 : 0,0
3 : 3054,3390 : 0,3390 : 0,0

Appendix D – Dataset from (Hopper, 2000a)

Number of Items : 16 or 17 items
BinWidth : 20
Optimal Bin Height : 20
Category : 1

Problem 1 (width x height)	**Problem 2** (width x height)	**Problem 3** (width x height)
2 x 12	4 x 1	4 x 14
7 x 12	4 x 5	5 x 2
8 x 6	9 x 4	2 x 2
3 x 6	3 x 5	9 x 7
3 x 5	3 x 9	5 x 5
5 x 5	1 x 4	2 x 5
3 x 12	5 x 3	7 x 7
3 x 7	4 x 1	3 x 5
5 x 7	5 x 5	6 x 5
2 x 6	7 x 2	3 x 2
3 x 2	9 x 3	6 x 2
4 x 2	3 x 13	4 x 6
3 x 4	2 x 8	6 x 3
4 x 4	15 x 4	10 x 3
9 x 2	5 x 4	6 x 3
11 x 2	10 x 6	10 x 3
	7 x 2	

Appendix E – Dataset from (Hopper, 2000a)

Number of Items : 25 items
BinWidth : 40
Optimal Bin Height : 15
Category : 2

Problem 1 (width x height)	**Problem 2** (width x height)	**Problem 3** (width x height)
11 x 3	11 x 2	12 x 7
13 x 3	2 x 3	7 x 7
9 x 2	10 x 7	7 x 1
7 x 2	8 x 4	5 x 1
9 x 3	9 x 5	3 x 2
7 x 3	7 x 2	6 x 2
11 x 2	4 x 1	7 x 2
13 x 2	6 x 1	5 x 2
11 x 4	4 x 5	3 x 1
13 x 4	8 x 3	6 x 1
3 x 5	1 x 3	12 x 6
11 x 2	5 x 5	9 x 6
2 x 2	3 x 1	12 x 2
11 x 3	12 x 4	7 x 2
2 x 3	6 x 2	10 x 3
5 x 4	2 x 4	4 x 1
6 x 4	11 x 4	5 x 1
12 x 2	10 x 2	16 x 3
1 x 2	3 x 2	5 x 3
3 x 5	11 x 2	4 x 2
13 x 5	3 x 4	5 x 2
12 x 4	26 x 4	10 x 3
1 x 4	8 x 4	9 x 3
5 x 2	3 x 2	16 x 3
6 x 2	6 x 2	5 x 3

Appendix F – Test Data from (Blazewicz, 1993)

Number of Items : 28
BinWidth : 15
Optimal Bin Height : unknown

Blazewicz, 1998 (no. of vertices : *x*, *y* coordinates)
6 : 0,0 : 2,-1 : 4,0 : 4,3 : 2,4 : 0 3
8 : 0,0 : 3,0 : 2,2 : 3,4 : 3,5 : 1,5 : –1,3 : –1,1
8 : 0,0 : 2,0 : 3,1 : 3,3 : 2,4 : 0,4 : –1,3 : –1,1
8 : 0,0 : 2,1 : 4,0 : 3,2 : 4,5 : 2,4 : 0,5 : 1,3
7 : 0,0 : 5,0 : 5,5 : 4,5 : 3,2 : 2,2 : 0,1
3 : 0,0 : 2,3 : –2,3
4 : 0,0 : 2,0 : 2,2 : 0,2

Each shape appears four times, making a total of 28 shapes

This data is re-produced in (Oliveira, 1998)