Two-dimensional Packing utilising Evolutionary Algorithms and other Meta-Heuristic Methods

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

by

Eva Hopper

University of Wales, Cardiff
School of Engineering

May 2000

DECLARATION

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any other degree.

Signed.....(candidate)

Date.....

This thesis is the result of my own investigations, except where otherwise stated.
Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed.....(candidate)

Date.....

I hereby give consent for my thesis, if accepted, to be available for photocopy and for inter-library loan, and for the title and summary to be made available to other organisations.

Signed.....(candidate)

Date.....

ACKNOWLEDGEMENTS

I would like to thank all of the people who contributed their time and support to me during my studies in Cardiff:

The staff of the Circuits and Systems Group, especially my supervisor Dr. B. Turton, for giving me the opportunity, arranging my grant and his invaluable support and guidance throughout my work, the Head of Department, Professor Tasker and the former Head of Department Dr. Horrocks for their support and guidance, and the secretarial staff Louise and Sheila for their organisational support.

The financial support from EPSRC and the WDA for providing the CASE studentship as well as Cardiff University for the A. A. Read scholarship.

J. Turton for her very useful comments on my thesis.

My fellow PhD students for their inspiration to my studies, in particular Michael, who was always willing to help out with any software and hardware problems and also maintained, updated and continuously extended the computing equipment to provide sufficient computing power for simulations and calculations.

To my family, especially my parents and my brother Peter, for their immense and never ending help during my stay abroad. A very special thanks to my boyfriend Wolfgang, who encouraged, helped and advised me so much during this work.

I could not mention everybody who helped, to all the people I have left out: Thank You.

SUMMARY

2D packing problems are a variety of combinatorial problems with a very large solution space that cannot normally be exhaustively searched. A series of hybrid approaches have been developed in this work to solve 2D rectangular and irregular strip packing and 2D bin packing problems. As the recent literature encourages the use of genetic algorithms in particular, along with other meta-heuristic methods for their solution, the main focus of this investigation has been on the application of evolutionary algorithms.

The meta-heuristic search methods have been implemented as hybrid algorithms, which use simple packing algorithms in the decoding stage for the layout generation. Apart from genetic algorithms, the meta-heuristic methods include simulated annealing, naïve evolution and stochastic optimisation. Hill climbing, random search and the downhill simplex method have also been applied to the same packing tasks.

The performance of the algorithms has been tested on a number of different sized packing problems. Some rectangular and irregular packing problems do exist in the literature, however the lack of benchmark problems has necessitated the implementation of a problem generator for rectangular packing tasks. For comparison, two commercial nesting packages were included and applied to the test problems. A heuristic packing method, which allows the user to generate rectangular layouts of very high quality, is proposed as a test method to benchmark packing algorithms.

Meta-heuristic techniques produced good results compared with published algorithms. Genetic algorithms were better suited to the solution of small to medium sized problems, at the expense of significant computation time. Simulated annealing produced very dense layouts but with an even greater increase in computation time. All of these techniques produced a limited improvement in the solution quality compared with standard packing techniques.

Table of Contents

Appendices

List of Figures

List of Tables

Nomenclature

	description	**unit**
A	area	unit
d	packing density	%
h	height	unit
N	number of items in packing problem	-
p	perimeter	unit
t	elapsed time	s
u	object utilisation	%
w	width	unit

List of Acronyms

search methods:

ANN	artificial neural network
DHS	downhill simplex method
EA	evolutionary algorithm
GA	genetic algorithm
HC	hill-climbing
NE	naïve evolution
RS	random search
SA	simulated annealing

genetic operators:

CX	cycle cross-over
PMX	partially matched cross-over
PMX1	partially matched cross-over; 1 cross-over point
PMX2	partially matched cross-over; 2 cross-over points
OBX	order-based cross-over
OX	order cross-over
OBM	order-based mutation
PBM	position-based mutation

sorting criteria:

DA	decreasing area
DH	decreasing height
DW	decreasing width
DP	decreasing perimeter
DRA	decreasing area of rectangular bounding box
DRP	decreasing perimeter of rectangular bounding box

packing routines for rectangular problems:

BL	bottom-left algorithm for rectangles based on sliding technique
BLLT	bottom-left algorithm for rectangles based on sliding technique
BLF	bottom-left algorithm for rectangles capable of hole filling
BLD	bottom-left algorithm for rectangles based on sliding technique
BF	Best Fit algorithm
BFD	Best Fit Decreasing algorithm
FF	First Fit Decreasing algorithm
FFD	First Fit Decreasing algorithm
FFDH	First Fit Decreasing Height algorithm

HFF	Hybrid First Fit = FFDH * FFD
NF	Next Fit

packing routines for irregular problems:

BLi	heuristic packing algorithm for polygons based on sliding technique and enclosing rectangles
BLLTi	heuristic packing algorithm for polygons based on sliding technique and enclosing rectangles
BLFi	heuristic packing algorithm for polygons based on enclosing rectangles; capable of hole filling
BLDi	heuristic packing algorithm for polygons based on sliding technique and enclosing rectangles
BLPi	heuristic packing algorithm for polygons based on sliding technique
BLDPi	heuristic packing algorithm for polygons based on sliding technique

other:

NFP	No Fit Polygon
TSP	Travelling Salesman Problem

1. Introduction

Packing problems are optimisation problems that are concerned with finding a *good* arrangement of multiple *items* in larger containing regions (*objects*). This type of problem is encountered in many areas of business and industry and is forms part of the combinatorial problems found in operational research. The usual objective of the allocation process is to maximise the utilisation and hence to minimise the "wasted" material.

1.1 Industrial Applications

The reduction of production cost is one of the major issues in manufacturing industries in an ever more competitive world where products are available almost instantly from anywhere on the globe. High material utilisation is of particular interest to industries with mass-production, since small improvements of the layout can result in large savings of material and considerably reduce production cost. Cutting and packing problems are encountered in many industries. The complexity of the problem and the solution approach depend on the geometry of the items to be placed and the constraints imposed. Whereas the wood, glass and paper industry are mainly concerned with the cutting of regular figures, irregular, arbitrary shaped objects are to be packed in the ship building, textile and leather industry.

The duration of the layout generation is also relevant in industrial applications. As manual generation can easily require several man-days, the packing time is an important economic factor. It is therefore understandable that industries are looking into ways to automate the packing process. Figure 1.1 summarises the steps typically involved in the industrial nesting process.

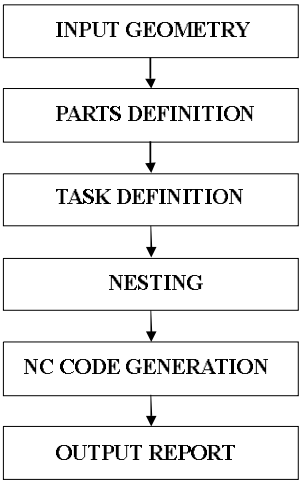


Figure 1.1: Typical steps involved in the nesting process

- input geometry from other CAD systems via DXF, DWG etc. file format or create geometry with the aid of built-in CAD module
- define parts: material, thickness, part quality, etc.
- define nesting task: number and size of object, part clearance, cutting process etc.
- automatic, manual or interactive nesting on single or multiple objects
- generate order in which parts are to be cut; output NC code for cutting process
- generate report including layout details, e.g. scrap percentage, length of strip material or sheet utilisation; cutting time and cost

The research community therefore started some time ago to investigate automatic solution approaches for cutting and packing tasks. The first scientific publications in this area appeared in the 1950s when a number of researchers highlighted the relationship between the stock cutting problem and linear programming theory, which belongs to the standard methods of operational research (Dantzig, 1951; Kantorovich and Zagaller, 1951). These early approaches concentrated only on rectangular problems.

The majority of industrial packing tasks involve irregular shapes and stock material. Certain material properties usually impose additional constraints on the packing task (section 1.1.1 to 1.1.4). Automatic packing systems were first introduced into the textile and shipbuilding industry in the 1970s and worked in interactive mode (Catastini et al., 1976; Øian et al., 1976). In the meantime a large number of fully automatic nesting systems have become available. Some are suitable for the general packing task; others are designed to meet the special requirements of certain industries (section 4.5).

Depending on the complexity of the shapes in irregular nesting tasks, the computation of layouts may require a long time. But in some applications packing problems need to be solved in real-time. As a consequence, the computation time is of paramount importance in the solution method applied. A great number of so-called on-line algorithms were proposed for this dynamic version of the packing problem (Coffman et al., 1984; Runarsson et al., 1996). The geometric algorithms in irregular nesting problems are complex and usually apply a certain degree of approximation to reduce the computation time. In cases where the solution of the nesting task is not coupled with the timing of the manufacturing process, computation time is a minor issue. When the nesting task is solved off-line, a higher degree of accuracy is possible. It is common practice in layout generation to utilise the available computational power efficiently by using overnight runs.

Recently, a class of solution methods found in the field of artificial intelligence have attracted the interest of the algorithm designers in this area (section 1.2). Inspired by nature, meta-heuristic search principles apply strategies, which work successfully in the natural process, to find good solutions. Modelled on evolution and annealing, the artificial counterparts consider a number of solutions and condense them to a few good ones (section 3.3). As the terminology suggests, the methods describe search principles rather than straightforward computation. Computation times are expected to be high which makes these methods only suitable for a limited range of nesting applications.

1.1.1 Packing in the Sheet Metal Industry

The sheet metal industry has to deal with regular as well as irregular nesting problems. Certain constraints regarding the material properties, the cutting process and scheduling aspects distinguish the packing task in this field from other industries. Apart from reducing the wastage to a minimum, there are a number of other factors that decide the final layout of the parts.

Material Properties:

The raw materials are either available in the form of sheet material with fixed dimensions or as coiled material with a fixed width. This has an impact on the objective of the nesting task. In the first case the shape of a possible remnant decides whether it can be used in a future nesting task or has to be treated as wastage. For coiled material this is usually not a constraint.

As the materials have inhomogeneous properties such as grain orientation, the number of possible orientations the parts can be nested in may be limited. This entirely depends on the application and the further processing of the shapes. If there are no bending operations to follow, the parts can be rotated in any direction; otherwise a specified angle with respect to the grain orientation needs to be applied.

Nesting Process:

The parts to be nested can contain void areas, some of which may be large enough to be considered for the nesting of smaller items. This technique is referred to as in-hole-nesting and is very common in the shipbuilding industry. To reduce waste, the nesting algorithm needs to be capable of tracking and nesting into void areas of irregular shapes. Sometimes the current nesting task does not contain a sufficient number of comparatively small shapes. As the raw material is often too precious to be wasted certain filler parts can be designated and used instead. These are not part of the current order and therefore may not be required immediately, but are produced for stock.

Larger nesting tasks might involve material of different types, e.g. thickness. In the shipbuilding industry for instance sheets of different thickness can be involved in the nesting process. Whereas a number of parts require a certain sheet type, often several thicknesses are suitable for a subsection of the order list. Consequently, depending on the availability, the nesting algorithm needs to decide on the best allocation.

In certain applications, the geometry of the parts may allow special layout configurations. Especially when the order list contains many congruent shapes, rotation along a certain axis can allow two shapes to be nested efficiently in a cluster that can be repeated in the layout. Grouping is a similar technique, where two or more shapes are combined to a cluster, which can be used more than once for the layout generation. Clusters offer sub-optimal configurations of a limited number of shapes. Their repeated use increases the speed of the layout generation once a combination is computed.

Cutting Process:

The cutting technique used to obtain the parts has a great impact on the layout generation. Depending on the cutting technology (e.g. laser and plasma cutting, stamping) a minimum distance between the parts is required. This parameter is referred to as bridge width. In laser and plasma cutting the process operates with a certain width. In order not to damage the parts, a certain distance between neighbouring shapes is necessary. In stamping processes the material tends to slip at the cutting edges if the bridge width is too small. Another important parameter that determines the cutting process is the cutting length. Layouts can be optimised so that the cutting of all parts can be carried out under minimisation of the total distance.

Scheduling:
The sequence in which the parts are cut can be important for the subsequent manufacturing process. This is the case where parts need to be processed in different steps. If the layouts are large, a special allocation of the parts with respect to the sheets facilitates this. The sequence of the parts may also be important for packaging and shipping. Geometrical or weight constraints may require the parts to be packed in a certain order. Sometimes different order lists are nested in one layout to maximise material utilisation. Hence the order sequence of the parts also plays a role in despatching.

1.1.2 Packing in the Textile Industry

The textile industry usually utilises coiled material and hence is mainly concerned with the strip packing problem. The material properties limit the layout generation. As the fabric often has certain directional properties and a pattern, the orientation of the parts is usually restricted to rotation intervals of 0° and 180°. In many cases it may not be possible to mirror the parts as the fabric has different properties at the other side.

1.1.3 Packing in the Leather Industry

The nesting task in the leather industry is very complex since both the parts to be nested as well as the objects, the so-called hides, are highly irregular. As leather is a natural material, as opposed to the manufactured ones used in the textile and sheet metal industry, the hides consist of areas having various qualities. The quality difference can be due to defects and colour differences. The nesting process therefore needs to match the parts with their respective quality zones on the hide. Usually, image processing takes place prior to the nesting process to determine the shape and the quality of the hides. As this nesting task is very complex and needs to consider many specific requirements specially designed nesting packages are available to the leather industry (section 4.5).

1.1.4 Industries with Rectangular Packing Problems

The group of rectangular packing problems appears in the paper, wood and glass industry. The cutting technique, which involves shear cuts, imposes a characteristic constraint on layouts in this area. The packing patterns are required to be guillotineable, such that the parts can be obtained by straight cut through the remaining layout only (section 2.2). Packing in the glass and wood industry also needs to consider various quality ranges and defects of the raw material. Rectangular packing tasks can occur as strip and bin packing problems.

Strip Packing:
The paper industry is mainly concerned with strip packing problem, as the raw material is available in the form of rolls. Hence the packing process aims at reducing the height of the layout.

Bin Packing:

Bin packing refers to packing of multiple bins and can be found where the stock material is available in the form of sheets rather than rolls. Bin packing is not restricted to the rectangular case. In industrial applications this problem is referred to as stock cutting (section 2.3) such as in the glass, wood and metal industry. The objective usually is to find the set of sheets to accommodate all parts of the order list under minimisation of the total material used. Depending on the application, the sheets can be identical or have different size. Regular bin packing is frequent in three dimensions including problems such as container and pallet loading.

Non-Guillotineable Layouts:

Industrial applications that involve non-guillotineable layouts are less frequent. The 2D stock cutting problem occurs as a partial problem in pallet and container loading. A commonly adopted approach is to reduce the 3D problem to two dimensions. Once a set of boxes for a certain layer has been determined, the layer represents a 2D non-guillotineable packing problem. The loading patterns are then created for each layer separately (Bortfeldt, 1994). Processor allocation can also be regarded as a 2D packing problem (Hwang, 1997). In some branches of the metal industry the rectangular packing problem can occur in the non-gullotineable form when different cutting methods such as laser and plasma cutting are used to obtain the parts.

1.2 Solution Approaches

For the solution of small packing problems deterministic methods such as linear programming have been developed. Since these methods are exact, they find the optimal solution. For problems of higher complexity, the solution space is larger due to a higher number of possible combinations. In this case it is not possible to calculate the optimal solution in reasonable computing time. In order to approach these problems heuristic techniques can be used to find near-optimal solutions at reasonable computational cost. Solutions will be within a tolerable range of deviance from the optimal solution. The way the placement has to take place is described by a set of rules. Since these rules are tailored to a particular packing task, the problem-specific heuristic algorithms can only be applied successfully to that particular problem type.

More flexibility is offered by general heuristic methods, so-called meta-heuristics, which describe general search principles rather than special rules. Some of them are inspired by optimisation processes in nature, such as evolutionary algorithms and simulated annealing (section 3.3). The search through the large solution space is guided by problem-specific information. The quality of the solution depends on the implementation of this knowledge and the parameters applied to control the search process. The success of the meta-heuristics can be explained by their great flexibility in taking into account problem-specific

constraints and their good trade-off between solution quality and computational effort as opposed to a full search.

In order to decide how profitable general heuristic search methods are as a solution approach to packing problems, their performance needs to be evaluated against the cost of manually generated layouts or problems-specific heuristics, which can both offer solutions of high quality. With the increase in computing power meta-heuristic search techniques have become highly competitive in complex packing problems with large solution spaces. Genetic algorithms in particular, which allow exploration and manipulation within a large search space, have frequently been applied in the literature (section 4.3).

1.3 Outline of the Thesis

This investigation has concentrated on 2D rectangular and irregular packing problems. The rectangular packing task has been considered in the form of strip packing and bin packing. The major objective has been to develop a set of meta-heuristic solution approaches sufficiently flexible to tackle various packing tasks.

Chapter 2 introduces a classification for cutting and packing problems. Since this project focuses particularly on the research of meta-heuristic methods, the basic concept of genetic algorithms and simulated annealing problems is briefly introduced in chapter 3 along with some other heuristic search principles.

Chapter 4 provides an overview of the research activities in the area of 2D and 3D packing problems. It outlines the current state of the research in the application of problem-specific heuristics and meta-heuristics to packing problems. Particular emphasis is hereby put on genetic algorithms. The chapter finishes with an overview of commercial software packages available for industrial nesting problems and concludes with a summary of the major aspects important for the application of genetic algorithms to packing problems.

Chapter 5 introduces the solution approaches developed in this work for 2D strip and bin packing problems. Several algorithms are introduced for the generation of the rectangular test problems. The outcome of a comprehensive performance evaluation also includes a comparison with some test problems from the literature and two commercial packages and is discussed in chapters 6 to 8.

Chapter 9 summarises the major findings of this investigation and gives recommendations for possible future work in this area.

2. Cutting and Packing Problems

Packing problems arise in a large number of situations. The variety of the problems is as large as their application areas including disciplines like management science, engineering, mathematics, computer science or operational research with different industries incorporating different constraints and objectives (Figure 2.1).

Due to this diversity of problems and application areas similar packing problems appear under different names in the literature. Dyckhoff (1990) proposed a systematic classification of packing problems. Packing problems are combinatorial problems of high complexity, which are NP-complete (section 2.4). Depending on the complexity and the practical requirements different solution strategies are applied.

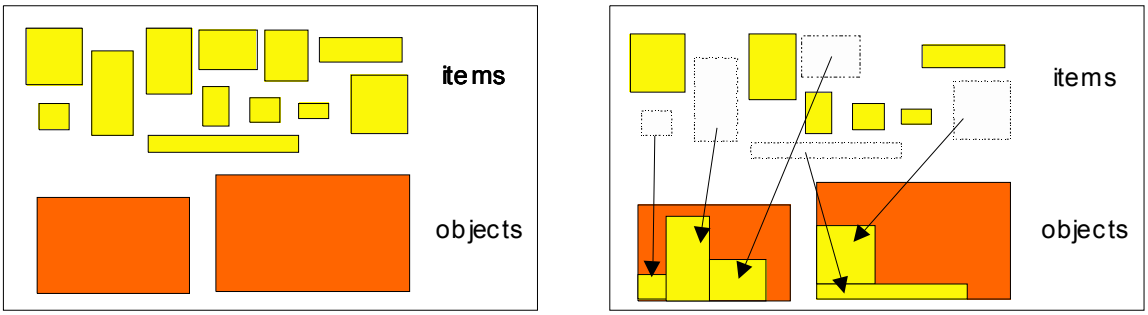


Figure 2.1: Packing task

2.1 Dyckhoff Classification

Analysing different packing problems shows that many of them have the same basic logical structure, although they are encountered in different application areas. In order to facilitate the information exchange across different disciplines and hence research and application of solution approaches Dyckhoff (1990) identified common characteristics and properties and systematically classified packing problems.

2.1.1 Classification and Definitions

In general, packing problems belong to the field of geometric and combinatorial computing. Dyckhoff (1990) distinguishes between packing problems involving spatial dimensions and those involving non-spatial dimensions. The first group consists of cutting and packing or loading problems that are defined by up to three dimensions in Euclidean space. The other group covers abstract “cutting and packing” problems including non-spatial dimensions such as weight, time or financial dimensions (Figure 2.2).

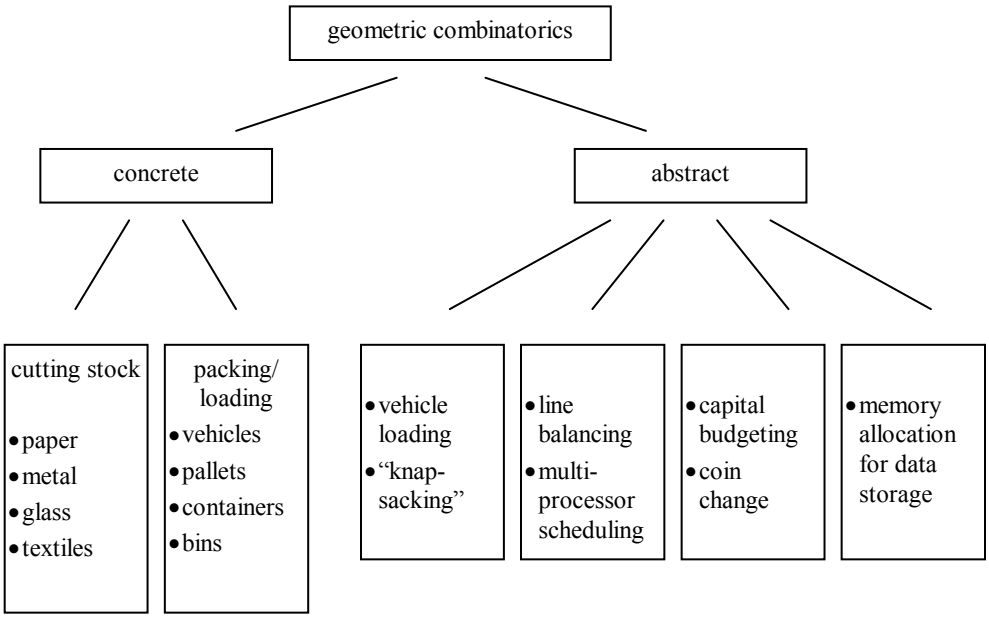


Figure 2.2: Phenomenology of cutting and packing problems (Dyckhoff, 1990)

Cutting and packing problems describe patterns consisting of geometric combinations of large objects (e.g. stock) and small items (e.g. order book). In the case of packing problems the large objects (e.g. container, bin) are defined as empty and need to be filled with small items (e.g. boxes). Cutting problems are characterised by large objects (e.g. sheet, roll) that need to be cut up into small items (e.g. 2D shapes). The residual objects, that occur in the pattern and do not belong to the order list, are called trim loss. The objective of most cutting and packing problems is to minimise the trim loss or wastage.

Dyckhoff emphasises the strong relationship between cutting and packing problems, which results from the duality of material and space. In this sense cutting stock problems can be seen as packing the space occupied by the small items into the large objects. Vice versa packing problems can be seen as cutting the large objects into small items.

2.1.2 Typology

Dyckhoff's classification system describes four important characteristics of packing and cutting problems. These characteristics and the values they can take on are summarised in Table 2.1. Examples for various problem types are given in (Dyckhoff and Finke, 1992).

The most important characteristic is the dimensionality defining the minimum number of dimensions necessary to describe the geometry of the pattern. Problems with more than three dimensions are obtained when they expand to non-spatial dimensions, e.g. time or weight.

The kind of assignment describes whether all objects and items or only a selection has to be assigned. Four types of assignment are possible, the two most important ones are differentiated in Dyckhoff's typology:

- a selection of small items has to be combined to patterns such that a non-trivial (corresponding) pattern is assigned to *each* large object (Beladeproblem B)
- all small items have to be combined to patterns that are then assigned to a proper *selection* of large objects (Verladeproblem V).

The assortment not only considers the shape of the objects and items but also their number. The different types for large and small objects are stated in the table below. The term "figure" refers to items and objects being uniquely determined by their shape, size and orientation.

Table 2.1: Dyckhoff's typology of packing and cutting problem (Dyckhoff, 1990)

Characteristic	Symbol	Description
dimensionality	1 2 3 N	one-dimensional two-dimensional three-dimensional N-dimensional with N > 3
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 different (non-congruent) figures congruent figures

2.2 Additional Classifications

The objective of a packing problem is the efficient allocation of figures in a containment region without overlap. Hence, the complexity of packing problems is strongly related to the geometric shape of the items to be packed. Concerning the geometry two types of shapes can be distinguished: regular shapes, that are described by a few parameters (e.g. rectangles, circles) and irregular shapes including asymmetries and non-convexities.

In 2D packing, the following layout types can be distinguished on the basis of the geometry of the items to be packed. In the case of regular items packing patterns can be orthogonal describing cuts only parallel to the sides of the stock sheet and non-orthogonal (Figure 2.5). Orthogonal cutting additionally distinguishes between guillotineable (Figure 2.3) and non-guillotineable (Figure 2.4) layouts. Packing of irregular shapes is known as nesting e.g. in the shipbuilding industry and as marker layout problem in the textile industry (Figure 2.6).

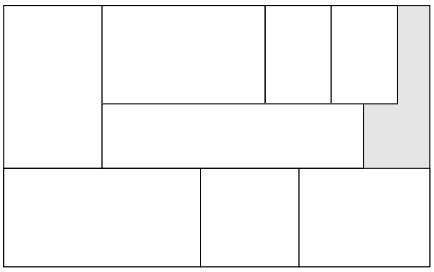


Figure 2.3: Orthogonal, guillotineable packing

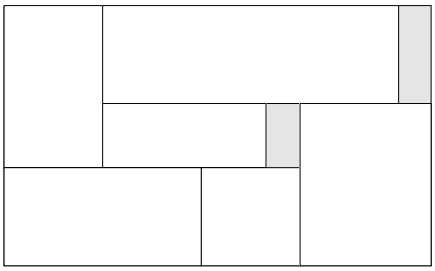


Figure 2.4: Orthogonal, non-guillotineable packing

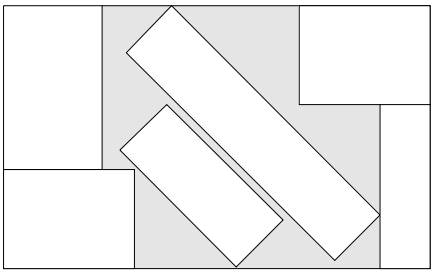


Figure 2.5: Non-orthogonal packing

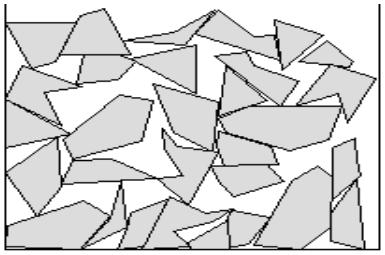


Figure 2.6: Nesting of irregular shapes

Although cutting and packing problems may have a common structure, which allows a classification like the one introduced by Dyckhoff (1990), they can differ to a great extent in the constraints imposed by various industries. Two general criteria common to all problems are that the items have to fit completely onto the objects and must not overlap. Special industrial needs are expressed as additional constraints concerning the details about possible allocation of the items. Some items are required to be placed only onto a certain area of the object (e.g. shoe manufacturing) or in certain directions (e.g. textile industry). In metal cutting for instance the items have to be placed with a minimum distance between each other. In the wood, glass or paper industry the subsequent cutting process requires the layouts to be guillotineable. Some examples of packing problems involving different geometric primitives are listed in Table 2.2.

Table 2.2: Examples of geometric shapes in 2D packing problems

geometric shape	**example for application**
circle, ellipse	loading of containers with pipes
polygons: convex or concave	packing of irregular shapes in the metal industry
free-form: parts consist of a series curved and straight line segments	marker layout problem in the textile industry
shapes with complete or partial enclosures	shipbuilding

2.3 Types of Packing Problems

Packing problems occur in various application areas involving different constraints and objectives. In the following some of the most important problems are defined briefly. Larger industrial problems can as well appear as combinations of two or more of these basic types. Further details about individual problems and examples can be found in (Dowsland and Dowsland, 1992; Dyckhoff and Finke, 1992; Hinxman, 1980).

Cutting stock problem:
The cutting stock problem is concerned with the cutting of pieces of a given order list from a set of stock sheets. This problem can be split into two sub-problems, an assortment problem (determination of the sheets to keep in stock) and a trim-loss problem (determination of the cutting pattern to minimise waste).

Trim-loss problem:
The trim-loss problem concerns the allocation of the order list onto the given stock sheets. The order list describes the set of pieces that must be allocated. The objective is to minimise the total cost of the stock sheets needed to fulfil the order.

Assortment problem:
This problem involves the determination of the stock sizes necessary to fulfil the order list. The order list needs to be assigned to a supply of stock sheets such that the best selection of sheets is used.

Knapsack problem:
Each of the order pieces has a given value. The objective is to pack the items into fixed stock objects such that the total value of the items packed is maximised (Martello and Toth, 1990). This type of problem often occurs as a sub-problem in other areas.

Bin packing:
1D bin packing is the allocation of items, whose width is identical to the ones of the bins. Hence, for the packing process only one dimension is important. As the height of the bins is fixed, the packing process aims at minimising the number of bins. Using items of different width results in 2D bin packing. The concept of bin packing appears in more abstract versions in other application areas e.g. assembly line balancing (Falkenauer and Delachambre, 1992).

Loading problem:
The loading problem describes the process of fitting a maximum number of boxes onto a pallet or into a container. The pallet loading problem can be regarded from the manufacturer's viewpoint, where identical boxes have to be loaded onto a pallet (manufacturer's pallet loading problem), as well as from the distributor's side, where the pallet has to be packed with non-identical items (distributor's pallet

loading problem). Container loading is similar to pallet loading, though in practical applications the two variants of the loading problem can be distinguished by their constraints.

2.4 Packing and NP-Hardness

One of the most important conventions in complexity theory is the definition of problem classes. Combinatorial problems can be stated as decision problems where a solution corresponds to a correct yes or no response. An optimisation problem is converted by posing the question of whether of not there exists a feasible solution which has an objective function value equal or superior to a specified threshold (Garey and Johnson, 1979; Gary Parker, 1995). According to the algorithm complexity, the following problem classes can be distinguished.

Class P: Class P describes problems, which can be solved by algorithms that require at most a polynomial function of the instance size. The membership in this class means that even large problem instances can be solved with exact routines. Increases in problem size normally have a small impact on the computation time. However, the solution of large problems might become impractical using conventional computer systems. Thus, it may be still worthwhile to search for better algorithms that are of lower polynomial order.

Class NP: The definition of this class is based on the solutions to a problem rather than the algorithm. If the correct solution (i.e. a yes response) can be checked for feasibility in polynomial time, a problem is known as non-deterministic polynomial (NP). Although problems in this class are 'easy' to verify, they are not 'easy' to solve. By this definition, the class NP contains the class P.

In order to understand the class of NP-hard and NP-complete problems, first the concept of reduction is introduced. A problem P reduces to a problem P' if any algorithm that solves P' also solves problem P provided an appropriate conversion is found. Polynomial reductions are the most important ones in this context. A problem P polynomially reduces to P', if a polynomial time algorithm for P' implies the existence of a polynomial time algorithm for P.

Class NP-Hard: Problems to which all members of NP polynomially reduce are referred to as NP-hard. Thus, a polynomial time algorithm for an NP-hard problem, would produce an algorithm for every member of NP at the same time. Many optimisation problems are known to be NP-Hard.

Class NP-Complete: Problems that are in class NP-hard and also in class NP are called NP-complete. As all the members of class NP-complete are decision problems, it does not contain optimisation problems in the strict sense. Optimisation problems are NP-hard, but have analogue decision problems which are NP-complete.

The rectangular packing problem or rather its decision analogue has been shown to be NP-complete (Fowler et al., 1981). As the irregular and 3D versions of this problem are more complex, they can also

be regarded as NP-complete. Various constraints can be imposed on a packing problem depending on the application. Adding constraints may add to its complexity and thus the constrained versions can also be regarded as NP-complete. It should be noted that adding constraints could have the opposite effect on the search space. In specific circumstances it may make the search space easier to 'navigate' and a problem which in its general form is regarded as NP-complete can then be solved with an exact algorithm.

According to the aforementioned definition the NP-complete class has the important characteristic, that all algorithms currently known for finding optimal solutions require a number of computational steps that grows exponentially with the problem size rather than according to a polynomial function. It is not worthwhile to search for an exact (optimal) algorithm, since it does not appear that any efficient optimal solution is possible. Alternative approaches that are not guaranteed to find an optimal solution are considered instead (section 2.5). Thus, by giving up solution quality, computational efficiency can be gained.

This point of view is often adopted in cutting and packing and has lead to the development of approximation algorithms, i.e. heuristics. Knowing that a given packing problem is NP-complete is extremely significant in the design of an automatic packing strategy, since it helps researchers to direct their effort towards those approaches that have the highest likelihood of leading to a useful solution.

2.5 Solution Approaches

Packing problems belong to the field of geometric combinatorial computing. The complexity of the geometric properties of the items influences the magnitude of the problem and the solution approach. The search for an optimal solution for the placement of rectangular pieces is simpler involving fewer possibilities in terms of the orientation of the pieces. With the number of possible orientations and arrangements being larger for irregular shapes, the search space is much larger for this problem class and gets more complex when concave features are involved.

The solution approaches to packing problems can be divided into two groups, deterministic and heuristic methods. Deterministic methods are exact techniques that guarantee to find the optimal solution for a problem. They are usually based on linear programming or enumeration approaches such as branch-and-bound or dynamic programming. These methods are explained in more depth in (Sedgewick, 1992).

For problems of higher complexity these techniques become inefficient due to the vast number of possible solutions that have to be evaluated in enumeration based approaches. Since conventional methods fail to produce an optimal solution in reasonable amount of computing time, heuristic approaches have been developed. Unlike exact methods heuristic techniques do not guarantee to find an optimal solution to a problem. They only produce solutions that are “good enough” or near optimal, but

at a reasonable computational cost. Solutions will be within a tolerable range of deviance from the optimal solution. Heuristic techniques applied to packing problems can be problem-specific consisting of a set of placement rules. Since these rules are tailored to a particular packing task the disadvantage of problem-specific heuristic algorithms is that they can only be applied successfully to that particular problem type. On the other hand they may be extremely efficient in terms of computational cost.

Packing problems can also be seen as combinatorial optimisation problems. The application of search techniques allows finding solutions, which satisfy predefined criteria. A combinatorial optimisation problem can be formulated in the following way:

With

- the solution space S being the finite set of all solutions,
- the cost function C being a real valued function defined on members of S and the quantity to be minimised (or maximised depending on the objective)
- the problem is to find a solution or state, $i \in S$, which minimises (or maximises) C over S.

In general, search techniques operate as iterative improvement techniques attempting to find the global minimum in the solution space. Downhill techniques also operate as iterative improvement techniques. As they only accept moves that result in an immediate improvement in the objective function, they may only find a local minimum, which may be far from the global minimum. If an optimisation method is to converge to a global minimum and be insensitive to the starting point, then it must either enumerate all the local minima and then select the global minimum (multi-start method) or be able to accept moves that increase the objective function in a controlled manner.

In contrast to local descent algorithms, which may get trapped in a local minimum, other heuristic search methods can overcome this problem by accepting an increase in the value of the cost function and allowing for uphill moves. The acceptance of such moves depends on problem-specific information used to guide the search process. Heuristic search techniques, which are able to search large, constrained solution spaces due to built-in mechanisms allowing for uphill moves, are genetic algorithms, simulated annealing, tabu search, artificial neural networks, Langrangean Relaxation and others. A description of these techniques can be found in (Reeves, 1993; van Laarhoeven, 1987).

3. Optimisation with Heuristic and Meta-heuristic Search

The quality of the layout, which is constructed using heuristic placement algorithms, strongly depends on the sequence in which the rectangles are presented to the routine. Thus the full state of solutions consists of every permutation of the items with each one packed in every feasible position and orientation. Since the number of combinations is too large to be explored exhaustively in reasonable amount of time, heuristic and meta-heuristic algorithms are used as a more efficient search strategy. Efficiency of a heuristic search technique is dependent on how domain-specific knowledge is exploited and its ability to overcome combinatorial explosion.

A tree structure is a graphical representation of the numerous solutions that exist for a combinatorial problem. At each stage of the search algorithm a new intermediate solution is generated for each of the remaining elements not yet allocated. Depending on the strategy applied to search the tree several procedures can be distinguished. Depth-first, breath-first, hill-climbing, beam search and best-first belong to the basic strategies (Winston, 1984). The branch-and-bound technique, discrete dynamic programming and the A* procedure are more sophisticated search methods involving backtracking and heuristic information. Other optimisation techniques applied to simple packing problems are linear and mathematical programming.

Large combinatorial problems typically have extremely large solution spaces, so that traditional search techniques are totally impractical. Many approaches are proposed in the literature involving meta-heuristic search principles (section 4.3 and 4.4). These techniques result in a good, however, not necessarily optimal solution within reasonable computing time. This chapter describes the basic principles of the heuristic and meta-heuristic optimisation methods used in this investigation.

3.1 Hill-Climbing

Hill-climbing is a local search technique performing moves in the neighbourhood of the current state. If the quality of the new solution is better than the current one, this move is accepted and the search continues from there. If the neighbouring state does not result in an improvement, the move is rejected and the search continues from the current state. The main disadvantage of this method is that the search process might get trapped in a local minimum, which is different from the global one. A useful variation of simple hill-climbing considers a series of moves from the current state and selects the best one as the next state. This method is known as gradient search or steepest-ascent hill-climbing.

3.2 Downhill Simplex Method

The downhill simplex method was extended by Nelder and Mead (Press et al., 1995) for multi-dimensional function optimisation. One of the major advantages of this method is that it only requires function evaluations and not derivatives. This method is not very efficient in terms of number of function evaluations and therefore is mainly suitable for smaller dimensions.

The method makes use of the geometrical concept of a simplex. The geometrical figure of the simplex depends on the number of dimensions of the optimisation (N) and describes a shape with N+1 vertices. In two dimensions the simplex is a triangle and in three it is a tetrahedron.

The downhill simplex method is started with N+1 points that describe the initial simplex. It then attempts to find a way downhill through the complex N-dimensional landscape until it reaches a (local) minimum. This is done by a series of iterations which consist of moving the highest (worst) point of the simplex through the opposite face of the simplex to a lower point. These steps are called reflections. If a reflection has been successful, i.e. if a lower point than the best point so far has been found, the algorithm tries to expand the simplex in this direction (expansion). If the reflection fails and the method cannot eliminate the highest point, it will try a contraction step around the lowest (best) point. The minimisation routine ends when a certain tolerance for the decrease in the function value has been reached.

3.3 Meta-Heuristic Search Strategies

In order to overcome the main disadvantage of local search algorithms such as hill climbing, whose weakness lies in the inability to escape from local minima, more sophisticated heuristic search strategies have been designed. This can mean the temporary acceptance of a state of lower quality. A meta-heuristic is an iterative master process that guides the operations of subordinate heuristics (Voss et al., 1999).

Various meta-heuristic search principles have been developed during the last twenty years. Some of them have been inspired by nature and are modelled on processes such as evolution and annealing. Genetic algorithms, simulated annealing and tabu search are the most widely applied meta-heuristics for large combinatorial problems. The meta-heuristic search process may manipulate a single solution (e.g. simulated annealing) or a collection of solutions (e.g. genetic algorithms) at each iteration. The subordinate heuristic can be high (or low) level procedures, local search or a construction method (Voss et al., 1999).

In the area of packing it is very common to use hybrid approaches combining meta-heuristics with heuristic algorithms. The task of the meta-heuristic is to search a good ordering of the items. A placement routine is then needed to interpret the permutation and evaluate its quality.

3.3.1 Genetic Algorithms

Genetic algorithms are optimisation and search procedures that operate in a similar way to the evolutionary processes observed in nature. They are based on Darwin's theory of evolution and simulate natural selection and genetics. They are well suited for complex optimisation problems with a large search space. As in nature genetic algorithms attempt to find new and better solutions to a problem by improving present solutions. The search is guided towards improvement applying the principle known as "survival of the fittest". This is achieved by extracting the most desirable features from a generation of solutions and combining them to form the next generation.

The quality of each solution is measured by a fitness function. According to their fitness values individuals are selected for reproduction usually using a weighted probability function. Hence each individual will contribute to the next generation in proportion to its fitness. The motivation is to continue this process through a number of generations in order to reach convergence on optimal or near-optimal solutions. Mimicking the natural evolution process genetic algorithms work with probabilistic rather than deterministic rules. The probabilistic nature of genetic algorithms becomes obvious when looking at the genetic operators such as reproduction, cross-over and mutation.

The origins of genetic algorithms can be traced back to the late 1950s (Bäck et al., 1997). In the early 1970s, genetic algorithms were developed further by Holland and his colleagues at the University of Michigan (Holland, 1975). Together with "evolutionary programming" (Fogel et al., 1966), "genetic programming" (Koza, 1992) and "evolutionary strategies" (Rechenberg, 1973) they belong to the group of "evolutionary algorithms". Although genetic algorithms are an iterative improvement technique they still remain heuristic strategies and cannot guarantee to find an optimal solution. The basic concepts of genetic algorithms are described briefly in the following using simple binary encoding. They can also be applied to other encodings requiring appropriate modifications. Further theoretical and practical details can be found in (Davies, 1991; Goldberg, 1989; Mitchell, 1996; Fogel, 1998).

3.3.1.1 Problem-Specific Design Variables

The problem-specific design variables describe the nature of the problem and thus reflect its specific characteristics. These decisions involve the problem representation along with suitable genetic operators.

Representation

The representation technique, i.e. the encoding of the solution space is of paramount importance for the successful operation of genetic algorithms. First of all, the parameters of the objective function influencing the optimisation problem need to be identified and coded. The classic approach uses binary coding where the parameters are represented by strings of 0's and 1's. The parameters are then

combined resulting in a multi-parameter string - each representing a possible solution to the problem. These multi-parameter strings correspond to chromosomes in natural systems.

The classical genetic algorithm involves binary strings as genotypes that are manipulated by the genetic operators. This representation becomes less effective for more complex problems such as combinatorial problems. In this case the representation can be numerical. Additionally, order-based chromosomes are proposed (Davis, 1991; Goldberg, 1989). The encoding technique can be further adapted to the nature of the problem implementing special data structures where an individual can consist of several chromosomes rather than a single chromosome. A good representation of the problem reduces the effort at coding and decoding but also demands more complex genetic operators in the sequel.

Order-based encodings are especially suitable for combinatorial problems with a permutation representing the complete set of elements. As the standard cross-over and mutation techniques would result in invalid solutions modified genetic operators are required. Techniques appropriate for this data structure are stated along with the standard operators.

Initial Population

The initial population is the first set of strings generated before the search process is started. Usually, the strings are chosen randomly. Alternatively, the initial population can include a high-quality solution generated with heuristic rules. This technique is known as seeding and may help the genetic algorithm to find better solutions. The size of the population is usually fixed.

Fitness Function

The fitness function provides a measure for the goodness of the individuals of a population with respect to the optimisation problem. The goodness of an individual is the main criterion for the quality-driven selective decisions in genetic algorithms. Before the reproduction process the quality of each member in the population is evaluated in the context of the problem using the fitness function. The optimisation problem is described mathematically in the so-called objective function.

The simplest way to state the fitness of an optimisation problem is to use the objective function. In this case the fitness of a string is associated with the value of the objective function of this string. The main disadvantage of this method is that it cannot discriminate very well between good and bad chromosomes when the population converges to a set of similar chromosomes. Scaling or ranking procedures are superior in this case. Extending the fitness function by a so-called penalty function excludes infeasible solutions in a population from the reproduction process.

Genetic Operators

A set of genetic operators is applied to a population in order to generate a new and better generation of possible solutions. These operators are probabilistic and operate according to the “survival of the fittest”

principle. The following three operators are most frequently used. For complex problems further genetic operators were proposed which are described in connection with specific applications in section 4.3.

Cross-over

As in natural systems heredity is the motivation for cross-over. Useful features, that have been adapted in previous generations, shall be transmitted from the parents to the children. Once the members are selected for the reproduction process, the cross-over operator is applied in order to produce new offspring. During the cross-over features of two or more parents are combined to generate one or more offspring. As in nature the cross-over involves an exchange of sections of the parents' chromosomes. Therefore one or more cross-over points are randomly selected along the genetic strings. Swapping the determined parts of the parents' strings generates the new offspring. The example of the one-point cross-over is shown in Figure 3.1.

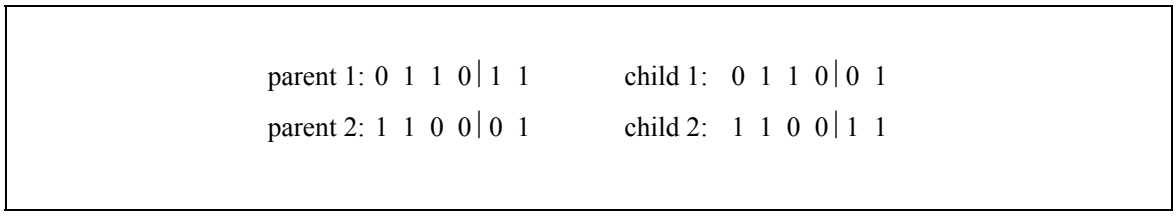


Figure 3.1: One-point cross-over

The rationale behind the cross-over operators is to produce new alternative solutions to the problem which are possibly better. As the parent chromosomes are usually the fitter individuals in the population according to the selection scheme and the creation of the new individuals is based on the combination of the their features, the idea is transfer good features to the next generation and achieve individuals which offer better solutions.

In cases where a more complex representation technique is used the cross-over operator needs to be adapted. The following operators are suitable in the recombination process when using a permutation as data structure to describe combinatorial problems (Goldberg, 1989).

- Partially Matched Cross-over: PMX crossover was developed by Goldberg (Goldberg, 1989); PMX can be implemented as two-point and one-point cross-over (Jakobs, 1996; Smith, 1985)
- Order Cross-over (OX) (Oliver et al., 1987; Goldberg, 1989)
- Order Based Cross-over (OBX) (Syswerda, 1991)
- Position Based Cross-over (PBX) (Syswerda, 1991)
- Cycle Cross-over (CX) (Goldberg et al., 1985)

Mutation

Mutation enables the recreation of genes, which are lost from the current population. This information cannot be gained if only the existing material is combined. The mutation operator is applied after the cross-over operation and randomly introduces minor modifications to the genetic strings of the

offspring. Mutation is implemented by altering the values of one unit in the genetic code with a certain probability. The principle of simple mutation is shown in Figure 3.2 using a binary coded string. The effect of this operator is to produce new unexplored areas in the search space and hence prevent the process from rapidly converging on a local optimum. Mutation therefore maintains the diversity of the population and allows a wider exploration. Since mutation also has a destructive effect as well it is only applied with a very low frequency.

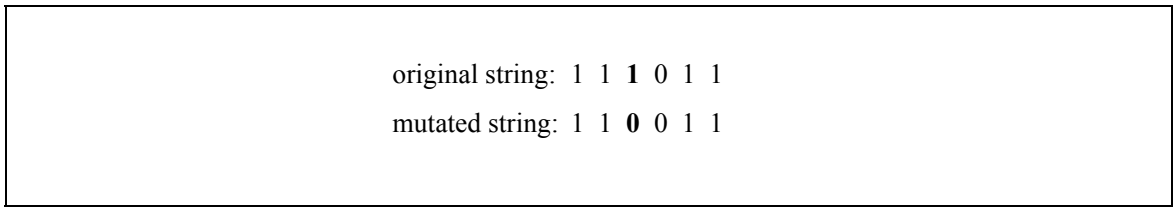


Figure 3.2: Principle of the simple mutation operator

Depending on the encoding other mutation operators can be superior to the simple one shown in Figure 3.2. In more complex encodings, usually an order relationship between adjacent loci on the strings (e.g. permutations) exists and reordering a sequence may be important. In this case the inversion operator can be applied, cutting out a section of a string and reinserting it in the reverse order (Goldberg, 1989). The following mutation operators were specifically developed for order-based encoding.

- Order Based Mutation (OBM) (Syswerda, 1991)
- Position Based Mutation (PBM) (Syswerda, 1991)

3.3.1.2 Generic Design Variables

The generic design variables concern the probability of the genetic operators and the reproduction scheme. The selection scheme and replacement strategy can be implemented in different ways.

Reproduction Scheme

During the reproduction process individuals are selected from the population and copied into a mating pool for the production of the next generation. The generation of the mating pool is a probabilistic process where the fitter individuals are more likely to receive more than one copy and the unfit ones are more likely to receive no copies. In the next step pairs of individuals are taken out of the pool and mated at random. The new generation can either replace the previous generation entirely (generational replacement) or partially (steady-state model).

With the selection of the strings taking place randomly according to their fitness, individuals with a higher value have a higher probability of contributing one or more offspring. The probability function that determines the selection process can reflect the actual fitness value, a scaled value or the rank. The reproduction of fit individuals provides the pressure for improvement. At the beginning of the search the values for each gene of an individual are randomly distributed. This produces a wide spread of

fitnesses. During the progress of the search certain values for each gene start to dominate. With the population converging, the range of fitnesses of a population reduces.

Proportional Selection

The proportionate selection technique chooses an individual for the reproduction process on basis of its objective function value. The probability of selection of an individual is proportional to its fitness. This technique can be implemented in different ways. The simplest one is known as roulette wheel selection, because it simulates the spin of a roulette wheel. Each string of a population is allocated a slot sized in proportion to its fitness. Spinning the weighted wheel corresponds to generating a random number and selecting a parent. As a fitter individual has a higher chance of being selected it thus contribute more copies to the mating pool.

In order to prevent premature convergence the reproductive trials should not be allocated in direct proportion to the raw fitness. Alternative methods re-map the raw fitness onto a new scale and include fitness scaling, fitness windowing and fitness ranking. Some of the most important selection techniques are briefly described in the following.

Rank Selection

If the fitness of some strings becomes very large, the fittest individuals can dominate the recombination process. This may lead to a loss of the genetic material. In order to prevent this reduction of genetic diversity ranking selection can be applied. After sorting the individuals according to their raw fitness value reproductive fitness values are assigned according to rank: linear (Baker, 1985) or exponentially (Davis, 1989). A parent string is selected with a probability proportional to its rank rather than the absolute fitness value. Consequently, the ratio between maximum and average fitness is normalised to a particular value. Hence the effect of one or two extreme individuals will be negligible. According to experiments carried out by Baker (1985) and Whitley (1989) ranking is superior to proportionate selection.

Scaling Mechanisms

Scaling mechanisms are applied to maintain a certain level of competition throughout the search process. Since there is a tendency that a few “super-individuals” may dominate the search process in the first generations, a take-over can be avoided by scaling down their fitness values. At the later stages when the population converges scaling up helps to differentiate between individuals with otherwise similar fitness values. Several scaling techniques exist including linear scaling and sigma truncation (Goldberg, 1989).

Termination criterion

Genetic algorithms are an iterative search technique. Being heuristic a convergence on the optimal solution cannot be guaranteed. Hence criteria for the termination of the process need to be stated. This

can be done by fixing the number of generations for the search process. An alternative method is to terminate when the process converges on an optimum, i.e. when no more improvements have been achieved for a specified number of generations.

3.3.1.3 A Brief Description of the Algorithm

Figure 3.3 shows the general description of a simple genetic algorithm. After the creation of the initial population the processes of reproduction, cross-over and mutation are repeated until the termination criteria is reached. The actual population at this stage represents the solutions to the problem. The characteristics of the genetic algorithm method are briefly summarised below.

Characteristics of genetic algorithms:

1. random choice as tool to guide, but not random search
2. searches from population of points, not a single point
3. uses objective function, not auxiliary knowledge
4. probabilistic, not deterministic transition rules

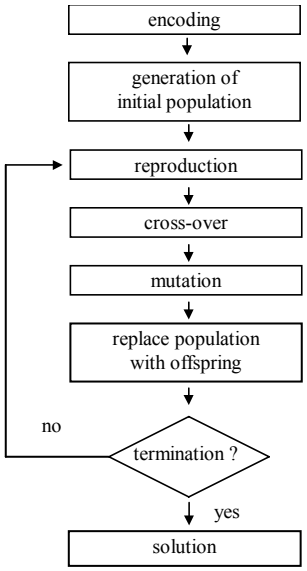


Figure 3.3: Flow diagram of genetic algorithm

3.3.2 Naïve Evolution

The rationale for naïve evolution is the same as for genetic algorithms. The algorithm operates according to the same principle, but does not apply cross-over to manipulate the search space. Only the mutation operator is used for the generation of the next population. A naïve evolution algorithm can be used to test the efficiency of the crossover operator in a genetic algorithm. Falkenauer (1998) applied this technique in experiments on 1D bin packing problems. Naïve evolution has been used throughout this work to establish the performance of the cross-over operators implemented in various approaches to strip and bin packing problems.

3.3.3 Simulated Annealing

Eglese (1990) investigated the application of simulated annealing as a tool for Operations Research. A number of researchers applied it to packing problems (section 4.4.1). Simulated annealing was introduced as an optimisation tool in the 1980's when the concept of physical annealing was first applied in combinatorial optimisation.

In a thermodynamic system low energy configurations of a solid are achieved by initially melting the substance and subsequently decreasing the temperature slowly. If the cooling process takes place too fast, the configuration does not achieve its lowest energy state and will be frozen into a locally optimal structure. Transferring this model to combinatorial problems the energy states correspond to the various feasible solutions and the energy of the system to the cost function to be minimised.

Simulated annealing can be seen as a variant of hill-climbing. Instead of only accepting neighbouring solutions that result in an improvement, also inferior solutions may be accepted with a certain probability. This probability depends on the increase in cost and a control parameter, i.e. temperature in physical annealing. The temperature refers to the analogy with physical annealing. The smaller the increase in the cost and the higher the temperature the more likely uphill moves will be accepted.

During the annealing process the temperature is gradually lowered according to the cooling schedule. This means the algorithm becomes more and more selective in accepting new solutions. At the end of the process only moves, which result in an improvement are accepted in practice. The search process terminates when it arrives at a lower bound for temperature or cost.

3.3.3.1 Problem-specific Design Variables

The problem-specific design variables describe the characteristics of the problem and consist of:

- representation
- fitness
- initial solution
- neighbourhood: i.e. set of states that can be reached from the current state

3.3.3.2 Generic Design Variables

The generic choices for the implementation of a simulated annealing algorithm are summarised in the annealing or cooling schedule and include:

- initial temperature
- length of Markov chain: number of moves the temperature is held constant based on the number of successful and unsuccessful moves
- decrement rule: mathematical expression describing how the temperature is lowered
- termination criterion: number of iterations, time, convergence, unsuccessful moves

4. Automated Packing – State of current Research

Over the last 30 years interest in packing and cutting problems has been very large. This is reflected by the great number of publications in this area. Cutting and packing problems arise in various industrial applications and are not restricted to the manufacturing industry. Packing problems for instance are encountered in operational research and the financial sector in a more abstract form. In terms of solution methods a number of approaches were proposed depending on the type and the size of the problem. For less constrained, simpler packing tasks exact algorithms were developed along with problem-specific heuristic procedures. For more complex packing tasks, i.e. irregular problems heuristic search methods have been applied successfully for their solution. Their success can be explained by the great flexibility in taking into account problem-specific constraints. They also offer a good trade-off between solution quality and computational effort regarding the size of the search space.

Since cutting and packing is an important issue in industrial applications, a substantial number of commercial nesting packages have become available recently. They are specially designed to meet industrial requirements and usually include a variety of features directed at the manufacturing process.

The following review considers only work on concrete packing problems in the area of 2D rectangular and irregular strip packing as well as bin packing. Particular emphasis is hereby put on solution approaches involving genetic algorithms. A brief introduction into techniques for 1D and 3D packing problems is also given. With regard to the solution methods, the research activities on other meta-heuristic and heuristic search techniques are also included. In addition, an overview of commercial software packages in the area of 2D and 3D cutting and packing is given.

4.1 Surveys and Reviews

Over the years a considerable amount of literature on cutting and packing problems has been presented. A number of reviews tried to keep track of the recent developments in this area - a task that is virtually impossible due to the size of the subject. Despite the vast extent of the literature, two surveys attempt to cover the total area of cutting and packing. Dyckhoff and Finke (1992) developed a classification method, on which they base their analysis of the concrete and abstract packing problems. Sweeney and Paternoster (1992) chose the opposite approach and addressed the subject from the perspective of the solution approach. The problems encountered in the literature are grouped into three classes according to their solution methodologies consisting of sequential assignment heuristics[1], single-pattern

[1] set of rules determining the order and the orientation of the items

generating procedures[2] and multi-pattern generating procedures[3]. Their work covers more than 400 problems including books, dissertations and working papers and is the most exhaustive bibliography published in this area to date. Table 4.1 provides an overview of the more recent reviews and surveys in the area of concrete packing problems.

Table 4.1: Reviews and surveys on packing problems in the literature

authors	**topic**	**classification of packing problems**
Dyckhoff and Finke (1992)	analysis of large variety of problems	Dyckhoff classification
Sweeney and Paternoster (1992)	more than 400 problems including books, dissertations and working papers	dimension, solution methodology, special topics
Golden (1976)	2D cutting stock problems	solution methodology
Hinxman (1980)	2D trim-loss and assortment problems	dimension, solution methodology
Rayward-Smith and Shing (1983)	1D and 2D bin packing	dimension
Sarin (1983)	2D cutting stock problems	solution methodology
Coffman et al. (1984)	bin packing	type of bin packing, dimension
Dowsland (1985)	2D and 3D rectangular problems	problem type, dimension
Coffman and Shor (1990)	2D regular packing problems	on-line, off-line; probabilistic analysis
Haessler and Sweeney (1991)	1D and 2D cutting stock problems	dimension, solution methodology
Dowsland (1991)	3D problems	solution methodology
Dowsland and Dowsland (1992)	2D and 3D packing problems, mainly regular	problem type, dimension
Whelan and Batchelor (1993)	industrial implementations of automated packing systems for 2D irregular packing problems	application, focus on leather industry
Dowsland and Dowsland (1995)	2D, irregular packing problems	methods for clustering, packing, computational geometry
Hopper and Turton (1997)	2D and 3D, regular and irregular packing problems and genetic algorithms	geometric characteristics of items, dimension

4.2 Application of Heuristic Algorithms to Packing Problems

A number of exact and heuristic algorithms were proposed for regular packing problems. Some researchers developed exact methods, which solve 1D and 2D problems of moderate size to optimality such as the linear and dynamic programming models used by Gilmore and Gomory (1961, 1963 and

[2] dynamic-programming based algorithms, which attempt to reapply a single 'optimal' pattern configuration

[3] linear programming based algorithms, which consider interactions between pattern; the solutions are approximated, hence heuristic

1965). The majority of packing problems are constrained and hence more complex. With respect to practical packing tasks arising in industrial applications, the solution quality is not always the major issue. It is traded against ease of implementation and computational cost. Near-optimal layouts achieved in reasonable time are usually sufficient for industrial needs. Heuristic methods, which provide fast solution approaches, were therefore studied intensively in the literature.

4.2.1 Regular Packing Problems

A significant amount of research is devoted to regular packing problems. Most frequently rectangular packing has been investigated. Dowsland's (1985) survey provides an excellent overview of exact and heuristic algorithms for the 2D and 3D rectangular packing tasks. This review was extended by Dowsland and Dowsland (1992) to address a number of bin packing problems.

4.2.1.1 1D Strip and Bin Packing Problems

In 1D packing problems the width of the items is equal to the object width. Hence only a single dimension is significant for the packing process. Several variants of this problem are encountered in the literature covering bin packing (Coffman et al. 1984), knapsack problems (Rönnquist, 1995) and strip packing problems (Dyckhoff, 1981; Gilmore and Gomory, 1961, 1963). Typical industrial applications concern the cutting of paper, pipes and wood. More abstract problems in this area deal with process scheduling, timetabling, processor allocation and file distribution.

Since 1D problems have a small solution space compared to higher dimensional ones they can be solved to optimality at reasonable computational cost. Exact methods, which were successfully applied, are linear and dynamic programming and branch-and-bound techniques. A number of heuristic algorithms for larger strip packing as well as bin packing problems were used such as the First Fit (FF), Next Fit (NF) and Best Fit (BF) (Rayward-Smith and Shing, 1983; Coffman et al., 1984). As the name suggests the FF-routine places an item in the first bin with sufficient space. Whereas the FF-rule searches the list of bins from the beginning, the NF-algorithm starts at the previously used bin and continues with a new one if the item does not fit. The BF-algorithm searches through all available bins and chooses the one that results in the highest performance defined by an evaluation function. In particular, pre-ordering the items according to decreasing height in combination with the FF- and NF-algorithms improves the average performance of the simple placement routines. These routines are referred to as FFD and NFD respectively. Figure 4.1 and Figure 4.2 illustrate the operation of the FFD- and NFD-algorithms placing a sequence of items, numbered from A to G. After sorting them by decreasing height, the sequences are allocated with the FF- and the NF-routine to the bins.

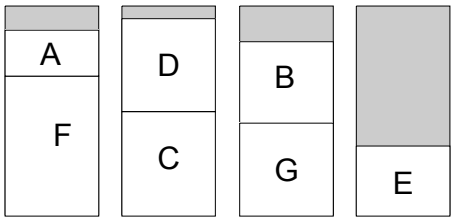


Figure 4.1: FFD-rule (First Fit Decreasing)

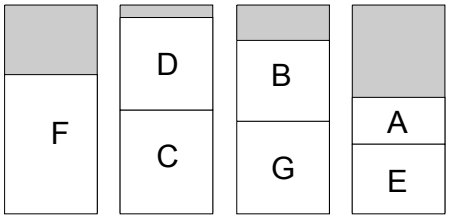


Figure 4.2: NFD-rule (Next Fit Decreasing)

4.2.1.2 2D Strip and Bin Packing Problems

The majority of the literature in this area concentrates on 2D rectangular packing problems. 2D packing tasks are either concerned with packing items into an object of fixed width minimising its height (Bengtsson, 1982) or into objects of fixed size minimising of the number of bins (Bengtsson, 1982; Berkey and Wang, 1987; Frenk and Galambos, 1987).

Many heuristic algorithms have been developed over the years to solve this problem. A fairly simple approach is the Bottom-Left-rule (BL), which places an item as near to the bottom of the strip as it will fit and then as far to the left as it can be placed at that bottom-most level. A number of placement rules which belong to the class of algorithms preserving bottom-left stability in the layout are described in section 5.5.4.1 to 5.5.4.4. A different approach is taken by the so-called level-algorithm, where the rectangles are placed in different rows of the strip in a left-justified manner. The packing is constructed as sequence of levels, each rectangle being placed so that its bottom side rests on one of these levels. The height is determined by the tallest rectangle at that level. The level technique is then combined with 1D rules like FF, NF or BF to determine at which level the next item is positioned. Again, pre-ordering according to height or width (e.g. FFDH or FFDW) can improve the average performance (Coffman et al., 1984). Figure 4.3 and Figure 4.4 highlight the operation of the FFDH and NFDH-algorithms, which apply a height-sorted sequence in combination with the FF-rule and the NF-rule respectively.

Level oriented methods are also used in two-stage algorithms for the bin packing problem such as the FFDH * FFD, where the FFDH is first used to solve a strip packing task which serves as basis for the subsequent bin packing process with the FFD. This method is described in greater detail in section 4.3.2.3 where it was used in connection with a genetic algorithm.

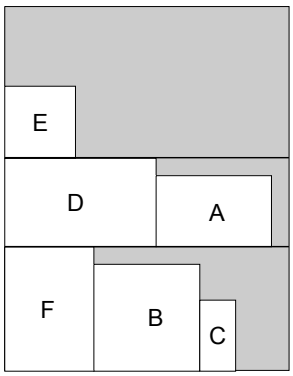


Figure 4.3: FFDH (First Fit Decreasing Height)

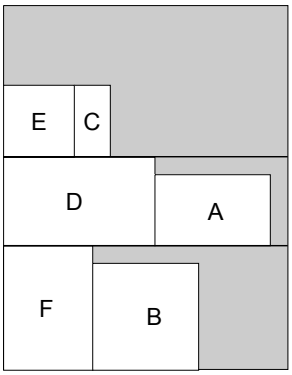


Figure 4.4: NFDH (Next Fit Decreasing Height)

Regarding industrial applications, the concept of bin packing is found in rectangular stock cutting problems, where an order list has to be assigned to a supply of stock sheets. These problems arise frequently in manufacturing processes, which are concerned with the cutting of large and flat rectangular objects as in the glass and wood industry. A number of publications introduced solution approaches for the two variants of the stock packing problem, the trim loss problem (Adamowicz and Albano, 1976b; Albano and Orsini, 1980; Wang, 1983; Rode and Rosenberg, 1987; Riehme et al., 1996) and the assortment problem (Beasley, 1985; Yanasse, 1991). Industrial packing techniques need to accommodate more constraints than the general 2D packing task. Layouts usually have to be guillotineable, which is the case in all examples quoted above, and may contain defective areas, like in the glass industry, which have to remain unpacked (Hahn, 1968).

Pallet loading can be regarded as bin packing, where a number of boxes have to be packed onto a pallet (Bischoff et al., 1995). The classical problem, the so-called manufacture's pallet loading problem only deals with identical boxes (Smith and De Cani, 1980; Dowsland, 1984; Scheithauer and Terno, 1996), for which Dowsland (1987) proposed an exact algorithm.

Some special packing problems like the classical pallet loading problem or moderate guillotineable packing problems were solved with exact algorithms using linear programming techniques (Gilmore and Gromory, 1963, 1965) and tree search algorithms (Hadjiconstantinou and Christofides, 1995; Christofides and Hadjiconstantinou, 1995). Most of the 2D packing tasks, however, were approached by heuristic techniques.

4.2.1.3 3D Strip and Bin Packing Problems

3D packing problems occur less frequently in the literature. Container and pallet loading problems are mainly concerned with the packing of rectangular boxes, which can be either identical (George, 1992) or non-identical (Bischoff and Marriott, 1990). In the general case, certain parallels can be noticed between pallet loading and container loading problems. Once practical constraints are involved such as stability (Carpenter and Dowsland, 1985), pallet and container loading have to be treated as two different

problems. Instead of volume utilisation, the objective may be the maximisation of the value of the boxes, in which case the problem can be formulated as a knapsack problem (Mohanty et al., 1994).

In terms of solution approaches many algorithms transfer the task into a 2D problem applying layer-by-layer strategies. Due to the enormous size of the solutions space, only heuristic approaches are applied to the 3D packing problem in the literature. The methods range from simple heuristics such as the 3D First Fit and the 3D Next Fit (Corcoran and Wainrwright, 1992) to meta-heuristic algorithms, some of which are described in section 4.3.4.

4.2.2 Irregular Packing Problems

Irregular packing tasks occur in the manufacturing processes of the leather, shipbuilding and sheet metal industry dealing with different types of irregular shapes. Whereas in the shipbuilding industry many items have rectangular features, packing problems in the leather industry are concerned with highly irregular shapes. Irregular items and objects increase the complexity of the packing process. Since this additional complexity is highly dependent on the nature of the shapes, irregular packing tasks can be distinguished according to the types of irregular items involved. Whereas most packing techniques use rectangular or polygonal approximations of the items (Adamowicz and Albano, 1976a; Nee et al., 1986), others like commercial nesting packages are capable of true-shape nesting (section 4.5).

Despite the irregular features of the items, the complexity of the packing task is determined by the degree to which the irregularity of the shapes is considered in the basic packing strategy. This refers to the representation of the problem. Irregular problems can be easily transformed into regular ones simplifying the packing process to a great extent. Approximating irregular items by simpler geometric shapes like rectangles reduces the complexity of the packing task. The complexity of the overlap computation for instance strongly depends on the representation method. The shape description is therefore one of the key elements in irregular packing problems.

The techniques used for shape description depend very much on the demands of a given application and its underlying objectives. The question of shape representation has to be seen together with the placement strategy as it influences the type of placement rules to be applied and vice versa. Due to this mutual influence the irregular packing problems reviewed in the following section are categorised by the shape description used rather than the type of the original shape and the main placement strategy. In the majority of problems the items are approximated by rectangles and polygons representing the irregular shape by a sequence of segments. Using a higher number of segments can increase the accuracy of the approximation. Other techniques are grid approximation and quad-tree representation. The major goals of these geometric representations are to reduce the time for overlap and intersection computation and the complexity of the nesting algorithm. With respect to the application in packing, they need to be regarded as a trade-off between accuracy and computational effort.

4.2.2.1 Packing of Rectangular Modules

Many irregular packing problems are transformed into rectangular ones by enclosing arbitrary shaped items by rectangles, because simple, existing algorithms can then be used in the packing process. Some researchers studied the sub-problem of finding the minimum enclosing rectangle. Freeman and Shapira (1975) developed an algorithm that computes the minimal rectangle for an arbitrary closed curve. Martin and Stephenson (1988) proposed a number of algorithms, which pack 2D items such as arbitrary polygons and curved objects into rectangles.

Haims and Freeman (1970) applied the minimum enclosing rectangle concept to a cluster of irregular items. Before the packing stage, up to eight irregular shapes are clustered and circumscribed with the minimum enclosing rectangle. These modules are then optimally packed into the object by means of dynamic programming.

Adamowicz and Albano (1976a) addressed the problem of nesting irregular items into rectangular modules. They introduce a technique that allows the efficient clustering of two polygons by calculating the No Fit Polygon (NFP). The NFP was originally presented by Art (1966) and describes all possible locations that an orbiting polygon (P2) can take with respect to a fixed one (P1) so that the polygons touch but do not overlap (Figure 4.5). This technique can be used to find the minimum enclosing rectangle of two polygons. The authors applied this method to cluster two and more polygons that are then packed on basis of the minimum enclosing rectangle approach. The ideas in that work were later developed further by Albano (1977) in the form of an interactive packing system that allowed the operator to manipulate the proposed layout. The rectangular modules are first grouped then allocated on the rectangular object using dynamic programming.

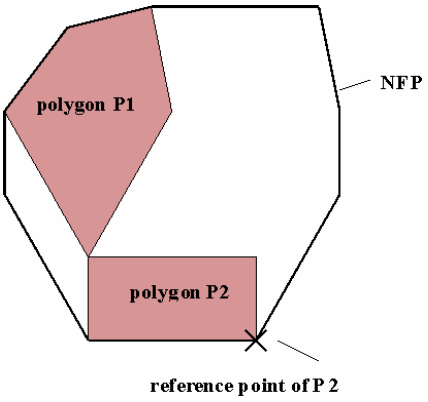


Figure 4.5: No Fit Polygon for two polygons

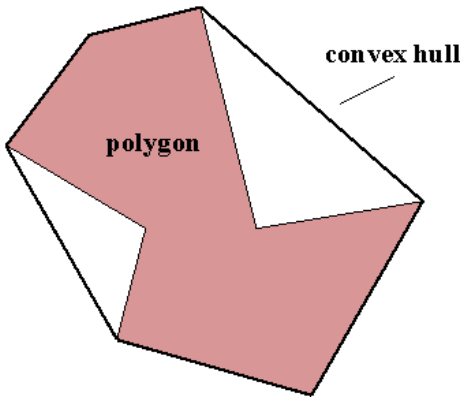


Figure 4.6: Convex hull of a concave polygon

Dagli and Tatoglu (1987) solved an irregular packing problem involving multiple objects. Using a two-stage approach the irregular part of the packing task is simplified to a problem only dealing with single objects. In the first step, the irregular items are enclosed by rectangles and allocated to objects applying mathematical programming techniques. Based on this initial allocation they are then packed by a heuristic procedure on the respective objects. After selecting an item according to a set of priority rules it is moved towards the cluster of previously placed items. By repetitively matching the sides of the two figures and rotating the new item, the algorithm searches for the configuration, which has the smallest enclosing rectangle. Although this procedure may achieve good layouts, it is computationally inefficient for irregular figures with a large number of sides.

The ideas developed by Nee (1984a, 1984b) regarding the pairwise clustering of blanks, were used by Nee et al. (1986) for the nesting of irregular items. Arbitrary items are approximated by polygons and pairwise clustered, if they are repetitive in small batches, before they are circumscribed by the minimum enclosing rectangle. Since pairwise clustered modules may not always be advantageous for the subsequent packing process, the clustering procedure is only carried out if a certain utilisation ratio is achieved. A subsequent rectangular packing procedure places the modules in both orientations onto the object trying all valid Pivot points, which are created by neighbouring boundaries of the already placed rectangles or the object. The non-intersecting configuration, which produces the smallest enclosing rectangle with its neighbour, is finally selected. The list of modules is sorted by decreasing area.

Cheok and Nee (1991) chose a rectangular packing approach for the automatic nesting of ship/ offshore plates, which consists of three steps. The first one is concerned with the shape representation, which approximates the items by a series of line segments ignoring minor features such as drain holes and fillets. In the next step the items are grouped into a number of classes according to their geometric properties as well as their relative sizes and void areas. The authors have developed two separate local optimisation processes. The first one is directed at the clustering of congruent items according to predefined arrangements, since in particular ship/ offshore plates are symmetrical about the centre line and come in pairs. After the clustering process the new figure is enclosed by a rectangle. The second local optimisation process attempts to place smaller items in the void areas of the enclosing rectangle of the larger ones. In the last stage of the packing system, the rectangular modules are allocated onto the object according to a rectangular placement procedure applying various sorting criteria to the sequence of rectangles. The classification of the shapes and the size in this approach helps to select the most appropriate local optimisation process and therefore contributes significantly to the efficiency of the search process.

Since in the shipbuilding industry a high percentage of the items have rectangular features, rectangular packing strategies may be applied in order to reduce the computational effort. Qu and Sanders (1987) use composites of rectangles to describe the irregular items decomposing them into maximal five non-overlapping rectangles. The system orientates each part such that its length is larger than its height and its largest complimentary void area is at the upper right corner. After sorting the list according to the height

of the parts, the items are placed onto the object according to a bottom-left procedure, composing the layout from the lower-left corner to the top-right one. Since the height of the next item cannot exceed the one of its left neighbour, the series of placed items can be enclosed by a stair-shaped line. At the same time the algorithm attempts to fill the blank areas below this line testing these positions first. When no more parts can be fitted within the stairs, the layer determined by the height of the first item is filled, before a new layer is started.

One of the main advantages of the shape representation of this approach is that the complexity of the arbitrary shapes is reduced to rectangles. Since their geometric properties can be stored efficiently, the computation of the layouts is fast. At the same time this description is very flexible hence if necessary a higher accuracy can be achieved by increasing the number of rectangles in the shape approximation. This approach offers many advantages for packing problems that involve a high percentage of rectangular items. For problems that consist mainly of highly irregular shapes the approximation technique and the orthogonal manner in which the layout is built have disadvantages.

4.2.2.2 Packing of Polygons

Some approaches to irregular nesting problems attempt to find efficient clusters of two or more identical or different shapes, which are combined to form larger polygons. These configurations can then be packed onto the object by a placement procedure. With respect to that, Grinde and Cavalier (1995) developed an algorithm that finds the convex enclosure with the smallest area for two concave polygons. The majority of solution approaches to irregular nesting use heuristic techniques which model the problem as a search tree and apply a placement strategy to allocate an item in the partial layout. The tree structure is a graphical representation of the numerous solutions consisting of permutations. The objective is to find the best path from the initial state to the goal state. At each node, a set of heuristic rules is applied to decide which branch to expand next taking into account the layout information. By expanding only the best node at each level the selection process eliminates the majority of permutations. A different field of research in the area of nesting focuses on layout compaction of existing layouts as opposed to layout generation.

In particular in the metal cutting industry, many cutting tasks are concerned with the nesting of congruent items, which are small compared to the object size. This problem is referred to as blank nesting. A possible approach is to find an appropriate description of the item such that it can be placed repeatedly on the infinite object without leaving any gaps or overlaps. This technique is called paving or tiling. Dori and Ben-Bassat (1984) developed a two-stage technique for the efficient nesting of congruent convex figures, which circumscribes an item by a convex polygon with a sufficient number of sides. In the second step, the convex polygon is enclosed by another polygon that is suitable to tile the plane. Koroupi and Loftus (1991) later extended this approach to concave polygons. The objective is to find a paving polygon, which adds as little area as possible to the original shape.

A different approach towards the nesting of congruent figures is taken by Nee (1984a, 1984b). Instead of constructing paver polygons the items are pairwise clustered before they are placed onto the object. A number of configurations are tried rotating the polygons at various angles and placing them side by side. The solution with the highest utilisation ratio is adopted taking into consideration a number of constraints such as bridge width. Various aspects of metal cutting to be considered for the layout generation are highlighted. Prasad and Somasundaram (1991) and Prasad et al. (1995) addressed the blank nesting problem. Their nesting algorithm is based on a sliding technique and finds all feasible arrangements of two items. This technique can also be applied to the nesting of three shapes. After clustering two selected blanks their boundary is traced and then clustered with a third item. The paper by Prasad and Somasundaram (1991) also contains a comprehensive list of practical constraints to be considered in packing systems for the sheet metal industry.

Albano and Sappupo (1980) developed a tree-search approach to address a problem from the textile industry. The items are placed by a leftward placement strategy allocating them to the right of the current profile through the construction of the No Fit Polygon. This single pass algorithm is improved by backtracking when the current solution path produces less desired results than a previous node. Since it is impossible to search the whole solution space, two bounds are used to reduce the increasing number of expanded nodes. The first one describes the quality of the partial layout and measures the true waste in the area left of the profile. For the second one, the potential waste, which will be generated by the remaining items, is estimated using a fixed proportion of their area. The branch that minimises both bounds is chosen next. The search tree is pruned restricting the number of options at each node by a set of rules. Each remaining part generates only one branch at each node. Only a fixed number of branches are explored and backtracking is only allowed to nodes that are significantly further up the tree.

Amaral et al. (1991) developed a sliding algorithm for a problem from the textile industry. This algorithm moves a new item towards the polygonal profile of the partial layout until a collision occurs. Several iterative sliding steps follow depending on the type of collision and the collision distances, until an appropriate position is found. The layout is built up from the left to the right. The intersection tests in this algorithm are based on the concept of D-functions that reduce them to a number of arithmetical operations proportional to the product of the number of sides of the polygons tested (Dowsland et al., 1998). For the selection of the next shape an interactive as well as a heuristic procedure is introduced. In the heuristic technique, the choice of the next shape is related to the analysis of the current profile. An appropriate item is then chosen ensuring that the larger items are placed first in the leftmost placement area. The layout quality could be improved through pairwise clustering of identical shapes.

Unlike the steel industry, which mainly involve rectangular stock sheets, the leather industry deals with objects (hides) and items that are highly irregular and can have defects and various quality levels. The fast solution method developed by Heistermann and Lengauer (1993) approximates the irregular figures by polygons, which are partitioned into various polygonal quality zones. Consequently, each point of an item must be placed on a point in the hide of the same or higher quality. In contrast to placement policies that

fill an object from a certain direction (e.g. leftward-strategy) the authors place a new item along the contour of the unfilled hide. The location for the next placement is called focus and is selected on basis of several parameters such as smoothness of the contour, curvature and distance to defects. The profile of the items is coded by a set of parameters and grouped into several topology classes. A number of items are chosen for the next placement according to their suitability for placement on the current focus. The item that best fits contributing the least to the added waste is finally positioned on the object. The authors claim that their method outperforms other comparable software in this area.

Lamousin et al. (1996) adopted Albano and Sappupo's (1980) heuristic search approach for the nesting of irregular items. In order to deal with the complexity of industrial shapes they applied an efficient representation technique and developed a new placement strategy that allows packing into void areas of larger items. In terms of shape representation the authors used several modification steps in order to obtain a simplified profile of the arbitrary shape. The convex hull serves as first approximation (Figure 4.6). Since this approximation introduces considerable wastage to certain types of shapes (e.g. L-shapes), concave regions of the shape are incorporated in the simplified profile if they are large compared to the area of the part. The placement strategy constructs the No Fit Polygon (NFP) to find a feasible allocation in the partial layout using leftmost, lowest strategy. For the nesting of voids the authors develop a technique called the internal NFP, which determines all feasible locations for a polygon inside void regions.

Oliveira et al. (2000) developed a heuristic search method using the No Fit Polygon to determine a set of the feasible positions in the layout. The best allocation is selected according to three nesting strategies minimising area, length or overlap between the rectangular enclosures of the two items to be nested. In each iteration several items in several orientations are considered for placement. The process of adding a new item to the layout is controlled in two different ways. The pieces are either ordered according to a sorting criterion and tried in all orientations or are tried in all orientations according to the chosen nesting strategy.

Unlike the heuristic search strategies described above, Dowsland et al. (1998) presented a "jostle" algorithm, which was inspired by granular products. When stored in a container any unevenness in the surface can be removed by shaking the container up and down. Simple pass algorithms, which use a set of pre-determined rules to pack a sequence of items towards one end of the stock sheet with a fixed width, often generate layouts with a jagged profile at the open end of the bin. Less area will be required for a layout that has a flatter profile. The jostle method models the shaking process by a series of repetitions of the leftmost placement strategy. After the items have been placed according to a leftmost placement policy they are re-ordered in decreasing order of their rightmost points and packed according to a rightmost placement strategy. In the next iteration the selection is done by the leftmost points of the parts and the placement policy is again the leftmost one. This process is continued for a fixed number of times. The authors used a modified bottom-left placement strategy, which fills holes behind the current profile.

Results showed that the performance of the method depends on the characteristics of the items, but is very effective for a variety of different problems.

Li and Milenkovic's (1995) work concentrates on a different aspect of nesting problems in the cloth industry, i.e. compaction of existing layouts. The efficiency of manually generated layouts can be increased by shortening the length of the layout, removing the overlaps in an overlapping layout and moving the items such, that larger void areas are created. Two types of motion can be distinguished, the compaction which will only lead to feasible (non-overlapping) configurations and separation, which transfers an overlapping configuration into a non-overlapping one. The authors discuss two optimisation models for the compaction and separation tasks.

4.2.2.3 Packing Based on Grid Approximation and Quad-Tree Representation

The common feature of the packing techniques discussed is their approach to the shape representation. Prior to the packing process, the item is approximated by geometric figures like polygons or rectangles. The shape is then described by a set of geometric entities such as vertices and angles. The number of vertices determines the resolution. In contrast to this method are shape representation techniques, which store the digitised image of an item as a set of pixels. Although this type of representation can be computationally expensive, it is advantageous for the description of highly irregular figures. In some cases it can be very difficult to elaborate a suitable set of geometric parameters that sufficiently describe the geometric properties of a shape for further processing. It allows the description of the geometric features of a shape with less loss of information as when the enclosing rectangle or the convex hull is applied to concave polygons. The resolution of the approximation is determined by the grid size.

Batchelor (1991) described a technique, which digitises the object in order to determine a suitable position for the next item. The arbitrary items are circumscribed with their minimum area bounding rectangle prior to packing. During the packing process the object is scanned until a position is found which is sufficiently large to place the rectangle without causing overlap. Due to the scanning procedure this method is able to pack smaller shapes between larger ones. This technique achieves denser patterns than the conventional rectangular placement strategy, which positions the rectangles rather than the original shapes.

Whelan and Batchelor (1991, 1992) and Whelan (1996) extended the ideas developed in Batchelor's (1991) approach to solve a nesting problem from the leather industry. The digitising technique is used to describe highly irregular objects as well as items. Since the search for a feasible location based on the enclosing rectangle contains some limitations (Batchelor, 1991), the authors have developed a packing method which makes use of computer vision techniques namely mathematical morphology. The morphological transformations allow to determine the feasible placement region and to describe the new partial layout again as a digitised image. The placement policy works on a sorted list of rectangles. Since this technique allows items to interlock it achieves higher packing densities than the one developed earlier

(Batchlor, 1991). Prior to placement, a heuristic component is implemented that determines the orientation and the order in which the items are positioned. The heuristic algorithm distinguishes between polygon and "blob" shapes. The ordering and orientation rules for the blob shapes mainly concern the circularity and size of concavities, whereas the polygons are ordered according to their size and are aligned prior to positioning such that their main direction corresponds with the main direction of the unpacked region of the object.

Han et al. (1997) developed a special part decomposition procedure to reduce the overlap computation time at the nesting stage. After calculating the minimum enclosing rectangle of the item, the scrap areas outside as well as holes inside the polygon are encoded using a quad-tree technique. The nesting procedure consists of two main heuristic algorithms using a SOAL (self-organisation assisted layout) to generate an initial layout (section 4.4.3) and simulated annealing for compaction (section 4.4.1).

4.3 Application of Genetic Algorithms to Packing Problems

A considerable amount of research has been carried out to develop algorithms for the solution of packing problems. Deterministic methods such as linear programming techniques have been developed to obtain exact solutions. Since exact algorithms only work efficiently up to a certain degree of complexity heuristic techniques are applied to more complex combinatorial problems. Apart from a vast number of problem-specific heuristics, that were proposed for certain packing problems, there are general heuristic methods like genetic algorithms, simulated annealing and tabu search. These algorithms are suitable for packing tasks as well as other combinatorial problems.

Although genetic algorithms were developed in the early 70s, it was not until the mid 80s that they were applied to packing problems. The first researcher who implemented genetic algorithms in this domain was Smith (1985) applying them to a 2D rectangular packing problem. At the same time Davis (1985) summarised the techniques for the application of genetic algorithms to epistatic domains using the example of 2D packing. During the last ten years various types of packing problems were approached ranging from regular to arbitrary shapes in two or more dimensions.

Complex epistatic problems are commonly approached by a two-stage procedure, a so-called hybrid genetic algorithm. The genetic algorithm manipulates the encoded solutions, which are then evaluated by a decoding algorithm transforming the packing sequence into the corresponding physical layout. Since domain knowledge is built into the decoding procedure the size of the search space can be reduced. The packing strategy for instance may only generate non-overlapping configurations, which restricts the search space to valid solutions only. The need for a decoding heuristic excludes certain information of the layout from the data structures the genetic algorithms operate upon. Therefore not all the information concerning the phenotype is available to the genetic operators and may therefore not be transferred to the

next generation. In the following review, the solution approaches using genetic algorithms are distinguished according to the classification described in the next section.

4.3.1 Classification of Packing Problems Approached with Genetic Algorithms

To date, a variety of packing problems have been approached by genetic algorithms. Most of the problems encountered in the literature are 2D strip packing tasks dealing with regular, mainly rectangular shapes. The complexity and the implementation of a genetic algorithm depend on the type of problem. In order to compare the quality of the solution approaches suggested in the literature the following review distinguishes between several problem types.

Since the geometric properties influence the complexity of the problem and the size of the search space, the various packing tasks are distinguished according to their geometric features. The spatial dimensions of the problem are also very important criteria. Both characteristics have been used to classify the solution approaches in the literature (Figure 4.7). The problems are grouped into regular or irregular packing problems. Whereas regular figures such as rectangles and circles can be determined by a few parameters (Dyckhoff, 1990), the term irregular applies to convex and concave polygons as well as highly irregular, arbitrary shapes.

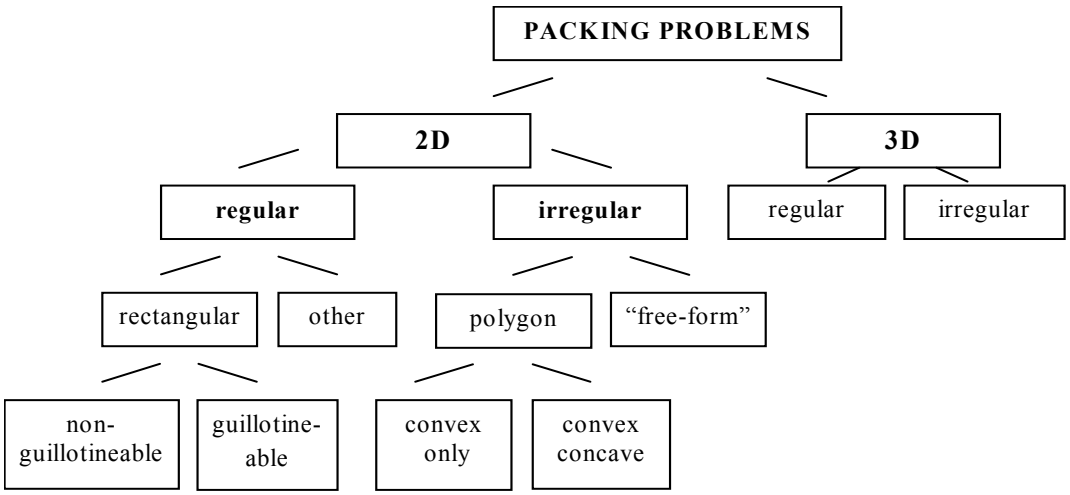


Figure 4.7 Classification of packing problems approached with genetic algorithms

The representation of the problem is the key to the efficient application of genetic algorithms to packing problems. The following review therefore puts particular emphasis on the description of the encoding technique and the genetic operators and summarises most of the problem-specific design variables in form of tables. Where available, the performance of the genetic algorithms in comparison to other solution approaches is briefly stated.

4.3.2 2D Regular Strip Packing Problems

The vast majority of the literature that concentrates on regular packing problems is concerned with the rectangular strip packing problem where a set of rectangles has to be allocated onto a rectangular object of unlimited height. In general, the packing of rectangular items distinguishes between guillotineable and non-guillotineable patterns. A modification of this problem is 2D bin packing that consists of multiple objects of fixed size. To date, only one approach has been described in the literature that uses other regular items. In all cases the aim is to find the arrangement of items producing the least waste either minimising the height of the object or reducing the number of bins.

4.3.2.1 Non-Guillotineable Packing Problems

Several researchers approached the non-guillotineable strip packing problem with genetic algorithms. Many of these methods are hybrid algorithms combining the genetic algorithm with a placement routine. In this two-stage approach a genetic algorithm finds the sequence, in which the items are to be packed with the aid of a placement routine (Table 4.2 to Table 4.5). A second group of genetic methods incorporates more layout information into chromosomes using a tree structure (Table 4.6). Some research concentrated on an entirely different genetic approach, which works without encoding, but manipulates the figures in the 2D layout directly.

Hybrid Approaches:

One of the first researchers who implemented genetic algorithms in the domain of packing is Smith (1985). He experimented with two heuristic decoding routines, one of which implements backtracking. The first one (Slide algorithm) places the rectangle in one corner from where it "falls" to the corner furthest away under orthogonal movements zigzagging into a stable position. The second procedure (Skyline) tries all stable positions in the partial layout. Comparisons between the two hybrid approaches show that the combination with the more sophisticated procedure generates better layouts, but is computationally more expensive. The performance of the genetic algorithms is compared with a packing method that is based on heuristics and dynamic programming. According to the author the genetic algorithms achieve the same packing densities in less time.

Jakobs (1996) uses the bottom-left heuristic (BL) to hybridise an order-based genetic algorithm. In order to reduce computational complexity the heuristic does not necessarily search for the lowest position available in the layout, but preserves bottom-left stability in the layout. Starting at the top-right corner of the object, each rectangle is moved as far as possible to the bottom and then the left in the partial layout. The initial population is seeded with a sequence in which the rectangles are sorted by decreasing width. During the reproduction process the worst individual in the population is identified and replaced with the offspring according to steady-state replacement. The hybrid concept of this genetic algorithm was extended to polygons using a modified placement rule (section 4.3.3.1).

Table 4.2: Hybrid genetic algorithms for non-guillotineable 2D packing problems

	Smith (1985)	**Jakobs (1996)**	**Liu and Teng (1999)**
problem	packing of single closed bin; 90° rotation	strip packing 90° rotation	strip packing 90° rotation
objective	maximise number of items in the bin	minimise height	minimise height
representation	permutation	permutation	permutation
fitness	ratio of packed to unpacked area	remaining area and height	remaining area and height
cross-over	OX (1point)	OX (1point)	OX (2point)
mutation	random reordering of string; rotation	inversion, swap of 2 elements, rotation	inversion, swap of 2 elements, rotation
decoder	Slide algorithm Skyline algorithm	BL-algorithm[4]	improved bottom-left algorithm[5]

The work by Liu and Teng (1999) was aimed at improving the decoder used by Jakobs (1996). The improved bottom-left routine is based on a sliding principle and gives priority to the downward shifting of the rectangle. The authors demonstrate the better performance of the new bottom-left placement routine using the two packing problems of Jakobs' work. The BL-algorithm as well as the improved version have been used for comparison purposes in this investigation and are described in detail in section 5.5.4.1 and 5.5.4.2.

The order-based approach using a bottom-left packing routine has attracted particular attention over the recent years. Leung et al. (1999) developed a placement routine, which preserves the bottom-left stability in the layout (Table 4.3). The improved BL-algorithm can access enclosed areas in the partial layout and is called 'Difference Process Algorithm'. Every insertion of a new item in the layout creates two empty rectangular spaces at its top and right side. The algorithm keeps track of the newly generated spaces selecting the one that is closest to the bottom-left corner of the object and sufficiently large for the allocation of the next rectangle. This packing routine was combined with genetic algorithms and simulated annealing. In comparison to the sliding algorithms the Difference Process Algorithm generated better results, because it is capable of filling enclosing empty areas in the layout.

Dagli and Poshyanonda (1997) used the genetic algorithm to generate an input sequence for the placement algorithm, which is based on a sliding method and combined with an artificial neural network (Table 4.3). The sliding routine places a new item next to the previously allocated one along the width of the object. If the space is not sufficient, a new row is formed. During the packing process the newly generated scrap areas are recorded and stored for subsequent allocations. Before an item is positioned

[4] BL = Bottom Left heuristic; based on sliding principle (section **Error! Reference source not found.**)

[5] based on sliding principle; referred to as BLLT-routine throughout this work (section **Error! Reference source not found.**)

onto the object, the available scrap areas are tested with an artificial neural network selecting the best match between the item and the empty areas. If no match can be found the item is allocated with the sliding routine. The matching process tries all admissible orientations of the item and is based on a matrix representation of the items and scrap areas using a grid approximation.

Lai and Chan (1997) used an evolutionary algorithm, which is combined with a heuristic routine. This algorithm does not use any cross-over operator and is only based on selection and mutation processes. The heuristic decoder is similar to the bottom-left algorithm used by Leung et al. (1999) and places the item in the position that is closest to the lower-left corner of the object. The packing task used by Lai and Chan is a stock cutting problem. Since the area of the object is limited it may not be possible to allocate all items. In addition to the classic mutation operator, a hill-climbing operation is applied during the decoding process that rearranges the rectangles of the permutation. If an item in the sequence cannot be allocated on the stock sheet the corresponding element in the permutation is shifted to the end of the sequence. Comparisons with a mathematical programming algorithm show that the evolutionary approach is computationally more efficient, but generates patterns with slightly higher trim loss.

Table 4.3: Hybrid genetic algorithms and evolutionary algorithm for non-guillotineable 2D packing problems

	Leung et al. (1999)	**Dagli and Poshyanonda (1997)**	**Lai and Chan (1997)**
algorithm	**GA**	**GA**	**EA**
problem	strip packing; no rotation	strip packing; 90° rotation	packing of a single closed bin; no rotation
objective	minimise trim loss	minimise height	minimise trim loss
representation	permutation	permutation	permutation
fitness	trim loss	height, width	trim loss
cross-over	PMX, CX, OBX, OX (1 point and 2 point)	OX	none
mutation	swap of 2 elements	inversion	swap of 2 elements; hill-climbing during allocation process
decoder	'Difference Process Algorithm'	sliding algorithm and ANN to match enclosed areas with item to be placed	placement closest to the bottom-left corner

Hybrid Algorithms Using Additional Layout Information

The data structures of the hybrid algorithms summarised in Table 4.3 may not recognise characteristic features of packing schemes in the encoding as most of them are hidden in the placement algorithm. A second category of solution approaches involving genetic algorithms is therefore directed at incorporating some of the layout information in the encoding technique (Table 4.4). Two approaches described in the sequel are based on binary tree structures using some additional rules to fix the position in the layout.

Another approach that deals with the manufacturer's pallet problem applies a representation technique, which contains all the information about the phenotype.

The genetic algorithm by Kröger et al. (1991a, b; 1993) is based on a directed binary tree to encode the problem in which each node represents a rectangle. Two sets of edges identify those parts that are adjacent in the vertical and the horizontal direction. This representation fixes one dimension of the position of the current item in the partial layout. The second dimension is determined by the bottom-left condition. In order to generate a unique packing scheme each node is assigned a priority value, so that the rectangle with the highest priority is placed next in case of a conflict. The data structure encodes the set of rectangles and also contains information about orientation and priority. The fitness evaluation of a packing pattern considers the height and the width. The genetic operators have been adapted to the problem with the mutation operators modifying the set of edges, the orientation and the priority values. The cross-over consists of taking a sub-tree from the first parent and placing it at the root position of the offspring. The missing rectangles are then taken from the second parent while the orientations are kept and the priority values are modified such that the packing sequence is maintained. Results show that the genetic algorithm outperforms the BL-heuristic. The evaluation against the BL-heuristic allows a relative comparison with the hybrid algorithms developed in this work (section 6.6).

Herbert and Dowsland (1996) developed a 2D encoding technique for a manufacturer's pallet loading problem, which contains only identical rectangles. The layout is represented by a 2D matrix indicating available positions for vertical and horizontal placement, whereby the horizontal one has priority. Since this encoding contains all the information necessary to represent the geometrical layout, no decoding algorithm is required for the fitness evaluation of the layout.

The boxes as well as the pallet are considered as checkerboards of unit squares. In a 1D model the binary strings are composed of all rows in the pallet, where every bit represents a possible placement cell for the box. In order to reduce the solution space, the authors developed a reduction technique to limit the placement positions to feasible co-ordinates that are integral combinations of box lengths and widths. The geometrical meaning of this representation can be seen best in connection with cross-over, which has the effect of cutting the layout horizontally. Hence the string representation reflects proximity in the horizontal direction within the same row, but not in the vertical direction. Vertically close box positions will appear widely separated on the string.

This has been the motivation for developing a 2D matrix encoding. In order to consider the orientation of the items two rows are used in the matrix to encode each row of the pallet, with the one representing horizontal and the other one representing vertical positions. Two cross-over operators were developed cutting the layout horizontally and in a random fashion. This cross-over operation can lead to infeasible solutions, which either can be penalised in the fitness function or repaired. The authors experimented with both options investigating several fitness functions and a repair operator. After removal of overlapping boxes the optimal packing over the corresponding set of positions is calculated using a graph-theoretic model. This repair operator can be used to transform the solutions of the final population into valid

layouts. An enhancement operator can also be applied throughout the search process. The enhancement operator optimally packs the removed boxes into the empty areas in the layout. Experimental results indicate it may be more profitable to remove overlap than to penalise it by the fitness function. For the small to moderately sized problems investigated 2D techniques did not have any advantages over the 1D ones. The authors concluded that their 2D approach might prove more beneficial in more complex problems.

Table 4.4: Comparison of the genetic algorithms for non-guillotineable 2D packing problems - approaches with encodings including layout information

	Kröger et al. (1991a, b; 1993)	**Herbert and Dowsland (1996)**	**Herbert and Dowsland (1996)**
problem	strip packing 90° rotation	pallet loading; 90° rotation	pallet loading; 90° rotation
objective	minimise height	maximise number of boxes placed	maximise number of boxes placed
representation	directed binary tree	1D binary string	2D binary matrix
fitness	height, width	number of boxes placed; penalty for overlap	number of boxes placed; penalty for overlap
cross-over	problem-specific	uniform (1 and 2 point)	problem-specific
mutation	variation of set of edges, orientation, priority	bit change	bit change
decoder	encoding structure + bottom-left condition	none	none

Algorithms operating on the 2D layout

The third type solution approach operates without encoding and solves the problem in the 2D space. So far, Ratanapan and Dagli (1997a, b; 1998) developed the only evolutionary approach in this area. Starting of from an initial solution, the layout is manipulated by three groups of operators including hill-climbing, mutation and recombination. The evolutionary algorithm is summarised in Table 4.5.

Various layout modifications move one item only and are implemented in the form of hill-climbing accepting the layout change if the fitness value is better or at least remains the same. These operations include translation, rotation and relocation of an item. An additional operator rotates an item around the touch point with another item. Further to that, two operations perform translation and rotation simultaneously. The series of mutation operators aims at rearranging several items. One operator reallocates an item into a different region of the object to create room for the reorganisation of other items. If the target area is occupied the item is reallocated to the upper right corner of the partial layout. In case overlap is created in the target area, a mutation operation is performed which moves all overlapping parts out of this region. Whereas the hill-climbing and mutation operators involve one layout only, the recombination process works on two or more exchanging individual parts or a whole area. Since this can lead to invalid configurations, multiple occurrences of an item and overlap need to be eliminated.

Experiments on rectangular packing problems showed that this approach could generate layouts of up to 97% packing density. One of its major drawbacks is the complexity of the various modification operators involving overlap determination and reallocation of partial layouts. Since no comparisons are made to other solution approaches in the literature it is difficult to establish the efficacy of this method.

Table 4.5: Evolutionary algorithm for non-guillotineable 2D packing problems - solution in 2D space

	Ratanapan and Dagli (1997a,b; 1998)
problem	strip packing; 90° rotation
objective	minimise height of bin
representation	2D geometric objects
fitness	packing density
cross-over	none
mutation	series of hill-climbing, mutation and recombination operations
decoder	none

4.3.2.2 Guillotineable Packing Problems

Guillotineable packing problems have been approached with genetic algorithms by four researchers. Most algorithms are based on tree representations applying various genetic operators (Table 4.6). Two methods take a different approach and use a permutation and a heuristic decoder to generate guillotineable layouts (Table 4.7).

Hwang et al. (1994) tackled three rectangular packing problems involving strip packing, packing under minimisation of the area and bin packing (section 4.3.2.3). The strip packing problem is approached with two different encodings. The first one uses a directed binary tree that can be described in the form of a string in polish notation. An operator is assigned to each tree-node indicating either the vertical and horizontal combination of two rectangles. Before the cross-over operation, the polish expression is spilt into permutation and operator parts that are manipulated separately. Four different mutation operators are applied to the chromosome (Table 4.6).

The second representation that has been implemented by the authors is order-based using a level-oriented packing procedure (Table 4.7). The packing is constructed as a sequence of levels, each rectangle being placed left justified so that its bottom rests on one of these levels. Each level is defined by a horizontal line drawn through the top of the tallest rectangle on the previous level. A new level is started whenever the remaining width of any of the previous levels is too small. Two versions of this decoding algorithm were implemented placing the current rectangle into the level where it fits first (First Fit strategy, FF) or positioning it where it fits best (Best Fit strategy, BF).

Comparisons show that the order-based approach achieves higher packing densities. The authors conclude that the penalty term is not sufficient to deal with the width constraint. The two versions of genetic

algorithms are compared to the First-Fit-Decreasing-Height heuristic (FFDH), which sorts the rectangles according to their height before placing them sequentially in the first available position. The two hybrid algorithms using the simple decoding routines perform equally well. Their performance is better than the one of the FFDH-heuristic.

The slicing tree representation proposed by Kröger (1995) ensures that the packing pattern is guillotineable. The relative arrangement of the rectangles stored in the leaf nodes is described with the aid of two operators at the node above indicating either a horizontal or a vertical combination. In order to preserve the knowledge stored in the sub-trees, a special cross-over operator exchanges sub-trees. Only sub-trees with a certain packing density and at most four rectangles are transmitted to the offspring. After reducing the first parent to the sub-trees to be inherited, the sub-trees from the second parent are separately inserted into the new string together with a new cut-line. The offspring is completed by the insertion of single rectangles that are missing from the complete set. In terms of mutation five different operators are applied (Table 4.6). A hill-climbing strategy is implemented in the genetic algorithm aimed at improving the fitness of a recently mutated or recombined string. The solutions produced by the genetic algorithm are superior to those found by heuristic algorithms as well as random search and simulated annealing. Genetic algorithms and simulated annealing achieve significantly better results than the primitive heuristics with the genetic algorithm performing best.

In order to reduce the complexity of the problem, Kröger (1995) introduces the concept of meta-rectangles, which describe a group of adjacent, densely packed rectangles that are combined to one large rectangle. In this way partial layouts are frozen yet the shape is still flexible enough to be grouped with other rectangles. In terms of recombination the cross-over operator has to ensure that the meta-rectangles are transmitted to the offspring. This produces a significant reduction in the run times and leads to an improvement in the average best solutions.

The solution approach applied by Rahmani and Ono (1995) is based on a binary tree, where each leaf node represents a rectangle. The node at the hierarchy level above indicates whether two rectangles perform a horizontal or vertical combination. In order to preserve the feasibility of the offspring a special cross-over operator was developed. Unlike the classical genetic algorithm where a certain amount of the population is selected for the recombination process, each individual is considered for cross-over. Once an individual is selected for cross-over using a certain node, a suitable candidate is searched and crossed. Since only sub-trees are crossed the solution only needs to be evaluated partially during the fitness calculation.

András et al. (1996) used a tree representation for this problem, where each node is either further cut into two pieces or remains uncut. In order to encode this problem a data structure has been developed with each node containing information about the dimensions of the piece, the position and orientation and the occurrence of a cut. The fitness of the individuals is related to the packing density. A combined crossover - mutation operator exchanges sub-trees between two parent strings. It may be necessary to modify the offspring after the crossover to guarantee feasible solutions, which adds a mutational component to the

operation. As the quality of the solutions is not measured against another method, the general performance of the genetic algorithm cannot be evaluated.

Table 4.6: Comparison of the genetic algorithms for guillotineable 2D packing problems using tree representations

	Hwang et al. (1994)	**Kröger (1995)**	**Rahmani and Ono (1995)**	**András et al. (1996)**
problem	strip packing 90° rotation	strip packing; 90° rotation	packing of a single closed bin; no rotation	packing of a single closed bin; no rotation
objective	minimise height	minimise height	minimise waste	minimise area
re-presentation	directed binary tree	string representing tree structure	tree representation	tree representation
fitness	bounding rectangle to be close to square; excess width penalised	height	utilisation ratio	packing density
cross-over	PMX and uniform	exchange of sub-trees under certain conditions	exchange of sub-tree under certain conditions	exchange of sub-trees
mutation	rotation, swap of two items; move of operator, complement of operator	swapping of sub-trees, inversion of cut-line or rectangle orientation, rotation of rectangle	inversion of cut-line; shifting of a cutting position	combined with cross-over: repair of infeasible configurations
decoder	combination of 2 items: position in containing larger rectangle is bottom-left justified	none	none	none

Corno et al. (1997) developed a hybrid genetic algorithm to solve a trim-loss problem from the glass cutting industry. The problem involves a number of constraints such as guillotineable layout, maximal distance between parallel cuts and defect areas of the glass sheet. A heuristic algorithm was developed that considers all technological constraints. The chromosome consists of a permutation of the items and contains a sequence of genes. Each gene corresponds to one item describing the geometrical characteristics, the orientation and a placement criterion flag, which links the piece with its previous one. The mutation operators work on these genes, changing the orientation and the placement flag. The layout constraints are entirely left to the decoder that searches through the available objects and positions to find the best. Comparisons to commercial packages show that the performance of the genetic algorithm is equal or better, especially for larger packing tasks.

Table 4.7: Comparison of the genetic algorithms for guillotineable 2D packing problems using order-based representation

	Hwang et al. (1994)	**Corno et al. (1997)**
problem	strip packing 90° rotation	packing of a single closed bin; 90° rotation; constraints: defects, distances, etc.
objective	minimise height	maximise utilisation
representation	permutation	permutation with flags for orientation, placement, geometry
fitness	height	utilisation ratio
cross-over	PMX	OBX
mutation	rotation, swap of 2 elements	swap of 2 elements, flip rotation, flip placement criterion flag
decoder	level-oriented FF[6] and BF[7]	heuristic algorithm that considers all technological constraints

4.3.2.3 Bin Packing

The genetic algorithm approaches to bin packing concentrate mainly on the 1D version. The objective of bin packing is to allocate a set of items into a minimum number of bins. Consequently the fitness function has to take into account the utilisation of the bins. Two of the following algorithms are concerned with bin packing in the classical sense and one method is aimed at dynamic bin packing (Table 4.8). Less work has been done on 2D bin packing.

The hybrid algorithm by Hussain and Sastry (1997) is order-based and uses the Next Fit algorithm (NF) to generate the bin layout. Comparisons with the First Fit Decreasing (FFD) and the Best Fit Decreasing (BFD) algorithm which sort the items according to decreasing height show that the genetic approach outperformed the simple heuristics for many test cases. The heuristic routines are described in section 4.2.1.2.

The representation scheme applied by Falkenauer and Delchambre (1992) and Falkenauer (1996, 1998) in their so-called grouping genetic algorithm focuses on the bins rather than the items. According to the authors an encoding technique that uses one gene per item to represent the bin, into which it is packed, does not perform well in combination with the classic crossover and mutation operators. Therefore an alternative data structure was proposed using a two-part chromosome, which consists of an item part describing the permutation of items and a group part. The group part contains the bins used in the solution. The order of the bins in the chromosome is irrelevant to the genetic algorithm. The idea behind this encoding scheme is that groups, which are the meaningful building blocks in grouping problems, are represented by the genes of the chromosome.

[6] FF = First Fit heuristic (section 4.2.1.2)

[7] BF = Best Fit heuristic (section 4.2.1.2)

The genetic operators only work on the group part and have to handle strings of variable length. They make use of two heuristic procedures, the First Fit (FF) and the First Fit Descending heuristic (FFD). The FF places an item into the first bin with sufficient space and starts a new bin in case there is not enough space. The FFD first sorts the items according to decreasing height before applying the FF-rule (section 4.2.1.2).

The cross-over procedure ensures that important information concerning the items stored in certain bins is transmitted. After random selection of two cross-over points in the group part, the bins between the cross-over section are inserted into the first offspring. In order to avoid the multiple appearance of items in the solution, the relevant bins originating from the first parent are deleted. Since the deleted bins may contain items that are not present in the bins coming from the second parent, these missing items are reinserted with the FFD-procedure creating additional bins if necessary. For the mutation process a few bins are selected randomly reinserting their items in random order with the FF-algorithm. An inversion operator is implemented changing the order of the bins in the group part of the chromosome. This can improve the transmission of certain genes to the offspring. Comparison with the FFD-algorithm showed that the performance of the genetic algorithms is superior.

The genetic grouping algorithm was extended implementing the so-called dominance criterion (Falkenauer, 1996) which was used earlier in an approximation algorithm for the bin packing problem. This criterion is based on the observation that under a certain geometric condition the arrangement of the items in a bin is better than another one. This is exploited in the form of local optimisation in the genetic algorithm. Experiments showed that this technique works better than the original grouping algorithm.

Table 4.8: Comparison of the genetic algorithms for 1D bin packing problems

	Falkenauer et al. (1992) Falkenauer (1996, 1998)	**Hussain and Stastry (1997)**	**Runarsson et al. (1996)**
problem	bin packing	bin packing	dynamic bin packing
objective	minimise number of bins	minimise number of bins	minimum weight for each bin, maximise utilisation
representation	two-part chromosome	permutation	binary strings representing bin allocation
fitness	bin capacity	unfilled proportion of the bin	set of fuzzy objective functions
cross-over	problem specific operator	special cross-over operator from TSP	binary crossover (2 point)
mutation	deletion of bins, reinsertion of objects with FFD-rule	swap of 2 elements	binary
decoding	First Fit-algorithm	Next Fit-algorithm	

Runarsson et al. (1996) applied genetic algorithms to a 1D dynamic bin packing problem. A continuous supply of items has to be packed into a limited number of bins assuring minimum weight of each bin. Once a bin is filled, an empty one replaces it. Similar to on-line algorithms, the items must be packed sequentially, though the algorithm sees a number of items simultaneously. The algorithm uses the binary

coding technique for the chromosomes, which assign the series of 'visible' items to a limited number of bins. The authors developed a set of fuzzy objective functions to guide the search.

The only work, which has so far approached the 2D bin packing problem with genetic algorithms, was conducted by Hwang et al. (1994). The items are represented by a permutation and packed into the bins by a two-stage heuristic. In the first stage the level-oriented First Fit heuristic procedure as described to the 2D strip packing approach in section 4.3.2.2 places them onto a strip of unlimited height constructing the layout as a sequence of levels. Each level forms a rectangular block containing one or more items. In the next step the packed strip is decomposed at each level. Each block can be viewed as a rectangle of fixed width and the height of the level. The blocks are then packed with the FFD or BFD-rule into fixed size bins reducing the problem to a 1D bin packing task. The authors implemented two hybrid genetic algorithms using the FFD and the BFD-routines in the heuristic decoding algorithm. Comparisons are made to the Hybrid First Fit-algorithm (HFF), which is a combination between the FFDH and FFD-routines (Coffman et al., 1984). The genetic algorithms consistently outperformed the heuristic one (HFF), whereby the one using the BFD-heuristic performed best (section 4.2.1.2).

4.3.2.4 Packing of Regular Shapes other than Rectangles:

The only genetic algorithm designed for packing regular shapes other than rectangles was proposed by George et al. (1995). A hybrid genetic algorithm is combined with a heuristic method to pack different-sized circles into a rectangular area. During the packing process so-called position numbers are used to indicate possible locations for the remaining circles in the partial layout.

The encoding technique of the genetic algorithm makes use of the position numbers, which are defined with respect to the sides of the object and the circles already placed. Instead of evaluating every possible position of a circle in the packing pattern, only an initial position is allocated to each circle. This serves as a default position and is only modified if it causes an infeasible packing configuration. The initial positions of all circles are stored in the chromosome, with the first cell containing the position of the first circle etc. As a measure of fitness the density of the circles in the rectangle is used. The genetic operators applied are proportional selection, one-point cross-over and a mutation operator that generates a random position number. The decoding procedure attempts to place a circle at a position number contained in the string. If this position is not feasible or not defined, the position number is incremented until a feasible position is found.

The genetic algorithm is compared to heuristic methods using the same decoding procedure. The comparison includes a heuristic method that generates the position number randomly. Performance comparisons for different problem types showed that genetic algorithms and random search outperformed the other heuristics, when a balance must be reached between quality and computational effort. The advantage of the data structure in George et al. (1995) is that domain information is implemented in the

genetic algorithm as part of the procedure. The task of the decoder is to check the feasibility of the layout and eventually to find a new position.

4.3.3 2D Irregular Strip Packing Problems

This category of irregular problems includes the packing of polygons and arbitrary shapes. A number of researchers have approached the packing of polygons some including holes inside the shapes. Depending on the nesting algorithm, some approaches are only suitable for convex polygons. In most solution approaches the irregular items are either polygons or approximated by polygons consisting of a list of vertices. Geometric algorithms are then required to determine feasible positions in the partial layout and eventually to calculate the overlap. A second shape description technique is grid approximation where items and objects are represented by a set of equal sized squares using 2D matrices. The nesting process therefore usually involves scanning of the various matrices and matching with empty cell clusters. An outline of the algorithms is given in Table 4.9 to Table 4.12.

4.3.3.1 Packing of Polygons

Fujita et al. (1993) proposed a hybrid approach combining an order-based genetic algorithm with local minimisation to solve a nesting problem involving convex polygons only. The local minimisation algorithm is used to optimise the position of an item in the layout, after initial placement in the leftmost-lowest position next to its preceding neighbour. This algorithm uses a Quasi-Newton method to manipulate the relative positions between the objects defined by a set of variables. The fitness of an individual is related to the waste and the distance of the polygons from the origin of the object and deals with the width and overlap constraint. Since the performance of the hybrid genetic algorithm was not compared to other methods, it is not possible to judge its efficiency.

Jakobs (1996) used an order-based genetic algorithm for nesting and extended the work on packing of rectangles (section 4.3.2.1) to polygons. The decoder only operates on the enclosing rectangles of the polygons during the evaluation stage. When the polygons are fed into the system they are first rotated into the position where the enclosing rectangle has the smallest area. The irregular aspect of the packing task is considered after the genetic algorithm has converged applying a shrinking-algorithm to the layout. This algorithm moves the polygons closer together shifting them as far as possible to the bottom and the left whilst avoiding overlap. To improve the shrinking-step, reflections of the original polygons are also tested. The shrinking-step reduces the height of the layout, since it allows utilisation of the space "wasted" by the embedding process. Applying the shrinking-routine to the final layout has a major drawback. Since the polygons are repositioned sequentially, empty areas in the layout may not always be reached, since items, which are not yet iterated, can block the sliding motion of the current one.

Hybrid algorithms which operate on a sliding principle and pack the polygon rather than the rectangle in the partial layout have been found to achieve better packing densities (section 7.5), although they are

computationally more expensive. Since no comparison was made to other techniques for irregular nesting tasks, it is difficult to establish the overall performance of the method proposed.

Table 4.9: Hybrid genetic algorithms for 2D irregular packing problems

	Fujita et al. (1993)	**Jakobs (1996)**	**Dighe and Jakiela (1996)**	**Dighe and Jakiela (1996)**
problem	convex polygons only; free rotation	polygons; 90° rotation	polygons; free rotation	polygons; free rotation
objective	minimise waste	minimise height	maximise density	minimise height
re-presentation	permutation	permutation	binary tree	permutation
fitness	waste, distance to origin, width; penalty for overlap	height, remaining area	packing density	height
cross-over	OX (1point)	OX (1point)	exchange of sub-trees	OX (1 point)
mutation	random removal and reinsertion of one element	inversion; exchange of 2 elements; rotation	none	random removal and reinsertion of 1 element
decoding	placement in leftmost-lowest position; local minimisation algorithm	placement of enclosing rectangles in BL-position; then shift algorithm; overlap omitted	determined by low-level GA: pairwise clustering of nodes items; overlap omitted	determined by low-level GA: vertical sliding from top of object into partial layout; overlap omitted

Dighe and Jakiela (1996) developed two genetic algorithms for the nesting task. The first approach is order-based and uses a sliding algorithm to move the irregular item into the partial layout in vertical direction. An additional (low-level) genetic algorithm is applied to find the best horizontal position at the upper side of the object from which to "drop" the item and its orientation for the sliding process.

The second genetic algorithm uses a binary tree representation technique and is also hierarchical. The tree determines in which way two items are clustered. The nesting process of the two polygons is controlled by a low-level genetic algorithm, which searches for the configuration with the smallest enclosing rectangular area. Both approaches avoid overlap during the nesting process. The two methods were tested on jigsaw puzzles with a known optimum solution and achieve packing densities between 69% and 72%. The major drawback of these techniques is the hierarchical structure using two genetic search processes. The low-level search is extremely wasteful in terms of computation time.

Bounsaythip and Maouche (1997) applied a binary tree approach to a problem from the textile industry. Before the nesting step, the polygons are circumscribed by the bounding rectangles. The nodes in the tree contain two operators that determine the side at which the second rectangle is packed with the stationary one and its orientation. The actual nesting process is carried out by a low-level routine which finds the smallest enclosing rectangle of the cluster using a special encoding technique described in their earlier work (see below; Bounsaythip et al., 1995). A single tree in this approach does not necessarily represent

the complete set of items, but rather a strip in the textile layout. The algorithm therefore has to deal with trees of different length. The cross-over and mutation operators are stated in Table 4.10.

In an earlier approach, Bounsaythip et al. (1995) used a different genetic algorithm to address the textile nesting problem. Instead of dealing with a complex marker layout they focus on the generation of one strip only in the layout. The polygons are circumscribed by the bounding rectangles. The shapes are represented with a special encoding technique that describes the contour of the polygon relative to the enclosing rectangle using a set of integer values. For each of the four rectangle sides such a contour description is generated. This representation technique is very practical for nesting two shapes. Unlike many other genetic algorithms a single shape represents one individual in the population. The fitness of an individual is determined by the utilisation ratio of the bounding box. Cross-over and mutation operators are domain-specific and merge selected shapes. The performance of this genetic algorithm was enhanced through hybridisation with simulated annealing which slightly increases the packing density.

Table 4.10: Comparison of the genetic algorithms for 2D irregular packing problems

	Petridis and Kazarlis (1994)	**Bounsaythip and Maouche (1995)**	**Bounsaythip and Maouche (1997)**
problem	polygons; no rotation	irregular items from textile industry; considering one strip only; 90° rotation	irregular items from textile industry; 90° rotation
objective	minimise height	minimise length of strip	minimise waste
representation	binary string encoding position in layout	string consisting of 4 sub-strings; represents a single shape or cluster	binary tree
fitness	dynamic; overlap, used area; x-position of shapes	packing density	density of the strip layout formed by each tree in relation to the overall layout
cross-over	multi-point, binary	interchange of sub-strings	exchange of sub-trees
mutation	binary	swap of sub-string within one individual	change of operator; swap of 2 elements; deletion of 1 element
decoding	none	none	best relative position of 2 clusters determined by low-level algorithm together with operator info in tree; overlap omitted

Petridis and Kazarlis (1994) developed a genetic algorithm, which does not require a decoding algorithm in the nesting process. Instead, the position of an item in the layout is encoded in the chromosome in from of two binary strings. Due to the simplicity of the encoding technique, the binary cross-over and mutation operators can be used. Furthermore, a set of mutation operations were defined to work directly on the phenotype swapping two shapes or repositioning shapes into gaps in the layout. Since the position of the items is determined by the encoding, overlapping configurations can occur and are penalised in the fitness function. The fitness function is dynamic, increasing the penalty term gradually in order to drive the population away from invalid solutions towards the end of the search. Similar to some simulated

annealing approaches in the literature (section 4.4.1), the rationale behind the dynamic nature is to penalise overlap less at the beginning when it is important for the shapes to pass over one another in order to reach enclosed areas. In addition to that, a local search technique was applied to the best solution at fixed generation intervals. Petridis and Kazarlis (1994) tested their algorithms on jigsaw problems consisting of less than 15 shapes. Comparisons showed that the optimal solution was more often found using the dynamic fitness function. The local search had a positive impact and accelerated the search process.

4.3.3.2 Packing based on Grid Approximation

Compared to the shape description based on geometric primitives such as polygons, fewer approaches use a digitised representation. Grid approximation offers the advantage that holes inside items or gaps in the partial layout can be easily described. Since the object is usually scanned for a suitable position these areas are automatically considered. One of the major advantages of this technique is that no additional routines are required to identify enclosed areas in the shapes or the partial layout. The different solution approaches are outlined in Table 4.11 and Table 4.12.

Ismail and Hon (1992) developed a genetic algorithm for the pairwise clustering of two identical polygons. After circumscribing the shape with the minimum enclosing rectangle, a grid is superimposed to convert the shape into a binary 2D matrix. When clustering two shapes, two parameters are used to describe their relative position to each other. Another four parameters are introduced to represent the mirroring of the shapes along the two axes. These parameters are combined into a binary multi-parameter string, defining a clustering solution. The fitness reflects the best orientation for maximising the material utilisation and includes a penalty for overlapping. Subsequent decoding of the string into a layout is straightforward. Comparisons to the performance of another clustering method that was developed by the authors earlier showed that the genetic algorithm produces denser packing of figures with concave features. This is mainly due to the limitations of the other method, whereas the solutions have been identical for other shape types.

Ismail and Hon (1995) extended the clustering method proposed in (Ismail and Hon, 1992) to dissimilar shapes in combination with a heuristic rule. Applying the above representation technique, the shapes and the object are first digitised and represented as a 2D grid array. Additional to the two parameters, which describe the relative position of a shape to the others, three parameters define mirroring and rotation. The overall genetic string is a sequence of the encodings for each individual shape. This data structure can result in infeasible solutions, which are penalised in the fitness function. The decoder uses a complex set of parameters and rules to describe the relative positions and the placement of the polygons.

Poshyanonda and Dagli (1993) extended the order-based genetic algorithm developed for rectangles to the nesting of irregular shapes (section 4.3.2.1). The decoder consists of an artificial neural network that matches an incoming shape with the available empty areas in the partial layout. For this purpose the items

and the object are presented as binary 2D matrices. The algorithm selects the best match or triggers a sliding algorithm if no match is found.

Table 4.11: Hybrid genetic algorithms for 2D irregular packing problems

	Poshyanonda and Dagli (1993)	**Ismail and Hon (1995)**	**Gwee and Lim (1996)**
problem	irregular items; 90° rotation	rectilinear shapes; 90° rotation	polyominoes; 90° rotation
objective	minimise height	minimise area used	optimal solution
representation	permutation	multi-parameter string including relative position and rotation of both items; binary	permutation
fitness	height	density of packing, penalty for overlap	number of boundary edges; number of void and overlapping cells; number of items without overlap
cross-over	OX	binary (1point)	PMX
mutation	inversion	bit change	
decoding	ANN to match scrap areas with item + sliding algorithm; overlap omitted	set of heuristic rules	circular placement starting from the centre of the object

Gwee and Lim (1996) studied a special type of irregular packing problem originating from the world of jigsaw puzzles. The items consist of rectilinear blocks, so-called polyominoes, which are placed onto a rectangular board. Since these puzzles have a known optimal solution, the performance of the genetic algorithm can be easily measured. The objective function considers three aspects, which are important in the search for the optimal configuration (Table 4.11). The set of polyominoes is represented as a permutation. The decoding stage uses a circular placement technique, which places the shapes in circular fashion starting at one corner of the board, and continues in anti-clockwise direction towards the centre of the board. Several orientations are tried selecting the one that yields to the highest number of contact edges. The idea of this technique is to build up good groupings of polyominoes starting from the corners. Comparisons with two hill-climbing techniques show that the genetic algorithm finds the optimal solution quicker, in particular when the problems consist of a higher number of pieces.

Jain and Gea (1998) designed a special encoding method, which describes the complete layout as a 2D matrix. Before the encoding step, the items are digitised and consist of a cluster of unit squares. In the 2D matrix the corresponding cells are marked with the item number. In that way the phenotype is completely contained in the genotype making a decoding algorithm redundant. A set of problem-specific cross-over and mutation operators were developed to work on this representation scheme and are stated in Table 4.12. Since these operations can easily result in overlapping configurations, repositioning of items is frequently required. In order to increase the density of the layout, subsequent compaction steps shift the items left and down in order to fill vacant positions.

The method developed by Ratanapan and Dagli (1997b, 1998) is different from the other approaches described so far, since it does not make use of a data structure to represent the problem. The irregular items are represented using a grid approximation. After the initialisation process, which places all items into non-overlapping positions on the object, a series of genetic operators is applied consisting of hill-climbing, mutation and recombination processes. These operations are described in connection with their earlier work on rectangle packing in section 4.3.2.1.

Table 4.12: Genetic algorithms operating on the phenotype for 2D irregular packing problems

	Jain and Gea (1998)	**Ratanapan and Dagli (1997b, 1998)**
problem	irregular items 90° rotation	strip packing; free rotation
objective	minimise layout area	minimise height
representation	2D matrix	2D geometric objects
fitness	used area; total moment of inertia	packing density
cross-over	exchange of items in sub-area of matrix	none
mutation	rotation; swap of 2 items; random new position for 1 item	series of hill-climbing, mutation and recombination operations
decoding	none	none

4.3.4 Overview of 3D Packing Problems

Unlike 2D packing problems, the 3D versions deal mainly with regular and in most cases cuboid objects, which have to be loaded onto pallets or into containers. A number of problems were approached with genetic algorithms in the literature. An overview of the algorithms for regular 3D problems is given in Table 4.13. The complexity increases when irregular objects are to be placed. So far, only one group of researchers have applied genetic algorithms to the packing of arbitrary 3D items.

4.3.4.1 Regular Packing Problems

Prosser (1988) developed two genetic algorithms for a highly constrained pallet loading problem. The problem consists of loading stacks of plates onto a minimum number of pallets without violating geometrical and weight constraints. The main drawback of the first genetic algorithm is the evaluation function calculating the number of pallets used. Since it only produces a few distinct values for densely packed pallets, there is a lack of guidance for the search process.

To overcome these disadvantages, a hybrid genetic algorithm was developed in order to find one maximally loaded pallet. Loading one pallet after the other up to the weight limit reduces the number of possible loading patterns and results in a faster convergence. Apart from the weight, the number of items on one pallet is also limited. The pallet is loaded up to the first violation of the constraints. The genetic algorithm is then applied iteratively to the reduced problem until the allocation of the stacks is concluded.

For example if a maximum of six elements of a string can be loaded onto a pallet the inversion operator is modified such that elements are randomly swapped between the first six positions of a string and beyond. The computational effort of the evaluation is lower than in the first genetic algorithm. The comparison with a branch-and-bound method showed that the second genetic algorithm produces better solutions in less time.

Corcoran and Wainwright (1992) proposed a hybrid genetic algorithm for a 3D loading problem using a single, open bin. The algorithm is based on the so-called level technique, which places items level by level. In extending this strategy to three dimensions the bin is divided into "slices" along the height and "levels" along the length. In order to evaluate the fitness of the packing pattern two heuristic algorithms are used. The Next Fit algorithm (NF) places an item into the next available position on the level without exceeding the width of the bin. If that fails a new level is started. The First Fit procedure (FF) searches from the beginning until a suitable position is found. The performance comparison for various sets of boxes shows that the genetic algorithms produce better packing patterns than the two heuristics, which have equal packing efficiencies.

House and Dagli (1993) approach the container loading problem with a permutation and use a heuristic decoder to pack the boxes. Lists keep track of the empty spaces that are generated after allocating a box.

A different approach to a container loading problem was taken by Lin et al. (1993). Their algorithm takes into account additional constraints such as the weight of the boxes. A heuristic method enhanced by a simulated annealing algorithm is combined with a genetic algorithm. In order to consider the weight constraint, i.e. placement of heavier boxes below lighter ones, the loading layout of the container is encoded in the form of a multi-chromosome string with each partial chromosome representing the loading pattern of one layer. Cross-over and mutation only operate on the corresponding relative chromosomes without inter-chromosome exchanges. A set of heuristic rules is used in the decoding stage. In terms of solution quality the genetic algorithm outperforms the heuristic method enhanced with the simulated annealing algorithm.

Bortfeldt (1994) developed a similar method to the level technique by Corcoran and Wainwright (1992) for a container loading problem, where a set of different boxes is packed maximising space utilisation. The genetic algorithm does not require a decoding stage since all information concerning the phenotype is stored in the encoding. Instead a heuristic algorithm is used during the cross-over and mutation operation based on a layer technique. The depth of a layer is determined by the first box (layer defining box = LDB) in the layer, width and height are the same as the container dimension. A further procedure is implemented to fill the layer, where each allocated box generates three spare spaces inside the layer: beside, in front and above. The fill algorithm searches for the best fitting pair of boxes and places them into the layer.

In order to keep the computational effort low, a problem-specific data structure is applied. The chromosome includes the number of layers and for every layer the LDB, its rotation and the set of boxes

allocated in that layer. The main advantage of this encoding technique is that the genotype of the individuals is sufficient for evaluation of the fitness. It guarantees the feasibility of the individuals generated by the genetic operators without the need for decoding. The rationale behind the cross-over technique is the transfer of the best layers of both parents. When no more layers can be taken and the offspring is still incomplete, new layers are generated by the heuristic algorithms described above. During the mutation stage the boxes with the lowest frequency of appearance in the new population are determined. New packing plans are then generated by the heuristic algorithm, which uses these boxes as the LDB of the first layer. The individuals generated in this way are exchanged for the ones with the lowest fitness in the population. A performance comparison shows that the genetic algorithm produces packing patterns with a high space utilisation.

Table 4.13: Comparison of the genetic algorithms for 3D regular packing problems

	Prosser (1988)	**Corcoran and Wainwright (1992)**	**House and Dagli (1993)**	**Lin et al. (1993)**	**Bortfeldt (1994)**
problem	pallet loading with weight constraints	loading of single open container	loading of single container with weight	loading of single container with weight constraint	loading of single container
objective	minimise number of pallets	minimise container size	maximise number of items	maximise number of items	maximise number of items
representation	permutation	permutation	permutation	multiple-chromosome	problem-specific
fitness	weight of pallets	height	container utilisation		utilisation and deviation of utilisation of single layers
cross-over	OBX, adapted to loading constraint	PMX, Cycle, Order2; Rand1	not stated	PMX	problem-specific with FF rule
mutation	exchange of elements from first 6 positions with rest	elements swapped and rotated	not stated	inversion	problem-specific
decoder	set of constraints	First Fit rule Next Fit rule	heuristic that caters for constraints	set of heuristic rules	none

4.3.4.2 Irregular Packing Problems

Ikonen et al. (1996) and Ikonen and Kumar (1997) proposed a framework of genetic algorithms for packing irregular items into a cylindrical container. The items need to be packed utilising cavities of larger ones with no orientation restrictions. In order to represent the order and the orientation, in which

the parts are packed a multi-chromosome representation is used. The first chromosome consists of a permutation, which encodes the packing order. The second one contains the orientation of each element. A swapping operator in the mutation stage changes the position of two parts in the chromosome and a mutation operator is applied to the second chromosome changing orientation values randomly. The geometry of the parts is included in the decoding procedure that translates the strings into their packing layout and evaluates their fitness. The fitness of an individual is related to the distance of each part from the origin. In order to reduce the probability of infeasible solutions an adaptive penalty function is incorporated in the fitness function. This penalty function is especially advantageous when it is not known in advance how difficult it is to find feasible solutions and how effectively the objective function differentiates between solutions.

4.4 Application of Meta-heuristic Methods to Packing Problems

Genetic algorithms are not the only meta-heuristic techniques that have been applied successfully to packing problems. A number of researchers experimented with simulated annealing, tabu search and neural networks. Since simulated annealing is one of the meta-heuristic methods, which are used to compare the performance of the hybrid genetic algorithms developed in this work (chapters 6 to 8), the major solution approaches in this area will be briefly described.

4.4.1 Simulated Annealing

Simulated annealing is a meta-heuristic search method whose design was inspired by the metallurgical process of annealing (section 3.3.3). Simulated annealing was applied to rectangular and irregular packing tasks, a selection of which is described below.

4.4.1.1 Regular Packing Problems:

A number of approaches in the literature have applied simulated annealing to 2D rectangular packing problems (Table 4.14). One of the first works on simulated annealing and packing problems was carried out by Kämpke (1988). He applied simulated annealing to 1D bin packing comparing different cooling strategies. Dowsland (1993) experimented with simulated annealing on pallet loading problems involving identical as well as non-identical boxes. In the identical case, the number of feasible positions is reduced to the co-ordinates, which are multiples of the item length. The neighbourhood is defined as the set of solutions, which is obtained, when each item is moved to any other position with some restrictions. Since these movements lead to overlapping patterns, this constraint has been dealt with in the objective function. In the extension to non-identical boxes, the condition for the feasible position is that it needs to be at a valid combination of lengths and widths of the other item types starting from the container edge.

The results indicate that simulated annealing is only capable of producing near optimal solutions, which could be improved by other optimisation routines.

Faina (1999) developed a hybrid simulated annealing algorithm for guillotineable and non-guillotineable stock cutting problems. The set of items is represented as a permutation indicating the order of packing. Two heuristic decoders are used to pack the objects whilst taking into consideration the guillotine constraint. The algorithm for the non-guillotineable layout places the current item either at the top-left or the bottom-right corner of the previously positioned rectangle. The choice between the two insertion points is random. In the guillotineable case, the algorithm keeps track of the remaining empty areas in the layout. After placing a rectangle, two empty areas are created at the top and the right side, which are stored and treated as objects in the subsequent packing processes. Although the placement algorithms are formulated for a stock cutting problem, the performance evaluation only involves one object of unlimited height (i.e. strip packing). Comparisons show that the algorithm for non-guillotineable layouts achieves much higher packing densities than the non-guillotineable one due to the better nesting technique.

Leung et al. (1999) also applied the order-based approach developed for use with genetic algorithms to simulated annealing (section 4.3.2.1). Their results indicate that genetic algorithms outperform simulated annealing.

Table 4.14: Comparison of the simulated annealing approaches for 2D rectangular packing problems

	Dowsland (1993)	**Faina (1999)**	**Leung et al. (1999)**
problem	pallet loading with identical and non identical boxes; 90° rotation	strip packing; guillotineable and non-guillotineable; no rotation	strip packing; no rotation
objective	finding a feasible arrangement of a fixed number of boxes	minimise area used	minimise trim loss
representation	position in layout; overlap allowed	permutation	permutation
fitness	minimise number of overlapping boxes	packing density	height
neighbourhood move	set of position composed of width and length of boxes	swap position of two elements	swap position of two elements
cooling schedule	geometric	geometric	geometric
decoder	none	left-justified routines considering guillotine constraint	'Difference Process Algorithm' (section 4.3.2.1)

4.4.1.2 Irregular Packing Problems

Most simulated annealing approaches for irregular packing tasks do not make use of any encoding technique like genetic algorithms. The packing problem is represented as an allocation of 2D items, which must be compacted. Usually, overlap is permitted during the search process and penalised in the evaluation function. With one exception (Ratanapan and Dagli, 1997b, 1998) the genetic methods from

the literature operate on an encoding. Overlap is usually avoided through the application of placement rules and only permitted in few approaches. A number of 2D nesting tasks have been approached with simulated annealing (Table 4.15), whereas less work has been done in the area of 3D irregular problems (Szykman and Cagan, 1995).

Jain et al. (1988) addressed a blank nesting problem from the metal cutting industry, where two congruent items are nested for continuous strip stamping applications. The blanks have arbitrary shape and are approximated by polygons. In order to accommodate interlocking shapes it is necessary to allow the shapes to move over one another producing intermediate overlap. Overlap is penalised in the fitness function consisting of two terms: the wasted area and a penalty for the total overlap.

Marques et al. (1991) developed a simulated annealing algorithm for the packing of polygons and applied it to a problem from the textile industry. A neighbourhood move is achieved by translation, rotation or reflection of an item accepting only valid configurations. The quality of the layout is described by the sum of three components: the area of smallest enclosing rectangle and parameters indicating the distance of each item from the centre of the object and proximity of the items to each other. In order to reduce the processing time for the verification of the layout legality, only the overlap between corresponding enveloping circumferences of the items is tested initially.

The efficiency of the search process conducted by simulated annealing largely depends on careful construction of the cooling schedule. Theodoracatos and Grimsley (1995) experimented with polynomial-time cooling schedules and showed their impact on computational efficiency. The authors applied simulated annealing to the packing of circles and polygons. In addition to that, an adaptive penalty function was proposed to penalise overlapping configurations to a minor extent at the beginning when items need to slide over each other in order to find feasible positions in the partial layout.

The simulated annealing algorithm in Han and Na's work (1996) is used to improve an already existing layout created by an artificial neural network (ANN). The irregular items are first approximated by polygons and circumscribed by the minimum enclosing rectangle. The polygon is then described as a composition of basic geometric shapes i.e. rectangles and circles that are placed in the void areas within the enclosing rectangle as well as within the item itself. A move to a neighbouring solution consists of translation, rotation or a swap of two items. Since these operations can result in invalid configurations the fitness function considers the overlap constraint in the form of a penalty. In order to achieve dense layouts a second parameter describes a force driving an item leftwards and downwards. The neighbourhood move achieved through translation is implemented in two ways. The large perturbation within the entire object area is directed at the global optimisation of a layout whereas the small perturbation in the lower leftward direction is used to optimise the layout locally. Low starting temperatures were used in this approach, since the starting solution obtained from the ANN has already generated a reasonably good quality. It is difficult to judge the merits of these hybrid approaches involving two intelligent search processes due to the lack of comparisons with other methods.

Burke and Kendall's work (1999b) is different from the approaches described above, since the neighbourhood moves are not performed directly in the layout. Instead the problem is represented as a permutation. A neighbouring configuration is reached through one of the re-ordering techniques stated in Table 4.15. As a consequence a placement routine is required to transform the list of items into the layout. The authors developed an algorithm, which nests two polygons in turn using the No Fit Polygon (NFP, Figure 4.5) and local search to determine the best position. Before a new polygon is placed, all positions along the vertices of the NFP are tried and the cluster with the smallest convex hull is used. In case the configuration exceeds the bin width a new 'row' is started. Results show that the simulated annealing technique produces better results than hill-climbing and there is a difference between the various neighbourhood operators.

Table 4.15: Comparison of the simulated annealing approaches for 2D irregular packing problems

	Jain et al. (1988)	**Marques et al. (1991)**	**Theodoracatos and Grimsley (1995)**	**Han and Na (1996)**	**Burke and Kendall (1999b)**
problem	polygons; clustering of 2 and 3 identical shapes; free rotation	polygons; textile industry; 90° rotation	1. circles; 2. polygons; free rotation	circles, polygons with enclosures; improvement of existing layout 90° rotation	polygons; strip packing
objective	minimise waste	minimise area	maximise number of circles	minimise height	minimise height
repre-sentation	2D layout overlap permitted	2D layout overlap omitted	2D layout overlap permitted	2D layout overlap permitted	permutation overlap omitted
fitness	waste; penalty for overlap	area of enclosing rectangle; sum of distances from centre; proximity to neighbours	waste; penalty for overlap	overlap area; moment of area in the bottom-left direction	area used by each 'row' in layout
neigh-bour-hood move	translation, rotation	translation, rotation: large and small perturbation; reflection	1. translation 2. translation and rotation	translation, rotation; swap of 2 elements	swap 2 adjacent items; swap 2 random items; re-order polygons according to type
cooling schedule	geometric	geometric	polynomial-time	geometric	linear; geometric
decoder	none	none	none	none	routine using NFP and local search

4.4.2 Tabu Search

Tabu search is a search technique that is guided by the use of adaptive or flexible memory structures. In contrast to heuristic search methods such as genetic algorithms and simulated annealing tabu search contains some in-built memory mechanisms that prevent the search algorithm from returning to recently executed moves for a number of iterations. A tabu list is maintained which contains all moves, which are

not allowed the current iteration step. The search is guided by an objective function in order to find the best admissible move in a neighbourhood (Reeves, 1993, 1996; Glover and Laguna, 1993, 1997; Pirlot, 1996). With respect to packing problems, fewer solution approaches with tabu search have been proposed than with genetic algorithms and simulated annealing. The first work in this area was conducted by Blazewicz and his co-researchers, which dates back to the early 90's.

The only investigation into the application of tabu search to rectangular problems was presented by Lodi et al. (1999a, b) who focused on 2D bin packing. The two constraints that are imposed on the packing process concern the fixed orientation of the items and the layout, which has to be guillotineable. The initial layout is generated by a simple heuristic algorithm, which is then improved by tabu search. The tabu search algorithm is based on two possible neighbourhood moves. The first one attempts to remove an item from the worst bin redistributing it among the other used bins. In the latter move the algorithm tries to accommodate the item by recombining the items of two other bins. The bin layout is generated with a heuristic level-oriented algorithm (section 4.2.1.2). The performance of the tabu search is better than any one of the two bin packing heuristics and comparable to a branch-and-bound algorithm.

Blazewicz et al. (1993) were the first ones to apply tabu search to irregular packing problems. Starting with a feasible layout solution, which is produced by a simple placement procedure, a tabu search process is used to further improve the existing layout. After selecting a single item, several new positions are tried and the best one is kept. The move describes a change of the allocation of one item from one position to the other, prohibiting overlapping configurations. Items that have changed their position during recent iterations are members of the tabu list. The best admissible move is determined by the objective function aiming at placing the rightmost elements into the void areas of the layout. In comparison with Albano and Sapuppo's (1980) heuristic search algorithm, the tabu search achieved better results.

4.4.3 Artificial Neural Networks

In some approaches to rectangular and irregular packing problems, neural networks have been used. They were also applied in combination with other meta-heuristic methods where they either served to generate the initial layout or to perform the nesting process. Two examples are briefly described below.

Dagli and Poshyanonda (1997) used a neural network in combination with a genetic algorithm for a rectangular packing problem (section 4.3.2.1). The genetic algorithm is used to generate an input sequence, which is decoded into the layout by the neural network. Every time a new item is placed into the partial layout all new scrap areas are recorded and stored for subsequent nesting processes. Before an item is allocated the neural network searches through all empty areas and returns the best match. If no match is found the item is allocated next to the partial layout using a sliding algorithm. The matching process is based on a grid description of the items and scrap areas.

Han and Na (1996, 1997) used a neural network to produce an initial solution for a 2D irregular problem. After a 'good' initial solution is obtained the non-overlapping layout is further improved by simulated

annealing (section 4.4.1). The learning algorithm of the neural network is based on a Kohonen network. At the beginning of the nesting process all shapes are allocated around the centre of the object by assigning small random values to their position vectors describing the distance to the centre. The position vectors, which only indicate the direction for the motion of the items, are then modified by the neural network. The finite position is determined such that the overlap of the items is minimal using leftmost-lowest placement. The cost function is a combination between a penalty term for the overlap and the moments of area driving items to the left and to the bottom side of the object.

4.4.4 Other Heuristic Search Techniques

One of the main characteristics of meta-heuristic search processes as opposed to local search is that they contain a means of escaping local minima. Whereas optimisation with hill-climbing terminates when a locally optimum solution is found, meta-heuristics can escape this situation by temporarily accepting solutions of lower quality. Some researchers used the concept of these uphill moves and implemented new meta-heuristic search principles in addition to the standard methods like genetic algorithms and simulated annealing. The stochastic optimisation method proposed by Pargas and Jain (1993) has been used during this investigation to compare the performance of the hybrid genetic algorithms with other meta-heuristic methods (section 6.4.2 and 8.4.2).

Healy and Moll (1996) proposed a minimisation algorithm for a 2D rectangular packing problem. The algorithm is a variant of a hill-climbing technique and designed such that it allows moves in the other direction in order to escape local minima.

Pargas and Jain (1993) developed a stochastic optimisation algorithm, which borrows some principles from hill-climbing and genetic algorithms. The method was applied to a 2D irregular packing problem. Stochastic optimisation operates on a population of solutions manipulating them with the aid of a mutation operator. Unlike in genetic algorithms, only one solution is modified at a time. A new state in the search process can be obtained in two ways. The first one selects a solution from the population using ranking and generates a certain number of neighbouring states as in steepest-ascent hill-climbing. The best solution of the neighbourhood compared with the current solution is taken. If its fitness is better, it replaces the current one in the population. With a probability of around 10% the second method generates a new state randomly in order to maintain diversity in the population. The termination criteria are based on convergence or a maximum number of iterations.

In the implementation for an irregular packing problem the items are represented as a permutation. A grid approximation technique is applied. The allocation routine scans the object for the first leftmost-uppermost cell, which allows a valid configuration. If overlap occurs the item is rotated by 90°. Unfortunately, the authors did not compare this approach to other meta-heuristic techniques. Therefore relative performance in terms of solution quality and speed are not known.

4.5 Commercial Packing Software

Cost reduction via efficient production is one of the major issues in the manufacturing industries where products are mass-produced. In particular, the efficient use of raw material can result in significant savings. Therefore the industry started some time ago to automate packing and cutting processes. With the increased use of computers in the manufacturing environment, fully automatic and interactive nesting systems have become available offering a large variety of tools for nesting.

Most of these products are not only directed to improve material utilisation, but the manufacturing process as a whole containing modules on scheduling, cost estimation and inventory. Some of them have been developed for special industries, e.g. the leather industry. In general, they are multi-purpose nesting systems for a wide range of industrial applications. Table 4.16 and Table 4.17 give an overview of commercial packing software for 2D rectangular and irregular packing tasks (see Appendix C for Internet addresses). Most irregular nesting systems include or provide optional modules with algorithms for rectangle packing.

Table 4.16: Commercial packages for rectangular packing problems

product	**company**	**description**
Cut Planner	Remarkable Software Ltd.	cutting optimisation for rectangular parts for multiple sheets
DEC++	Advanced Technology Institute	packing software for rectangular parts for multiple sheets
Itemizer	MID-OHIO SYSTEMS	PC-based packing system for rectangular items
Ritmo4	ALMA Manufacturing Software	rectangular and linear cutting optimisation software package for woodwork, mirror manufacturing, sheet metal and cutting plastics or composites
Shear/Miser	Fab/Trol Systems, Inc.	packing system for rectangular parts for multiple sheets with remnant tracking

Table 4.17: Commercial packages for rectangular and irregular packing problems

product	company	description
UG/Sheet Metal Nesting	Unigraphics Solutions Inc.	fully automatic and interactive nesting for the sheet metal industry
PowerNest	ALMA Manufacturing Software	fully automatic nesting of system mainly for the mechanical industry with; includes a specific function for the nesting of rectangular parts; available as library containing the nesting algorithms
Nestix2	Nestix Oy	automatic nesting module part of 3D CAD package for the shipbuilding industry
HUMANCAD Optitex	Humantec Industriesysteme	template construction and nesting for the apparel, the upholstery and the automotive industry; interactive and automatic nesting is supported
Materials Manager 2.7	MID-OHIO SYSTEMS	general purpose true-shape nesting and optimisation system; runs inside of AutoCAD; automatic and interactive nesting
OptiNest	Remarkable Software Ltd.	general purpose true-shape nesting program; automatic and interactive mode
SigmaNEST	SigmaTEK Corporation	true-shape automatic nesting system; automatic and interactive mode; for the metal and wood cutting industry
SS-NEST	Striker Systems	automatic nesting software for sheet metal
SS-STRIP	Striker Systems	automatic strip layout; also interactive
AccuFABTM	DynaSys, Inc.	multi-purpose fully automatic nesting system; also interactive
Nestlib	Geometric Software Solutions Co. Ltd.	fully automatic true-shape automatic nesting; also available as DLL containing the nesting algorithms
SMP/IS	Merry Mechanization, Inc.	fully automatic and interactive true-shape nesting system sheet metal fabrication; with modules for just-in-time requirements, special cutting processes etc.
PINS XL 2000	Generative N/C Technology	automatic nesting system
Helix Manufacturing	MICROCADAM, Inc.	automatic true-shape nesting system for the sheet metal industry
AutoNest	Virtek	fully automatic dynamic nesting software specifically designed to optimise yields on leather hides
Friendly 2D	DNT	automatic nesting software for the leather cutting industry; fully automatic and interactive mode
Lasernest	Humantec Industriesysteme	fully automatic nesting software for the leather industry

Since material utilisation is one of the major industrial issues, most nesting algorithms work with true-shape representation rather than approximations. Other desired features of nesting packages are a high speed of operation and the capability to perform fully automatic nesting. Most of the commercial products on the market at present offer the following major features with respect to the packing process:

- flexibility aimed at industrial problems from a wide area of applications
- support for a range of file formats, e.g. DXF
- fully automatic nesting
- possibility for interactive modifications

- true-shape nesting
- suitability for regular and irregular problems involving rectangular or non-rectangular sheets
- facility to nest multiple sheets of different sizes in a single run
- in-hole-nesting, i.e. facility to optionally nest parts within the holes of larger parts
- support for incremental nesting, i.e. nesting on a pre-nested sheet(s)
- part rotation (user defined step angle for each part)
- suitability for several cutting techniques
- cutting sequence generation

The features listed above are not a full description of the commercial packages but focus mainly on the packing aspect. Commercial nesting software typically offers more features especially with respect to the cutting process, providing support for many cutting techniques such as laser and plasma cutting and punching. Additional modules are available for special industrial needs such as part scheduling, cost estimation and inventory control. The price for the commercial products starts at $400 for rectangular packing systems and can be as much as $20,000 for irregular nesting products, depending on the complexity of the software and the modules acquired. Two of the nesting packages listed in Table 4.17 have been used for the evaluation of the rectangular and irregular packing algorithms developed in this work (section 5.5 to 5.7).

A number of products are available for 3D packing tasks specialised on container and pallet loading (Table 4.18). Although a first step has been made towards packing of irregular items and spaces in two of the packages in Table 4.18, there is still no commercial product available, that supports complex 3D shapes as they occur in the field of rapid prototyping. Research activities in this area have already started (Ikonen et al., 1996). With the availability of computational power continuously widening, it should not be long until commercial products for this task will be launched.

Table 4.18: Commercial packages for 3D container/ vehicle and pallet loading problems

product	**company**	**description**
PowerPack	Remarkable Software Ltd.	flexible 3D packing tool for packing boxes on containers, pallets etc.
CubeIQ	Remarkable Software Ltd.	loading optimiser for rectangular and irregular shaped containers, i.e. air craft
CARGOMANAGER	Gower Optimal Algorithms Ltd	3D system for packing boxes into containers, pallets etc.
PALLETMANAGER	Gower Optimal Algorithms Ltd	powerful package to pack pallets with rectangular boxes and cylindrical boxes

4.6 Conclusions

This review of solution approaches to 2D and 3D packing problems reveals that heuristic and meta-heuristic search methods have been implemented for the solution of a large variety of problems. The solution space of combinatorial problems is enormous and increases rapidly with the complexity of the problem, in particular with the geometry of the objects to be allocated. With most of the packing problems being NP-complete, heuristic search procedures are used, since exact algorithms cannot solve the problem in polynomial time. During recent years, researchers have proposed an increasing number of meta-heuristic approaches for the solution of rectangular and irregular packing tasks that offer the ability to search large and complex solution spaces in a systematic and efficient way.

4.6.1 Meta-heuristics

Genetic algorithms and to a smaller extent simulated annealing have been applied to 2D packing tasks. Despite some comparisons with problem-specific search processes and local optimisation methods such as hill-climbing, only a few attempts have been made so far to compare the performance of various meta-heuristics in the area of packing. Burke and Kendall (1999a) and Leung et al. (1999) carried out some research in this area. The first work indicated that tabu search and simulated annealing outperform genetic algorithms, whereas genetic algorithms were better in the second investigation.

In addition to that, most researchers in the area of genetic algorithms seem to take a successful search process of their particular implementation for granted. Furthermore, the operation of the genetic operators with respect to the outcome of the search process is hardly verified. This is of paramount importance where novel encoding structures and as a consequence problem-specific operators are proposed. A verification step would normally be quite straightforward and easy to implement. As most genetic algorithms make use of both genetic operators, omitting the cross-over operation reveals its impact on the final outcome and on the course of the search process. Despite the simplicity, this technique is not part of the standard test tools researchers use in this area. So far, it was only applied by Falkenauer (1996) and is referred to as naïve evolution.

A second, at least as powerful tool for the performance evaluation of genetic algorithms, is random search. Executed over the same number of iterations as the meta-heuristic algorithm, it can be used to assess the quality of the search space operators. Since the genetic operators as well as the neighbourhood moves are intended to guide the search process to good solution areas in the extremely large solution space, the outcome of a search which conducts a pure random exploration reveals how well this objective has been met.

4.6.2 Benchmarks

The discussion of performance comparison leads on to another topic which has not entirely been neglected, but certainly has not been resolved to satisfy the needs of a coherent research community. A key weakness behind much of the work is the lack of comparisons with known benchmarks in the widest sense by pointing at standard test methods as well as standard test problems. Despite some effort by two on-line libraries (section 5.3), there is no test suite available which could enable at present comparisons between algorithms intended for packing problems. Although some researchers acknowledge and regret this fact in their work, no further work has been done in this area. Performance evaluation mainly continues to only consider 'self-made' test problems, which are not publicly available in most cases. A commonly agreed test suite benefits the development of algorithms as well as the industrial user, who has to select the most appropriate packing method considering various criteria. Solution quality and computation time are only two out of many criteria to be considered.

Apart from the range of benchmark problems a number of standard packing methods is also of advantage for performance comparison. Especially in the area of rectangular packing, a large number of simple heuristics exist which could be applied as such a standard method for this purpose. Simple heuristics are easy to implement and achieve very dense layouts under certain conditions (section 4.2.1). As meta-heuristics are expected to perform comparatively better in terms of solution quality this may seem to be a waste of time. However, even a relative comparison to a standard method is a useful and a valid measure for comparisons between more complex algorithms. Although the task of determining a benchmark method may be more difficult in irregular packing, some of the heuristic search techniques (Albano and Sappupo, 1980; Oliveira et al., 2000) have proved to be flexible and extremely powerful on a variety of test cases. They also were already used by a few other researchers. Therefore even a benchmark method for irregular packing could be established.

In order to keep test problems flexible regarding parameters such as problem size, aspect ratio of items or availability of known optimum solution, a further task is the design and implementation of problem generators. Whereas this is certainly simpler for the rectangular strip and bin packing problems, a careful consideration of parameters to determine the irregular packing task in certain applications is necessary for the irregular case.

4.6.3 Solution Approaches

The literature review shows that a large variety of genetic algorithms and simulated annealing approaches were developed for strip and bin packing problems. The major features of the existing solution approaches with respect to encoding technique, shape representation and algorithm design are briefly summarised in the following section which highlights their major advantages as well as disadvantages.

4.6.3.1 Encoding

The strength of genetic algorithms lies in the ability to search large and complex solution spaces in a systematic and efficient way. Not being dependent on a particular problem structure allows the user to utilise different methods for the encoding of the genotype. The performance of a search process is strongly related to the representation of the packing problem. It is important that the encoding technique, which describes possible packing patterns, utilises characteristic features in the packing schemes. It may be advantageous to design the data structure such that sub-structures of layouts are accessible and can easily be manipulated. For packing problems, order-based chromosomes can be used to represent packing sequences. An appropriate modification of the data structure may maintain certain efficient sub-structures of the layout. At the same time the genetic operators need to be adapted to the encoding technique, so that they support the inheritance of important layout features, which are meaningful and effective for the packing objective.

4.6.3.2 Type of Approach

With respect to the packing problems described in the review, three types of solution approaches involving genetic algorithms can be distinguished. The common feature of most genetic algorithms developed for packing problems is their two-stage approach. The concept of the genetic algorithm is used to explore and manipulate the solution space, whereas a second procedure is needed to evaluate the solutions generated. Therefore the phenotype needs to be constructed in order to check quality and feasibility of packing scheme. As the genetic algorithm is used to determine the sequence of packing, a placement routine is then needed to find the allocation of the items on the object.

A heuristic decoder can limit the genetic algorithm. It may not support the inheritance of certain features by the offspring since the domain knowledge is hidden in the placement routine. In order to avoid the dependency of the performance of the genetic algorithm on the decoding method, it seems beneficial to develop a data structure that calculates the fitness from the genotype rather than the decoded phenotype. A second category of solution approaches attempts to incorporate more layout information into the data structure of the genetic algorithm. Some additional rules are still needed to fix the position in the layout. The third group of genetic solution methods resolved this matter by transferring the genetic search process into the 2D layout domain. Since the genetic operations are performed directly on the 2D shapes this method does not require an encoding technique.

The concept of performing a search process entirely in the layout domain has long been applied in simulated annealing and tabu search (Marques et al., 1991; Marques et al., 1998) and is common to most approaches in this area. Applying an indirect optimisation process via the use of an encoding is a very recent idea (Faina, 1999; Burke and Kendall, 1999b).

The benefits of an operation in the 2D space are evident, since it enables a meaningful implementation of the abstract meta-heuristic principles and operators describing concepts such as neighbourhood and

neighbourhood moves as well as features of the phenotype, cross-over and mutation. The operation on the layout rather than an encoded data structure raises a number of other issues to be considered, such as overlap. Overlapping configurations are invalid solutions and need to be resolved either by rejecting, correcting or temporarily accepting them. Rejection wastes precious computation time and may result in less dense layouts for highly irregular shapes, since the slightest change in position or rotation could lead to invalid configurations, which will no longer contribute to the search process. Correcting invalid configurations seems a better option, since often only minor re-positioning is necessary to obtain a valid solution. This contributes additionally to the computation time, especially, if the re-positioning task turns out to be more complex involving several steps.

Accepting an invalid configuration temporarily offers a balance between these two extreme measures, since often a series of moves automatically will result in a valid solution. This is beneficial when shapes pass over each other in order to reach enclosed areas in the layout or other shapes. The acceptance of an invalid layout requires a penalty term in the evaluation function. The penalty expression needs to be carefully designed balancing between layout compaction and overlap generation. According to Davis (1991) penalties are less efficient to guide the search than using a decoding algorithm that avoids producing constrained results.

The opposite approach is taken by hybrid algorithms, where the search process operates on an encoding. The packing rules applied by the decoding algorithm guarantee that all solutions considered in the search process are valid. There has been much speculation on whether this is beneficial with respect to the transmission of specific layout to the next generation and the next state in the neighbourhood respectively. The literature is reluctant so far to give a satisfactory answer to this problem. The different solution approaches have not been compared with each other. Since much of their performance strongly depends on the packing task with respect to the formulation of the objective and the shapes involved it is not sufficient to judge their performance purely on the basis of the packing densities achieved. This emphasises the need for commonly accepted benchmark tests and problems (section 4.6.2).

4.6.3.3 Computation Time

The decoding method has a great influence on the computational effort of the hybrid algorithm. The importance of computation time in a certain nesting task depends on the respective application. Meta-heuristics are computationally very expensive due to the high number of function evaluations. This results in long run times especially in irregular problems, where geometric computations required for the nesting process are time intensive. Type and implementation of geometric algorithms contribute to the computation time, especially when a high accuracy for the shape approximation and description is used.

4.6.3.4 Shape Representation

The representation of the shapes to be placed is strongly related to the strategy chosen to tackle the nesting task. Two main methods can be distinguished in the irregular examples in the literature. In approaches where the allocation in the object is found on the basis of a scanning process, shapes are represented as matrices. The second option is the description in the form of geometric primitives such as polygons and circles and implies that geometric routines are used to compute the relation between items in the layout. The approximation of arbitrary items as they occur in the textile and metal industry by concave polygons or the convex hull raises the issue of accuracy. For instance, a popular method in the nesting process is the clustering of two polygons using the convex hull or some outer boundary of the configuration in the subsequent nesting steps (Burke and Kendall, 1999b). The convex hull is not an accurate description of the partial layout, but might be sufficient for the generation of a layout of acceptable quality. The basic question to resolve in this context is how much accuracy is needed in terms of layout quality and how much is affordable in terms of computation time.

The issue of shape representation reflects on the encoding technique. In a hybrid algorithm the domain knowledge is stored outside the meta-heuristic part, since an additional procedure is used for decoding into the phenotype. In approaches, that do not involve a decoding algorithm, the geometry of the figures necessarily needs to be considered in the data structure (e.g. Jain and Gea, 1998).

4.6.4 New Solution Approach to Rectangular and Irregular Packing Problems

The major objective of this investigation is to develop a general meta-heuristic solution approach to 2D rectangular and irregular packing problems involving strip as well as bin packing. As the review of the literature shows genetic algorithms are most favoured for this task and obtained promising results in a large range of applications. They are therefore primarily considered in this work. At the same time a comparison with other meta-heuristic methods is regarded as highly important.

4.6.4.1 Problem Representation and Outline of the Algorithm

The issue of representation, which refers to a description either in the 2D domain or in form of encoding technique, has not been resolved satisfactorily in the literature considering all its advantages and disadvantages. Therefore it has been decided to use an encoding technique. After all the natural example genetic algorithms are derived from is based on this principle. Before any assumptions concerning the quality of a certain encoding can be made, it is important to first investigate the traditional, though very basic, order-based technique, which by now can be seen as the standard encoding technique for combinatorial problems. After comparison with more sophisticated principles it may eventually be developed further.

Rectangular Strip Packing:

The chosen approach is a two-stage order-based technique, where the meta-heuristic method acts as tuner for a packing routine. Approaches which used this principle (Kröger et al., 1991a, b and 1993; Hwang 1994; Jakobs, 1996; Liu and Teng 1999) outperformed heuristic solutions with respect to rectangular strip and bin packing problems. The heuristic decoders used by Jakobs, Liu and Teng as well as Hwang et al. show a certain weakness since they do not necessarily place the rectangle into the best position from the point of view of a human operator. One of the design tasks of the hybrid algorithm is to improve the packing routine with respect to the packing density allowing access to enclosed areas in the partial layout.

At the time this work was started the publication by Leung et al. (1999), which presents a similar idea, did not exist, but was conducted independently. In order to establish the performance of the improved heuristic decoder comparisons are made to other hybrid techniques in this area as well as the heuristic algorithms themselves. The latter one will help to identify the advantages of the hybridisation concept for the layout generation.

In packing problems, the simulated annealing approaches have always operated directly on the 2D layout. The order-based encoding technique had not been incorporated into this search method at the time this investigation was outlined and seemed worth exploring. In the meantime, a number of simulated annealing approaches were based on this principle (Faina, 1999; Burke and Kendall, 1999b; Leung et al., 1999). However, the rectangular ones do not consider the orientation of the items as a neighbourhood move.

Rectangular Bin Packing:

In addition to strip packing, 2D rectangular bin packing is considered as a second application of the hybrid meta-heuristic algorithms. Apart from one investigation by Hwang et al. (1994), most genetic algorithm approaches focus on 1D bin packing. Although Hwang's method uses a heuristic decoder, it does not necessarily allocate an item in the best way to generate a dense layout. The improved decoder used in the strip packing case should also achieve better bin layouts. Unlike in the literature (Hwang, 1994; Faina, 1999), different-sized bins are used incorporating the sequence of the objects into the encoding such that it can be manipulated by the search process.

Irregular Strip Packing:

The concept of the hybrid algorithms developed for rectangular packing tasks is also applied to irregular packing extending the bottom-left placement routines to polygons. Although hybrid principles have been applied in the literature in this area they have some drawbacks. The irregular placement algorithm used by Jakobs (1996) considers the irregularity of the packing task only after the search process. This can be improved by treating the shapes as polygons during the decoding stage rather than as rectangles. Many redundant calculations in Dighe and Jakiela's (1996) method, that applies a low-level search process in connection with a sliding routine to position an item, can be avoided using a simple decoder. In this work

a number of decoding methods are investigated which are based on a sliding principle, placing the polygons in a bottom-left manner similar to the rectangular placement routines.

4.6.4.2 Performance Testing

When testing algorithms, it is important to consider their application. Apart from the solution quality, computation time is also an important issue in industrial applications and strongly depends on the problem size. To judge the suitability of the algorithms for a certain packing task, the impact of the problem size on the performance of the hybrid algorithms is investigated and compared with simple heuristic methods.

Problem Generators:
Due to the lack of benchmark problems in this area (section 5.3.2) new problem generators are developed for the rectangular case to create test problems automatically under consideration of certain parameters. So far, no algorithms for the automatic test case generation for 2D strip and bin packing problems have been published.

Benchmark Problems:
One of the major objectives in this work is to compare the performance of the hybrid algorithms with other meta-heuristic solutions. Since there is no standard test method, comparison is only possible where authors compare their work with other algorithms in the area or where the same test problems can be used. Unfortunately, the first option is not standard praxis, apart from a few exceptions (Blazewicz et al., 1993; Oliveira et al., 2000). Therefore a list of test cases will be compiled that were applied in earlier solution approaches to rectangular as well as irregular packing tasks. In that way, at least comparisons to a few other methods are possible.

Benchmark Tests:
At the same time an attempt is made to identify a possible benchmark method for rectangular packing tasks, which is easy to implement and performs reasonably well with respect to packing density. The heuristic nature of the method will help to reveal the merits of meta-heuristic search techniques over heuristic ones.

Commercial Nesting Software:
Since the meta-heuristic methods in this work have been developed for looking at possible industrial applications, standard industrial practice should be taken into consideration when judging the suitability of algorithms for certain tasks. Currently, the manufacturing industry has the choice between a number of commercial nesting packages. Apart from one work (Corno et al., 1997), researchers in this area do not seem to regard them as a possible and inevitable way to test whether their algorithms match industrial practice, if intended to do so. A number of software packages, which are suitable for rectangular and

irregular strip and bin packing tasks, have been identified in section 4.5. Two of them are used to measure the performance of the hybrid algorithms developed in this work.

Testing of Meta-Heuristics:

The performance of the hybrid algorithms is measured by applying a comprehensive test procedure. A comparison is made with the simple heuristic packing routines, which are used to hybridise the meta-heuristic methods. Since the meta-heuristics function as a tuner for the simple algorithm, they are expected to perform better in terms of layout quality.

In order to see whether a possible performance gain is purely due to a higher number of iterations, random search is applied testing the suitability of the genetic and neighbourhood operators to guide the search process into good areas of the solution space. The operation of the cross-over operator is verified studying its impact on the final solution quality with naïve evolution. Possible merits of seeding are analysed.

Apart from genetic algorithms, other meta-heuristic and heuristic methods were applied to packing tasks in the literature. Since little evidence exists whether some are more favourable than others, a selection has been included in this work involving hill-climbing, simulated annealing and stochastic optimisation, which has been previously used in the literature (Pargas and Jain, 1993). A comparison with some of the heuristic search techniques in the literature is possible through the application of the same test problems.

4.6.4.3 Implementation

One of the key issues in algorithm development is efficiency and reliability. Especially, the nesting algorithms used in irregular packing require a large number of complex and time intensive geometric computations. The implementation of geometric algorithms does not only consume precious time during the algorithm development, which could be better spent at different stages of the project, but also seems like "re-inventing the wheel" considering the amount of professional software available in this area. A software library has been used in the project, which offers reliable data types and routines for geometric primitives. An overview of software packages for computational geometry as well as description of the ones used is given in the following chapter.

5. Computational Geometry, Benchmarks and Algorithms for Rectangular and Irregular Packing

5.1 Introduction

Packing and nesting programs make intensive use of geometric algorithms. Unlike rectangular problems, where it is sufficient to use comparison operations of co-ordinates to determine where to allocate a rectangle, the packing of irregular shapes relies heavily on the use of geometric algorithms. Geometric algorithms are especially used in the decoding stage of meta-heuristic hybrid algorithms. The frequent use of the decoder function during the simulation process makes an efficient implementation of geometric data types and algorithms of paramount importance in achieving acceptable computation times.

A number of geometric computing packages have been developed over recent years that provide libraries of geometric and non-geometric data types and algorithms. In addition, they offer tools for the implementation, application, evaluation and visualisation of geometric algorithms. The ability to visualise geometric objects and to animate geometric algorithms is extremely important for the demonstration and for the development of geometric programs. Since they address all major issues of computational geometry algorithms such as robustness, arithmetic precision, efficiency and visualisation, whose implementation is no small task, one of these packages has been used in this project for the implementation of the nesting algorithms. An overview of some of the available software packages is given in section 5.2.

In this work a family of genetic algorithms has been developed to tackle rectangular as well as irregular packing problems on strip material. In addition, the rectangular case has been extended to multiple objects. To test the performance of new algorithms relative to existing algorithms, benchmark problems are needed. The scope of published benchmark problems for 2D rectangular and irregular packing, has been fairly limited in terms of object size, number and dimension of items. A systematic investigation into the performance of new algorithms requires the use of a range of test problems. Therefore problem generators have been developed to create 2D rectangular strip packing tasks.

The comparison of new meta-heuristics and heuristics for packing with published methods has limitations when benchmark problems are used. A true picture of the capabilities of packing algorithms can only be obtained if they are also compared to industrial software. Therefore two commercial nesting packages have been included in this investigation and are briefly introduced in section 5.4.

In sections 5.5 and 5.6 a family of meta-heuristic hybrid algorithms is developed to solve rectangular and irregular packing problems on strip material. The main focus will be upon evolutionary techniques as a

result of other research in this area (section 4.3). First the 2D rectangular packing problem is considered, where a fixed set of items has to be allocated on a single object. Four heuristics, which belong to the class of packing procedures that preserve bottom-left stability, are hybridised with meta-heuristic algorithms and local search heuristic. These techniques are then extended and applied to irregular nesting problems. Finally, a third problem category involving rectangular items and multiple objects is approached with evolutionary techniques (section 5.7).

5.2 Computational Geometry

Over the last decades researchers in computational geometry have developed a number of efficient algorithms for solving a wide spectrum of geometric problems. These research activities focused on design and analysis of geometric algorithms, many of which have never been implemented. Most of the functioning implementations have been treated in isolation and therefore have remained inaccessible to application oriented researchers within the computational research community and other areas such as computer graphics, robotics and image processing. Unlike other areas of computing there is no standard library of data structures and algorithms for combinatorial and geometric computing. Continuous re-implementation of basic data structures and algorithms slows down progress of algorithm development for research and industrial applications.

The implementation of geometric algorithms is not a small task. Software for computational geometry must deal with a number of issues not faced by other software applications. The (built-in) library types such as integers, reals, vectors, and matrices are usually sufficient in statistics, numerical analysis and linear programming. Geometric data typically involve a combination of numerical and combinatorial relationships. Combinatorial and geometric computing therefore relies heavily on types such as stacks, queues, sequences, graphs, points, lines and convex hulls. The visualisation of geometric objects and the animation of geometric algorithms are important in computational geometry for the purpose of demonstration and debugging.

The increased interest in implementation issues of geometric algorithms in many disciplines has resulted in the publication of several collections of geometric algorithms and software on the Internet (see Appendix D). Several software projects have been launched with the aim of developing comprehensive libraries of geometric data types and algorithms. Here effort is directed towards building a geometric computing environment, which provides an efficient and robust framework of tools for the implementation, application, evaluation and demonstration of geometric algorithms. Table 5.1 provides an overview of the major software developments for computational geometry. Some of these projects are still ongoing.

Table 5.1: Overview of software packages for computational geometry

name of package	description	language	references
CGAL (Computational Geometry Algorithms Library)	library of geometric and non-geometric data structures and algorithms	C++	Overmars (1996), Schirra (1996)
LEDA (Library of Efficient Data Types and Algorithms)	library of geometric and non-geometric data structures and algorithms	C++	Mehlhorn and Näher (1995, 2000), Mehlhorn et al. (1997)
PlaGeo SpaGeo	library for planar geometry library for spatial geometry	C++	Giezeman (1994)
Computational Geometry Workbench	library of geometric data structures and algorithms; also geometrical programming environment for creation and manipulation of geometric objects; animation of geometric algorithms	Smalltalk/ V	Knight et al. (1990), Epstein et al. (1994)
XYZ GeoBench	library of computational geometry algorithms and user interface; focus on fundamental algorithms	Object Pascal	Schorn (1990), Nievergelt et al. (1991)
GeomNet	system for performing distributed geometric computing over the Internet	Java	Barequet et al. (1997)
Geomview	interactive 3D viewer for geometric objects provided by external programs		Munzner et al. (1995)

5.2.1 Brief Description of Major Computational Geometry Packages

The common objective of the software projects listed in Table 5.1 is the development of powerful, robust and efficient tools for computational geometry, which can be used by non-experts. The implementation of the packages is directed towards different objectives, so that they complement each other. Whereas most of the packages focus on the development of comprehensive libraries containing a large number of geometric data types and algorithms, other packages provide an interactive user interface. Two of the projects mentioned in Table 5.1 have a different objective. One of them is intended as a powerful viewer for external programs, the other provides a large computational geometry resource available over the Internet using distributed computing. Some of these projects are still under development. Most geometry libraries and programs are not in the public domain, but can be used free of charge for academic research and teaching and under a commercial license. Some of these software packages are briefly introduced in the sequel.

LEDA (Library of Efficient Data Types and Algorithms) is a C++ library of efficient data types and algorithms. It serves as a platform for combinatorial and geometric computing on which application programs can be built. It also contains a large range of non-geometric data types like *stack*, *dictionary*,

linked list and *graph* and can be used like the STL libraries on all major platforms. Section 5.2.2 provides a more detailed description of LEDA.

The goal of the CGAL (Computational Geometry Algorithms Library) project is to design and implement a robust, easy to use, and efficient C++ software library of geometric data types and algorithms (Overmars, 1996; Schirra, 1996). CGAL is a joint project between several universities and is conducted in close collaboration with a number of European companies. In order to deal with the complexity in geometric computing, the LEDA number types and combinatorial data structures are used. The library consists of a kernel that contains basic geometric primitives and operations on them, a basic library with a large collection of standard geometric algorithms, and support libraries, which provide interfaces to other packages, e.g. for visualisation and I/O operations. In addition, collections of geometric routines will be provided that are directed to specific application domains, such as geographic information systems, computer graphics, robotics, and computer vision. Compared to LEDA, CGAL has a more extensive collection of 3D data types and algorithms, however, it is so far only available for UNIX systems.

The Computational Geometry Workbench by Epstein et al. (1994) is a Smalltalk V/Mac environment for creating, editing and manipulating geometric objects. It provides tools for the demonstration and animation of geometric algorithms and allows the implementation of new algorithms. The software is no longer supported.

The XYZ GeoBench (eXperimental geometrY Zürich) by Nievergelt et al. (1991) is a workbench for geometric computation on Macintosh computers. It provides an interactive user interface similar to a drawing program which can be used to create and manipulate geometric objects such as points, line segments, polygons, etc. In addition, the user interface provides access to algorithm animation. The XYZ program library contains abstract data types, a large number of standard algorithms for 2D problems and some for three and higher dimensional computational geometry.

PlaGeo (planar geometry) by Giezeman (1994) is a library of basic planar objects and geometric algorithms. SpaGeo (spatial geometry) is the 3D counterpart of PlaGeo containing descriptions for geometric objects such as planes, polyhedra and spheres. The PlaGeo/ SpaGeo and the XYZ GeoBench project have ceased and are superseded by CGAL.

GeomNet is different from the packages mentioned above since it provides a system for performing distributed geometric computing over the Internet. There is no standard for the representation of geometric data and implementations. Geometric programs typically use their own formats, making it difficult to combine elements from various packages. Barequet et al. (1994) have started to develop an 'Internet computing' framework called GeomNet. GeomNet consists of a family of geometric computing servers that execute a variety of geometric algorithms on behalf of remote clients. This can be either users interacting through a Web browser interface or application programs connecting directly through sockets. A prototype implementation of part of this is available.

Geomview is an interactive 3D viewer written by the Geometry Center at the University of Minnesota (Munzner et al., 1995) for viewing and manipulating geometric objects. It can be used as a viewer for static objects or as a display engine for external programs, which produce dynamically changing geometry. The author of the external program can therefore concentrate on the implementation of the desired algorithm and leave the display aspects to Geomview that also provides an interactive control for motion and appearances such as lighting, shading, and materials.

Despite the large number of geometry packages, the choice for this investigation has been somewhat limited due to the project specifications and the state of development of the respective packages. For this research the LEDA package has been chosen, because of the support for C++ on the Windows NT platform. It provides a comprehensive and flexible range of 2D geometric data types and algorithms, which correspond well to the needs of geometric computation for packing and nesting problems. In addition, it can be easily implemented in external programs and offers graphical support. A more detailed description of the LEDA package is given in section 5.2.2. The ongoing development and support of the library make it an even more powerful tool to use. Throughout this project version 3.8 has been used. Subsequently, version 4.0 with extended capabilities on graph algorithms has been released.

5.2.2 LEDA (Library of Efficient Data Types and Algorithms)

The LEDA project was started in 1988 by Mehlhorn and Näher (1995, 2000) at the Max-Planck Institut für Informatik, Saarbrücken with the objective of building a small, but growing library of data types and algorithms in a form that allows them to be used by non-experts. The LEDA package was first distributed in the summer of 1990 consisting of a first version of the combinatorial part. Shortly after, the library was extended by a first implementation of the 2D geometry library and an interface to the X-Window system for graphical input and output.

Following this, the template mechanism to realise parameterised data types was introduced along with a more efficient graph data type and arbitrary precision number types used for robust implementations of geometric algorithms. In addition, the LEDA memory management system was extended to user-defined classes while at the same time the efficiency of many data types and algorithms were improved. Over the years the LEDA user community has grown substantially in size. Industrial use started in 1994 with the foundation of LEDA Software Ltd., which has recently been transferred into Algorithmic Solutions Ltd. The further development of the computational geometry aspects of the library will be carried out in close co-operation with the CGAL project.

5.2.2.1 Principal Features of LEDA

The LEDA library consists of two major parts: a combinatorial part containing data structures and graph algorithms and a geometric part which describes data types and algorithms for 2D and 3D geometry. The main features of LEDA are:

- LEDA provides a sizeable collection of data types and algorithms. This collection includes most of the data types and algorithms described in the text books of the area (O'Rourke, 1998; Laszlo, 1996).
- LEDA gives precise and detailed specification for each data type and algorithm mentioned above. The specifications are short, general allowing several ways of implementation, and abstract in order to hide all details of the implementation.
- LEDA contains efficient and robust implementations for each of the data types and algorithms.
- LEDA is implemented in form of a C++ class library. It can be used with almost any C++ compiler that supports templates.
- LEDA is available for all major system platforms (UNIX, Windows NT etc.).
- For many data structures access by position is important. LEDA uses an item concept to cast positions into an abstract form. An abstraction of 'pointers into a data structures' allows access to a data structure by position and is known as the item concept. Most of the LEDA data types use this concept.
- LEDA uses the iterator concept of the C++ Standard Template Library (STL) to deal with collections of objects. An iterator is used to access all items in a linear sequence step-by-step.
- LEDA contains a comfortable data type *graph* that also offers the standard iterations.
- LEDA provides a collection of simple and basic geometric data types and routines for 2D and 3D computational geometry.
- The graphics interface of LEDA provides the data type *window* that is used for the graphical input and output of basic 2D geometric objects. LEDA offers a separate data type *GraphWin* which can easily be used to construct, display and manipulate graphs and to animate and debug graph algorithms.
- LEDA is available via the Internet (Appendix D). The distribution contains all sources, installation instructions, technical reports, and the user manual. LEDA can be used free of charge for academic research and teaching as well as commercially under license.

5.2.2.2 Organisation and use of the LEDA library

The implementation of most LEDA data types and algorithms are precompiled and contained in five libraries. They can be linked to C++ application programs. The main LEDA library contains the

implementations of all simple data types, basic data types and priority queues. The graph library consists of the implementations of all graph data types and algorithms. There are two separate libraries for the geometry in the plane and in the 3D space. The functions for the graphic interface under the X11 window system are described in the graphics library. Table 5.2 gives an overview over the major categories of data types and algorithms contained in the various LEDA libraries including some examples for each category.

Table 5.2: Major categories of data types and algorithms in the LEDA library

data type/ algorithm category	**examples**
simple data types and basic support operations	strings, input/ output streams, files
number types and linear algebra	rational numbers, integers of arbitrary length, matrices
basic data types	1D arrays, 2D arrays, stacks, queues, linear lists, sets, dynamic trees
dictionaries	dictionary arrays, hashing arrays, maps
priority queues	priority queues, bounded priority queues
graphs and related data types	parameterised graphs, undirected graphs, node arrays, edge arrays, Markov chains
graph algorithms	shortest path, maximum flow, minimum cost flow
graph iterators	node iterators, edge iterators, adjacency iterators, breath first search, depth first search
basic data types and algorithms for 2D geometry	point, segment, ray, circle, polygon, rational points convex hull algorithms, triangulation algorithms
advanced data types for 2D geometry	2D dictionaries, set of intervals
basic data types and algorithms for 3D geometry	3D points, planes convex hull algorithms
graphics	colours, windows, menus, postscript files

The usage of the data types and algorithms is described in the LEDA manual. The class declarations are contained in individual header files that are included in the user program before a specific data type can be used.

In order to use LEDA together with software packages like CGAL the avoidance of name conflicts becomes an important issue. Since name spaces are not yet available, LEDA offers a prefixing mechanism to avoid name conflicts. The libraries contain class name symbols with the common prefix '`leda_`'. Macro routines, which are automatically implemented by the inclusion of a LEDA header, allow the user to work with unprefixed class names as used throughout the LEDA manual. During compilation all unprefixed type names will then be transferred into prefixed ones.

Iterators are a powerful technique in object-oriented programming. In general, an iterator is a small object associated with a specific kind of linear sequence. An iterator is used to access all items in a linear sequence step-by-step. For instance, it can be used to feed an algorithm with a set of input data of the same type. There is no need to copy the data from the applied container to a library container before they can be fed into an algorithm. LEDA provides different iterator classes in connection with the graph type offering the user more control than the simple step-by-step access (Table 5.2).

LEDA supports the use of user-defined parameter types. A user-defined class type *T* can be used as actual type parameter in a container class. For instance *list<T>* describes a doubly linked list of the class type *T*. In order to use a user-defined class type, the following operations have to be provided: a constructor taking no arguments, a copy constructor, an assignment operator, input/ output operators, a compare and a hash function.

Since the use of LEDA throughout this project has primarily concentrated on 2D geometry library, the basic geometric data types and algorithms will be explained in the following section.

5.2.2.3 Overview of Basic Geometric Data Types and Algorithms

The 2D geometry library of LEDA provides a collection of simple data types for computational geometry, such as *point*, *circle*, and *polygon*, and operations on these types. The first kernel (e.g. the data types *point* and *segment*) was restricted to 2D geometry and used floating point arithmetic as the underlying arithmetic. Since it is difficult to implement reliable geometric algorithms based on this kernel, the kernel was further developed and is now based on exact rational arithmetic (e.g. the data types *rat_point* and *rat_segment*). This kernel is, however, still restricted to two dimensions. The geometric classes listed below are included in the 2D library.

- points
- segments
- straight rays
- straight lines
- circles
- polygons
- generalised polygons
- rational points
- rational segments
- rational rays
- straight rational lines
- rational circles
- rational polygons
- rational generalised polygons

Since the more complex geometric objects such as polygons can be created from primitive ones, like a polygon built from a series of vertices or segments, the user normally has a choice between a number of constructors. Typically, each class has a series of member functions, which allow the return of the individual co-ordinates and execution of translation and rotation operations, and the calculation of intersection and distances. All classes support equality and identity tests as well as I/O operations. The library contains a series of non-member functions for each class. Table 5.3 lists some of the member and

non-member functions that are available for each class. With respect to the polygon classes there are two iterators available which repeat over the vertices or segments of a polygon. Similarly, the generalised polygon class, which describes a series of polygons in the plane, has an iterator accessing all polygons.

Table 5.3: The basic 2D classes of LEDA with some member and non-member functions

class	**examples of member functions**	**examples of non-member functions**
points rational points	translation by distance/ vector; reflection; rotation by angle/ point; distance to point	test if three points are collinear/ right-turn/ left-turn; area included
segments rational segments	return of co-ordinates, slope; horizontal/ vertical test; intersection with segment	test if two segments are parallel; compare slopes; compute orientation
straight rays rational rays	contains segment/ point; compute slope	compare slopes; compute orientation
straight lines straight rational lines	distance to point; return point/ segment on line, compute perpendicular segment	compare slopes; compute orientation
circles rational circles	return centre/radius; contains point; compute tangent	-
polygons rational polygons	return vertices/ segments; test if simple; intersection with segments; size; area	generate polygon with all points on circle
generalised polygons rational gen. polygons	return vertices/ edges/ polygons; intersection with segment/ line; union; intersection	

The LEDA 2D library also provides a number of plane algorithms. All functions listed below work for geometric objects based on both floating-point and exact (rational) arithmetic. In particular, *point* can be replaced by *rat_point*, *segment* by *rat_segment*, and so on. Only the rational data types, however, are guaranteed to produce correct results for all inputs.

- line segment intersection
- triangulations
- convex hulls
- closest pairs
- Voronoi diagrams
- miscellaneous functions (e.g. bounding box, test for simple polygon)
- functions to obtain properties of geometric graphs (e.g. clockwise or anti-clockwise ordering etc.)

5.3 Benchmark Problems

Efficiency and solution quality are important aspects for the evaluation of algorithms. In order to establish the performance of new algorithms relative to existing ones benchmarks are needed. A series of benchmark problems for cutting and packing was published in the literature. Collections of benchmark

problems for cutting and packing are provided in two online libraries on the Internet. The OR-library which is maintained by Beasley (1990) at Imperial College, London (Appendix E) contains an extensive collection of benchmark problems for a variety of areas within operations research including a section on cutting and packing problems. The second library focuses exclusively on cutting and packing and is provided by SICUP (Special Interest Group in Cutting and Packing, Appendix E).

The number of problems published, which are suitable for the needs of this project, however, is fairly small. So far, the online libraries only provide rectangular test problems, where the focus is on 1D strip and bin packing problems. These benchmark tests are limited in terms of problem type, problem size and object and item dimensions. For a systematic investigation of algorithm performance a large range of test problems is needed. For that reason new problem generators for the 2D rectangular strip and bin packing have been developed (section 5.3.2). They allow the user to automatically create guillotineable and non-guillotineable problems with known optimum solutions as well as packing problems where the optimum layout is not known.

Despite the lack of benchmarks in irregular packing it is difficult to develop a problem generator in this area. Especially in cutting and packing, most packing algorithms are developed for industrial purposes and therefore performance testing should also be carried out using industrial data. Industrial packing problems are very specific and it is not an easy task to mimic their properties by algorithms. Artificial data tests can be suitable, mainly at the early testing stage. Since they do not have to be very extensive, they can be created manually. It is possible to obtain a series of benchmark problems for a number of industrial applications from the literature. Therefore the effort of designing and implementing algorithms to generate sets of irregular problems was not within the scope of this research.

An overview of the benchmarks published for rectangular and irregular problems used to measure the performance of our algorithms is provided by section 5.3.1. Section 5.3.3 lists the test problems constructed either with the aid of the above mentioned problem generators as in the case of rectangular packing or manually for irregular packing.

There is a need for widely available benchmark problems and problem generators in many areas of cutting and packing. Researchers should be encouraged to both publish the instances of their test problems in conjunction with their work and make them accessible in one of the online libraries, which are updated regularly. This would facilitate performance testing and comparison of algorithms, avoid the time consuming manual design of test problems and the re-implementation of algorithms for problem generators. The series of test problems used in recent work (Hopper and Turton, 1999; 2000) have been made available in the OR and SICUP online libraries.

5.3.1 Published Benchmark Problems

A number of 2D rectangular and irregular problems in the area of strip packing have been used for the illustration of algorithms in the literature. Some of them were published by the authors in conjunction with their work. The lack of clearly defined benchmark problems in the area requires the use of these problems to establish the algorithm performance along with test problems constructed with the aid of problem generators. The following two sections provide an overview of the benchmark problems extracted from the literature. Details of all rectangular and irregular problem instances are listed in Appendix A and B.

5.3.1.1 Published Benchmark Problems for Rectangular Packing

This section describes the rectangular benchmark problems obtained from literature. The exact problem instances are listed in Appendix A.1.2. In the following two categories of problems are distinguished on the basis of their construction. For the first group at least one optimum layout is known (Table 5.4). This is not the case for the second category (Table 5.5). The tables also indicate the source of the problem instances. With the exception of problems J1 and J2, the dimensions were stated in the respective papers. For problems J1 and J2 sufficient information was given in sample layouts to extract the dimensions of the rectangles.

Table 5.4: Rectangular test problems from literature with known optimum solution

reference	name	size	source
Jakobs (1996)	J1	25	extracted from a sample layout
Jakobs (1996)	J2	50	extracted from a sample layout
Ratanapan and Dagli (1997b)	D2	21	dimensions stated
Kendall and Burke (1999)	Kendall	13	dimensions stated

Table 5.5: Rectangular test problems from literature with unknown optimum solution

reference	name	size	source
Ratanapan and Dagli (1997b)	D1	31	dimensions stated
Ratanapan and Dagli (1998)	D3	37	dimensions stated
Dagli and Poshyanonda (1997)	D4	37	dimensions stated

5.3.1.2 Published Benchmark Problems for Irregular Packing

A vast amount of literature is concerned with irregular packing. Most authors use industrial or artificially created test problems to demonstrate the operation of their algorithms. These problems can be divided into three main application areas: the textile industry (Table 5.6), the sheet metal industry with respect to

the shapes of the geometric objects (Table 5.7) and jigsaw puzzles (Table 5.8). The problem instances can be obtained from Appendix B.1.2.

In contrast to most of the rectangular test cases, authors fail to provide detailed data on their test problems. Recently some authors (e.g. Oliveira et al. 2000) have started to include problem instances in their publications. Apart from the test problems used by Oliveira et al. (2000) the test problems listed below have been extracted from sample layouts in the publications. The layouts were scanned and processed with digitising software. This software allows extraction of the co-ordinates of objects from an image by clicking on the respective vertices. Scanning and digitising implies a degree of inaccuracy. However, even if digitised geometric figures are not completely identical to ones in the published work, for most comparisons with other algorithms they will be sufficient to give an indication of the algorithm performance. Some authors published dimensions, however, these have not been suitable for this work and have therefore been scaled by a factor as indicated in Table 5.6 and Table 5.7.

Table 5.6: Irregular test problems from literature: textile industry

reference	name	size	problem type	shapes	source	factor
Ratanapan and Dagli (1997b)	Dagli	30	textile (assumed)	polygons, non-polygonal pieces with arcs	scanned from sample layout; approx. by polygons	1
Bounsaythip and Maouche (1997)	Mao	20	textile	polygons, non-polygonal pieces with arcs	scanned from sample layout; approx. by polygons	5
Marques et al. (1991)	Marques	24	textile	polygons, non-polygonal pieces with arcs	scanned from sample layout; approx. by polygons	
Albano and Sapuppo (1980)	Albano	24	textile	polygons, non-polygonal pieces with arcs	scanned from sample layout; approx. by polygons	
Oliveira et al. (2000)	SHIRTS	99	textile	polygons	co-ordinates stated	1
Oliveira et al. (2000)	TROUSERS	64	textile	polygons	co-ordinates stated	1
Oliveira et al. (2000)	SWIM	48	textile	polygons	co-ordinates; rounded to the nearest integer	1

Table 5.7: Irregular test problems from literature: artificially created problems

reference	name	size	problem type	shapes	source	factor
Jakobs (1996)	Jakobs1	25	artificial	polygons	constructed from sample layout	1
Jakobs (1996)	Jakobs2	25	artificial	polygons	constructed from sample layout	2
Fujita et al.	Fu	10	artificial	convex polygons	scanned from sample	

(1993)					layout	
Han and Na (1996)	Han	23	artificial	polygons	scanned from sample layout	
Blazewicz et al. (1993)	Blaz1	28	artificial	polygons, non-polygonal pieces with arcs	Oliveira et al. (2000)	1
Blazewicz et al. (1993)	Blaz2	20	artificial	polygons, non-polygonal pieces with arcs	Oliveira et al. (2000)	1
Oliveira et al. (2000)	SHAPES	43	artificial	polygons, recti-linear shapes	co-ordinates stated	1
Oliveira et al. (2000)	SHAPES2	28	artificial	polygons	co-ordinates stated	1

The jigsaw problems of Dighe1 and Dighe2 presented in Table 5.8 are not identical to the problems used by Dighe and Jakiela (1996), however similar in terms of number and shape of the items and the object. Dighe and Jakiela allowed clearance between the parts, so that the optimal packing density is less than 100%. The problems below have been modified such there is no clearance. Hence the optimum layouts in both cases have a packing density of 100%.

Table 5.8: Irregular test problems from literature with known optimum

reference	**name**	**size**	**problem type**	**shapes**	**source**
Dighe and Jakiela (1996)	Dighe1	16	jigsaw	polygons	constructed according to a sample layout in the paper
Dighe and Jakiela (1996)	Dighe2	10	jigsaw	polygons	constructed according to a sample layout in the paper

5.3.2 Generation of Benchmark Problems

For this work, a family of genetic algorithms has been developed to tackle packing problems on strip material as well as on multiple objects. For a systematic investigation of the performance of the algorithms developed, a large range of test problems was needed. The benchmark problems published in the online libraries and the literature are very limited with respect to the properties that are regarded as fundamental for this investigation. The problem generator needs to be flexible in terms of problem type and size as well as number and dimensions of items and objects.

A set of algorithms for the automatic generation of 2D rectangular packing problems has been designed and implemented. They allow creation of guillotineable and non-guillotineable problems with known optimum solutions (section 5.3.2.1) as well as packing problems where the optimum layout is not known (section 5.3.2.2) for single and multiple objects. The input of the problem specification is carried out via a user interface. The generated problem instances are stored in the form of text files listing the width and

height of the rectangles. Each problem can be displayed via the graphic interface provided by LEDA (section 5.2.2).

5.3.2.1 Perfect Packing Problems

A perfect packing problem is one for which at least one optimal solution is known. The optimal solution in this work is defined as the arrangement of a set of rectangles on a given object without leaving any gaps in the layout, i.e. the layout has a packing density of 100%. In orthogonal packing rectangles can only be packed parallel to the edges of the object distinguishing between two layout types: guillotineable and non-guillotineable patterns (section 2.2). Table 5.9 lists the parameters that have to be supplied by the user for these two problem types. The algorithms for the generation of instances of both problem types will be described in detail in the following sections. For the generation of guillotineable problems two algorithms have been developed referred to as Algorithm1 and Algorithm2.

Table 5.9: Input parameters for perfect packing problems

parameter	description
problem type	guillotineable or non-guillotineable
problem size	number of items the problem consists of
object	size of object
item ratio	maximum ratio between the two rectangle sides

a) Generators for guillotineable problems

Algorithm1:

In order to create problems with a known optimum layout, an object is chosen and then divided into smaller rectangles according to the design rule described in Figure 5.1. Applying Algorithm1 the total number of rectangles in the problem is provided in Equation 5.1.

$$n = 1 + 3 * i \qquad \text{Equation 5.1}$$

with n = total number of rectangles in problem

i = number of times loop has been executed

```
user input
calculate number of rectangles to produce (according to Equation 5.1)
repeat
    select randomly one of the existing rectangles (i.e. large rectangle)
    if start
        use object as large rectangle
    place random point into large rectangle
    construct horizontal AND vertical lines through point
    determine intersection points with borders of large rectangle
    create 4 new smaller rectangles (according to Figure 5.2)
    check ratio of these 4 rectangles
    if acceptance
        delete large rectangle
        add four small rectangles
        increment rectangle counter by 3
until number of rectangles reached
```

Figure 5.1: Algorithm1 for the creation of a guillotineable problem instance in pseudo code

An estimate of the problem size is provided by the user. Using the relation in Equation 5.1 the number of times the loop in Figure 5.1 needs to be executed can be calculated and will be rounded to the larger integer. The total number of rectangles is then re-calculated which may be equal or larger than the user input. A sample problem created with this algorithm is shown in Figure 5.3.

It is worth mentioning that the accurate number of items requested by the user could easily be achieved by including a modification step at the end of the algorithm that executes an appropriate number of splits or mergers on a few rectangles. This has not been implemented in the problem generators described here.

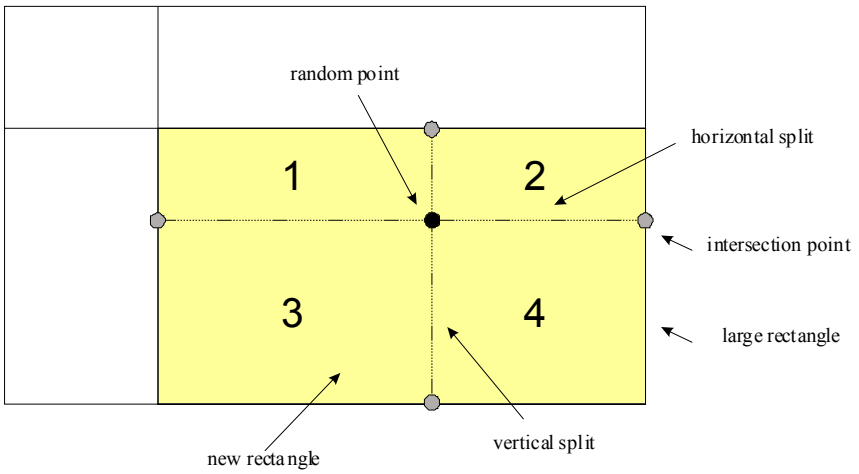


Figure 5.2: Splitting an existing rectangle into 4 new rectangles for the guillotineable case; Algorithm1

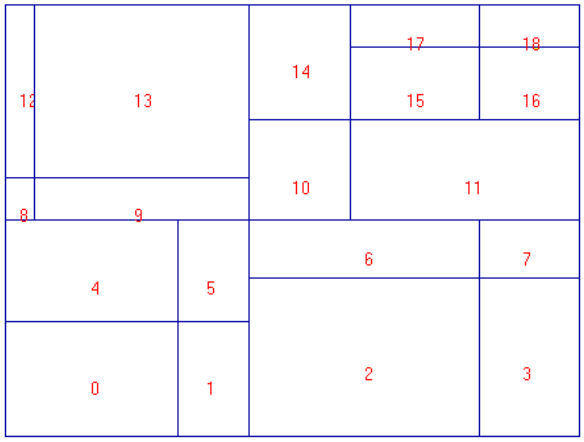


Figure 5.3: Example of guillotineable problem generated by Algorithm1; n=19

Algorithm2

The second algorithm for the generation of rectangular test problems creates only two new rectangles per insertion point. Using Algorithm1 a newly created rectangle always has one of the dimensions with the two neighbouring rectangles in common unless the adjacent rectangles get selected for a further split in a subsequent step. Algorithm2 avoids this by only creating two new rectangles per large rectangle (Figure 5.4 and Figure 5.5). The total number of rectangles in the problem is provided in Equation 5.2. An example of a problem instance created with Algorithm2 is shown in Figure 5.6.

$$n = 1 + i \qquad \text{Equation 5.2}$$

```
user input
calculate number of rectangles to produce (according to Equation 5.2)
repeat
    select randomly one of the existing rectangles (i.e. large rectangle)
    if start
        use object as large rectangle
    select random side of large rectangle
    place random point on this side
    construct line perpendicular this side through point
    determine intersection point with borders of large rectangle
    create 2 new smaller rectangles (according to Figure 5.5)
    check ratio of these 2 rectangles
    if acceptance
        delete large rectangle
        add 2 small rectangles
        increment rectangle counter by 1
until number of rectangles reached
```

Figure 5.4: Algorithm2 for the creation of a guillotineable problem instance in pseudo code

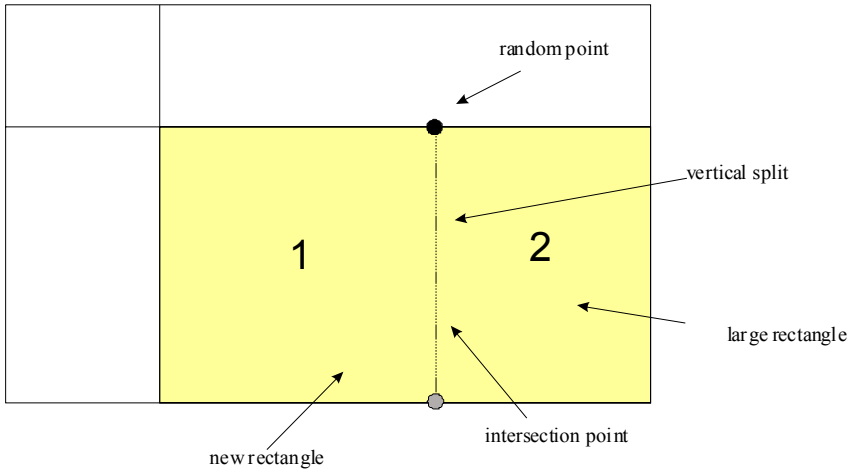


Figure 5.5: Splitting an existing rectangle into 2 new rectangles for the guillotineable case; Algorithm2

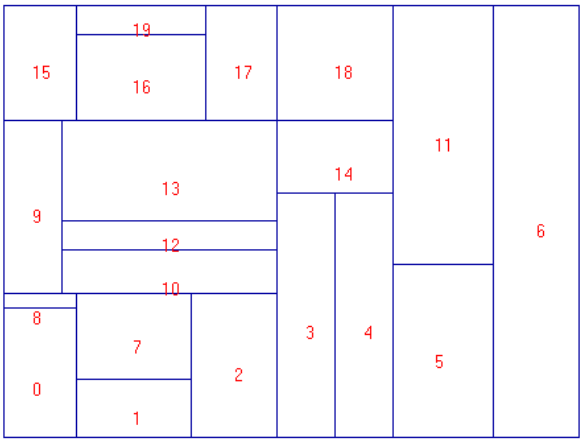


Figure 5.6: Example of guillotineable problem generated by Algorithm2; n=20

b) Generator for non-guillotineable problems

According to the algorithm for guillotineable problems, first an object with a certain size is chosen, which is then divided into smaller rectangles. The design rule for non-guillotineable problems is stated in Figure 5.8. Equation 5.3 describes the total number of rectangles in a problem instance created by this rule. In Figure 5.9 an example of a test problem generated with this problem generator is shown.

$$n = 1 + 4 * i \qquad \text{Equation 5.3}$$

with n = total number of rectangles in problem, i.e. problem size

i = number of times loop has been executed

The user input provides an estimate of the problem size. Using the relation in Equation 5.3 the number of times the loop in Figure 5.8 needs to be executed can be calculated and will be rounded to the larger integer. The total number of rectangles is then re-calculated and may be equal or larger than the user input.

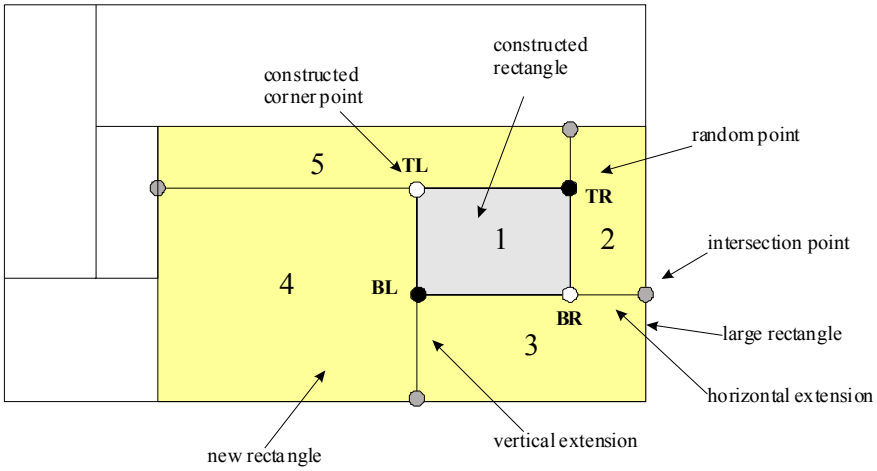


Figure 5.7: Splitting an existing rectangle into 5 new rectangles for the non-guillotineable case

user input
calculate number of rectangles to produce (according to Equation 5.3)
repeat
  select randomly one of the existing rectangles (i.e. *large* rectangle)
  if start
    use object as large rectangle
place 2 random points into large rectangle (Figure 5.7)
  create *small* rectangle using these two points
  check ratio of the *small* rectangle
  if acceptance
    select direction for extension of the BL and TR points: vertical or horizontal
    extend BL and TR corner in selected direction
    determine intersection points with borders of *large* rectangle
    extend TL and BR corner in other direction
    determine intersection points with borders of *large* rectangle
    create 4 new *small* rectangles (according to Figure 5.7)
    check ratio of the *small* rectangle
    if acceptance
      delete *large* rectangle
      add five *small* rectangles
      increment rectangle counter by 4
until number of rectangles reached
where
TR = top right; TL = top left; BL = bottom-left; BR = bottom right

Figure 5.8: Algorithm for the creation of a non-guillotineable problem instance in pseudo code

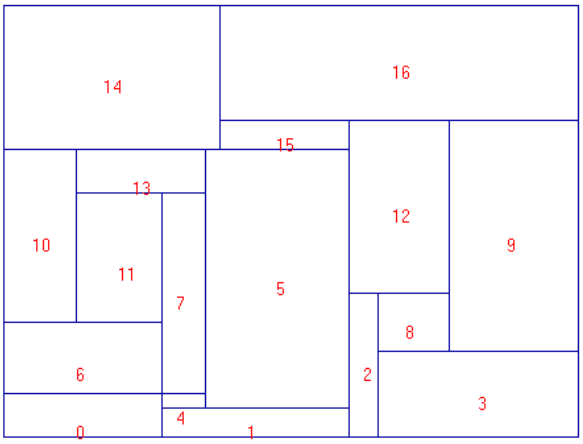


Figure 5.9: Example of non-guillotineable problem generated with the problem generator; n=17

5.3.2.2 Packing Problems without Known Optimum

Not all industrial packing problems have a known optimum corresponding to an object utilisation of 100%. The algorithm applied in this problem generator for strip packing problems without known optimum is based on uniform distribution. The rectangle sides are created so that they are uniformly distributed in the range supplied by the user (Table 5.10) obeying the ratio constraint. In some problems multiple instances of one item can occur. The user can indicate this by supplying the number of items that should be created for each item type.

Table 5.10: Input parameters for packing problems without optimum solution

parameter	**description**
problem size	number of items to the problem consists of
object	width of object
item dimension	minimum and maximum proportion of larger item side in relation to the object width
item ratio	maximum ratio of the two rectangles sides
item types	number of different box types
scaling	yes/ no; to produce a series of problems which have roughly the same total area

For some tests it is useful to have a set of problems which not only consist of the same number of items, but also have roughly the same total area. Therefore, an optional scaling procedure has been implemented in the algorithm and the total area of the rectangles in a set of problems is calculated. The user inputs a larger area to which all packing problems produced in this series should be scaled. The scaling procedure calculates by which factor the dimensions of the rectangles in each problem set need to be scaled and re-calculates the dimensions for each rectangle.

5.3.3 Constructed Test Problems

The following test problems have been used for the evaluation of algorithms developed in this work. For the construction of the rectangular problems the problem generators introduced in sections 5.3.2.1 and 5.3.2.2 have been used. The irregular problems have been designed manually.

5.3.3.1 Test Problems for 2D Rectangular Strip Packing

The following problems have been constructed with the aid of the problem generators described in section 5.3.2. In order to eliminate any possible influence of the object dimensions a square object has been used in all problem instances. Whereas the first series of problems has been constructed such that the known optimum layout is guillotineable (Table 5.11), the second series has optimum layouts which are not guillotineable (Table 5.12). For both categories various problems of different size have been created, each consisting of five instances labelled ‘a’ to ‘e’. The dimensions of the rectangles are produced randomly with a maximum aspect ratio of 7. The dimensions of the items used by the individual problem instances are given in Appendix A.1.2.

Table 5.11: Guillotineable rectangular problems

name	size
T1a, T1b, T1c, T1d, T1e	17
T2a, T2b, T2c, T2d, T2e	25
T3a, T3b, T3c, T3d, T3e	29
T4a, T4b, T4c, T4d, T4e	49
T5a, T5b, T5c, T5d, T5e	73
T6a, T6b, T6c, T6d, T6e	97
T7a, T7b, T7c, T7d, T7e	199

Table 5.12: Non-guillotineable rectangular problems

name	size
N1a, N1b, N1c, N1d, N1e	17
N2a, N2b, N2c, N2d, N2e	25
N3a, N3b, N3c, N3d, N3e	29
N4a, N4b, N4c, N4d, N4e	49
N5a, N5b, N5c, N5d, N5e	73
N6a, N6b, N6c, N6d, N6e	97
N7a, N7b, N7c, N7d, N7e	197

5.3.3.2 Test Problems for 2D Irregular Strip Packing

The problems listed in Table 5.13 have been constructed manually and consist of a number of convex and concave polygons. The various problems differ in size and composition. The problem 'poly1a' contains a set of 15 polygons. The other four problems labelled 'a' consist of multiples of 2 to 5 of the polygon set in 'poly1a', depending on the problem size. All problems labelled 'b' contain a unique set of polygons, i.e. each polygon is only contained once. The problem instances are stated in Appendix B.2.2.

Table 5.13: Constructed irregular test problems; object width: 40

name	size	problem type	shapes	name	size	problem type	shapes
poly1a	15	artificial	polygons				
poly2a	30	artificial	multiple of poly1a	poly2b	30	artificial	polygons
poly3a	45	artificial	multiple of poly1a	poly3b	45	artificial	polygons
poly4a	60	artificial	multiple of poly1a	poly4b	60	artificial	polygons
poly5a	75	artificial	multiple of poly1a	poly5b	75	artificial	polygons

5.3.3.3 Test Problems for 2D Rectangular Bin Packing

Three different sized problem categories have been used to analyse the performance of the hybrid algorithms with respect to 2D bin packing that contain different sized items and objects (Table 5.14). Each problem category contains five test problems generated with the problem generator described in section 5.3.2.2. The test problems have been constructed such that the optimum solution is not known, but the number of objects is large enough to accommodate all items. The item dimensions have been generated randomly with the longer side ranging between 10% and 100% of the object width. After all items have been created, their dimensions have been scaled by a factor (>1) so that the five problems in one category have roughly the same total item area. The individual problem instances are stated in Appendix A.2.

Table 5.14: Constructed 2D bin packing problems

problem name	item number	object number
M1a, M1b, M1c, M1d, M1e	100	16
M2a, M2b, M2c, M2d, M2e	100	18
M3a, M3b, M3c, M3d, M3e	150	20

5.4 Commercial Nesting Packages for Performance Testing

Solving packing tasks with meta-heuristics has so far only been a research topic. In industrial applications layouts are usually generated manually or by the use of problem-specific heuristics. Most researchers who developed meta-heuristic and heuristic algorithms use industrial data to demonstrate the performance of their algorithms. Apart from one study (Corno et al., 1997), no one else evaluated them with respect to commercial nesting packages that were especially designed for industrial needs. A true picture of the capabilities of meta-heuristics and heuristics will only be obtained if they are not only tested on industrial data but are also compared to industrial software. An evaluation needs to include solution quality and

flexibility of packing algorithms as well as their efficiency regarding computation time, which is a major issue for industrial applications. Not every industrial application needs to be solved in real-time, but there can be a demand for speed, especially when last minute changes in the layout or stock availability are likely to occur. A list of commercial software for 2D rectangular and irregular packing tasks is presented in section 4.5. Further details can be obtained from Appendix C.

Two commercial nesting packages have been used for this investigation: Nestlib by Geometric Software Services Co. Ltd. and SigmaNEST by SigmaTEK Corporation. Apart from true-shape nesting of irregular items on single and multiple objects, the major reason for using these two packages was the availability of demonstration versions that allow the application of user data. Both packages are powerful tools that support both the packing task and also the subsequent cutting process. Since this work concentrates on the packing aspect, features with respect to layout generation are described below. The performance comparison reported in this work is not intended as a full evaluation of the commercial products and focuses on the packing aspect rather than the whole range of features implemented in the packages. The nesting software offers many more features than discussed here, in particular for the cutting process such as cutting sequence generation. A number of features which are common to most commercial packages such as high speed of operation and fully automatic true-shape nesting have been already briefly described in section 4.5. Therefore in section 5.4.1 and 5.4.2 only distinctive features of Nestlib and SigmaNEST are highlighted. Although suitable for multiple sheets none of the packages include features to find the optimal set of objects required for a certain nest.

5.4.1 Nestlib Nesting Software

Nestlib version 9.0 by Geometric Software Services Co. Ltd., India can be used as a fully automatic and standalone nesting package via a user interface. In addition, it also provides a nesting library in the C programming language that contains 2D true-shape nesting algorithms. This allows software developers to incorporate it as a nesting module in their code. Nestlib receives the part and sheet data from the external application, fits the parts on the sheet and outputs the nesting result (Figure 5.10).

Nestlib can be used for the development of a wide variety of packing applications. Typical applications include sheet metal cutting (flame, plasma and laser), punching, fitting of patterns on cloth or leather, nesting of packaging material etc. Optional modules are also available for special nesting requirements such as multiple torch nesting, pairing and clustering. Major features of this product regarding the packing process are:

- support for filler parts to reduce wastage
- grouping of parts: facility to nest a collection of inter-related parts as a unit with no constraint on the relative positioning of these parts
- facility to attach priority numbers with each part
- choice of area or perimeter based sequencing of input parts

- processes its own file format for input data, which is simpler than DXF
- facility of localised nesting on a sheet

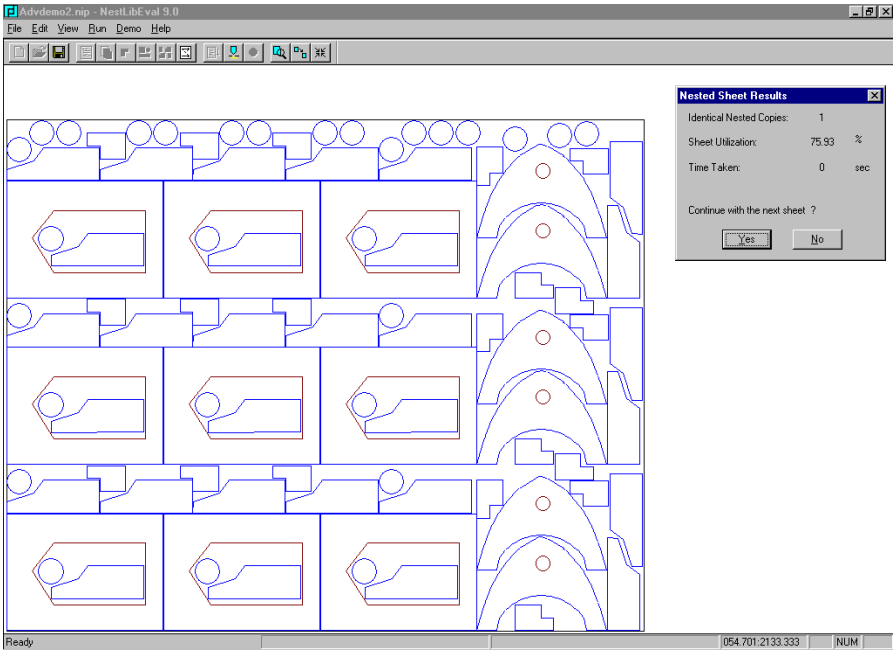


Figure 5.10: Nestlib user interface

5.4.2 SigmaNEST

SigmaNEST version 4.0 by SigmaTEK Corporation, USA is a nesting system, which allows generating layouts fully automatically as well as interactively (Figure 5.11). In addition to that, also manual positioning of shapes is possible. SigmaNEST comprises a complete built-in CAD system and therefore has capabilities to create geometry. This process is supported by a standard shape library containing a variety of geometric objects, which can easily be modified to form the parts required. Alternatively, the geometric parts can be imported through a variety of file formats. A packing task can be nested either automatically or manually. Once the nesting task has been carried out, it can be modified manually. SigmaNEST is optimised for a number of cutting techniques such as laser and plasma cutting and punching. In addition, SigmaNest offers many other facilities such as automated NC programming, part cost estimation, scheduling, job tracking and inventory control. Major features with respect to the packing process are listed below as far as they have not been mentioned in section 4.5.

- facility to nest parts and sheets of different types (e.g. thickness and material) in one nesting task
- processes a large number of file formats (e.g. DXF)
- allows processing of batch files: convenient for data input of a complete or partial nesting task
- choice of area or width based sequencing of input parts
- actual shape remnant tracking

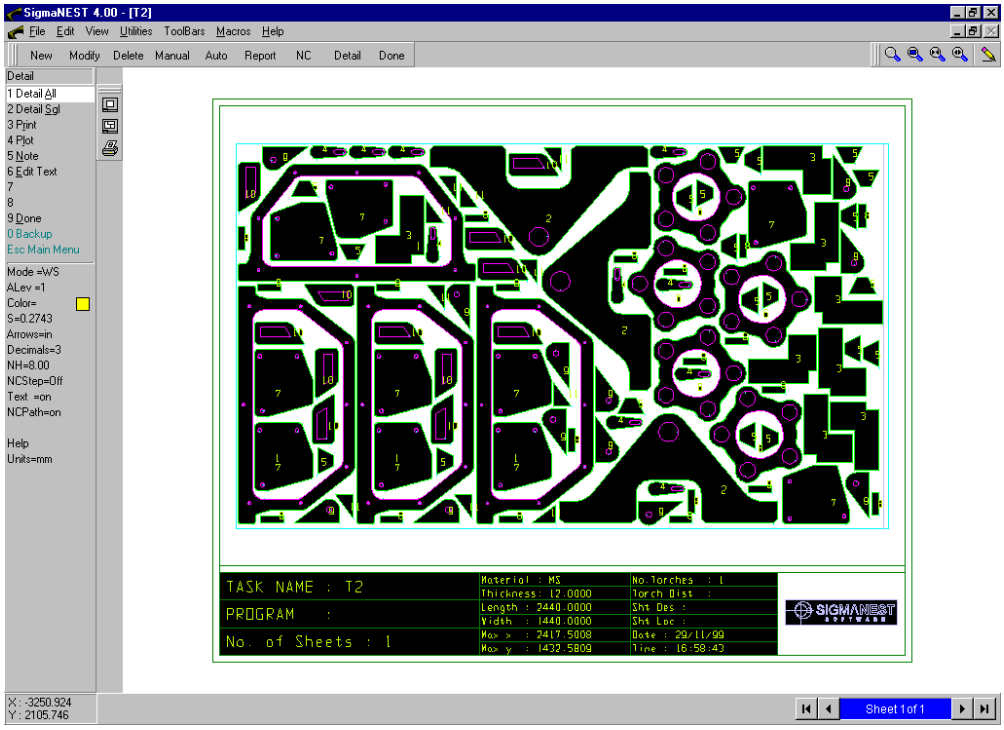


Figure 5.11: SigmaNEST user interface

5.4.3 Performance Testing

Since the module for strip packing is optional, the result of the nesting process in the demonstration version of Nestlib is only given in the form of the layout and the object utilisation of each object. To obtain the height, the layout was stored as an image and processed with digitising software.

Before the layout generation, the items belonging to the same nesting task are ordered either by the area or the perimeter. For the performance evaluation in this work both sequences have been applied and the better result has been used. SigmaNEST states the largest x- and y- co-ordinates of a layout as well as the packing density. Similar to Nestlib there is a choice of two parameters to generate the input sequence sorting the items by area or height. Again, both parameters have been applied to the benchmark problems and the better result has been used for further comparisons.

5.5 Application of Meta-heuristic Algorithms to Rectangular Packing Problems

In this section a family of meta-heuristic as well as heuristic algorithms is developed to solve rectangular packing problems for strip material. Due to promising results of other research in this area (section 4.3) the main focus in this work will be on evolutionary techniques. A two-stage approach has been chosen for

meta-heuristic methods, where a meta-heuristic algorithm is combined with a heuristic packing policy. In total four different heuristic methods have been used in hybrid combinations with three meta-heuristic search algorithms.

The hybrid algorithms will be compared later in terms of solution quality and computation time on a number of different sized packing problems, some of which are benchmark problems from the literature (chapter 6). In order to show the effectiveness of the design of the different algorithms, their performance is compared to local search, i.e. hill-climbing, random search and heuristic packing routines, which are used as decoding algorithms. Their implementation will be described along with the one of the meta-heuristic strategies in section 5.5.3.

5.5.1 The Packing Problem

The problem consists of packing a collection of items onto a rectangular object while minimising the used object space. The packing process has to ensure that there is no overlap between the items. The specific problem that is addressed in this section has the following characteristics (Figure 5.12):

- a set of items, which may contain identical items
- the complete set of items needs to be packed
- one single object of fixed width and unlimited height
- all pieces are of rectangular shape
- items can be rotated by 90°

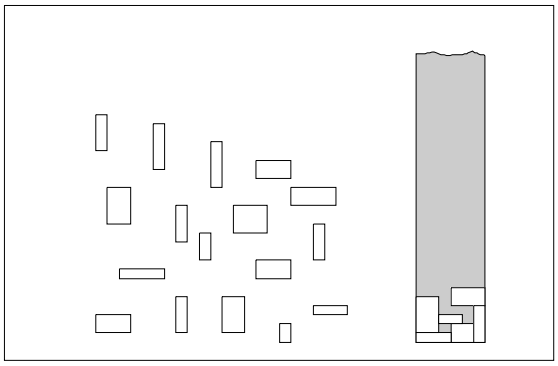


Figure 5.12: 2D rectangular strip packing problem

5.5.2 Meta-heuristic Hybrid Algorithms

The number of all possible positions of the rectangles on the object is very high, even if the overlap constraint is considered and the items are only placed in non-overlapping configurations. Heuristic algorithms can be used to generate a valid layout placing the items step by step onto the object according to a specific placement rule. The sequential placement of the items may result in a good, however, not necessarily optimal solution depending on the respective sequence. Each of these sequences represents a solution to the packing problem. The number of combinations with respect to the input sequence is too large to be explored exhaustively in a reasonable amount of time. Meta-heuristic search algorithms on the other hand which make use of built-in strategies to search the solution space more efficiently can be used to find suitable input sequences. In order to overcome the main disadvantage of local search algorithms like hill-climbing, whose weakness lies in the inability to escape from local minima, more sophisticated heuristic search strategies can be applied allowing the temporary acceptance of states of lower quality. Meta-heuristic algorithms can be considered to some extent as local search strategies, but they also include a means to escape from local minima.

In order to combine the strengths of heuristic and meta-heuristic methods a family of hybrid approaches has been developed. In the following hybrid algorithms the task of the meta-heuristic is to search for a good ordering of the items. A placement routine is then needed to interpret the sequence transforming it into a layout. Once the layout has been generated the quality of the input sequence can be calculated according to the evaluation function.

Hybrid algorithms can be powerful techniques that combine the strengths of meta-heuristic and heuristic algorithms. Since a meta-heuristic evaluates a large number of solutions in the search space before it will converge on a (sub-) optimal solution, the heuristic decoding algorithm has a strong influence on computation time. The efficacy of the hybrid is therefore strongly dependent on how domain-specific knowledge is exploited and its ability to overcome combinatorial explosion.

As mentioned above the main focus among the meta-heuristic techniques has been put on evolutionary techniques using genetic algorithms and naïve evolution. A number of experiments have been carried out in order to establish the influence of the problem-specific and generic design variables of the genetic algorithms investigating the influence of the type and rate of cross-over and mutation as well as the selection type and the population size. Various methods to obtain a seeding for the initial population have been examined and used to compare the final outcomes of the evolutionary search processes. In addition, simulated annealing has been applied. The four heuristic routines used to hybridise the meta-heuristic algorithms are described in section 5.5.4.

5.5.3 Implementation of the Search Algorithms

A number of problem-specific and generic decisions have to be made for the implementation of the meta-heuristic search algorithms. The problem-specific design variables concern the objective function, initial solution, representation scheme as well as the operators applied to manipulate the search space. The generic decisions are concerned with the probabilities at which the search space manipulators such as cross-over and mutation are applied, the cooling schedule in the case of simulated annealing and the population and generation sizes for the genetic algorithms as well as the stopping criteria for the search.

5.5.3.1 Shape and Problem Representation

In terms of implementation, rectangular packing problems manage with much simpler data structures and algorithms compared to their irregular counterpart. Rectangular items and objects can be sufficiently described using the following parameters:

- item number: unique number in permutation
- reference point:
  i.e. the offset in x and y direction
  from the reference point of the object
- height *h*
- width *w*
- orientation:
  in anticlockwise direction from positive x-axis

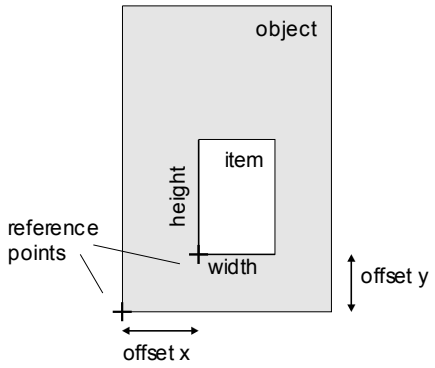


Figure 5.13: 2D shape representation in rectangular problems

For an efficient computation of the packing algorithms in terms of time and memory usage it is desirable to use simple, but fast data types. Due to the simplicity of the rectangle parameters described in Figure 5.13 the data structures for the description of the rectangular packing problem can be entirely based on

the fast *integer* type. Sometimes the data describing an industrial problem is in the form of real or rational numbers. In this case the application of a scaling factor used to multiply the dimensions of the items and rectangles allows transferring this data set into an integer data set. In rectangular problems there is no need for complex geometric data structures, since a layout can be easily constructed by series of co-ordinate comparisons and distance calculations. In particular, the use of integers allows an efficient implementation of the placement algorithms used in the decoding stage.

In the 2D strip packing problem the task is to place the complete set of items on the object space. The item set consists of all the rectangles, which have to be packed in a certain nesting task. The rectangles in the item set can be different or identical in size. Since they all have to be packed onto the object, they are treated as unique by the nesting task and are represented by a unique number. Hence the complete set of items forms a permutation with each item number occurring only once. In order to interpret the permutation in respect to the 2D layout a decoding algorithm is needed which describes a procedure for the allocation of each rectangle on the object. In the hybrid combination the task of the meta-heuristic is to search for a good sequence of all the items.

5.5.3.2 Search Space

On the one hand the search space needs to be sufficiently large to allow the meta-heuristic search process to explore a large range of layout configurations before it starts converging. On the other hand a very large search space may contain a high number of layout configurations which do not contribute to the search process due to a low quality. Therefore it can be useful to limit the search space. The search space for the hybrid implementations described in the following section has been limited in two ways.

The first limitation concerns the validity of solutions. Since overlapping layout configurations are not valid solutions to the packing problem, they can either be excluded completely from the search space, ignored or transformed into valid solutions with the aid of operators applied to the layout. In this investigation the first option has been chosen. The search space is restricted to valid solutions only, i.e. configurations which describe non overlapping layouts. In this work the packing problem is tackled with a two-stage approach, where the meta-heuristic methods search the solution space for input sequences, which are translated into layouts during the second stage. The validity of all solutions in the search space is achieved through the decoding stage. The placement routines are designed so that they only result in valid layouts.

In case a search space with invalid configurations is used a penalty term can be introduced into the evaluation function which assigns a low quality to these solutions. This will not prevent the occurrence, but should guide the search process. Excluding invalid solutions on the other hand will save the effort of determining a suitable penalty term and also save the computation time spent on invalid solutions. Secondly, when only valid solutions occur, there is no need for the time consuming implementation of correction operators that transform invalid layout configurations into valid ones. These correction

algorithms are not only complex due to the operation on the phenotype rather than the genotype, but also time consuming and increase overall computation time.

The second limitation of the search space refers to the orientation of the rectangles. On the one hand a high number of orientations allows the search heuristic to explore a large range of layout configurations. On the other hand many of these configurations might be of low quality. This is especially the case in rectangular packing. Since dense layouts are achieved when the edges are aligned, solutions that include rectangles in too many orientations can account for a large percentage of waste. Therefore it is useful to restrict the search space to a certain number of rotations. Figure 5.14 demonstrates this by packing the same set of rectangles allowing a rotation interval of 90° and 22.5° respectively. Although the search space in the second case is larger, the layout achieved is not better due to the characteristics of non-orthogonal packing. If, however, rotation is excluded completely from the search space, too many good layout configurations may be also excluded, which does not support the search process either.

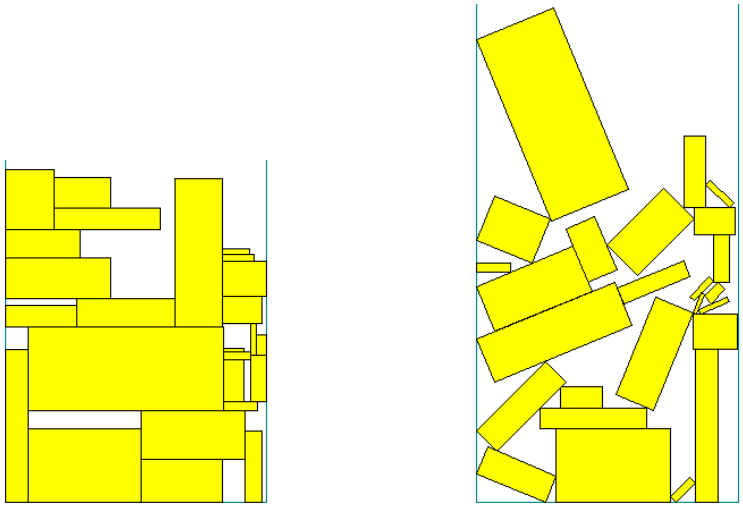


Figure 5.14: Packing a set of rectangles allowing a rotation interval of 90° (left) and 22.5° (right)

In this implementation the rectangles can be rotated by 90° and therefore are present in two orientations in the search space. In order to allow the meta-heuristic and local search algorithms to manipulate the orientation of the items, an operator is used which flips the orientation of each rectangle in the sequence with a certain probability.

5.5.3.3 Evaluation of the Quality of the Layout

The permutation represents the order, in which the finite set of items is packed. A decoding algorithm is then used to evaluate the fitness of a solution by constructing the layout. The quality of a packing pattern is first of all determined by its height. This value is not sufficient to describe the quality of a solution, since a number of sequences can result in layouts of the same height. Although both patterns in Figure 5.15 have the same height, the right one is better, since there are fewer gaps between the rectangles. In

order to examine how tightly the rectangles are packed, the remaining area of the object is calculated. All empty areas not entirely enclosed by the layout belong to the remaining area (Figure 5.15) and can either be completely or partially re-used in a future nesting process.

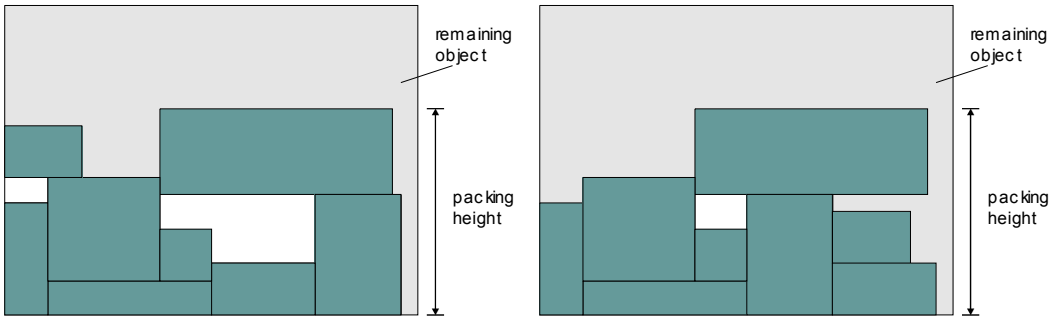


Figure 5.15: Two layouts with the same packing height, but different remaining object area

For the assessment of the quality of the layout both quantities have been used and are included in the fitness function in form of a weighted sum (Equation 5.4). In order to facilitate comparisons between packing problems both terms of the sum are scaled to the object dimensions.

$$fitness = a \cdot \left(1 - \frac{h_{layout}}{h_{object}}\right) + b \cdot \frac{A_{remaining}}{A_{object}} \qquad \text{Equation 5.4}$$

In the special case where the optimum solution and therefore the optimum height and object area are known the above evaluation function has been modified in order to include this additional problem-specific knowledge (Equation 5.5).

$$fitness = a \cdot \left(1 - \frac{h_{layout} - h_{optimum}}{h_{optimum}}\right) + b \cdot \frac{A_{remaining}}{A_{object}} \qquad \text{Equation 5.5}$$

The weights of the two terms have been established in a series of trials using the values of 0.7 for ‘a’ and 0.3 for ‘b’. This evaluation function has been used for all the experiments analysed in chapter 6.

5.5.3.4 Genetic Algorithm

Since order-based encoding is used for this problem, care has to be taken that valid chromosomes are generated during the cross-over and mutation operations. Partially matched cross-over (PMX) (Goldberg, 1989) and order-based and position-based mutation (Syswerda, 1991) are suitable for this type of encoding. Only one of the mutation operators has been applied in a recombination process. Proportional selection and generational replacement have been applied. The orientation of the rectangles is considered in the genetic algorithm in the form of an additional mutation operator. In the case of orthogonal packing only two orientations for an item are possible. The rotation operator is applied to every item in the chromosome and changes the orientation with a certain probability. The initial population has been

generated randomly. The specific values used for the generic design variables are stated in Table 5.15. Table 5.16 summarises the problem-specific decisions.

Table 5.15: Implementation of generic design variables for genetic algorithms and naïve evolution

variable	genetic algorithms	naïve evolution
cross-over rate	60%	-
mutation rate	3%	3%
selection type	proportional; elitism	proportional; elitism
replacement	generational	generational
population size	100	100
termination criteria	1000 generations	1000 generations

Further techniques that have been implemented include elitism and seeding (Goldberg, 1989). Since heuristic placements with pre-ordered input sequences can yield packing patterns, whose quality is above average (Coffman et al., 1984), the initial population has been seeded with the permutation that describes the rectangles sorted according to a certain parameter. A series of simulations has been carried out in order to establish the influence of this technique. Various methods to obtain the seeded solution have been investigated. The influence of the cross-over and mutation type as well as the selection strategy has also been investigated along with the affect of the probability rates for cross-over and mutation (section 6.5 and Table 5.28).

5.5.3.5 Naïve Evolution Algorithm

The basic idea behind naïve evolution is the same as for the genetic algorithm. However, no cross-over operator is applied to manipulate the search space. Only the mutation operator is used for the generation of the next population. A naïve evolution algorithm can be used to test the efficiency of the cross-over operator in a genetic algorithm. Falkenauer (1998) applied this technique on 1D bin packing problems.

Table 5.16: Implementation of the problem-specific design variables for genetic algorithms and naïve evolution for 2D rectangular strip packing

design variable	genetic algorithms	naïve evolution
representation	permutation (item sequence)	permutation (item sequence)
evaluation function	weighted sum of packing height and remaining area (Equation 5.4 and Equation 5.5)	weighted sum of packing height and remaining area (Equation 5.4 and Equation 5.5)
initial population	random	random
crossover operator	PMX (1-point PMX)	-
mutation operator 1	order-based	order-based
mutation operator 2	change of orientation by 90°	change of orientation by 90°

The problem-specific and generic decisions for the naïve evolution algorithm are the same as for genetic algorithms. The values concerning the generic design variables are listed in Table 5.15. Table 5.16

summarises the implementation of the evolution-based algorithms, i. e. genetic algorithms and naïve evolution, regarding the problem-specific decisions.

5.5.3.6 Simulated Annealing Algorithm

The packing problem is represented by a permutation that is interpreted as the order in which the rectangles are packed. The neighbourhood structure of the current solution is defined by the set of solutions that can be reached through one of the following three manipulation operations. The first two randomly change the sequence of the elements in the permutation and operate according to order-based and position-based mutation in genetic algorithms. The third operator considers only the orientation and flips the rotation variable of one randomly selected item. For the translation to the next solution only one of the operators is applied. The initial solution is generated randomly.

The generic choices for the implementation of a simulated annealing algorithm are summarised in the annealing or cooling schedule (Table 5.17). The schedule presented in Press et al. (1995) is used in this study. The temperature function is geometric and decreased by 10%. The initial value for the temperature has been determined using the random walk described by in Press et al. (1995). The temperature is held constant for 100N total moves or 10N successful moves at each step with N representing the number of items in the problem.

5.5.3.7 Hill-Climbing

The neighbourhood structure as well as the manipulators for the search space used in the local search procedure are the same as in simulated annealing. A neighbouring solution can be reached through two swap operators that work in the same way as order-based and position-based mutation in genetic algorithms, or a rotation operator, which flips the rotation of an item by 90°. Only one of the three operators is used at a time to reach an adjacent state. Hill-climbing is a local search technique where a neighbourhood move is made, when the neighbouring solution is better. The technique implemented here is called steepest-ascent hill-climbing. The idea is to explore a larger neighbourhood, before a move is made. Out of the neighbouring solutions evaluated the best one is only accepted if it is better than the current solution. In this implementation 100*N neighbouring states are explored before a decision on the move is made. Since uphill moves are not allowed the search process runs until it gets trapped in a solution that is locally optimum. The search process is stopped if the algorithm is unable to find a better solution after N iterations. N is the number of rectangles in the problem. The starting point for the search process is generated randomly.

Table 5.17: Implementation of generic design variables for simulated annealing and hill climbing for 2D rectangular strip packing

	simulated annealing	**hill-climbing**

initial temperature	random walk method	
size of neighbourhood		100 N
length of Markov chain	100 N moves maximum 10 N accepted moves	
decrement rule	$T_{k+1} = 0.9\ T_k$	
termination criterion	k=100; k= number of decrements	N unsuccessful trials

5.5.3.8 Stochastic Optimisation Algorithm

In addition to the meta-heuristics described above, a stochastic optimisation algorithm has also been implemented. It combines features of genetic algorithms and hill climbing and has been used by Pargas and Jain (1993) for a 2D irregular strip packing problem. Similar to genetic algorithms, stochastic optimisation operates on a population of solutions manipulating them with the aid of a mutation operator. The operation of the algorithm is described in section 4.4.4. Since the packing problems in this work are represented in form of a permutation the same neighbourhood operators as in simulated annealing and hill-climbing are used. Table 5.18 summarises the implementation of the problem-specific and generic variables for the stochastic optimisation algorithm.

Table 5.18: Implementation of the problem-specific design variables for stochastic optimisation for 2D rectangular strip packing

problem specific decisions		**generic decisions**	
representation	permutation (item sequence)	size of neighbourhood	50 N
evaluation function	weighted sum of packing height and remaining area (Equation 5.4 and Equation 5.5)	size of population	100
initial population	random	selection type	ranking
mutation operator 1 mutation operator 2	order-based position-based	termination criterion	200,000 iterations or N unsuccessful moves
mutation operator 3	change of orientation by 90°		

5.5.3.9 Random Search

In order to demonstrate the effectiveness of the design of the various meta-heuristic and local search techniques, a comparison with a random search process is made. The random search technique applied here uses the same search space as the other methods, and only results in valid layouts. Instead of navigating through the search space in a controlled way, random search generates the permutations randomly. The heuristic decoders described in section 5.5.4 are then used to evaluate the random input sequences. The search process is executed for the same number of function evaluations as the genetic algorithm, i.e. 100,000.

The random search technique used here is not entirely random with respect to the allocation of the items on the object. As it uses the problem-specific knowledge implemented in the heuristic placement rules, only certain layout positions are possible. It is therefore not equivalent to random positioning of the rectangles. The implementation as a hybrid algorithm, however, allows a comparison with the meta-heuristic hybrid algorithms on the same basis. The objective of the meta-heuristics is to find a good input sequence by navigating through the search space with the aid of sophisticated operators. Random search floats through the space and hence it can assess the effectiveness of the operators and also of the meta-heuristic search technique as a whole.

5.5.4 Heuristic Placement Algorithms

In this work four heuristic placement routines have been used to hybridise the above mentioned meta-heuristic search techniques. These packing routines belong to the class of bottom-left heuristics (Baker et al., 1980). In these techniques stability will not allow an item in the bottom-left position to be moved any further downwards or to the left.

Three of the techniques are described in the literature. Two of them have been used in hybrid packing algorithms. The first technique is the BL-heuristic used by Jakobs (1996) in a hybrid genetic algorithm. The second one was combined with a genetic algorithm by Liu and Teng (1999). It is referred to as BLLT-routine throughout this work. The two heuristics and their meta-heuristic hybrids have been used to evaluate the performance of the newly created algorithms. Chazelle (1983) implemented a different type of bottom-left placement technique, which is referred to here as BLF-algorithm. Unlike the other two allocation routines, this one is not based on a sliding technique. It is more sophisticated and hence computationally more expensive. A fourth placement routine has been developed in conjunction with the representation scheme used by the meta-heuristics and has been named BLD-rule. The heuristic placement techniques are described in detail in the following sections.

5.5.4.1 BL-Algorithm

The BL-algorithm described below was used by Jakobs (1996) in a hybrid genetic algorithm. Starting from the top-right corner each item slides as far as possible to the bottom and then as far as possible to the left of the object. The successive vertical and horizontal movement operations are repeated until the item locks in a stable position as described in Figure 5.17. A valid position is found when the rectangle collides with the partial layout at its lower and left sides. Figure 5.16 shows the placement of a sequence of rectangles described by the permutation (2, 6, 4, 7, 3, 0, 1, 5).

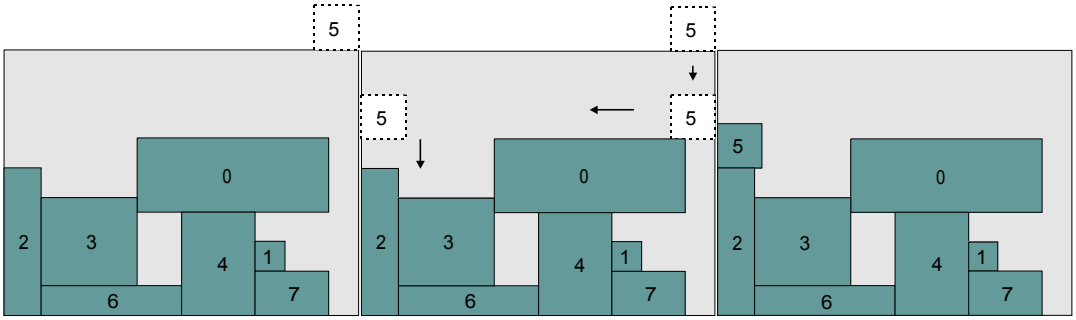


Figure 5.16: Placement of a rectangle into a partial layout using the BL-algorithm

```
repeat
    get next rectangle
place at the TR corner of object
    repeat
        calculate the vertical distance to the partial layout (to the bottom)
        move item by vertical distance
        calculate the horizontal distance to the partial layout (to the left)
        move item by horizontal distance
    until vertical AND horizontal distance = 0
until all items placed
```

Figure 5.17: BL-algorithm (pseudo code)

The major disadvantage of this routine consists of the creation of empty areas in the layout, when larger items block the movement of successive ones. On the other hand its time complexity is only $O(N^2)$ (Jakobs, 1996), where N is the number of items to be packed. The result of this low complexity is a favourable hybrid combination with a meta-heuristic, since the decoding routine has to be executed every time the quality of a solution is evaluated and hence contributes to a great extent to the run time of the hybrid algorithm.

5.5.4.2 BLLT-Algorithm

Liu and Teng (1999) also used a hybrid genetic algorithm to solve a non-guillotineable packing problem. For the decoding stage a bottom-left algorithm was developed, which is a based on a sliding technique as well. The authors claim that this combination outperforms Jakobs' algorithm and demonstrate this with the aid of the two test problems used by Jakobs. Since two test cases are not sufficient for a general performance evaluation, a more systematic investigation is needed using a greater range of benchmark problems. Therefore the algorithm by Liu and Teng has been included in this work and has been implemented as decoder in the meta-heuristic hybrid algorithms.

In order to avoid confusion with other bottom-left techniques the algorithm by Liu and Teng is referred to BLLT in this work, including the authors' initials in the label. Like the BL-algorithm it starts by placing the rectangle at the top right corner of the object. It is then moved as far as possible to the bottom. Instead of moving it the complete distance to the left in the next step until it collides as in the BL-technique, the BLLT-rule moves the rectangle along the partial layout. It is moved to the left as long as it can slide along upper sides of one or more rectangles below. As soon as it reaches the corner, it is moved vertically to the bottom again. In case a collision occurs with other items in the layout before that, the movement operation is stopped at this position. After a number of movement steps the item will be locked in a stable position. A stable position is reached when the item cannot be moved any further to the left or downwards without collision. In Figure 5.18 the allocation of the same sequence of rectangles that has been used in the example in section 5.5.4.1, is shown. Figure 5.19 summarises the BLLT-rule in the form of pseudo code.

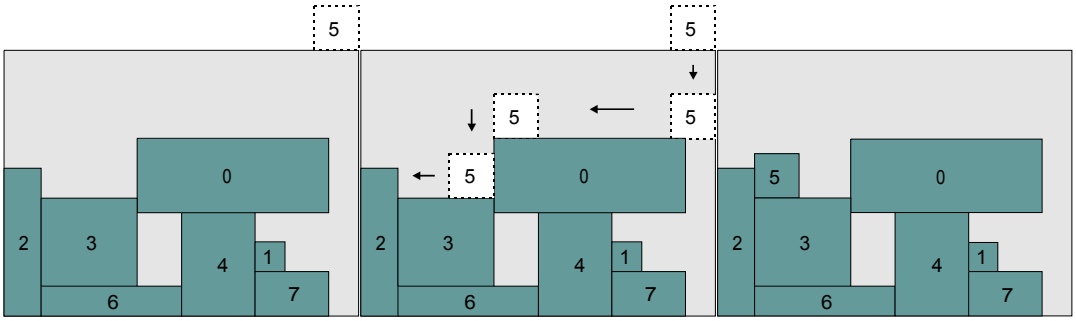


Figure 5.18: Placement of a rectangle into a partial layout using the BLLT-algorithm

```
repeat
    get next rectangle
    place at the TR corner of object
    repeat
        calculate the vertical distance to the partial layout (to the bottom)
        move item by vertical distance
        calculate the horizontal distance to the partial layout (to the left)
        calculate the horizontal length of the sliding path along the rectangle(s) below
        move item by the smaller of the two horizontal distances
    until vertical AND horizontal distance = 0
until all items placed
```

Figure 5.19: BLLT-algorithm (pseudo code)

5.5.4.3 BLF-Algorithm

Since the BL- and the BLLT-routines described above tend to generate layouts with relatively large empty areas, a more sophisticated bottom-left heuristic has been considered for hybridisation with meta-

heuristics. The strategy here consists of placing a rectangle into the lowest available position of the object and left-justifying it. Figure 5.20 demonstrates the placement policy using the same permutation example as above.

Since the generation of the layout is based on the allocation of the lowest sufficiently large region in the partial layout rather than on a series of bottom-left moves, it is capable of filling existing gaps in packing pattern. In order to distinguish it from the bottom-left algorithms described in sections 5.5.4.1 and 5.5.4.2 it is referred to as the Bottom-Left-Fill (BLF) heuristic. Compared to the BL- and BLLT-routines this method results in denser packing patterns. The major disadvantage, however, lies in its time complexity, which is $O(N^3)$ (Chazelle, 1983).

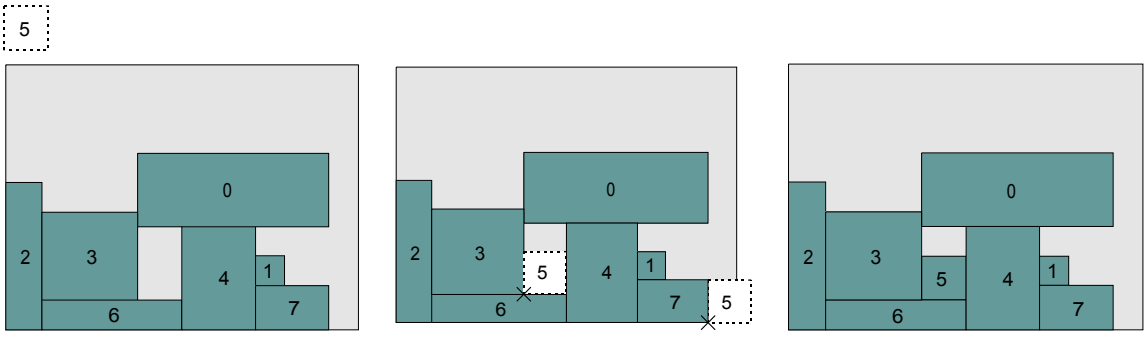


Figure 5.20: Placement of a rectangle into a partial layout using the BLF-algorithm

A time efficient implementation of the BLF-routine is extremely important, especially when used in combination with a meta-heuristic that operates on basis of a large number of function evaluations. Figure 5.27 outlines the implementation of this algorithm used in this work. In order to keep computation time low, it is important that the placement algorithm not only keeps track of the free positions in the layout, but also of the size of available area at the respective positions.

5.5.4.4 BLD-Algorithm

The BLD-algorithm has been developed specifically for use in hybrid combinations with meta-heuristics. The data structure chosen to encode the packing problem is a permutation decoding the sequence in which the rectangles are to be allocated on the object. The problem-specific operators of genetic algorithms and simulated annealing, which work on this data structure, are based on the idea of vicinity, attempting to preserve parts of the phenotype through cross-over or to perform changes in the neighbourhood through mutation and state changes respectively. In order to support the search process it is important that the decoding algorithm translates the genotype into the phenotype in a meaningful manner. Regarding the way in which a sequence of rectangles is transferred into a layout by the above placement rules, it can be seen that adjacency in the permutation not necessarily corresponds to adjacency in the layout. Chapter 6 investigates to which extent this effects the search process of the meta-heuristics.

An attempt has been made to develop a placement algorithm that also preserves the bottom-left stability in the layout, but allows allocating an item in vicinity to the preceding item in the sequence. Therefore the top-right corner of the previous item is chosen as starting position (Figure 5.22). A bottom-left stable position in the layout is found by iterative downwards and leftwards moves. In some configurations this procedure can lead to invalid layouts, when the final position intersects with the partial layout. If this is the case the rectangle is re-allocated to the top-right corner of the object and iterated from there according to the BL-routine (Figure 5.21).

```
repeat
    get next rectangle
    place at the TR corner of previously placed item
    repeat
        calculate the vertical distance to the partial layout (to the bottom)
        move item by vertical distance
        calculate the horizontal distance to the partial layout (to the left)
        move item by horizontal distance
    until vertical AND horizontal distance = 0
    intersection test
    if position is not valid
        start BL-routine
until all items placed
```

Figure 5.21: BLD-algorithm (pseudo code)

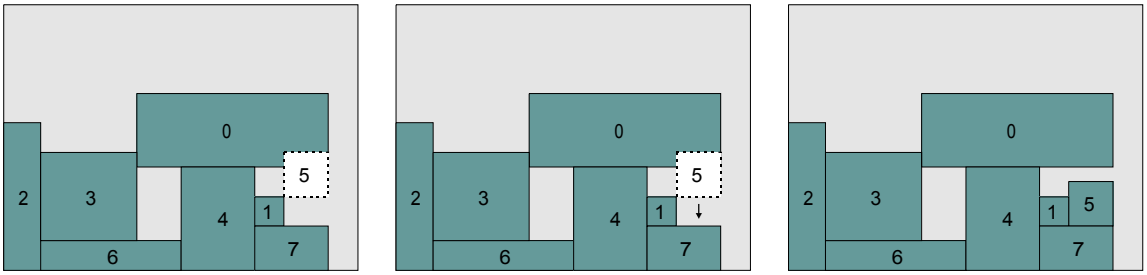


Figure 5.22: Placement of a rectangle into a partial layout using the BLD-algorithm

5.5.4.5 Implementation

The meta-heuristic search process is based on a large number of fitness function evaluations. In this particular hybrid approach the layout has to be generated according to the corresponding permutation before the fitness of a solution can be evaluated. Decoding algorithms need to be implemented as efficiently as possible. An increase in the decoder efficiency will hence result in a major reduction of the overall computation time of the meta-heuristic algorithm.

In terms of the implementation, rectangular packing problems manage with much simpler data structures and algorithms compared to their irregular counterparts. Rectangular items and objects are sufficiently described by reference point, height, width and orientation (section 5.5.3.1). The simplicity of these parameters allows the implementation of data structures and algorithms based entirely on the fast data type *integer*. In the rectangular case there is also no need for geometric data structures, since the layout can be easily constructed by a number of co-ordinate comparisons and distance calculations. The use of integers allows an efficient implementation of the placement algorithms introduced above.

Special care, however, has to be taken to keep the number of distance calculations to the rectangles in the partial layout to a minimum. A series of tests regarding the location in the layout is useful to reduce the set of rectangles involved in the calculation operations. Timing is usually not so critical in the placement algorithms based on the sliding techniques, i.e. BL-, BLLT- and BLD-technique. After a number of movements the new rectangle will be locked in a stable position in the partial layout. The BLF-algorithm, however, gives more reason for concern regarding computation time. Since an attempt is made to first fill gaps in the existing layout starting at the bottom-left corner of the object, a large number of intersection tests have to be carried out before a suitable position in the layout is found. Possible insertion points are repeatedly accessed by every new item to be placed. In order to keep computation times low, it is important that the placement algorithm keeps track not only of the free positions in the layout, but also of the size of available area at the respective positions. The implementation of the BLF-algorithm in this work considers these two aspects and is described in the following (Figure 5.27).

Apart from the list of items that have to be placed, the algorithm also maintains a list of possible insertion positions sorted in bottom-left order. After a rectangle has been placed its top-left (TL) and bottom-right (BR) corner are stored in this list. With every position a height and a width value are stored, which are initialised as the remaining distances to the top and to the right object borders respectively (Figure 5.23). The idea is to collect information about the size of the available gaps in the layout during the run of the algorithm, which accesses these insertion points sequentially. If the dimensions assigned are smaller than the rectangle dimensions the insertion point is rejected immediately. If they are sufficiently large an intersection test routine is started, which tests the rectangles in the neighbourhood of this possible insertion point for intersection (Figure 5.24). As soon as an intersection is detected this routine is stopped and the gap size assigned is updated by the latest distances obtained (Figure 5.25).

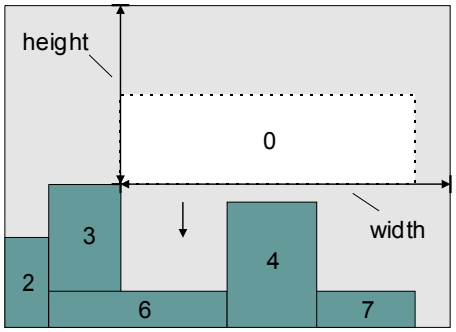


Figure 5.23: Insertion of two points with dimensions

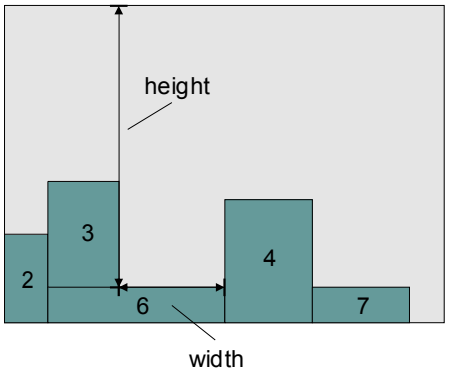


Figure 5.24: Attempt to place rectangle 3

In the case of an intersection the next possible insertion point from the list will be tried. As soon as the dimensions assigned are large enough and the outcome of the intersection test is negative, the position is accepted. The rectangle is then iterated into a bottom-left stable position by successive left and downward movements (Figure 5.26).

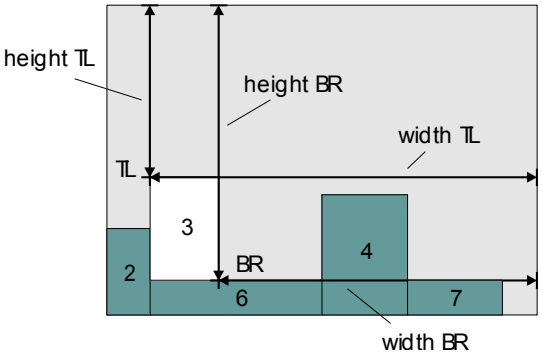


Figure 5.25: Update dimensions of insertion point

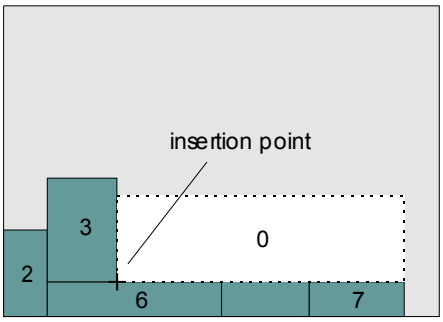


Figure 5.26: Place rectangle 0 at point with suitable dimensions

The general idea is to update the information on the empty area at a possible insertion point, every time it is accessed in the attempt to place a rectangle. The dimensions of the area are compared with the rectangle. If the area is too small to accommodate the item, this position is rejected immediately. Otherwise an intersection test with the partial layout around the insertion point is started. The information about the available empty area is stored with the point in the list. The intersection test is necessary, because the layout in the range of the insertion point might have changed due to placements at other positions.

This technique only stores the gap size information when it is obtained and still requires an intersection test at certain stages. However, it prevents redundant testing. If no layout information were stored at all, all the rectangles in the partial layout would have to be tested for intersection for every new insertion point tried. Alternatively, the gap size of every position could be updated after every insertion of a

rectangle. This involves many redundant calculations as well, since a number of larger rectangles will be rejected on the basis of the current information about the gap size stored in the list. Therefore it is not necessary to maintain information about the accurate area of the gaps at every stage of the algorithm. It is sufficient to store layout-specific information when it can be obtained from necessary calculations.

```
repeat
    get next rectangle
    repeat
        get next position from list
        check if area assigned to position is large enough
        assigned area is large enough
            intersection test with all rectangles that could intersect at that position
                                stopped when first intersection is detected
            intersection true
                update size of area at this point with values obtained from the first intersection incident
            intersection false
                insert rectangle at this position
                iterate rectangle into a bottom-left stable position
                get TL and BR point of this rectangle
                initialise assigned area with width = distance to right border; height =distance to top border
                update position list by inserting TL and BR point of this rectangle
                delete unnecessary positions from list
                sort list of positions in bottom left order
    until all positions tried AND intersection is true
until all rectangles placed

TL = top-left    BR = bottom-right
```

Figure 5.27: BLF-algorithm; pseudo code for a possible time efficient implementation

5.6 Application of Meta-heuristic Algorithms to Irregular Packing Problems

The ideas and techniques that have been developed for the rectangular strip packing problems are extended in this section to tackle irregular strip packing problems. Meta-heuristic as well as heuristic algorithms are designed and implemented for a number of irregular packing tasks. These packing problems originate from different industries in which different constraints need to be obeyed such as the orientation of the items.

The work done to-date in the area of irregular packing and meta-heuristics concludes that meta-heuristics and in particular genetic algorithms are a powerful tool to approach this problem (section 4.3.2). Therefore this work concentrates on the application of evolutionary techniques. Similar to the rectangular packing methods presented in section 5.5 a series of hybrid algorithms has been developed consisting of

meta-heuristic search and heuristic placement rules. A number of different heuristic packing policies have been tried in combination with meta-heuristic search.

Most published work has been regarded on its own and not put in context with other solution approaches. This is due to the lack of benchmark problems in this area, as it is very time consuming to re-implement published nesting approaches for comparison purposes. Benchmark problems are vital for the evaluation of the performance of new algorithms. Therefore an effort has been made here to compare the hybrid approaches in terms of solution quality and computation time on a number of different nesting tasks, most of which originate from the literature. In order to show the effectiveness of the meta-heuristic algorithms, their performance is compared to hill-climbing, the downhill simplex method, random search and heuristic packing routines, which are used as decoding algorithms. The implementation of all hybrid techniques is described in section 5.6.3.

5.6.1 The Packing Problem

Irregular packing problems are very frequent and occur in a various industries such as the metal, textile, leather, wood and shipbuilding industries. In industrial applications, additional constraints concerning the orientation of the parts versus the object need to be considered. In some cases the rotations of the items may be restricted to one or two orientations due to the material properties as in the textile industry. In other industries, which use materials with more homogenous properties it might be possible to rotate the parts freely, e.g. in certain applications in the metal industry.

The irregular problem discussed in this section is illustrated in Figure 5.28. The nesting task consists of allocating a collection of items onto a rectangular object while minimising the used object space. The packing process has to ensure that there is no overlap between the items. The problem characteristics are summarised below:

- a set of items, which may contain identical items
- the complete set of items needs to be packed
- one single object of fixed width and unlimited height
- the object has rectangular properties
- all pieces are of polygonal shape
- items can be rotated by a certain pre-specified angle depending on the

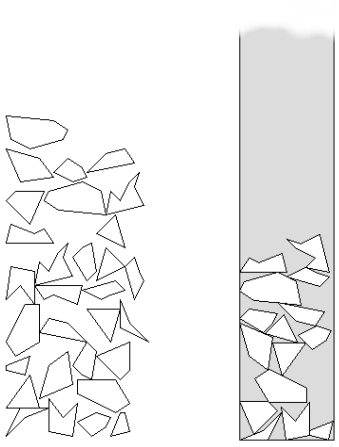


Figure 5.28: 2D irregular strip packing problem

5.6.2 Meta-heuristic Hybrid Algorithms

The meta-heuristic hybrid algorithms developed for rectangular strip packing in section 5.5 have been extended to irregular items. This type of packing task is also referred to as a nesting problem. In the two-stage approach the meta-heuristic algorithms are combined with a number of the heuristic placement policies. The design of the placement rules depends on the shape representation method used for the items. Some of the rules described in detail in section 5.6.4 make use of a simple approximation technique; others are based on a more accurate shape approximation.

The performance of the hybrid algorithms is evaluated in chapter 7 using a number of test problems from various industries, which have been used in the literature, as well as constructed test cases. In order to show the effectiveness of the design of the different meta-heuristic methods, their performance is also compared to random search, local search (hill-climbing), the simplex method and the heuristic packing routines, which are used in the decoding stage. The implementation of the various search strategies is described in section 5.6.3.

5.6.3 Implementation of the Search Algorithms

The design of the hybrid approaches for the irregular packing problem has evolved from the design of the hybrid algorithms for the 2D rectangular strip packing problem. The problem-specific decisions concerning objective function, initial solution, representation scheme as well as the operators applied to manipulate the search space have been made in accordance to the implementation in the rectangular case. The data structures and algorithms for this problem, however, are more complex, since they need to deal with geometric figures rather than pairs of integers representing the co-ordinates. For this purpose some of the variables have been modified to suit this problem. The implementation of the generic decisions like the probabilities for the search space operators, the cooling schedule in simulated annealing and the population and generation sizes in genetic algorithms as well as termination criteria for the search are also based on the rectangular strip packing design.

5.6.3.1 Shape and Problem Representation

Compared to rectangular packing tasks the data structures used to describe irregular parts are more complex. The geometric shapes that need to be packed vary from industry to industry. Some industrial problems only involve simple shapes such as circles e.g. loading of containers with pipes; others have to deal with a more complex geometry. In the metal industry for instance convex and concave polygons are frequent as well as free-form shapes that consist of a sequence of line segments and arcs. The latter are often found in the textile industry. In addition, a special type of geometric figure occurs in the metal and shipbuilding industries, which contains either partial or complete enclosures. If these enclosed areas are large enough, they can be used for nesting smaller parts.

The complexity of the data structures and algorithms for irregular packing problems depends on the geometric type of the parts and the shape description used. Since the objective here is to develop methods which can work with most irregular problem types, the description of the geometric shapes has to be flexible. It also needs to cater for many applications, be relatively time efficient with respect to the run time of the search algorithms and sufficiently accurate to ensure close nesting. Therefore the irregular parts have been described by polygons, which may be concave. All parts in the item set which are not polygonal, e.g. parts containing arcs are approximated by their enclosing polygon (Figure 5.29). This is also true for circular and ellipsoidal shapes. Description by solid polygons will neglect eventual enclosures in the part. The data structure used to describe a polygon involves the following parameters:

- item number: unique number in permutation
- reference point:
  i.e. the offset in x and y direction
  from the reference point of the object
- height *h* of enclosing rectangle
- width *w* of enclosing rectangle
- list of vertices in (enclosing) polygon
- orientation:
  in anticlockwise direction from positive x-axis;
  discrete step size

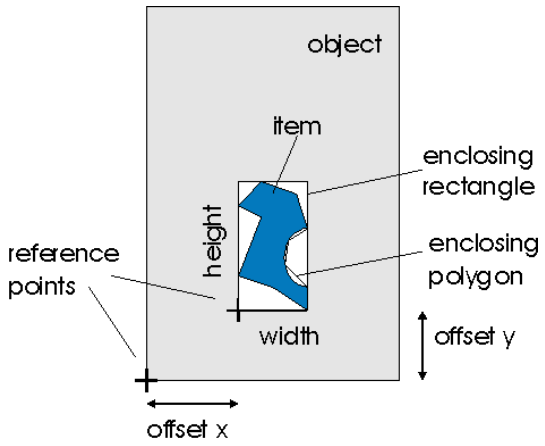


Figure 5.29: 2D shape representation in irregular problems

Shape approximation raises the question of accuracy. The more vertices are used to approximate a shape, the more accurate is the description of the shape. Higher accuracy in the shape representation enables close nesting of two parts. On the other hand every additional vertex will result in higher run times for the nesting routines such as intersection checks between two polygons. The approximation therefore is a trade-off between accuracy and computation time. Too much detail in description may increase the computational cost tremendously.

Other techniques for the shape description are approximation by rectangles and convex polygons. These methods are most suited in specific applications where many of the parts have rectangular features, e.g. in the metal industry, where parts have minor concave features which can be ignored in order to reduce computation time. Since these approximation techniques are only used under special circumstances they have not been further investigated in this study. The input data sets used for the performance evaluation of the hybrid algorithms in this work only contain polygonal shapes (Appendix B). Therefore the issue of accuracy in the approximation of arc structures has not arisen in the implementation.

The run times of the meta-heuristic search processes are higher than the ones of the problem-specific heuristics. It is important to keep the computational effort within the nesting algorithms as low as

possible, since they are used every time a solution is evaluated during the search process. The operators of the search method also manipulate the orientation of the items. Before a sequence of items can be placed by the decoding routine, the polygons have to be rotated into the correct orientation. Throughout the search process many orientations of a polygon will be tried repeatedly. In order to avoid repetitive rotation operations, all rotation calculations are done prior to the search process. Therefore the number of possible orientations of a polygon has been fixed. All configurations regarding the orientation are calculated and stored in a look-up table for each polygon. Before the construction of the layout through the decoding algorithm, copies of the polygons in the respective orientations are obtained from the table and presented to the packing routine. The step size of the rotation angle has to be chosen in advance and can have any value. This value is limited by the cost of the dynamic memory and is dependent on the problem type (Figure 5.14).

Much has been said of about the benefits of using integer data types with respect to computation time in section 5.5.3.1. It is advantageous to describe geometric shapes other than rectangles by integers. This is not possible as rotation operations involving angles other than multiples of 90° will result in co-ordinates that are real numbers rather than integers. The use of floating-point arithmetic for geometric algorithms is not reliable and can result in inaccuracies and errors. For this reason the kernel used by the LEDA library (section 5.2.2) was extended to exact rational arithmetic in order to deal with this issue (Mehlhorn and Näher, 2000). The use of data types for the exact rational arithmetic increases the computation time of the nesting algorithms. A time efficient execution of the nesting process allows the search process to finish within acceptable times.

Since the execution time is of paramount importance, a third approach has been used for the implementation of the data structures and algorithms in this work. In this shape description all polygon co-ordinates are of integer type. Rotations by angles other than 90° multiples can result in co-ordinates represented by floating point numbers and are rounded to the nearest integer. This involves a certain inaccuracy, which depends on the range of numbers used for the description of the polygon. If the polygon is in the vertical and horizontal range between 0 and 100, this description differs greatly from the accurate one, scaled to a higher range, the difference is smaller (Figure 5.30). This is demonstrated in Table 5.19 that shows the relative difference between the polygon area in the accurate shape representation and the polygon area in the approximated representation.

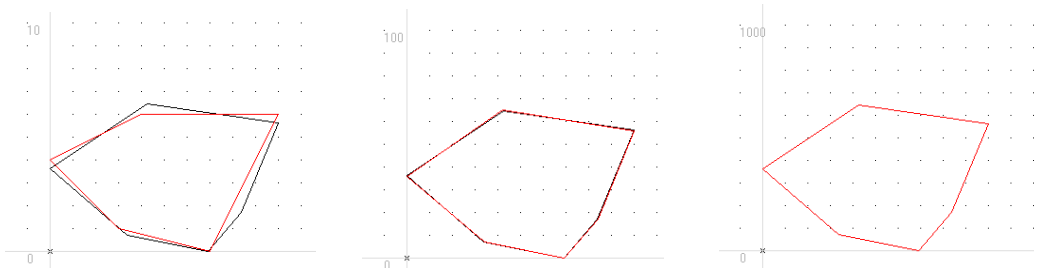


Figure 5.30: Approximated and accurate representation of a polygon in different ranges

Table 5.19: Polygon approximation in dependence of the scale

	polygon 1			**polygon 2**		
range	area of original polygon	area of rotated polygon	relative difference [%]	area of original polygon	area of rotated polygon	relative difference [%]
0.10	40.5	37.5	-7.4070	13.5	15	1.1111
0..100	4050	3165	0.3704	1350	1336	-1.0370
0..1000	405000	404990	-0.0026	135000	134909	0.0678

Before the look-up table with all possible orientation configurations of the polygons is created, all polygons are scaled by a suitable factor. The factor is limited by the integer number range (-32k..32k). The small inaccuracies in this representation technique require the validity of a layout to be verified after construction. This is done with a routine that tests if any of the polygons when replaced by the accurate copy intersect. In case of a conflict, the layout is either discarded and a correction attempt can be made if it is of very high quality using a series of translation operations involving the intersecting polygons and their neighbours. For some of the performance tests presented in chapter 7 the polygons are rotated into the orientation where the area of the enclosing rectangle is minimal. This orientation has then been treated as the orientation at 0° and used for the creation of the look-up table.

Using a polygon data type based on integer co-ordinates allows an efficient implementation of the placement algorithms for the decoding stage. In this implementation, however, a compromise has been made using the LEDA polygon class based on the float-point kernel. This has been mainly due to the additional time needed for design and implementation of a safe new polygon class, which is based on integers, as well as the member functions and routines operating on this class. The implementation and use of the special polygon class, which only involves integer co-ordinates, allows a further decrease computation time with respect to some geometric calculations. In spite of the restriction to the floating-point implementation in this work, the integer approximation of the polygons as described above has two major advantages. It ensures that the algorithms that make use of co-ordinate comparisons work reliably, and avoids the use of the less time efficient rational polygon class, which would be needed for the safe operation of some nesting routines.

As in the hybrid methods used in rectangular strip packing (section 5.5), the complete set of items is described by a permutation with each item number only occurring once. For the interpretation of the permutation in respect to the 2D layout, a placement routine is used. The task of the meta-heuristic in the hybrid combination is to search for appropriate item sequences that are translated into the packing pattern in the subsequent decoding stage.

5.6.3.2 Search Space

The hybrid search methods for the irregular problem use a search space that contains only solutions that result in valid layouts. Therefore solutions with overlapping items or solutions which accommodate only a part of the item set are excluded from the solution space.

Some of the approaches in the literature include incorrect layouts in the solution space and allow the search process to reach invalid states. This technique has been used frequently in connection with simulated annealing (Jain et al., 1988; Theodoracatos and Grimsley, 1995). Some researchers experimented with this type of solution space using genetic algorithms (Petridis and Kazarlis, 1994; Ismail and Hon, 1995). There are several ways to deal with invalid solutions during the search process. The easiest is to discard them (Marques et al., 1991). However, valuable information to guide the search process may be lost as well as precious computation time.

A second way allows invalid layouts to remain in the search process, but assigns a low fitness value to them (Ismail and Hon, 1995). Some effort needs to be spent to determine a suitable penalty term and its weight in the evaluation function. A third possibility is to correct invalid solutions and transfer them into valid layouts (Ratanapan and Dagli, 1997b; 1998). The validity of a solution is usually not reflected in the encoding at this stage because the correction takes place at the layout level. Correction algorithms are complex and the operations on irregular shapes increase the overall computation time if they are carried out frequently.

An advantage of a solution space containing invalid states with respect to packing is that empty and enclosed areas in the layout may be reached. Allowing the search to temporarily accept incorrect states, i.e. overlapping configurations may be useful for nesting into enclosed areas in the layout as well as inside individual shapes (Figure 5.31). In order to describe the quality of an overlapping configuration in strip packing, the height of the layout as a sole parameter is not sufficient, because overlapping layouts have lower heights, although they are not necessarily better and are invalid. A penalty term is needed in the evaluation function to reduce the fitness of the layout. The search process is still able to continue from such a solution, but must not converge on invalid states at the end. The correction process is expensive in terms of time and algorithm development. The route via overlapping configurations is not the only way to fill enclosures. Search algorithms based on search spaces, which do not allow overlapping configurations can use other techniques to fill enclosures in parts and in layouts.

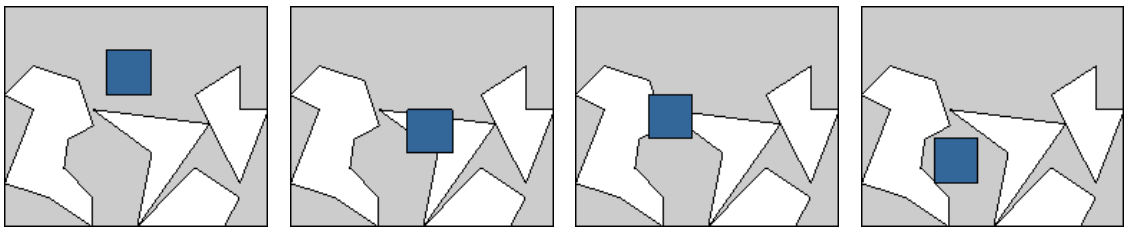


Figure 5.31: Nesting into empty enclosed areas via intermediate incorrect states

In this implementation the search space is also limited with respect to the orientation of the rectangles. The shape representation technique introduced in section 5.6.3.1 requires the number of orientations for the items to be fixed in advance. The number of orientations is limited by the availability of memory and the suitability in the context of the application. In packing applications where the material is not homogenous the number of possible orientations is limited. In the nesting problems that occur in the textile industry shapes can usually rotate by 180°. Reflection and 90° rotation depend on the properties of the fabric. In this implementation the orientation has been handled very flexibly. Depending on the context of the application the interval for rotation angle can be chosen and reflection can be included.

The higher the number of possible configurations, the larger is the search space. An investigation has been carried out to establish the influence of the number of orientations on the outcome of the search process. If not otherwise stated the orientation in the irregular case has been limited to rotations by 90° and reflection, which results in a total of eight orientations per item. Most of the approaches in the literature also work with rotations of 90°. In a number of benchmark problems from the literature the orientation is not considered at all and the items can only be packed in their original orientation.

5.6.3.3 Evaluation of the Quality of the Layout

The major objective of an irregular strip packing problem is to reduce the packing height. The evaluation function for the irregular case considers the height of the layout as well as the remaining area. The remaining area is calculated on the basis of the enclosing rectangles as shown in Figure 5.32. It consists of the area outside the enclosing rectangles, which is part of a continuous piece starting at the top object border. This approximation is also sufficient as an estimation of the remaining height. It facilitates the calculation process and is less expensive in terms of computation time compared to an accurate calculation of the remaining area.

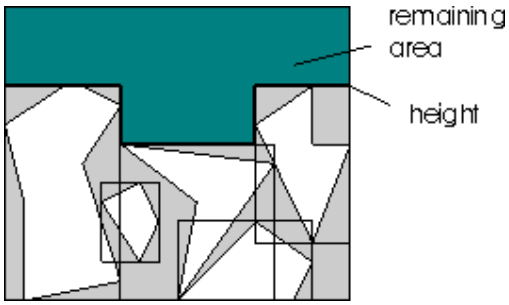


Figure 5.32: Determination of the remaining area and height of the layout

For the evaluation of a solution, the height and remaining area of the layout have been included in the fitness function in the form of a weighted sum (Equation 5.6). In order to facilitate comparisons between packing problems, both terms of the sum are scaled to the object dimensions.

$$fitness = a \cdot \left(1 - \frac{h_{layout}}{h_{object}}\right) + b \cdot \frac{A_{remaining}}{A_{object}} \qquad \text{Equation 5.6}$$

The weights of the two terms have been determined in a series of trials setting ‘a’ to 0.7 and ‘b’ to 0.3. This evaluation function has been used for all the experiments discussed in chapter 7.

5.6.3.4 Genetic Algorithm and Naïve Evolution

The design of the genetic algorithm and naïve evolution approach for the irregular strip packing problem is based on the rectangular problem. Similar problem-specific and generic decisions have been made and are summarised in Table 5.20 and Table 5.21. The orientation of the items is implemented in the meta-heuristic methods, in the form of a mutation operator. Depending on the application context only a certain number of rotations are allowed. The operator randomly chooses an orientation of all permissible ones. The rotation operator is applied with a certain probability to every item in the chromosome.

Table 5.20: Implementation of the problem-specific design variables for genetic algorithms and naïve evolution for the irregular strip packing problem

design variable	**genetic algorithms**	**naïve evolution**
representation	permutation for item sequence	permutation for item sequence
objective function	minimise height of layout (Equation 5.6)	minimise height of layout (Equation 5.6)
initial population	random	random
crossover	PMX (1-point PMX)	-
mutation operator 1	order-based	order-based
mutation operator 2	random change of item orientation	random change of item orientation

Table 5.21: Implementation of generic design variables for genetic algorithms and naïve evolution for the 2D irregular strip packing problem

variable	genetic algorithms	naïve evolution
cross-over rate	60%	-
mutation rate	3%	3%
selection type	proportional; elitism	proportional; elitism
replacement	generational	generational
population size	50	50
termination criteria	1000 generations	1000 generations

5.6.3.5 Simulated Annealing and Hill-Climbing

Since the design structure for simulated annealing and hill-climbing is similar, their implementation for irregular strip packing is presented together in this section. The initial solution is generated randomly. The two search methods then use the same three operators to explore the search space; these can be described as mutation operators. Order-based mutation and position-based mutation work on the permutation, hence changing the order of the elements. The third operator is applied to one element in the permutation and changes the orientation randomly with a certain probability. Only one operator is used to jump from the current state to the neighbouring state.

The generic choices for the implementation of a simulated annealing algorithm are summarised in the annealing schedule (Table 5.22). As in the strip packing problem the schedule presented in Press et al. (1995) is used with a few alterations concerning the length of the Markov chain. The number of iterations at each temperature has been modified in order to reduce the total simulation time. The initial temperature has been calculated separately for each problem at the beginning of the search process using the random walk method by Press et al. (1995).

Table 5.22: Annealing schedule for simulated annealing and parameter for hill-climbing for the 2D irregular strip packing problem

	simulated annealing	hill-climbing
initial temperature	calculated by random walk method	
size of neighbourhood		100 N
length of Markov chain	50 N moves maximum 5 N accepted moves	
decrement rule	$T_{k+1} = 0.9\ T_k$	
termination criterion	k=100; k= number of decrements	N unsuccessful trials

5.6.3.6 Downhill Simplex Method

The downhill simplex method was developed for multi-dimensional function optimisation and only requires function evaluations rather than derivatives (section 3.2). The method allows minimisation of functions in N dimensions, but is not very efficient for large dimensions, which makes it impractical for complex problems. It has been applied to the irregular strip packing problem for 2D and 3D optimisation.

In 2D optimisation the two independent variables in the packing problem are the x- and- y position of the polygon to be placed. In this case the simplex consists of three vertices, which represent possible insertion points for a polygon in the layout. The quality of an insertion point is assessed with an evaluation function. After experimenting with different evaluation functions, the y-co-ordinate of the highest vertex in the polygon has been chosen to describe the quality of the insertion point. It is possible for some of the insertion points to result in invalid layout configurations. Overlap in the layout is penalised with very large function values, in order to discourage the simplex method from further exploration of these points. In this implementation the vertices of the simplex consist of integer numbers.

The downhill simplex method has been used to minimise three independent variables of the strip packing problem. As explained above the first two variables represent the x- and y-position of the polygon. The third one describes the orientation of the polygon. In the 3D optimisation the simplex consists of four vertices, which describe the insertion point and the orientation of the polygon. In this implementation the insertion points can only take on integer values. An angle of 12.25° has been used for the minimum rotation interval. The evaluation function is the same as in the 2D implementation.

Using this evaluation function it is theoretically possible to nest items into enclosing areas of the layout or holes inside parts, because the vertices explored by the simplex method describe possible insertion points for a polygon. The allocation in the partial layout is determined by the insertion point rather than calculated by a series of sliding movements as in the case of the heuristic placement procedures introduced in section 5.7.4. that are therefore not capable of hole filling. Since the smallest overlap is severely punished in the evaluation function used here, hole filling is possible, but not very likely. It cannot be achieved through a series of intermediate (invalid) solutions as illustrated in Figure 5.31. For this purpose a different evaluation function with a more moderate penalty term would be needed.

5.6.4 Heuristic Placement Algorithms

For the construction of the layout packing, rules are needed that translate the item sequence described by the permutation, into the layout. The set of packing rules are similar to the ones used for the rectangular packing problem in section 5.5.4. These place each rectangle as far as possible to the bottom and the left borders of the object; the resulting packing patterns show bottom-left stability. A similar idea has been used for irregular strip packing problems. The sequence in the permutation of the irregular items is allocated using a series of successive downwards and leftwards movements. Five of the techniques

introduced in the following sections make use of a sliding technique. The sixth method is based on the BLF-routine where an existing layout is compacted by performing a series of downwards and leftwards movements on each polygon. These movements are restricted to the horizontal and vertical directions.

5.6.4.1 Distance Calculation between Two Polygons

Before a polygon can be translated either horizontally or vertically the distance to the partial layout needs to be calculated. The technique applied here to determine the distance between two polygons has two steps and is described for a horizontal translation. In the first step the distance between the polygon and its left neighbour is measured. Rays are constructed which start in the vertices of the right polygon and are directed towards the left polygon. The rays that are within the correct range will intersect with at least one segment of the polygon. The distances between the intersection points with the polygon and the starting points are calculated, keeping the smallest distance. In the second step the same procedure is then carried out for the left polygon, also determining the minimum distance (Figure 5.33). The smaller of the two values is the distance by which the polygon can be translated without intersection. The calculation of the vertical distance between two polygons is based on the same principle.

Figure 5.33: Calculation of horizontal distance between two polygons

For an efficient implementation of this routine, care has to be taken that not too many redundant calculations are made. Figure 5.34 illustrates some circumstances under which it is not necessary to carry out the distance calculation. Example 1 shows that rays do not have to be constructed in vertices that lie outside a certain range. This range is determined by the top and bottom point of the right polygon. Rays are not constructed in vertices that are not 'visible' from the other polygon (example 2). Example 3 illustrates how the distance between two polygons is determined using rays and segments. Redundant operations can be avoided when only the 'visible' segments are tested which lie in the range of the ray. These are only a few examples showing how redundant calculations can be avoided.

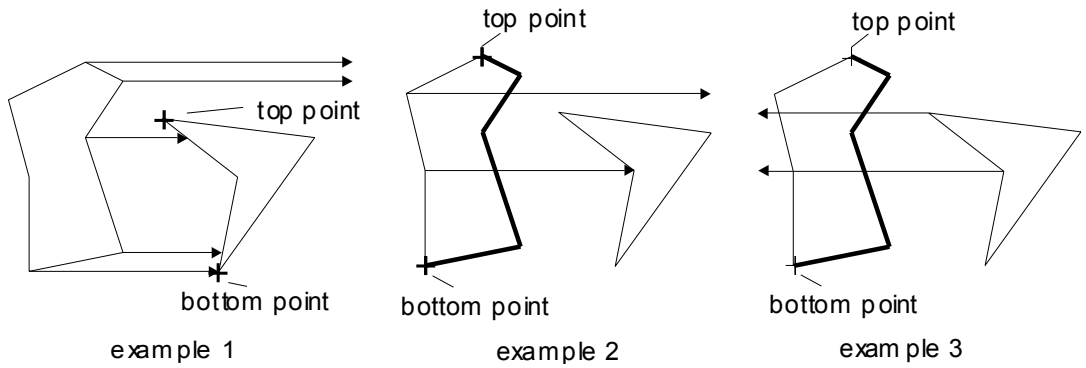


Figure 5.34: Redundant tests in the distance calculation

5.6.4.2 Sliding Algorithms Based on Rectangles

Five of the six packing routines used in the hybrid combination with the meta-heuristics belong to the group of sliding algorithms, which pack a polygon by a series of downwards and leftwards movements.

Three of the algorithms in this section are based on the packing rules developed for the rectangular strip packing in section 5.5.4. They have been modified to handle irregular shapes and consist of two steps. In the first step only the enclosing rectangles of the items are used. A new item is placed into the layout according to one of the placement techniques suitable for rectangles, which have been described in sections 5.5.4.1 to 5.5.4.4. Figure 5.35 shows the placement of the enclosing rectangle of a polygon using the BL-algorithm.

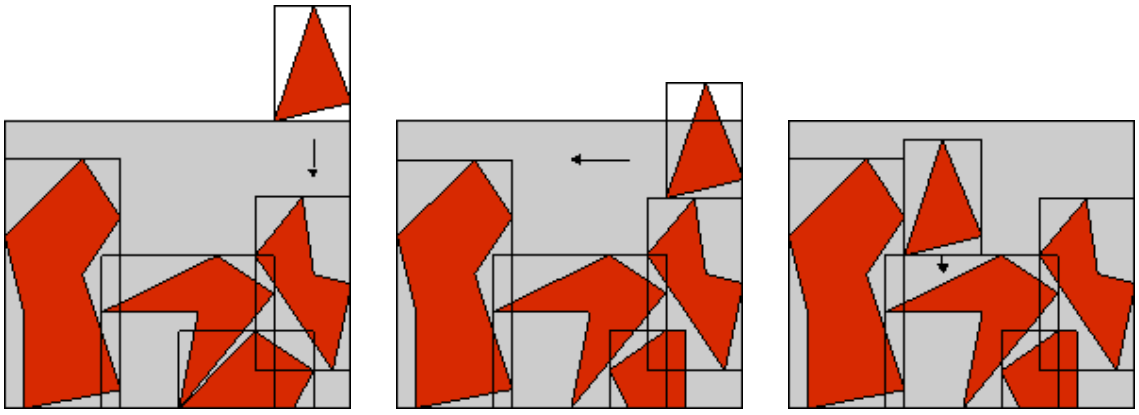


Figure 5.35: First step of BLi-algorithm

Once a stable position for the enclosing rectangle has been found, the polygon is nested using an iteration routine that applies a series of vertical and horizontal movements until the polygon is in a bottom-left stable position in the layout (Figure 5.36). Since the shape representation and some of the nesting algorithms are based on the integer data type, the distances the polygon is moved are rounded to the smaller integer.

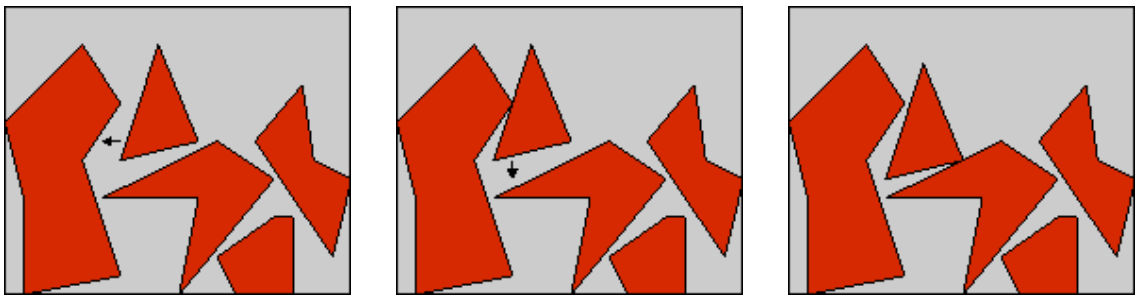


Figure 5.36: Second step of BLi-algorithm

The three routines applied to place the enclosing rectangles are the BL-, the BLLT- and the BLD-algorithm. The extended versions implemented for use with polygons are called BLi-, BLLTi- and BLDi-algorithm in this work. These three algorithms operate according to the routine outlined in Figure 5.37.

```
repeat
    get next item
    get enclosing rectangle
    place rectangle onto object according to placement rule (i.e. BL, BLLT or BLD)
    repeat
        move polygon vertically to towards the bottom side until collision
        move polygon horizontally to towards the left side until collision
    until polygon is in stable position
until all items placed
```

Figure 5.37: Sliding routine for polygon packing (pseudo code)

5.6.4.3 Sliding Algorithms Based on Polygons

BLPi-algorithm

The use of enclosing rectangles in the first stage of the above placement procedures may not be appropriate for nesting polygons because the fine placement does not start until a stable position for the rectangles has been found. A sliding procedure has been developed which operates on the polygon representation. The starting point for a new item is the top-right corner of the object as shown in Figure 5.38. From there it is moved vertically until a bottom-left stable location has been reached. The distances for the translations are rounded to the smaller integer. This packing routine is called BLPi-algorithm.

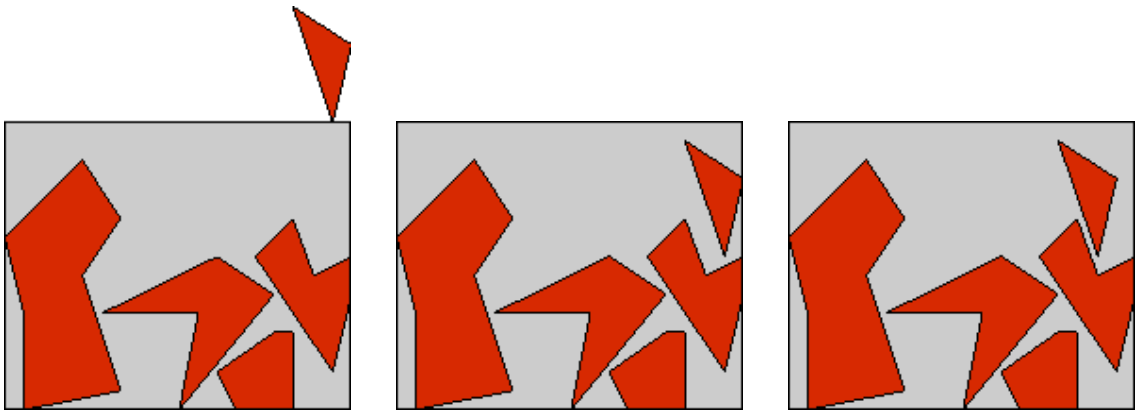


Figure 5.38: Placement of a polygon with the BLPi-algorithm

BLDPi-algorithm

The analysis of the BLPi-algorithm has shown that a packing routine, which uses a sliding principle based on polygons, has some disadvantages. It does not allow to generate efficient layouts in the current implementation (section 7.2). Therefore a second sliding routine has been implemented which uses an initial position different from the BLPi-routine. It starts in the partial layout instead of the top right corner of the object. It is based on the BLD-algorithm for rectangles where the starting point is close to the previously placed item. After placing the reference point of the polygon at the top right corner of the enclosing rectangle of the proceeding item, the polygon is iterated into a stable position like in the BLPi-routine. If the polygon cannot be placed next to the proceeding one, the top-right corner is chosen as a starting point instead. The final location is found when the polygon cannot be moved any further to the left and the bottom (Figure 5.39).

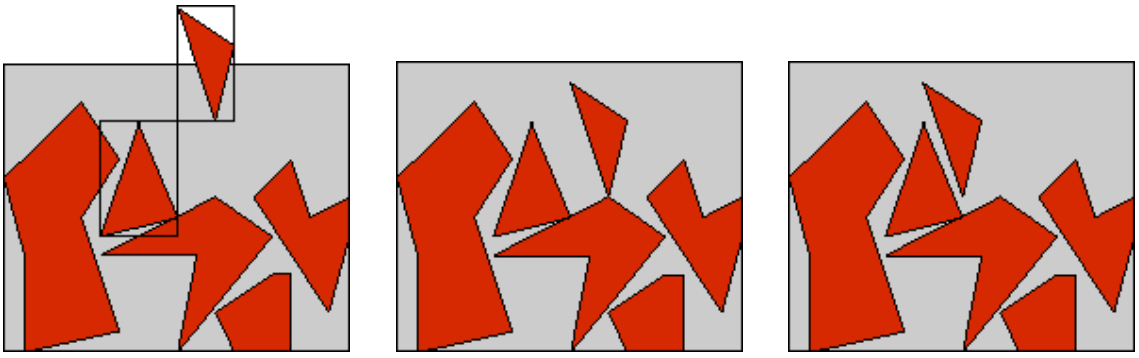


Figure 5.39: Placement of a polygon with the BLDPi-algorithm

5.6.4.4 BLFi-Algorithm

The packing routines introduced in section 5.6.4.2 are based on a sliding technique first applied to the enclosing rectangles. The BLF-algorithm allows construction of very dense layouts in the rectangular case as it can nest into enclosed areas in the layout. This technique has been modified to make it suitable

for the placement of polygons. Firstly, the complete layout is generated using the BLF-algorithm on the enclosing rectangles (Figure 5.41). Then an iteration routine is applied to each polygon until a bottom-left stable position for the polygons has been found. The reference points of the rectangles in the layout are first sorted in bottom-left order. Only one polygon is moved at a time. It is possible that polygons that have been iterated previously could be moved again. In the implementation used here, each item in the pattern has only been iterated once. The right picture in Figure 5.40 shows the layout after each polygon has been iterated in the horizontal and vertical directions. As in the previous packing routines the polygons are only translated by distances rounded to the smaller integer.

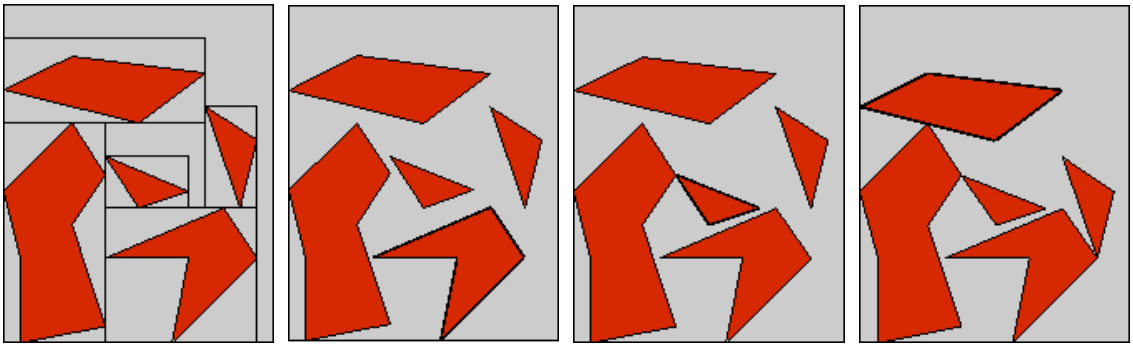


Figure 5.40: Layout generation with the BLFi-algorithm

```
repeat
    get next item
    place enclosing rectangle according to BLF routine
until all items placed
sort rectangle in bottom-left order according to their reference point
repeat
    get next polygon
    repeat
        move polygon vertically to towards the bottom side until collision
        move polygon horizontally to towards the left side until collision
    until polygon is in a stable position
until all polygons iterated
```

Figure 5.41: BLFi-routine for polygon packing (pseudo code)

5.6.4.5 Summary

The packing algorithms introduced in this section have been developed for use in hybrid combinations with meta-heuristic and heuristic search algorithms. The sliding mechanism used in some of these techniques does not provide a means to nest items into enclosed areas in the layout. An exception is the BLFi-algorithm. Unfortunately, this method is limited in respect to in-hole nesting as it uses the rectangle

approximation. None of the algorithms allows nesting inside holes that are part of a shape. The algorithms could, however, be easily modified for this purpose. In a possible design, holes could be treated as additional objects. The packing rules then would have to be extended to allow nesting into irregular objects. Packing tasks that involve multiple objects are known as bin packing problems. The solution of bin packing problems for the rectangular case with meta-heuristics and heuristics is described in the following section.

5.7 Application of Genetic Algorithms to 2D Bin Packing Problems

In this section the meta-heuristic hybrid algorithms that were designed for rectangular strip packing have been extended for use on bin packing problems. In a two-stage approach the meta-heuristic algorithms are combined with the heuristic packing rules introduced in section 5.5.4. In total four different heuristic methods have been used in hybrid combinations with three meta-heuristic search algorithms.

The hybrid algorithms will be compared in terms of solution quality and computation time for a number of different sized packing problems. Their performance is compared with random search and the heuristic packing routines, which are used as decoding algorithms. The implementation of the various search strategies is described in section 5.7.3.

5.7.1 The Packing Problem

The problem consists of packing of a collection of items onto a number of rectangular objects minimising the used object space. The objects can differ in their dimensions and in the set of objects to be used. The packing process has to ensure that there is no overlap between the items. The 2D bin packing problem is very frequent in the metal industry. In most cases it occurs in the irregular version, e.g. in the shipbuilding industry. The characteristics to the specific packing task illustrated in Figure 5.42 are listed below:

- a set of items, which may contain identical items
- the complete set of items needs to be packed
- a set of objects, which may contain identical objects
- all pieces are of rectangular shape
- items can be rotated by 90°

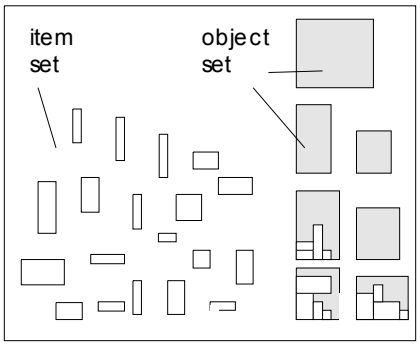


Figure 5.42: 2D rectangular bin packing problem

5.7.2 Meta-heuristic Hybrid Algorithms

The hybrid algorithms developed for the strip packing problem (section 5.5) have been extended to accommodate multiple objects for the 2D bin packing problem. Additional data structures and operators are needed to handle multiple objects. The decoding algorithms used are extensions of the ones introduced for the strip packing problem.

The main focus among the meta-heuristic techniques is put on evolutionary techniques using genetic algorithms and naïve evolution. A number of experiments have been carried out in order to assess the influence of the cross-over and mutation operators. The four heuristic routines used to hybridise the meta-heuristic algorithms are described in section 5.5.4.

5.7.3 Implementation of the Search Algorithms

5.7.3.1 Shape and Problem Representation

The data structures for the rectangular strip packing problem are based on the fast *integer* type. Rectangular items and objects occurring in the 2D bin packing problem can be described sufficiently with the parameters listed below. The item parameters have been extended by the object number, which represents the number of the object on which the rectangle has been located. The object can be represented with fewer parameters since no rotation is permitted.

item parameters:

- item number: unique number in permutation
- reference point:
  i.e. the offset in x and y direction from the reference point of the object
- height *h*
- width *w*
- orientation: in anticlockwise direction from positive x-axis; restricted to 0° and 90°
- object number

object parameters:

- object number: unique number in permutation
- height *h*
- width *w*

The complete set of items forms a permutation where each item number occurs only once. For the interpretation of the permutation in respect to the 2D layout, a placement algorithm is used. Since the objects can be different in area and dimensions the order of objects has an influence on the layout. In order to describe the object set a second permutation has been used containing a unique number for every object in the set. Suitable decoding algorithms are then used to transfer the two permutations into the 2D layout. The task of the meta-heuristic in the hybrid combination is to search for appropriate sequences of all the items and a sub-set of the objects, that is translated into the packing pattern in the subsequent stage.

5.7.3.2 Search Space

The search space for the 2D problem involving multiple objects is extended by object permutation. It is limited with respect to the validity of the solutions and the orientation of the rectangles. Overlapping solutions or solutions which accommodate only a part of the item set on the objects are invalid and therefore not contained in the solution space. For the reasons explained in section 5.5.3.2 the items can only be rotated by 90°. A two-stage approach is used. The validity of all solutions in the search space is guaranteed by the decoding stage. The placement routines are designed such that they only result in valid layouts. Care has to be taken that the area of the available objects is sufficiently large to accommodate the complete item set.

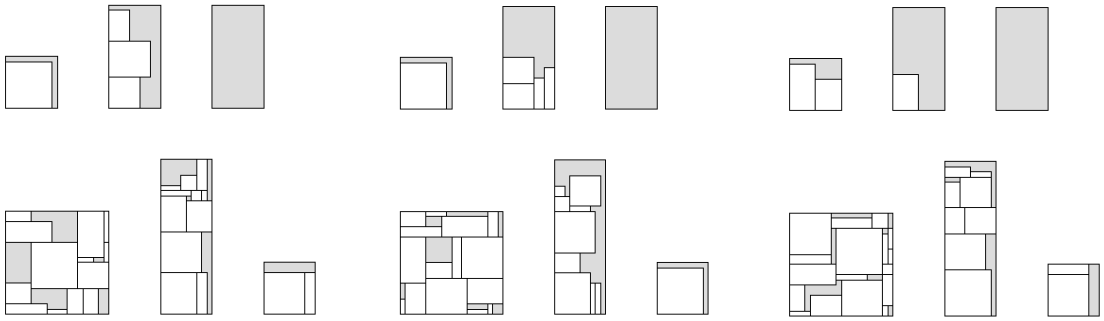
5.7.3.3 Evaluation of the Quality of the Layout

The two permutations represent the order, in which the finite set of items and objects is packed. In order to evaluate the fitness of the solution the layout is constructed using a placement algorithm. The quality of a packing pattern is determined by the utilisation of each object defined in Equation 5.7, where n indicates the number of items on the respective object. The denser the rectangles are packed the less waste is produced and the higher is the object utilisation.

$$object\_utilisation = \frac{\sum_{i=1}^{n} A_{item_i}}{A_{object}} \qquad \text{Equation 5.7}$$

The objective is to find the set of objects with the smallest total area, which accommodate the complete set of items. A simple cost function would be the total area of the objects used to pack all items. This is not sufficient to guide the search process since a large number of solutions result in the same total area. Figure 5.43 shows the solutions to the problem, which use the same total object area. The objects in example 1 are not very densely packed and show many empty areas enclosed between the rectangles. In example 2 the patterns are denser and the empty area consists mainly of one continuous piece on some of the objects. The solution in example 3 is the favoured one. The first four bins are very densely packed.

The last bin contains one item only, which one of the other bins probably could easily accommodate after some minor modifications to the layout. This solution is therefore a good state in the solution space. The objective function needs to indicate this in the fitness value. Applying the total area as the cost function will yield the same cost for all solutions hiding the sub-optimal solutions which could guide the search process.



example 1 example 2 example 3

Figure 5.43: Three solutions to the 2D bin packing problem using the same total object area

Falkenauer and Delchambre (1992) suggested a fitness function for the 1D bin packing problem that includes the capacity of the individual bins. In their work Equation 5.8 was used in a hybrid genetic algorithm for a 1D bin packing problem with M being the number of bins. A_i is the area of all the items in a certain bin with a capacity of A_{object}. The exponent k is a constant (k>1) and puts a higher weight on objects with higher utilisation. Falkenauer and Delchambre experimented with several values for k settling on k=2, which also has been used for the experiments in this work.

$$fitness = \frac{\sum_{i=1}^{M} \left(\frac{A_i}{A_{object_i}} \right)^k}{M} \qquad \text{Equation 5.8}$$

Minimising the number of bins may not always be the objective of 2D bin packing. In terms of pure material cost there is no difference between two bins which are half full and one bin which completely filled. Most of the unused material can be stocked and used again for a different packing process in the industrial application. This, however, will introduce cost regarding the additional handling and stocking. For this investigation the minimisation of the total object area has been used as the objective.

5.7.3.4 Genetic Algorithm and Naïve Evolution

The evolutionary techniques use chromosomes containing two separate permutations - one for the item sequence and one for the object sequence. The cross-over and mutation operators, which are applied to both permutations, are stated in Table 5.23 together with the other problem-specific decisions.

The orientation of the rectangles is considered in the meta-heuristic algorithms in the form of mutation. In the case of orthogonal packing only two orientations are possible for an item. The rotation operator is applied to every item in the chromosome and changes the orientation with a certain probability. In respect to the objects the orientation is kept constant. The specific values for the generic design variables in genetic algorithms and naïve evolution are summarised in Table 5.24.

Table 5.23: Implementation of the problem-specific design variables for genetic algorithms and naïve evolution for the 2D bin packing problem

design variable	**genetic algorithms**	**naïve evolution**
representation	permutation for item sequence permutation for object sequence	permutation for item sequence permutation for object sequence
objective function	maximise bin utilisation (Equation 5.8)	maximise bin utilisation (Equation 5.8)
initial population	random	random
crossover operator item string object string	 PMX (1-point PMX) PMX (1-point PMX)	-
mutation operator 1 item string object string	 order-based order-based	 order-based order-based
mutation operator 2 item string	 change of item orientation by 90°	 change of item orientation by 90°

Table 5.24: Implementation of generic design variables for genetic algorithms and naïve evolution for the 2D bin packing problem

variable		**genetic algorithms**	**naïve evolution**
cross-over rate:	item string object string	60% 60%	-
mutation rate:	item string object string	3% 3%	3% 3%
selection type		proportional; elitism	proportional; elitism
replacement		generational	generational
population size		100	100
termination criteria		1000 generations	1000 generations

A series of simulations has been carried out in order to establish the influence of the cross-over and mutation operator applied to the chromosome describing the objects. The naïve evolution algorithm has been implemented to test the efficiency of the two crossover operators in a genetic algorithm in this order-based approach. The problem-specific and generic decisions for the naïve evolution algorithm are the same as for genetic algorithms apart from the cross-over operator. In naïve evolution the navigation through the search space is done only with the aid of the mutation operators.

5.7.3.5 Simulated Annealing and Hill-Climbing

The 2D bin packing problem is presented in the form of two permutations to the second stage of the hybrid approach. Simulated annealing and hill-climbing are used to search for two good input sequences. The neighbourhood structure of the current solution consists of all solutions in the search space that can be reached by one of the following three operators. Two operators work on the item sequence and manipulate either the order through a swap between two elements or the rotation of one item in the sequence. The principle of the operator, which changes the order of the elements in the permutation, is the same as order-based and position-based mutation in genetic algorithms. The third operator is used for the object sequence swapping two elements according to order-based or position-based mutation. Only one of the operators is needed to transfer the current solution into the adjacent solution. The initial solution is generated randomly.

The generic choices for the implementation of a simulated annealing algorithm are summarised in the annealing schedule (Table 5.25). As in the strip packing problem the schedule presented in Press et al. (1995) is used. The generic decisions for the hill-climbing algorithm are also is summarised in Table 5.25.

Table 5.25: Annealing schedule for the simulated annealing and parameters for hill-climbing algorithm for 2D bin packing

variable	**simulated annealing**	**hill-climbing**
initial temperature	calculated by random walk method (Press et al., 1995)	
size of neighbourhood		100 N
length of Markov chain	100 N moves maximum 10 N accepted moves	
decrement rule	$T_{k+1} = 0.9\ T_k$	
termination criterion	k=100; k= number of decrements	N unsuccessful trials

5.7.3.6 Stochastic Optimisation Algorithm

As in the case of rectangular strip packing stochastic optimisation has been implemented in addition to the meta-heuristic methods described above. The operation of the algorithm is described in section 4.4.4. Since the bin packing problem is represented in the form of two permutations describing the item and the object sequence, the same operators as in simulated annealing and hill-climbing are used to generate the next state in the search. Table 5.26 summarises the implementation of the problem-specific and generic variables for the stochastic optimisation algorithm.

Table 5.26: Implementation of the problem-specific design variables for stochastic optimisation for 2D bin packing

problem specific decisions		**generic decisions**	
representation	permutation (item sequence)	size of neighbourhood	50 N

evaluation function	weighted sum of packing height and remaining area (Equation 5.8)	size of population	100
initial population	random	selection type	ranking
item and objects strings: mutation operator 1 mutation operator 2	 order-based position-based	termination criterion	200,000 iterations or N unsuccessful trials
item string: mutation operator 3	 change of orientation by 90°		

5.7.4 Heuristic Placement Algorithms

For the construction of the bin layout the same heuristic placement routines have been used as for the 2D strip packing problem. All four packing algorithms belong to the class of bottom-left heuristics that preserve bottom-left stability in the layout and are described in detail in section 5.5.4. The placement routines need to be extended in order to deal with multiple objects.

The solution obtained by the search process is presented to the packing algorithm in the form of two permutations. The first one describes the sequence of rectangles and the second one the sequence of objects. The construction process is started with the first elements in both sequences. Under application of one of the four packing rules the items in the first permutation are placed into the first bin. A new bin is started as soon as an item cannot be allocated in the current bin. The packing algorithms attempt to place an item in the partially filled bins first according to their sequence in the object permutation before a new bin is used (Figure 5.44). Care has to be taken that the set of available objects is sufficiently large to accommodate the complete item set.

```
repeat
    get first object
    get next rectangle
    repeat
        place item onto this object according to placement rule (BL, BLLT, BLF or BLD)
        get next object in sequence
    until item is placed
until all items placed
```

Figure 5.44: Placement routine for 2D bin packing (pseudo code)

5.8 Equipment and Simulation

5.8.1 Equipment

All simulations referred to in this work have been carried out on a Pentium II 350 MHz processor under MS Windows NT 4.0. The available RAM memory has been 128MB. All the algorithms have been implemented in C++ (MS Visual C++ 5.0) using the LEDA 3.8 package (Library of Efficient Data Structures and Algorithms) as described in section 5.2.2.

5.8.2 Simulation

The meta-heuristic and heuristic algorithms have been implemented as described in sections 5.5. to 5.7. For the problem-specific and generic variables the standard values listed in the respective section have been used. In order to evaluate the performance of the algorithms, a series of simulations has been carried out according to the parameters described in Table 5.27, Table 5.29 and Table 5.30 unless otherwise stated. The timing information given in connection with the performance analysis of the algorithms in chapters 6 to 8 refers to the above mentioned simulation system. At the time of the simulations the algorithm has been the sole application run on the processor.

The genetic algorithm and the naïve evolution algorithm have both been simulated over 1000 generations. The stopping criterion for the simulated annealing and the hill-climbing algorithm is based on the number of unsuccessful moves. Hence both search processes have been run until termination by this criterion. In order to establish the efficiency of the optimisation processes random search has been applied over the same number of iterations as the respective genetic algorithm (section 5.5.3.9). The outcome of the meta-heuristic methods is compared on basis of the number of iterations. In order to describe the solution quality achieved with a certain method the average result has been used for further analysis.

The heuristic packing methods applied in the hybrid combinations have also been used to evaluate special input sequences. These sequences describe the item set in a sorted order. Since sorted input sequences can achieve good layout quality, a number of sorting parameters have been investigated and compared with each other as well as with the meta-heuristic methods.

5.8.2.1 Simulation Set-up for 2D Rectangular Strip Packing

The performance of the meta-heuristic and heuristic algorithms has been tested with seven different sized packing tasks ranging from 13 to 50 items which have been used as benchmarks in the literature (section 5.3.1.1). A series of guillotineable and non-guillotineable tasks ranging from 17 to 199 items have been constructed (section 5.3.3.1). Each of these categories consists of five test cases. The average outcome for the five problems in a category has been used as the final result for the respective category. Details about

the test problems can be found in Appendix A.1. The various combinations between the genetic algorithms and the heuristic decoders have been analysed statistically according the Friedman Test for non-parametric statistics (Noether, 1990). In order to test for differences in the effectiveness of the methods, the algorithms described in section 5.5 have been simulated using the above mentioned benchmark problems. The parameters for the set-up of the respective simulations are given in Table 5.27.

Rectangles have been sorted by the following four parameters: decreasing height (DH), decreasing width (DW), decreasing area (DRA) and decreasing perimeter (DRP). The letters in the brackets describe how these sequences have been labelled. Random input sequences (Z) have been applied in order to compare the performance of the various sorting techniques. The outcome of the heuristics can vary to a large extent. The average solution quality is not so important and the best result out of 1000 runs has been used for the performance comparison with the meta-heuristics.

Table 5.27: Simulation parameters for rectangular strip packing problems

method	**stopping criterion**	**number of runs**
GA	1000 generations	10 runs
NE	1000 generations	10 runs
RS	number of iterations equivalent to GA, i.e. 100,000	10 runs
SA	based on number of unsuccessful moves or limited to 500,000	10 runs
SO	based on number of unsuccessful moves or limited to 200,000	10 runs
HC	based on number of unsuccessful trials	10 runs
heuristics		1000 runs

Since the main focus of this work is on genetic algorithms, the influence of some generic and the problem-specific design variables on the layout quality for this implementation has been studied. Table 5.28 provides an overview of the variation of these parameters that are explained in detail in section 3.3.1. This investigation has been carried out using four problem categories, i.e. T2, T3, T4 and T5, which differ in the number of rectangles (section 5.3.3.1). In each experiment only one parameter has been varied at a time using the average of five runs.

Table 5.28: Overview of the variation of the generic and problem-specific design for genetic algorithms

method no.	**population size**	**cross-over rate [%]**	**mutation rate [%]**	**selection type**	**cross-over type**	**mutation type**
1	26	0	0	pro-portional	partially matched cross-over (PMX1); 1-point cross-over	order-based mutation (OBM)
2	50	20	1	ranking	partially matched cross-over (PMX2); 2-point cross-over	position-based mutation (PBM)
3	100	40	3		position-based cross-over (PBX)	

4	150	60	5		order cross-over (OX)	
5	200	80	10		cycle cross-over (CX)	
6		100	20			

In order to analyse if a parameter within the chosen limits has an effect on the outcome of the genetic algorithms, the Friedman Test for non-parametric statistics (Noether, 1990) has been applied. Each parameter change is referred to as a 'method' in the following. The various parameter categories, which correspond to one column in Table 5.28, are examined separately. The methods in a category are ranked for each test problem according to the final packing height. For instance in the category cross-over rate, six different parameter values have been tried. Therefore six methods have been examined altogether and are ranked from 1 to 6 for each of the five test problems in a problem category, e.g. T2, T3 etc. In order to compare the various methods, the rank sum for the five the problems is calculated for each method and stated in section 6.5. The rank sums are then analysed according to the Friedman Test.

5.8.2.2 Simulation Set-up for 2D Irregular Strip Packing

A number of different sized packing tasks that were used as test problems in the literature have been applied to test the performance of the meta-heuristic and heuristic algorithms. The problems described were either constructed by the authors or originate from industrial applications (section 0). A series of irregular packing problems containing 15 to 75 polygons have been constructed. They can be divided in two categories. The problems in category 'a' contain multiples of the smallest problem set, i.e. ‘poly1a’, whereas category 'b' describes problems which consist of N different polygons (section 5.3.3.2). Details about the test problems can be found in Appendix B.2. For the simulation of the algorithms described in section 5.6 the above mentioned benchmark problems have been applied using the respective parameters for the set-up (Table 5.29). The analysis of the meta-heuristic and the heuristic search methods in chapter 7 uses the average outcome of the respective number of runs.

Table 5.29: Simulation parameters for irregular strip packing problems

method	**stopping criterion**	**number of runs**
GA	1000 generations	5 runs
NE	1000 generations	5 runs
RS	number of iterations equivalent to GA, i.e. 50,000	5 runs
SA	based on number of unsuccessful moves or limited to 500,000	5 runs
HC	based on number of unsuccessful trials	5 runs
heuristics		1000 runs
simplex method	tolerance for decrease in evaluation function	5 runs

The irregular items have been sorted by six parameters in total, i.e. decreasing height (DH), decreasing width (DW), decreasing area (DA) and decreasing perimeter (DP), decreasing area of enclosing rectangle (DRA) and decreasing perimeter area of enclosing rectangle (DRP). For comparison purposes random input sequences (Z) have been applied to the packing heuristics. The best result out of 1000 runs has been used for the comparison with the meta-heuristic methods.

Regarding the meta-heuristic search algorithms all experiments in sections 7.3 and 7.4 have been carried out using the minimum enclosing rectangle concept if not otherwise stated. This means that each polygon is rotated into the position where its enclosing rectangle has the smallest area after the data input to the system. This orientation is then regarded as the starting orientation, i.e. orientation 0° for the further manipulation. A rotation interval of 90° has been used, limiting the number of possible orientations for a polygon to eight.

5.8.2.3 Simulation Set-up for Rectangular 2D Bin packing

Three different-sized problem categories have been used for the experiments with 2D bin packing. Each problem category consists of five test problems that contain the same number of items and objects and have a similar total area of rectangles (section 5.3.3.3). Table 5.30 summarises the set-up for the simulations carried out with the various algorithms. As an additional stopping criterion the maximum number of iterations has been used for simulated annealing and stochastic optimisation in order to reduce the simulation time.

Since the objective of the 2D bin packing problem is to minimise the total area of the objects the performance comparison of the various algorithms is based on the average object utilisation. The average object utilisation is the average of the utilisation all objects needed to accommodate all items and has also been used by the commercial software package Nestlib to describe the quality of the nesting process. Each problem category consists of five test problems using the average of the five problems to describe the performance of an algorithm in connection with a certain test category.

Table 5.30: Simulation parameters for rectangular packing problems using multiple objects

method	**stopping criterion**	**number of runs**
GA	1000 generations	10 runs
NE	1000 generations	10 runs
RS	number of iterations equivalent to GA, i.e. 100,000	10 runs
SA	based on number of unsuccessful moves or 300,000 iterations	10 runs
SO	based on number of unsuccessful moves or 200,000 iterations	10 runs
HC	based on number of unsuccessful trials	10 runs
heuristics		1000 runs

The items have been sorted by the same parameters used in the case of the rectangular strip packing problems (section 5.8.2.1). The best result out of 1000 runs has been used for the performance comparison with the meta-heuristics.

Since the search process conducted by simulated annealing uses more iterations than the evolutionary methods in the standard implementation, two experiments have been carried out to establish the influence of the number of iterations. For this purpose the population size has been increased to 200. Together with a generation size of 1500, genetic algorithms roughly perform the same number of iterations as simulated annealing (section 8.4.1). The number of evaluations used in random search has also been increased to 300,000. All methods describing a variation from the standard algorithm have been simulated over the same number of runs as the original version (Table 5.30).

5.9 Summary

A series of meta-heuristic algorithms have been introduced for three types of packing problems. Namely rectangular and irregular strip packing problems and 2D bin packing involving rectangles. The meta-heuristic as well as the heuristic search methods are hybrid algorithms where the meta-heuristic is used to search for high quality item sequences, which are transformed into the respective layout with the aid of heuristic placement routines. Four different placement routines have been used for this study.

For the efficient implementation of the nesting algorithms in terms of development time as well as computation time the LEDA software package has been used which contains a library of data structures and algorithms for combinatorial and geometric computing. Some aspects of LEDA important for this work have been highlighted.

In order to evaluate the performance of new and existing algorithms a set of benchmark problems has been presented. With respect to the rectangular strip packing problem a part of the benchmark problems originates from other published research. The other part has been created with the aid of problem generators designed and implemented for a systematic investigation of the algorithm performance. The test cases for the irregular packing task have either been created manually or they have been obtained from the literature describing industrial as well as artificial packing tasks. For the 2D bin packing problem the test cases have been created with the new problem generators.

The objective of this investigation is to establish the performance of the evolutionary techniques with respect to other meta-heuristic and heuristic search methods. In order to determine the efficiency of the meta-heuristic search processes in general, their outcome is compared to simple heuristic packing algorithms in terms of solution quality and computation time. The hybrid approach, which has been used in this work to apply the meta-heuristic search technique to packing problems, is evaluated against other meta-heuristic and heuristic approaches from the literature. Since the key motivation for algorithm

development in the area of packing is the industry, the performance of meta-heuristic search techniques are compared with commercially available software packages specifically designed for industrial needs.

6. Meta-heuristic Algorithms and Rectangular Packing

6.1 Introduction

This chapter is concerned with the application of meta-heuristic algorithms to the 2D rectangular strip packing problem. The problem consists of a fixed set of items, which have to be allocated on a single unlimited object. As described in section 5.5.2, meta-heuristic hybrid techniques are used to approach this packing task. For this purpose, four heuristics, which belong to the class of bottom-left packing procedures, are hybridised with meta-heuristic and heuristic search algorithms.

Since the main focus of this investigation is on evolutionary methods, genetic algorithms have been studied in greater detail and more attention is paid to the problem-specific and generic decisions for algorithm implementation in this area. In order to show the effectiveness of the genetic algorithms, their performance is compared with other meta-heuristic hybrid approaches such as simulated annealing and naïve evolution. In this respect, naïve evolution is a useful technique to evaluate the design of the cross-over operation in the genetic algorithm.

One of the major drawbacks of the application of meta-heuristics is the relatively large computation time needed by the search process in comparison to heuristic methods. It is not only important to evaluate the various methods on the basis of solution quality, but also to consider the computation time. In particular, the meta-heuristic hybrid algorithms are compared to local heuristic search (hill-climbing) and random search. The packing heuristics used to hybridise the search algorithms are known to generate good layouts under certain conditions. A series of experiments has been carried out in order to investigate this aspect further and to evaluate the effectiveness of meta-heuristics in this context.

A number of different sized packing problems have been constructed in order to compare the meta-heuristics and heuristics. Some problems published in the literature have been used as benchmarks in order to compare the algorithms developed in this work with other research. Summaries of the test problems that are referred to in this chapter are given in section 5.3.1.1 and 5.3.3.1.

6.2 Comparison of the Heuristic Algorithms

Before the performance of the hybrid algorithms can be studied, it is important to determine what kind of solution quality the heuristic packing rules can achieve. The design of the four packing rules used for rectangular strip packing is described in section 5.5.4. Since the placement algorithms contain domain-specific knowledge concerning the layout, they contribute to the solution quality obtained with the hybrid combinations.

In order to establish to what extent the packing algorithms influence the layout quality achieved with the meta-heuristics, they have been executed on random input sequences taking the best packing height over 1000 executions. The heuristic placement algorithms have been tested on a number of different-sized problems ranging from 17 to 199 items. Details of the test problems can be found in section 5.3.3.1 and Appendix A.1.2.

Comparison of the Solution Quality of the Sliding Algorithms

Figure 6.1 summarises the relative distances between the lowest packing height found by the heuristic packing routines and the height of the optimal solution. The comparison of the various heuristics shows that the more sophisticated BLF-routine achieves better layouts for all problems. The layouts generated by the BLF-algorithm are between 3 and 21% better than the ones obtained with the other packing routines. The difference to results obtained by the other algorithms becomes larger with increasing problem size. This comparison also shows that the BLLT-rule works better than the BL-algorithm, as was suggested in the literature (Liu and Teng, 1999), however, they failed to prove their point, since only two test problems were used for demonstration. The difference between the result of the BLLT-routine and that of the BL-algorithm also increases with increasing number of rectangles. The newly developed BLD-algorithm does not perform very well for small problems (<T5, i.e. 73 items). It outperforms the BL-method for larger problems. Experiments have also been carried out with a series of non-guillotineable problems (Appendix A.1.2) which have the same number of items. The same ranking has been obtained for various methods. The relative differences to the optimal height also lie in the same range (Appendix F, Table F.24 and Table F.25).

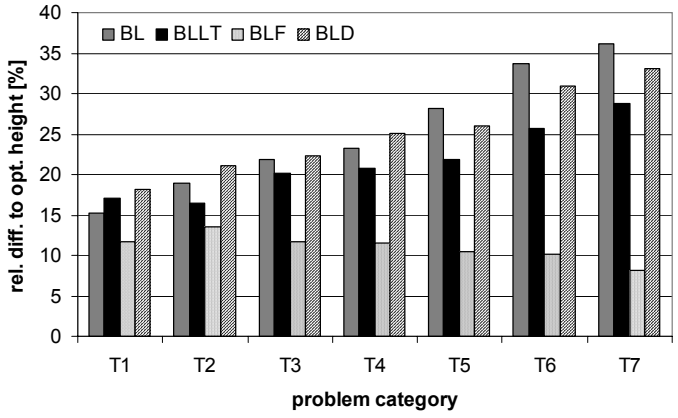


Figure 6.1: Relative difference to optimal height with heuristics for random input

Table 6.1: Standard deviation for height [units]

N	BL	BLLT	BLF	BLD
17	35	35	31	38
25	30	29	25	32
29	34	34	28	34
49	26	25	18	24
73	28	28	17	24
97	28	28	19	25
199	22	23	15	18

The results of the BLLT-routine are better than the ones of the other two sliding algorithms in each problem category with one exception (i.e. T1). This is due to the way a rectangle is moved downwards and leftwards by the BLLT-rule (section 5.5.4.2). Instead of moving it leftwards by the full distance that

is determined either by the left border or a rectangle to the left, it is moved in the bottom direction as soon as the partial layout allows it. This obviously allows the algorithm to achieve lower packing heights. The newly introduced BLD-algorithm only manages to outperform the BL-routine for larger problems (>T4). Its performance against other heuristics is not so important at the moment, since it has been developed especially for use with genetic algorithms (section 5.5.4.4). In order to evaluate its performance, it needs to be looked at in conjunction with the genetic algorithm.

Solution Quality of the BLF-Algorithm

The BLF packing algorithm achieves better packing patterns than the other three heuristics for the given example problems. Since the BLF-routine attempts to fill the gaps in the layout first, the majority of the small items are 'absorbed' within the existing partial layout and do not contribute further to the packing height, which is mainly determined by the larger items (Figure 6.4). The BL-rule cannot access unused regions in layout, so that also smaller rectangles contribute more to the packing height. The results in Figure 6.1 show that in the case of the BLF-routine the difference to optimal solution gets smaller with increasing problem size. The reason for this is that larger problems contain a larger number of small items, which are allocated in the empty areas contained in the partial layout. This is reflected in the standard deviation of the final packing height (Table 6.1), which is lowest for the BLF-routine and decreases with increasing problem size. The packing heights achieved by this routine vary the least, because it attempts to allocate rectangles in the enclosed areas of the layout. Hence they contribute less to the total packing height compared to the other algorithms. In general, the standard deviation decreases for larger problems. This is due to the fact that in the larger problems for this test series the rectangles have smaller proportions compared to the object. A change in the input sequence does not have the same effect on the total packing height as when there are only a few rectangles whose proportions to each other and to the object are larger.

Solution Quality with Pre-Ordered Input

According to Coffman et al. (1984), pre-ordered input sequences can achieve very good layouts compared to random input sequences. The packing heuristics have been applied to sequences, where the rectangles have been sorted by decreasing height (DH), width (DW), area (DRA) and perimeter (DRP). Sorted input sequences can produce lower packing heights than random one depending on the problem size (Appendix F, Table F.1 and Table F.2). Comparing the performance of sorted input sequences to random ones, several trends can be seen. Height- and width-sorted input achieves better layouts than random input for all packing algorithms for larger problems (Figure 6.2). For smaller problems (<T4, i. e. N=49) random input is better. Sorting the sequence by height works better than sorting it by width. Area- and perimeter-sorted input sequences work better for smaller problems than for larger ones. Usually, their outcome is worse than the one obtained for random input. The BLF-algorithm is an exception in this respect. Here, all sorting techniques perform better than random input and their performance increases for larger problems.

In general, pre-ordering the input sequences improves the outcome of all packing heuristics for problems which consist of 73 rectangles and more. The packing height can be improved between 2 and 20% depending on the heuristic used. The best result has been obtained by the BLF-algorithm achieving a packing height that differs only by 2% from the optimum using a height-sorted input sequence (Appendix F, Table F.2).

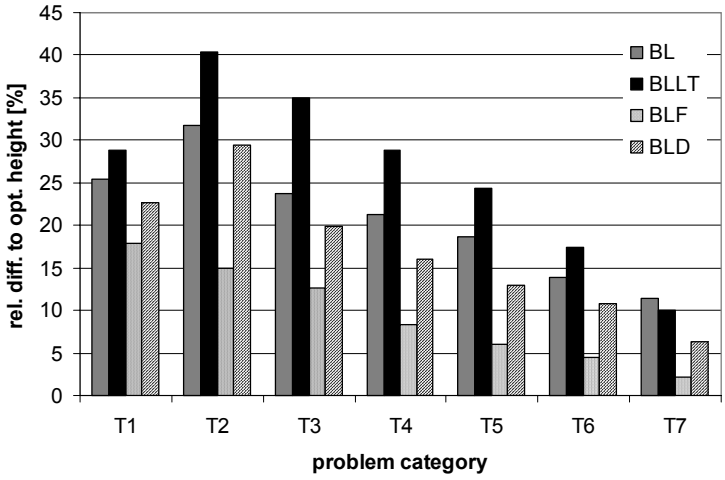


Figure 6.2: Relative difference to optimal height with heuristics using height-sorted input

Time Performance

The packing heuristics studied here are used as decoding algorithms in combination with meta-heuristics, where they are executed a large number of times. It is important to analyse their performance in terms of the computation time. Figure 6.3 shows that the average elapsed time increases exponentially with the problem size for all packing routines. As already mentioned in section 5.5.4.3, the BLF-algorithm has a higher computational complexity than the other sliding routines. This difference is particularly noticeable for larger problems. The average time needed to place one item in one run is highest for the BLF-algorithm (Table 6.2). The number of positions, which are tested on average, before a suitable position is found, is larger than the number of movements carried out using a sliding method. The reason for this is that the number of free positions, i.e. gaps in the layout, increases exponentially with the problem size. Whereas the sliding methods more or less ignore the additional enclosed empty areas, the BLF-algorithm makes an attempt to fill the enclosed areas first.

Among the sliding algorithms, the BL-algorithm is the most time efficient one. The elapsed time for the BLLT-algorithm is higher, because it performs more shifting operations per placement moving the item in a 'staircase' manner along the partial layout. The BL-routine on the other hand moves the rectangles by the full available distance to the left and needs fewer sliding steps in total. Concerning the computational efficiency, the BLD-routine lies between two other sliding techniques. Although it shifts the item by the

full distance like the BL-routine, some overlap checking is needed in order to verify the starting position and the following movements.

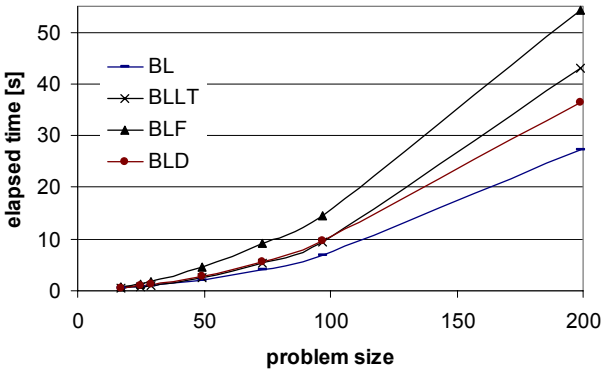


Figure 6.3: Average elapsed time per 1000 runs for heuristics

Table 6.2: Average elapsed time to place one item per run [µs]

N	BL	BLLT	BLF	BLD
17	24	24	47	35
25	28	32	56	36
29	31	34	62	41
49	43	51	94	59
73	58	73	126	78
97	72	98	151	100
199	137	216	272	183

Summary

Simple packing heuristics can achieve dense layouts in particular with input sequences, which are sorted by decreasing height and width (Figure 6.4). Although the execution time for the BLF algorithm is considerably larger, the performance gain especially for large packing problems justifies the application of the BLF-routine. The time complexity plays a more important role in combination with a meta-heuristic. The performance of meta-heuristics needs to be evaluated in the context of the application, since computation times are expected to be much larger. In general, a trade-off between the solution quality and execution time will have to be made in applications, where timing plays an important role.

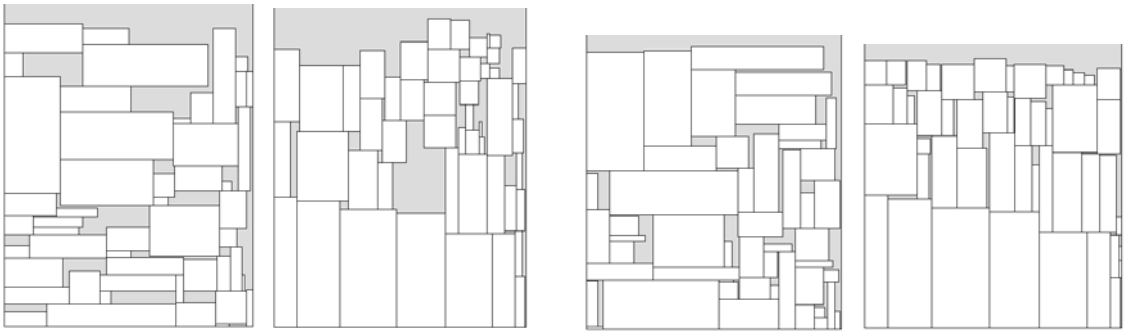


Figure 6.4: Best layouts for T4a with BL (left) and BLF (right); random and height-sorted input

6.3 Comparison of the Evolutionary Approaches

In the following section, the performance of evolutionary approaches is investigated. First of all, the hybrid genetic algorithms are compared to random search in order to study their efficiency in respect to

the simple heuristics. Another aspect investigated regarding the efficiency of the search process is the cross-over operation used by the genetic algorithms. The search processes will be compared to simple heuristics, which make use of sorted input sequences.

6.3.1 Genetic Algorithms versus Random Search

In order to allow a fair comparison between the simple heuristics and the meta-heuristic search processes, the various packing routines are executed over the same number of iterations as the genetic algorithm using 100,000 random input sequences. Figure 6.5 shows that the four genetic approaches outperform random search for all test cases. This means that the genetic algorithms in the current implementation manage to search the solution space more efficiently and find better sequences than pure random trials. The difference to the optimal solution becomes larger with increasing problem size for the genetic algorithms as well as the random search. The only exception in this respect is the combination with the BLF-algorithm where this trend goes in the opposite direction. This will be explained in detail in section 6.3.3.

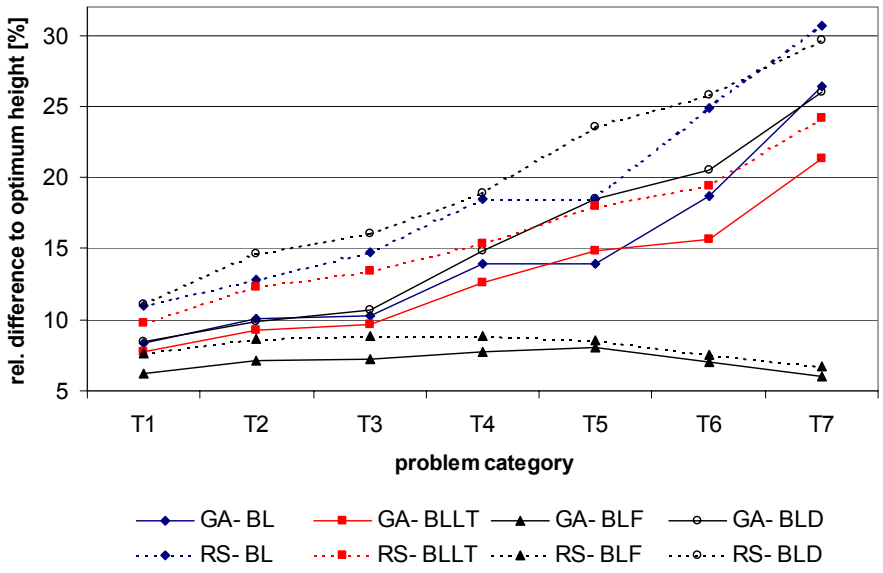


Figure 6.5: Relative difference to optimal height for GA and RS with various decoders

Table 6.3 shows the difference between random search and genetic algorithms when comparing the relative difference between the packing height and the optimal solution. The positive numbers indicate that the genetic algorithms find solutions, which are up to 6% better than the ones of the random search in terms of packing height. The difference to the genetic algorithm outcome first of all depends on the decoder type. With respect to the decoders based on a sliding technique, the genetic algorithms work more efficiently, because the difference to the random search is larger. For the combination with the BLF-algorithm, which generates very good layouts in general, the genetic algorithm achieves results that are

only up to 1.6% better than the random search outcome. Similar trends have been analysed for the series of non-guillotineable problems (Appendix F, Table F.26 and Table F.27). In order to facilitate the comparison between the various meta-heuristic and heuristic approaches, the relative difference to the optimal packing height for this problem set has been summarised in Table F.3 to Table F.5 and Table F.26 (Appendix F).

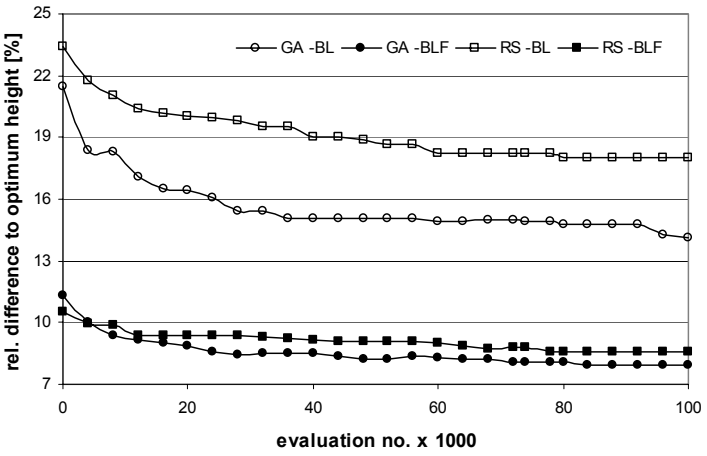


Figure 6.6: Relative difference to optimal height with GA and RS with BL and BLF; problem T4a

Table 6.3: Difference between RS and GA for the rel. difference to the optimal height [%]

	BL	BLLT	BLF	BLD
T1	2.6	2.0	1.4	2.7
T2	2.8	3.1	1.5	4.8
T3	4.4	3.8	1.6	5.3
T4	4.6	2.8	1.1	4.0
T5	1.2	3.1	0.5	5.1
T6	6.2	3.7	0.5	5.3
T7	4.3	2.9	0.5	3.7

Figure 6.6 illustrates the search process of the genetic algorithm and the random search over the number of iterations for the BL- and BLF-decoder. The genetic algorithm in combination with the BL-routine outperforms the random search in the beginning and exploits the solution space efficiently during the first third of the search process. After that, both methods only find marginal improvements. In the case of the BLF-decoder, the course of the search hardly differs between the random and the genetic algorithm. This shows that the BLF-heuristic is a very powerful packing routine, which even generates high quality layouts using random input sequences. Since the input sequence is less important for the BLF-rule, genetic search does not result in a large improvement over the random search.

The random search process which uses the same number of iterations as the genetic algorithm achieves better results than the simple heuristic packing routines whose outcome is based on 1000 iterations (Appendix F, Table F.3 and Table F.5). Since the outcome of the random search lies between the one of the genetic algorithm and the one of the packing heuristics some of the performance gain achieved by the genetic algorithm over the simple heuristics is due to the higher number of iterations.

6.3.2 Genetic Algorithms versus Naïve Evolution

In the following section the efficiency of the search mechanism of the genetic algorithms is examined. One of the main features that distinguish genetic algorithms from other meta-heuristic search processes, is the cross-over operation. Whereas simulated annealing navigates through the search space with the aid of

a mutation like neighbourhood manipulator, genetic algorithms attempt to conserve good aspects of the current solutions using a cross-over operation. In order to evaluate the efficiency of this operator, the outcome of the genetic algorithms has been compared to naïve evolution, which only uses mutation. Table 6.4 shows that the outcome of the two evolutionary methods is very similar with the naïve evolution performing slightly worse for most problems and decoding algorithms (with three exceptions). This is indicated by the positive numbers in Table 6.4. The difference to the results obtained with the genetic algorithms is very small and lies between 0.3 and 2.6%. This difference becomes smaller with increasing problem size.

It can be seen that the course of the search performed by the two methods is very similar with the difference being less than 3% at any stage of the search (Figure 6.7). The solution quality increases quickly in the beginning. Throughout the search, the genetic algorithm stays ahead of the naïve evolution before converging on a similar final value, which is 1% better for the genetic algorithm in this case. This shows that the cross-over operator can be useful for the search process as the example with the BL-decoder in Figure 6.7 illustrates.

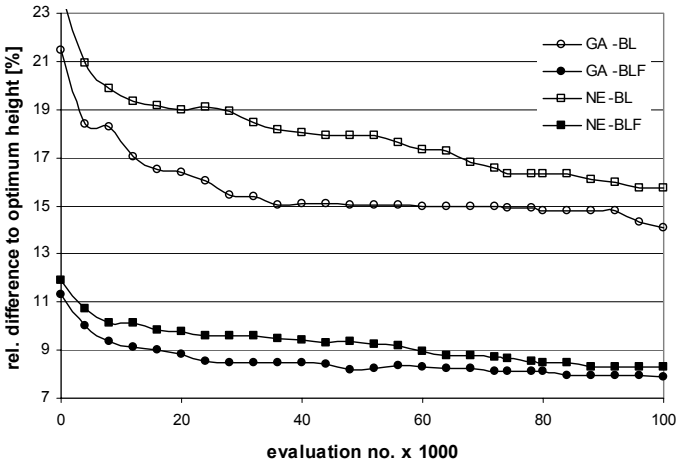


Figure 6.7: Relative difference to optimal height with GA and NE with BL and BLF decoder for problem T4a

Table 6.4: Difference between NE and GA for the rel. difference to the optimal height [%]

	BL	BLLT	BLF	BLD
T1	2.6	1.9	1.3	1.4
T2	1.6	1.8	1.0	1.7
T3	1.7	2.0	1.2	1.9
T4	1.2	1.0	0.6	0.4
T5	1.6	0.6	0.5	0.4
T6	1.9	0.9	0.5	-0.2
T7	0.3	-0.3	0.3	-0.2

This comparison reveals two important aspects. First of all, the cross-over operator is beneficial in the context of the hybrid algorithm, since the genetic algorithm not only performs better than the random search, but also better than the naïve evolution for nearly all problems and in combination with all decoding routines. Secondly, the cross-over becomes less useful for large problems (>T5, i.e. 73 items). Despite the performance gain achieved by the genetic algorithm over random search for larger problems compared to smaller ones (see above), the cross-over operator contributes less to this outcome as the comparison with naïve evolution shows. The difference between the genetic algorithm and the naïve evolution is very small for large problems. The fact that naive evolution is better than random search for

all test cases confirms the success of the evolutionary search principles using a population, mutation, selection and reproduction.

6.3.3 A Comparison of Hybrid Combinations for the Genetic Algorithm

Various decoding routines have been used in the second stage of the hybrid approach for the evaluation of a solution. The BLF-algorithm has been found to generate the best layouts. This is due to its packing principle that allows filling enclosed areas in the partial layout. In general, the layout quality is determined by two aspects, the sequence of the items and the packing mechanism. Although the sequence of the rectangles has an influence on the solution quality as Figure 6.6 shows, where the distance to the optimum becomes smaller with increasing number of random trials, the packing technique itself is more important.

Comparison for Random Input

The BLF-technique outperforms the sliding techniques at any stage of the random search independent of the number of iterations (Figure 6.8). The BLLT-algorithm is the best of the sliding techniques and outperforms the BL- and the BLD-routine by about 4%. The performance of the BL- and the BLD-packing rules is very similar. The graphs of the three algorithms, which reach distinctive final solution qualities, i.e. BL, BLLT and BLF, do not cross each other in Figure 6.8. This shows that the packing technique plays a more important role for the layout generation than the sequence of rectangles as already indicated above. The same performance can be observed using the packing heuristics in combination with the genetic algorithm (Appendix F, Figure F.2).

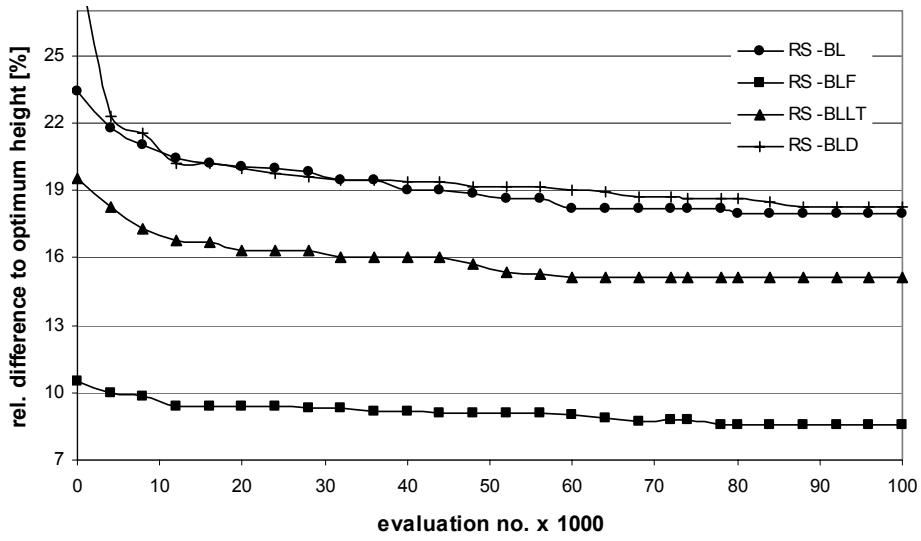


Figure 6.8: Relative difference to optimal height for random search with heuristics; problem T4a

Friedman Test

In order to verify if the differences in the outcome of the various genetic hybrid combinations are statistically significant, the Friedman Test has been carried out for the four different methods (Noether, 1990). The methods applied to all five test problems in one category are ranked between 1 and 4. The rank sums stated in Table 6.5 are obtained by building the sum of the ranks for each method. If the P-value associated with the test result is less than 0.05 there is statistical evidence for the difference in the effectiveness of the various methods. With one exception (i.e. T3) the P-values listed in Table 6.5 are below this limit for all problem categories. This indicates that the decoding method has a significant impact on the outcome of the hybrid genetic algorithms tested. In absolute terms, the BLF-decoder has the lowest rank sum and outperforms the other heuristic packing techniques. The BLLT-routine comes second according to the rank sums for all problem sizes. Concerning the remaining two routines, the BL-rule performs worst for small problems, whereas for large problems the BLD-technique has the highest rank sum. The main reasons for this ranking of the four decoders have already been discussed in section 6.2.

Table 6.5: Rank sums and P-values for the four genetic hybrid methods

method	decoder	T1	T2	T3	T4	T5	T6	T7
1	BL	18	19.5	15	15	16	15	18
2	BLLT	12	11.5	13	10	10	10	10
3	BLF	5	5	5	5	5	5	5
4	BLD	15	14	17	20	19	20	17
P-value p		0.025< p<0.01	0.025< p<0.01	0.05< p<0.025	p< 0.01	p< 0.01	p< 0.01	p< 0.01

Comparison of Genetic Algorithms

In Figure 6.6 the search process conducted by the genetic algorithm and the random search over the number of iterations is shown for the BL- and BLF-decoders. The genetic algorithm in combination with the BL-algorithm outperforms the search already from the beginning and exploits the solution space efficiently during the first third of the search process. After that, both methods only make small improvements. In the case of the BLF-decoder, the course of the search hardly differs between random and genetic algorithm. This shows that the BLF-heuristic is a very powerful packing routine, which even generates high quality layouts when it is applied to random input sequences. Since the input sequence is less important in connection with the BLF-rule, the genetic algorithm does not result in a large improvement over the random search (Figure 6.6).

The course of the search processes, which use the other two sliding techniques (i.e. BLLT and BLD), is similar to the ones with BL-algorithm concerning the relative performance (Appendix F, Figure F.1). Again, the genetic algorithm achieves an improvement over the random search. In absolute terms, the genetic algorithm using the BLLT-algorithm is the best combination among the sliding techniques. This

extends the work carried out by Liu and Teng (1999), who developed this packing rule in order to improve the known BL-routine, however, they did not demonstrate this on a comprehensive set of test cases.

Whereas the random search only 'explores' the solution space, the in-built search mechanisms allow the genetic algorithm to 'exploit' good regions. The difference between the genetic and random search for the BLF-case is smaller, which indicates that the 'exploitation' of the solution space is limited. Since the packing heights achieved with the BLF-heuristic on its own are already very close to the optimal height, the genetic algorithm cannot improve the performance of the heuristic as much as in the case of the sliding routines. In other words by using the 'poorer' sliding techniques as decoder, the genetic algorithm is needed to find a good input sequence, whereas by applying the BLF-heuristic better layouts are achieved in fewer iterations. This questions the use of a genetic algorithm combined with a 'poor' decoder for this type of packing problem. In order to achieve high quality layouts, which is the industrial objective, the simple BLF-heuristic may be sufficient under certain circumstances explained in section 6.3.5.

Benchmark Tests

For a comprehensive assessment of algorithms certain benchmarks tests are required. The most important parameters to analyse the performance of packing algorithms are solution quality and computation time. Benchmark tests can be carried out in two possible ways. The first one refers to a set of benchmark problems. Using the same test problems, algorithms can be compared in absolute terms with respect to solution quality and computation time. The type of benchmark test requires a set of benchmark problems, which is commonly accepted and applied by the research community. Unfortunately, this is not yet the case in the area of cutting and packing despite some effort by two online libraries (section 5.3).

A second possibility to describe the quality of algorithms consists in the comparison with benchmark methods. The performance of newly developed algorithms could be measured against a standard packing method. This could then be used for a comparison in relative terms. So far, the community has not identified a standard test suite. The experiments with the BLF-routine have shown that it is capable of achieving very efficient layouts (sections 6.2 and 6.6). In addition, its implementation is simple as opposed to problem-specific algorithms published in the literature. It also has a fairly short run time. This makes it attractive to be used as a benchmark algorithm in rectangular strip packing. In particular, the combination with pre-sorted input sequences has resulted in layouts of very high quality, which in many cases have been higher than the ones achieved by meta-heuristics (Table F.2 to Table F.4, Appendix F).

The application of this simple heuristic proves also valuable during the development of new algorithms. If new meta-heuristic and heuristic algorithms were compared to the BLF-method at an early stage, their performance would not be overestimated. The lack of simple but efficient benchmark tests contributed certainly to the enthusiasm concerning the powers of meta-heuristics in the area of rectangular strip packing. In many cases complex and problem-specific algorithms do not achieve a higher solution quality

than this simple heuristic (Table 6.12). Compared to meta-heuristic algorithms, the computation time of the BLF-heuristic should also be comparatively low.

Summary

The best layouts have been achieved using genetic algorithms in combination with the BLF-routine as already indicated. Their performance is up to 20% better than the ones achieved by a genetic algorithm using the best sliding technique, i.e. the BLLT-rule (Figure 6.5). For all techniques, packing heuristics as well as genetic algorithms, the difference between the packing heights achieved and the optimal height becomes larger with increasing problem size. Since larger problems have a search space, which grows according to combinatorial laws, the genetic algorithms cannot search the space as fully as for smaller problems. One exception in this respect is the BLF-routine due to the reasons explained in section 6.2.

6.3.4 Analysis of the Performance of the BLD-Algorithm

One of the main reasons for the development of the BLD packing routine has been to extend the adjacency concept by the order-based algorithm to the phenotype, i.e. layout generation. Whereas the final location of a rectangle in the layout found by the other packing rules is not necessarily in proximity to its proceeding element in the permutation, the BLD-routine attempts first to place the item next to its predecessor. Only if this fails, another rule is used to find a valid position. Since genetic algorithms transfer partial sequences to the next generation with the aid of the cross-over operation, it was considered advantageous to interpret the permutation in a way that pays more attention to the sequence of elements.

The comparison with random search shows that the genetic algorithm finds layouts, which are up to 5.3% better than using the same number of random input sequences (Table 6.3). This improvement over random search is larger for the BLD-combination than for the other packing routines with one exception (T6). This shows that the genetic algorithm can exploit some of the sequential information in the permutation and its correspondence to the phenotype successfully. This is not necessarily due to the cross-over operation as the comparison with naïve evolution in Figure 6.9 demonstrates. The difference to the search process without the cross-over operator is sometimes less than 1%.

In absolute terms, the genetic algorithm with the BLD-decoder does not reach the solution quality achieved with the other hybrid combinations (Table 6.3 and Appendix F, Figure F.2). This is probably due to the somehow 'wasteful' manner in which a rectangle is placed. Instead of shifting it all the way down from the top until it is locked in the partial layout, the BLD-rule enforces an allocation next to the proceeding element. Although the rectangle is shifted further down and left in the local layout neighbourhood in the next step, it is not able to move across the layout in the same way as is the case with the BL- and BLLT- routine. This allows the BL- and BLLT-technique to find more lower positions for a rectangle in the partial layout, on average, than the BLD-rule.

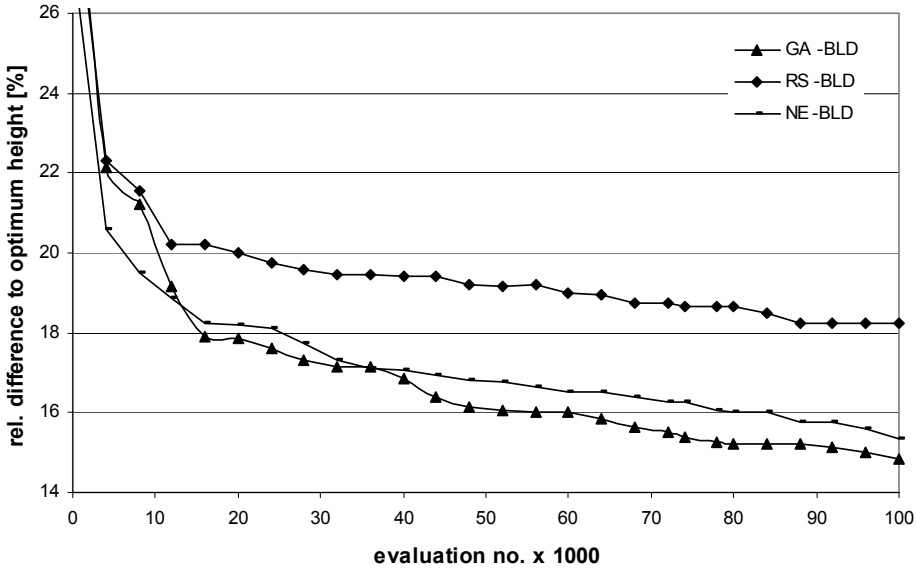


Figure 6.9: Relative difference to optimal height with GA, NE and RS for BLD-decoder for problem T4a

6.3.5 Performance with Seeding

As has been shown in section 6.2, pre-ordered input sequences can achieve high quality layouts. In particular, orderings where rectangles have been sorted by decreasing height and width are suitable for 2D strip packing. In order to make use of this observation, the initial population of the genetic algorithm has been seeded with a pre-ordered input sequence. For this purpose, the best solution that has been obtained by applying one of the sorting criteria examined in section 6.2 to the various heuristic packing rules has been used as a seed chromosome.

The performance gain achieved by the seeded genetic algorithm over the unseeded one is shown in Figure 6.10. These values indicate the relative difference by which the outcome of the seeded method is better than the unseeded one in terms of packing height. The performance gain over the unseeded genetic algorithm increases with increasing problem size. Seeding works particularly well for the largest test problem set, i.e. T7, where the improvements achieved by the hybrids with the sliding routines are between 10% and 15%. The BLF-hybrid is also improved through seeding, although the gain is smaller (<3%).

In order to examine if the genetic search has been able to improve the seeded solution itself, the relative difference between the outcome of the seeded genetic algorithm and the seed has been summarised in Table 6.6. In general, the performance gain over the seeded solution becomes less with increasing problem size. For some of the largest problems, it has not been possible to achieve a solution better than the heuristic one, which has been used to seed the initial population (e.g. T7).

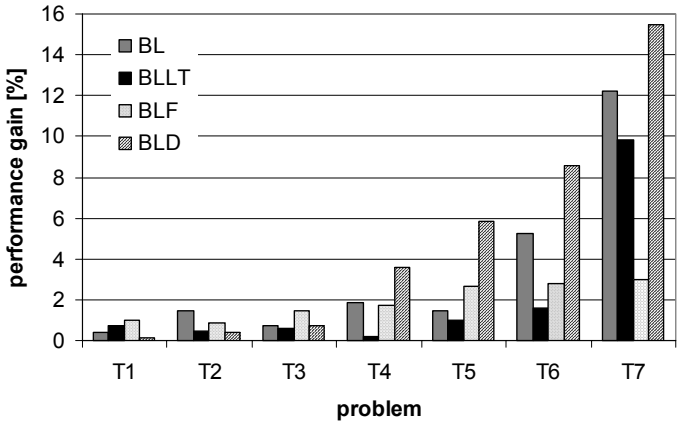


Table 6.6: Relative difference between seeded GAs and the seeded solution [%]

	BL	BLLT	BLF	BLD
T1	6.8	9.4	6.2	9.0
T2	9.6	7.2	6.9	10.6
T3	11.3	10.2	5.7	9.1
T4	8.5	7.5	2.3	4.8
T5	5.7	7.1	0.7	1.3
T6	1.2	3.2	0.3	0.6
T7	0.4	0.5	0.0	0.0

Figure 6.10: Comparison between seeded and unseeded GA

The various experiments using seeding have shown that it can be beneficial to use a seeded population. The success of the seeding technique depends on the size of the problem. The performance gain for small problems is due to the seeding technique, since the genetic algorithm manages to improve the seed between 2 and 10% for problems with up to 49 items, i.e. T4 (Table 6.6). For larger problems, the difference between the outcome of the genetic algorithm with seeding and the seeded solution itself becomes smaller. In this case, the seeded genetic search does not find better solutions than simple packing heuristics, which use pre-ordered input (Figure 6.11).

The application of genetic algorithms to packing problems is only appropriate for smaller problems, where a performance gain over simple heuristics is possible. Genetic algorithms have achieved good layouts for problems with up to 73 items (T5) in this investigation. Beyond that limit, the application of simple heuristics becomes competitive not only in terms of solution quality, but even more concerning the computation time (section 6.3.6).

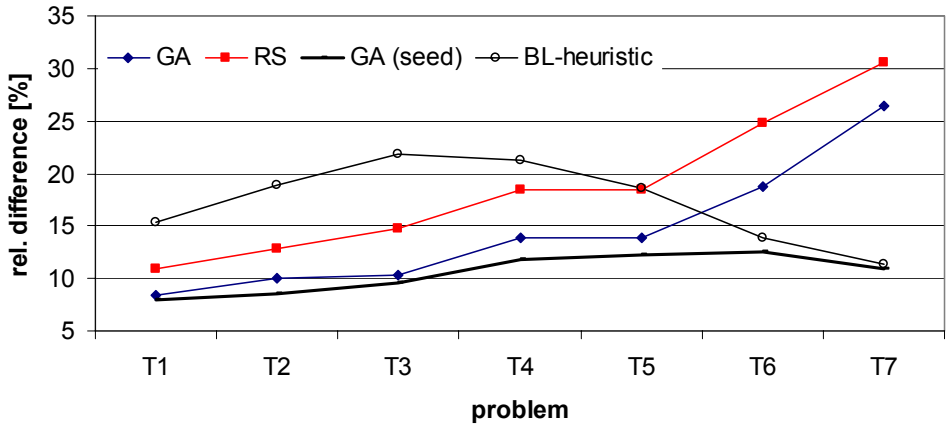


Figure 6.11: Comparison between GA, seeded GA, RS and best sorted input for the BL-algorithm (relative difference from optimum packing height)

6.3.6 Computation Time of the Evolutionary Methods

In the meta-heuristic search process the packing heuristics are used to evaluate the solutions encountered during the search process. Since the number of iterations performed by the search algorithm can be very high, the computation time of the decoding routines is crucial for the total run time of the meta-heuristics. Table F.7 (Appendix F) summarises the elapsed time for the evolutionary methods and the random search for a single run of the algorithm. It can be seen that the elapsed time for all hybrid combinations increases with increasing problem size.

The elapsed time depends on the problem size and the decoding method used. As indicated in section 6.2, the four packing heuristics differ in their run time. This is also reflected in the run time of the hybrid methods. The combinations with the BL-heuristic run faster than the combinations with the other packing routines. Whereas the elapsed time for the methods using the sliding techniques differs only slightly for small problems, the difference becomes higher for larger problem sizes. The hybrid, which uses the BLF-decoder for instance, runs 3 to 4 times longer than the one using the BL-heuristic.

For an evaluation of the computation time, the solution quality also needs to be considered. The BLF-decoder generates the best layout compared with other decoding heuristics (section 6.3.3). The layout quality achieved by the genetic algorithm in combination with the BLF-decoder is between 2 and 15% better than the one of the second best heuristic, i.e. BLLT-rule. Whereas the difference in run time is small for small problems, it becomes important when nesting a larger number of items.

In order to be able to discuss the timing issue, the practical packing application must also be considered. In applications, where the run time is not crucial for the next stage in the manufacturing process, the use of the more sophisticated decoding routine generates layouts of higher quality. If the timing is crucial for an application, a trade-off has to be made between layout quality and computation time. This is particularly the case, in applications where the packing task must be solved in real-time.

Comparing the layout quality of the meta-heuristics with the ones of the simple heuristics, it can be seen that the packing heuristics can outperform the search processes under certain conditions. These conditions depend on the problem size and the type of the packing problem. For 2D rectangular strip packing problems, height-sorted input sequences have been found to produce very good layouts. This is only valid for larger packing problems. Table 6.8 shows that the BLF-method is capable of outperforming the genetic algorithm using the BLF-routine by up to 4% when it is applied to height-sorted input sequences (indicated by negative values). This is not only the case for the BLF-heuristic, but also for the other packing routines. The difference to the corresponding genetic outcome can be up to 20% for problems having between 73 and 199 items depending on the packing rule. In absolute terms the BLF-heuristic achieves the best layouts with height-sorted input, which only differ between 4 and 6% from the optimal packing height for large problems (Appendix F, Table F.2).

Table 6.7: Average elapsed time of the simple packing heuristics for 1000 runs [ms]

	BL	BLLT	BLF	BLD
T1	40	40	80	60
T2	70	80	140	90
T3	90	100	180	120
T4	210	250	460	290
T5	420	530	920	570
T6	700	950	1460	970
T7	2720	4300	5420	3640

Table 6.8: Difference between height-sorted heuristics to GA for the relative difference to the optimal height [%]

	BL	BLLT	BLF	BLD
T1	17.2	21.0	11.6	14.1
T2	21.6	31.2	7.8	19.5
T3	13.4	25.3	5.4	9.2
T4	7.4	16.2	0.6	1.2
T5	1.3	9.5	-2.1	-5.5
T6	-4.9	1.7	-2.6	-9.7
T7	-15.0	-11.3	-3.9	-19.7

The major advantage of simple heuristics in comparison to the meta-heuristics is their extremely short run time (Table 6.7). At the same time, they achieve a higher solution quality for large problems than the respective approaches using genetic algorithms. For this reason the application of genetic algorithms is only appropriate for certain packing tasks. In terms of layout quality, genetic algorithms achieve better outcomes for problems that consist of less than 49 items (i.e. T4). Beyond that limit, the simple BLF-packing heuristic using pre-ordered input is capable of achieving better results (Figure 6.12). In theory, it is possible to improve the results obtained by the packing heuristics when they are used to seed the initial population of the genetic algorithm. The performance gain for larger problems is only marginal (<1% for BLF) as described in section 6.3.5. The use of the meta-heuristic search process with long computation times compared to the heuristic cannot be justified for most applications.

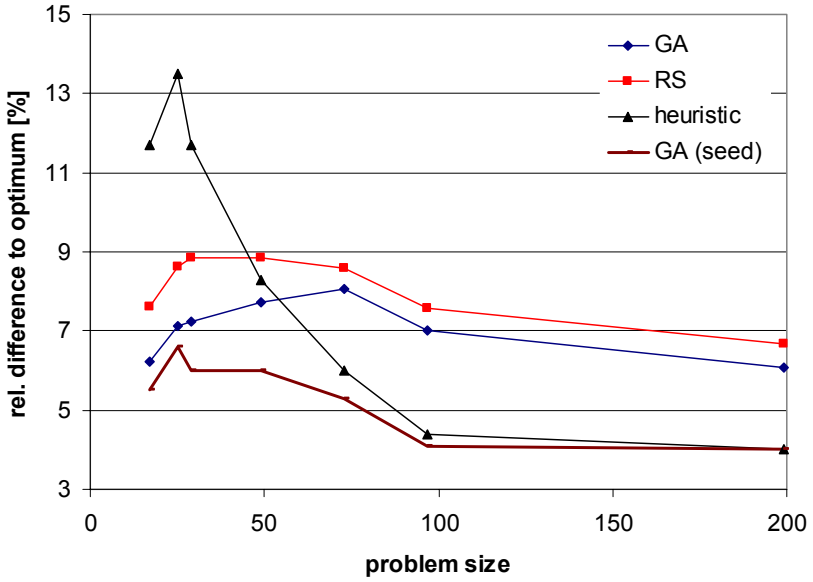


Figure 6.12: Comparison between GA, seeded GA and best sorting for the BLF-algorithm

Comparing the two meta-heuristics and the random search, it can be seen that the naïve evolution routine runs faster than the other hybrids. The reason for this is that naïve evolution only uses a mutation operator, whereas the genetic algorithm also applies a cross-over operation. Since the cross-over is performed on a larger percentage of the population than the mutation (i.e. 60% and 3% respectively) a larger proportion of the population needs to be evaluated in the next generation in the case of the genetic algorithm compared to naïve evolution. Although random search does not apply cross-over and mutation, its run times are longer. Every random sequence needs to be evaluated, whereas in the genetic algorithm and naïve evolution the decoding operation is only performed for the new individuals in the next generation.

6.4 Comparison of the Meta-Heuristic Approaches

In the following section the performance of the genetic algorithm approaches is compared with two other meta-heuristic methods. The first method is simulated annealing, which was successfully used by other researchers in packing applications (section 4.4.1). Instead of manipulating a whole population of solutions, simulated annealing navigates through the search space by a series of jumps between neighbouring solutions. The second meta-heuristic method used in this comparison is a stochastic optimisation algorithm developed by Pargas and Jain (1993). This technique is a combination of genetic algorithms, simulated annealing and steepest-ascent hill-climbing as indicated in section 4.4.4. The reasons for applying it in this context is explained further down. The implementation of the two meta-heuristic search techniques is based on the same hybrid principle as genetic algorithms (section 5.5.3).

6.4.1 Genetic Algorithm versus Simulated Annealing

The comparison between simulated annealing and genetic algorithms shows that simulated annealing is capable of generating very good layouts. In fact, it generates higher packing densities than genetic algorithms in each problem category and for each decoder combination. Table 6.9 shows the difference in the outcome between the genetic algorithm and the simulated annealing techniques in terms of the relative difference to the optimal height. The negative numbers indicate that simulated annealing finds layouts that are closer to the optimal solution. The difference depends on the problem size as well as on the decoder type. It is larger for the combinations with the sliding techniques and increases with increasing problem size. For the best decoder, the BLF-routine, simulated annealing outperforms the respective genetic algorithm by up to 3%.

The values for the relative difference to the optimal height are summarised in Table F.4 (Appendix F) in order to enable an absolute comparison. In absolute terms, simulated annealing achieves packing heights, which only differ between 4 and 6% from the optimum.

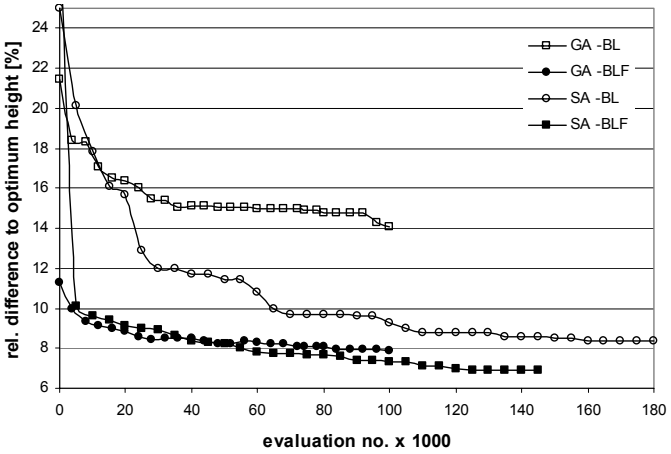


Figure 6.13: Comparison between GA and SA for BL- and BLF-algorithm for problem T4a

Table 6.9: Difference between GA and SA for the rel. distance to optimal height [%]

	BL	BLLT	BLF	BLD
T1	-1	-1	0	-1
T2	-2	-1	-1	-2
T3	-2	-2	-1	-3
T4	-5	-5	-2	-6
T5	-8	-7	-3	-6
T6	-7	-8	-3	-8
T7	-9	-8	-2	-3

Figure 6.13 shows that the simulated annealing algorithm in combination with the sliding technique (BL) converges very slowly compared to the genetic algorithm. Since simulated annealing can only explore one neighbourhood solution at a time, it makes less progress than the genetic algorithm in the beginning of the search process (<20,000 iterations). The genetic algorithm on the other hand can examine a larger section of the search space, since it is based on a population of solutions rather than a single one. The genetic search process therefore finds good solutions faster than the simulated annealing algorithms.

Due to slow convergence, the number of iterations required by the simulated annealing method can be high compared to the evolutionary techniques (Appendix F, Table F.6). This is particularly the case for larger problems. The elapsed time of the simulated annealing methods is larger than for the genetic algorithm (Appendix F, Table F.7 and Table F.8). In the case of the largest packing task i.e. T7, the simulated annealing algorithm runs six times longer and achieves a layout that is 2% better. Whether the improvement in the layout justifies the increased computational effort, depends entirely on the application (section 6.3.6).

The combinations of simulated annealing and the various heuristic packing algorithms produce a similar ranking as the hybrid genetic algorithms. The BLF-heuristic is the best packing routine and achieves solution qualities, which cannot be reached by the sliding techniques (Appendix F, Table F.4). The fact, that the ranking of the various packing routines is the same as in the case of the genetic algorithms (Figure 6.14), indicates that the outcome of the hybrid algorithms is strongly dependent on the capabilities of the decoding methods. Since the termination criteria of the search process used in the implementation of the simulated annealing algorithm is based on the number of unsuccessful neighbourhood moves, the number of iterations differs for the combinations with the various decoders (Appendix F, Table F.6).

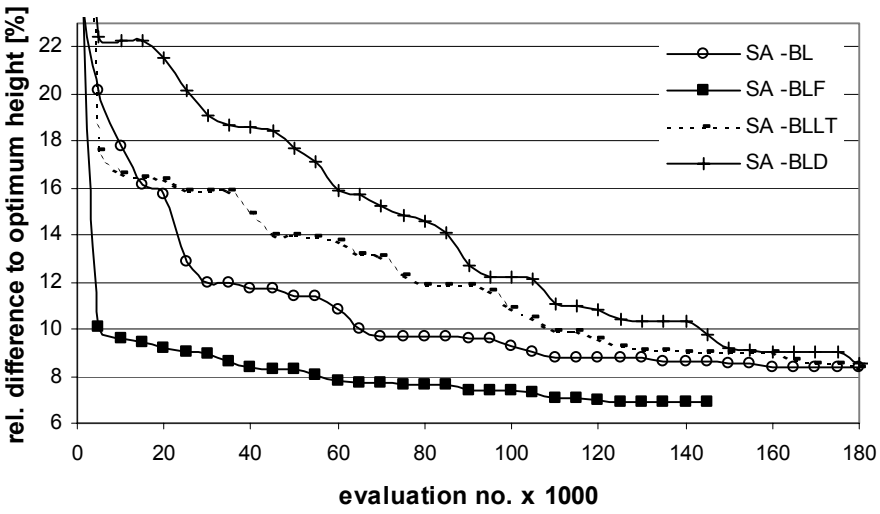


Figure 6.14: Comparison of different decoders for simulated annealing with problem T4a

6.4.2 Genetic Algorithm versus Stochastic Optimisation Algorithm

As seen in section 6.4.1 one of the major drawbacks of simulated annealing is the slow convergence rate. This is especially noticeable looking at the elapsed times of larger problems (Appendix F, Table F.8). Whereas the genetic algorithm converges very quickly, the search process conducted by the simulated annealing algorithm continues for much longer finding solutions that have slightly higher quality. In order to examine why simulated annealing outperforms genetic algorithms for rectangular packing problems, the neighbourhood operators and the reproduction process need to be considered. Concerning the neighbouring solutions, one of the distinct features of the two meta-heuristics is the cross-over operation. Simulated annealing uses an operator that performs the same solution manipulation as the mutation operator in the evolutionary methods. The neighbourhood operator, however, cannot be the only reason for the different outcome, since simulated annealing outperforms naïve evolution for all test problems (Appendix F, Table F.3 and Table F.4).

In order to examine the performance of genetic algorithms and simulated annealing in greater detail, a fourth meta-heuristic search process has been implemented and applied to the test problems. Pargas and Jain (1993) developed a stochastic optimisation algorithm, which combines features from genetic algorithms, simulated annealing and hill-climbing (section 4.4.4).

The comparison between the three meta-heuristics shows that the stochastic optimisation algorithm performs better than the genetic algorithms or at least equally well (Table 6.10). For most problems it achieves a packing height, which is 1% better than the corresponding genetic algorithm. Stochastic optimisation does not outperform simulated annealing. The difference between simulated annealing and stochastic optimisation methods becomes larger with increasing problem size (Appendix F, Table F.4.).

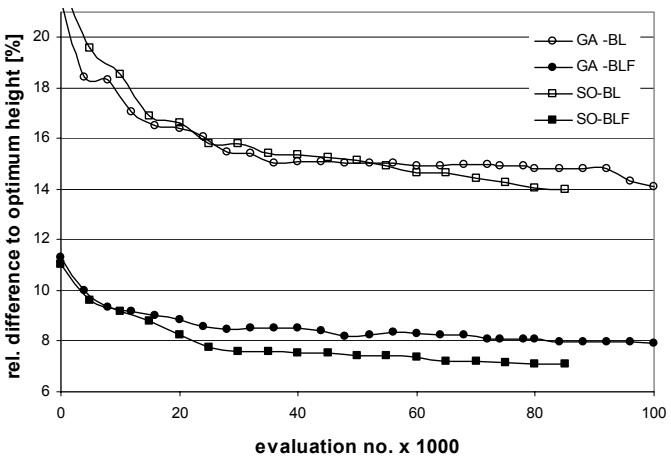


Figure 6.15: Relative difference to optimal height with GA and SO for BL- and BLF-algorithm for problem T4a

Table 6.10: Difference between SO and the GA for the rel. distance to optimal height [%]

	BL	BLLT	BLF
T1	-1	-1	0
T2	-1	0	0
T3	0	0	0
T4	0	-2	-1
T5	0	-1	-1
T6	2	-1	0
T7	0	0	0

Stochastic optimisation is also based on a population of solutions. The search progresses quickly in the beginning and its course is very close to one performed by a genetic algorithm (Figure 6.15). In most test cases examined the stochastic algorithm manages to "overtake" the genetic algorithm and converges on a solution that is between 1 and 2% better. With one exception within the 21 experiments consisting of three decoders applied to seven packing problem sets, the stochastic optimisation algorithm has been capable of finding better or equally good results when compared with the genetic algorithm (Table 6.10).

The comparison with simulated annealing shows the slower convergence rate of the simulated annealing algorithm (Figure 6.16). At the stage where the search of the stochastic optimisation algorithms terminates, its solution quality is better than simulated annealing with the BLF-decoder. As the simulated annealing process runs longer, it can find solutions that outperform the ones achieved by the other meta-heuristics. The complete search process for the two methods is illustrated in Figure F.3 (Appendix F).

One of the features the stochastic optimisation algorithm has in common with simulated annealing is that the search only progresses one state at time. Although a population of solutions is maintained, the neighbourhood operator is only applied to a single solution. The manipulated solution is accepted with a certain probability, also allowing uphill moves as in simulated annealing.

Towards the end of the search algorithm, when the packing height is already very low, a neighbourhood change can result in layouts of very low quality. It is very unlikely that such a step will be accepted by any of the meta-heuristic methods. If the neighbourhood change results in a minor deterioration of the layout, this move may remain part of the search process either in form of the next generation (GA) or the population (SO). In this respect the two meta-heuristics operate in a similar way.

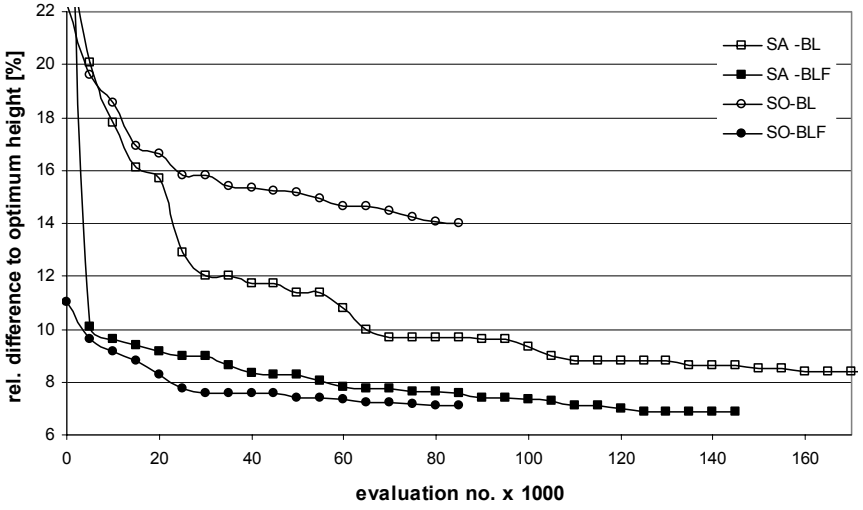


Figure 6.16: Comparison between SA and SO for BL- and BLF-algorithm for problem T4a

In simulated annealing the subsequent manipulation operation will be applied to the same solution that was accepted in the previous step. In this way simulated annealing differs from the other two meta-heuristics (i.e. GA and SO), which keep the solution, but which do not necessarily operate on it again. Concerning the evolutionary algorithms, the probability is very low that this particular solution is changed during the following mutation operation (3%) provided it is selected. In the case of the stochastic optimisation the solution may remain in the population, and not be elaborated for a certain time. Since the simulated annealing algorithm only operates on one solution, any (inferior) solution accepted will be further manipulated in the next step. Obviously, this feature is beneficial in the application of packing problems, since a layout of slightly lower quality may be transferred into a better solution through a chain of small changes involving one rectangle at a time. This might explain why simulated annealing manages to outperform genetic algorithms and stochastic optimisation for every single test problem and decoder combination.

6.4.3 Meta-Heuristics and Local Search

Hill-climbing has been applied to the same set of test problems in combination with the four heuristic decoders. Concerning the sliding techniques, it performs better for small problems. The relative distance to the optimal packing height becomes larger with increasing problem size (Appendix F, Table F.5). For the BLF-decoder the opposite trend has been found, where the best results are achieved for the larger problems. The relative performance regarding the problem size and the decoder type is similar to the combination with meta-heuristics. Unlike in the case of the meta-heuristics, the combination with the BLLT-decoder performs worst in every problem category apart from T1.

For most problems, hill-climbing does not reach the packing height obtained by random search (Appendix F, Table F.3 and Table F.5). The search process conducted by the hill-climbing technique only

allows exploration of a limited area of the search space, whereas random search can explore it completely. The difference in the outcome of both methods shows that the solution space is very "uneven" and contains a large number of local minima, which can easily trap the hill-climbing search. A random walk through the solution space is more successful, as a large number of solutions can be explored. Although hill-climbing is able to find the best solution within a certain (limited) neighbourhood, the solution quality obtained by random search is not only better in the beginning, but usually throughout the whole search process (Figure 6.17).

The success of the random search depends on the number of iterations. If this number is large, as in the case of Table F.3 (Appendix F), where it is based on 100,000 iterations, random search finds better solutions than hill-climbing. If only a small number of input sequences are used, hill-climbing performs better for small problems as the comparison with the heuristic methods using random and height-sorted input sequences shows (Appendix Table F.5). In this case hill-climbing has an advantage over random sampling, since it can explore the neighbourhood in order to reach the local minimum. Only a large number of trials allow random search to outperform hill-climbing. Problems with larger solution spaces are solved better by the simple heuristics using random or sorted input (section 6.3.5).

Figure 6.17 illustrates the performance of hill-climbing in comparison with several meta-heuristic methods. The meta-heuristic methods clearly outperform hill-climbing as well as random search from the beginning of the search process. Whereas hill-climbing only explores the enormous search space locally, meta-heuristic methods can exploit the space more effectively and concentrate on promising areas.

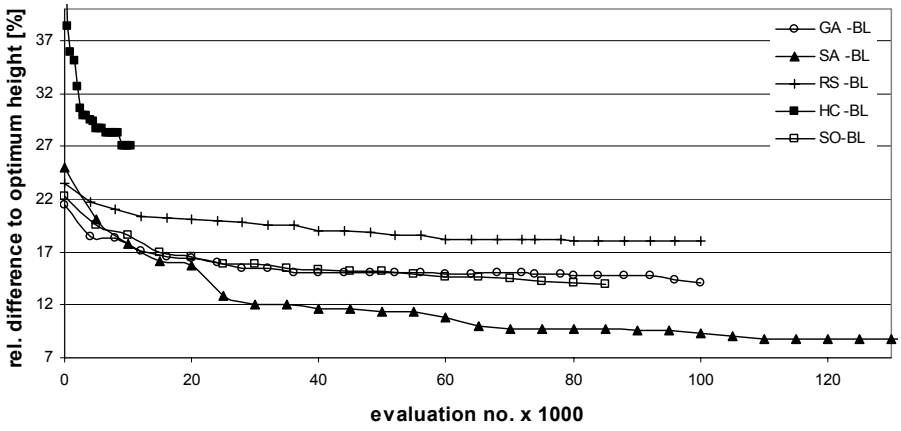


Figure 6.17: Comparison between HC, RS and meta-heuristics for BL-algorithm for problem T4a

Concerning the heuristic decoders, the BLF-routine has achieved the best results. The solution quality found for larger problems is very close to the outcome of the genetic algorithm (Table 6.11). The fact that the difference for problem category T7 is only 1%, shows that the decoding technique has a large influence on the outcome of the hybrid algorithm. This difference is larger for the sliding techniques, which illustrates the capability of genetic algorithms to escape local minima. In the case of the BLF-

decoder, a large number of sequences result in the same packing height. This leaves fewer possibilities to the genetic algorithm to exploit the search space through the intelligent use of recombination and evolutionary operators.

Table 6.11: Difference between HC and GA for the relative distance to optimal height [%]

	BL	BLLT	BLF	BLD
T1	20	17	8	17
T2	15	15	5	15
T3	15	24	5	18
T4	12	15	2	12
T5	12	17	1	12
T6	14	19	1	11
T7	9	20	1	6

Looking at the results stated in Table F.3 and Table F.4 (Appendix F), it can be seen that the meta-heuristic methods outperform the hill-climbing algorithm due to their ability to escape from local minima. Hence meta-heuristics offer a clear advantage over the local search algorithm in that respect. Since the hill-climbing algorithm terminates in a local minimum its run time is shorter than the one using meta-heuristics (Appendix F, Table F.7 and Table F.8).

6.4.4 Overall Comparison and Summary

The most successful search strategies in the beginning of the search process are genetic algorithms (Figure 6.17). Solutions found by the genetic algorithm improve rapidly over the number of evaluations and only get outperformed by the simulated annealing algorithm towards the end of the search (Figure 6.13). Hence the technique which genetic algorithms use to explore the space is initially more successful than the one applied by simulated annealing. One of the differences between the algorithms is that a cross-over operator is used in the genetic algorithm to manipulate the current best solutions. This obviously creates larger changes in the sequences than the neighbourhood operator used in simulated annealing and hence could explain the rapid progress the genetic algorithm makes in the beginning.

Comparisons with naïve evolution algorithm show that both evolutionary methods are equally successful and the difference during the search is only small (Appendix F, Figure F.3). Hence the cross-over operator used in this implementation is not the reason for the better exploration of the search space in the beginning of the search process. If cross-over was the main contributor, then the difference between the genetic algorithm and naïve evolution would be higher. The mutation operator implemented is obviously sufficient for this task. Unlike the simulated annealing method the evolutionary techniques work on a population of solutions, which they explore simultaneously. Whereas simulated annealing operates only on one solution at a time, the recombination method in evolutionary algorithms guarantees that the most

successful solutions are utilised in the following generation. Hence, they allow exploration of the solution space in parallel. The genetic algorithm is outperformed by the simulated annealing technique towards the end of the search process, when the population has converged. Only the simulated annealing technique, where solutions of inferior quality can be accepted over a series of moves, can then lead the search into promising regions. Figure 6.18 shows the best layouts that have been obtained with genetic algorithms and simulated annealing in combination with the BL- and the BLF-routine.

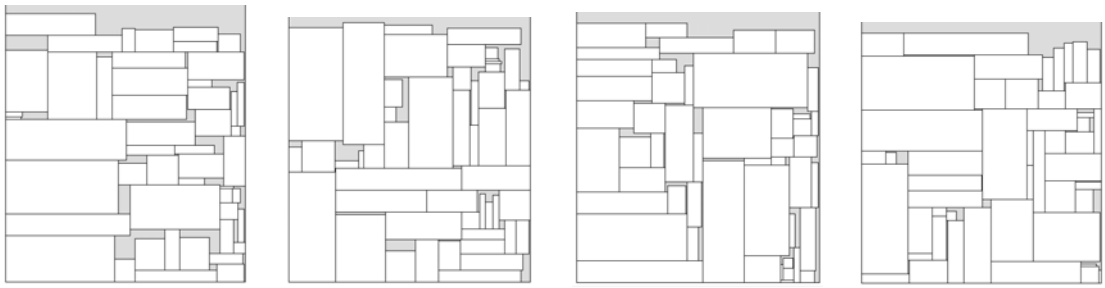


Figure 6.18: Best layouts for the following hybrids: GA+BL, SA+BL, GA+BLF and SA+BLF

Simulated annealing either only changes the rotation of one element at a time or the position of two elements in the permutation. Due to this manipulation technique the search process is slow at the beginning and even random input sequences result in better layouts. Random search is quickly outperformed, when the simulated annealing algorithm starts exploiting promising areas in the solution space and finds solutions which would not have been found on a random basis.

Hill-climbing is outperformed by random search, especially for the smaller problems. The packing heights achieved by the evolutionary algorithms are very similar and outperform random search and hill-climbing. Simulated annealing yields the best results in each problem category.

The results analysed in section 6.3 show that the combination between the meta-heuristics and the BLF packing routine achieve the better outcomes compared to the combinations with the decoders based on a sliding technique. The difference is higher for the larger problems. Since the packing heights achieved with the BLF-routine on its own are already very close to the optimal height (less than 8% from the optimum on average), the meta-heuristic cannot improve the performance of the heuristic as much as in the case of the other decoders. In other words by using the 'poorer' decoders, the meta-heuristic is needed to find a good input sequence, whereas applying the better BLF-heuristic good layouts are achieved with fewer iterations.

This questions the use of a hybrid combination between a meta-heuristic and a 'poor' decoder for this type of packing problem. In order to achieve high quality layouts it may often be sufficient to apply the BLF-routine over a small number of iterations. This is especially true for larger problems (>29 items, T3), where the hybrid methods only manage to improve the heuristic solution by less than 2%, but at a very high computational cost. For smaller problems on the other hand, where the execution times are

acceptable, the meta-heuristic with the BLF-rule outperforms the simple heuristic by about 7%. In this case the application of a meta-heuristic algorithm offers advantages. Summarising, the hybrid approach with the 'poorer' decoders, which are more efficient in terms of computation time, cannot be justified, since it is easily outperformed by the BLF-heuristic on large problems.

An implementation of the BLF-algorithm, which has a lower computational complexity, is of great benefit for the hybrid approach using the BLF-rule. Chazelle (1983) developed an implementation of this algorithm that has a complexity of $O(N^2)$. Using a more time efficient implementation the execution times of both heuristic packing rules become comparable, and will only differ by a factor rather than an order of magnitude. This means the hybrid algorithms using the sliding techniques lose their major advantage over the ones using the BLF-decoder.

6.5 Influence of Generic Design Variables on the Performance of the Hybrid Genetic Algorithm

In this work the main focus has been put on genetic algorithms. The influence of the generic and problem-specific design variables on the final packing height has been studied for the combination with the BL- and the BLF-algorithm. For this experiment the design variables have been modified according to Table 5.28 changing one parameter at a time. In order to analyse if a parameter has an effect on the outcome within a certain range the Friedman Test is applied (Noether, 1990). It has been examined if a possible effect on the outcome depends on the problem size using the test problems of the categories T2 to T5 whose item number ranges between 25 and 73 (section 5.3.3.1).

Population Size

Firstly, the influence of the population size has been examined using five values in the range of 26 and 200. Each genetic algorithm using a different population size is referred to as a 'method' in the Friedman Test. In order to compare the various methods, the sum of the ranks for the five problems of a problem category are calculated for each method. Table F.11 and Table F.12 in Appendix F show the rank sums for problem categories T2 to T5 for the combination with the BL- and the BLF-decoder. The rank sums are smaller for the larger population sizes. This indicates that larger population sizes generate better outcomes resulting in a smaller rank sum. In order to verify if there is statistical support for this conclusion, the Friedman Test has been carried out. In this case the P-value associated with the test result is less than 0.05. The P-values stated in Table F.11 and Table F.12 are below this limit for both decoders in all test categories.

This demonstrates that the population size influences the outcome when using genetic algorithms over the parameter range tested. The larger population sizes result in a better outcome. In order to see how large the improvement in packing height is in absolute terms, the relative differences to the optimal height for

the standard size of 100 and the largest size tested i.e. 200 are listed in Table F.13 (Appendix F). The final packing heights are up to 1% closer to the optimum than the ones obtained with the standard configuration. The improvement over the standard population size is higher for the larger problems (i.e. T5). With increasing problem size the search space increases tremendously according to combinatorial laws. A larger population size will explore a higher number of solutions. The improvement of less than 1% for a population size may not justify the higher computation time. The decision about the population size depends on the simulation time and solution quality desired for a certain packing application.

Cross-over and Mutation Rate

The following two experiments have been carried out in order to examine the influence of cross-over and mutation rate. Six methods have been tried with respect to the cross-over rate changing its values between 0 and 100%. The rank sums for the combination with the BL-decoder are stated in Table F.14 (Appendix F). The lowest rank sums are obtained for values around 60%. The Friedman Test failed to give any statistical evidence that any of the six methods performs best. However, it can seen that a cross-over rate of 0% performs worst in all four problem categories.

The cross-over rate certainly has an influence on the final outcome, but does not favour a certain value in this experiment. Table F.14 shows that appropriate values lie between 60 and 80%. For the BLF-decoder, the Friedman Test could prove an influence of the cross-over rate for all four problem sizes (Table F.15). The best outcomes have been obtained for a cross-over rate of 100%. This is probably due to the underlying packing technique of this routine where many different sequences can result in the same layout. A higher cross-over rate generates a large number of new solutions for the next generation and allows a better exploration of the search space.

A similar affect has been analysed for the influence of the mutation rate in combination with the BLF-decoder. Table F.17 (Appendix F) shows that lowest rank sums are obtained for the highest mutation rates with one exception, which is the smallest problem size tested (i.e. T2). In the case of the BL-decoder the affect of mutation rate on the outcome has been statistically proven by the Friedman Test. The preferred values lie between 1 and 3% (Table F.16, Appendix F).

Problem-Specific Design Variables

The experiments concerning problem-specific design variables of genetic algorithms studied different types of cross-over, mutation and selection. As Table F.18 to Table F.21 (Appendix F) show none of the cross-over and mutation types result in a distinct outcome of the genetic algorithms, which can be statistically proven. Rank selection has been found to be better once in combination with the BL-decoder and twice for the BLF-routine (Table F.22 and Table F.23). The results do not allow any further conclusions regarding the problem size.

6.6 Comparison with Approaches in Literature

Hybrid combinations between meta-heuristics and a heuristic packing rule as investigated in this study not only achieve high quality layouts. Their main advantage lies in simplicity of the implementation. The meta-heuristic search is hybridised with a heuristic algorithm and acts as tuner of the packing routine. The representation as a sequencing problem allows the use of well-known manipulation techniques for the search space, e.g. order-based cross-over operators, rather than developing problem-specific operators that can only be used in one specific context.

On the other side it can be argued that the geometric information concerning the layout is hidden in the heuristic decoder. Hence it cannot be exploited by meta-heuristic search processes to the same extent as if a representation was used that includes more geometric information in the definition of the neighbourhood structure and chromosomes respectively. In order to evaluate the hybrid methods developed in this work, their performance is compared with meta-heuristic approaches by other researchers using the same benchmark problems. A list of the test problems used for this investigation can be obtained from section 5.3.1.1. The principles of the approaches referred to in the sequel have been discussed in greater detail in section 4.3. The combination with BLF-decoder has been found to achieve the best solutions among the various decoding routines (section 6.3) and has been used in the following comparison. In order to evaluate the contribution of the intelligent search process to the final packing height, the outcome of the random search and the simple BLF-heuristic applied to 1000 random and sorted input sequences is also stated.

Hybrid Algorithms

The majority of literature concentrates on hybrid algorithms, where a genetic algorithm or another meta-heuristic method is combined with a heuristic placement routine (section 4.3.2). In this two-stage approach a genetic algorithm is used to determine the sequence, in which the items are to be packed. A second algorithm is then needed, which describes how this sequence is allocated onto the object.

Jakobs' approach (1996) is a hybrid genetic algorithm, which applies the BL-heuristic as a decoder. The difference in the outcome for the two test problems J1 and J2 is due to the decoding algorithm as the BLF-heuristic is capable of filling enclosed areas in the layout (Table 6.12).

Table 6.12: Packing heights obtained with GA, SA, RS and simple heuristic of benchmark problems from literature [units]

	J1	J2	D1	D2	D3	D4	Kendall
problem size	25	50	31	21	37	37	13
literature	17.5	17.3	48.0	42.0	118.0	164.8	168.1
GA +BLF	16.0	16.0	48.1	41.4	117.3	167.5	145.4
GA + BLF (seeded)	16.0	16.0	47.0	41.0	115.9	166.0	143.8
SA +BLF	16.0	16.0	47.4	40.0	114.3	163.9	145.6
RS + BLF	16.0	16.0	48.3	41.3	118.1	168.4	147.0
BLF-heuristic	16.0	16.0	47.0	41.0	116.0	166.0	148.0
best sorting	DW, DRA, DRP	DW, DRA, DRP	DW, DRA, DRP	DW	DW	DRA, DRP	DRA, DRP

The two test problems created by Jakobs were used by several researchers to test the performance of their algorithms. Liu and Teng (1999) used the Jakobs' BL-decoder as a basis for the development of the improved BL-heuristic, which is referred to in this work as BLLT-routine (section 5.5.4.2). As already indicated in section 6.3, the hybrid combination with the BLLT-algorithm outperforms Jakobs' approach. It also achieves better heights for the test cases J1 and J2 (Table 6.13). One of the most recent publications in this area (Leung et al., 1999) also refers to the hybrid approach by Jakobs. The heuristic decoder developed by Leung et al. (1999) belongs to the class of the bottom-left routines and allows keeping track of all empty areas in the layout. Unfortunately, the authors did not publish their results in absolute terms. They reported that their decoding routine manages to achieve better layouts than the meta-heuristics in combination with the BL-routine.

Table 6.13: Packing heights for benchmark problems J1 and J2 obtained with various GA approaches from the literature [units]

problem	N	Jakobs (1996)	Liu and Teng (1999)
J1	25	17.48	16
J2	50	17.28	16

Dagli and Poshyanonda (1997) developed an approach involving a hybrid between genetic algorithm and a neural network. The genetic algorithm generates an input sequence for the placement algorithm, which is based on a sliding method combined with an artificial neural network. This approach achieves packing densities between 95 and 97%. Applying the meta-heuristics in combination with the BLF-routine achieves densities between 95 and 96%. These figures have been converted into packing heights in Table 6.12 for the problem D4. The genetic algorithms do not outperform Dagli and Poshyanonda's method, but achieve a packing height that differs less than 2% from the best result stated in the literature. The test problem D4, however, is very special in the sense that the ratio between the width and the height of the object is very large. Even simple heuristics as the height- and width-sorted BLF-routines achieve a

packing density of 94%. This packing density corresponds to packing heights that differ less than 2% from the heights achieved by Dagli and Poshyanonda. Hence, the higher effort of the implementation in Dagli and Poshyanonda's approach is not reflected in a much better outcome and can be only justified if the solution quality is the main issue in the packing application.

The genetic algorithm developed by Burke and Kendall (1999) was originally intended for irregular problems. The sequence of items is placed with the aid of the No Fit Polygon (Albano and Sappupo, 1980) using local search to determine best position of the item. The authors also applied this technique to rectangular strip packing for test purposes. It has therefore been included in this comparison. Since the approach by Burke and Kendall is aimed at irregular items, it is outperformed by the simple heuristic as well as the meta-heuristics (Table 6.12). In one out of 10 runs the seeded genetic algorithm has found an optimal solution.

Hybrid Algorithms Using Additional Layout Information

A second category of solution approaches with genetic algorithms aims at incorporating some of the layout information into the data structure of the genetic algorithm. However, some additional rules are still needed to fix the position in the layout. The genetic algorithm by Kröger et al. (1991) is based on a graph structure to encode the problem, which includes some geometric information in the data structure. This representation fixes one dimension of the position of an item in the layout. The second dimension is determined by the bottom-left condition. Since the data sets used in the experiments are not published only an indirect comparison is possible through the BL-heuristic. The authors use the BL-heuristic to evaluate their approach. For problems consisting between 25 and 65 rectangles, the genetic algorithm achieves packing heights which are between 1 and 7% better than those obtained by the BL-heuristic (Appendix F, Table F.10). The family of hybrid genetic algorithms developed in this work achieves packing heights that are between 7 and 16% lower than those of the BL-heuristic for similar problem sizes (Appendix F, Table F.9). In particular, the hybrid combination with the BLF-heuristic performs very well and is on average about 15% better than the BL-heuristic.

Algorithms Operating on the 2D Layout

A third group of genetic algorithms attempts to solve the problem in 2D space. The evolutionary search developed by Ratanapan and Dagli (1997a, 1998) is different from the other approaches described so far, since it does not make use of a data structure to represent the problem. The items are represented as 2D pieces with their true geometric dimensions. Comparisons show that this does not necessarily generate better layouts. The packing densities achieved by Ratanapan and Dagli's technique for three small and medium-sized problems are about 92%. In order to facilitate the comparison with other methods, the corresponding packing heights are stated in Table 6.12 for the problems D1, D2 and D3.

Using the same data sets, the hybrid meta-heuristic techniques in combination with either of the packing routines were able to outperform this result (Table 6.12). The packing densities range between 92 and

98%, with the simulated annealing obtaining the best outcomes. For the smaller problem i.e. D2, the simulated annealing algorithm even found an optimal solution in 10 out of 10 runs. Also the combinations with the other heuristic decoders usually succeeded in finding an optimal solution in 50% of the trials. This shows that this test problem is simple compared to the others, since it contains a large number of identical rectangles. Unfortunately, Ratanapan and Dagli did not state the computational effort for the rearrangements in the layout after application of the mutation operators, which would allow comparison of the two distinct methods on the basis of computation time.

Summary

The test problems used for this comparison are only small to medium-sized compared to the test problems used in the previous sections. The evaluation of the approaches developed by various researchers is only valid for problems up to 40 items. With one exception (i.e. D4), the hybrid genetic algorithm in combination with the BLF-decoder performs better or at least equally well as the meta-heuristic methods found in the literature using the above benchmark problems. In particular, it outperforms the approaches which do not use the hybrid technique, but implement the layout information completely or partially in the genetic algorithm, which is the case in Dagli's and Koerger's work.

Concerning the hybrid approaches the hybrid genetic algorithm, which uses the more sophisticated decoder, i.e. the BLF-routine, achieves better layouts than Jakobs' method. It also outperforms the improved technique developed by Liu and Teng using different test problems (section 6.3 and Table F.3, Appendix F). This is due to the packing technique of the BLF-heuristic, which fills enclosed empty areas in the layout unlike the group of sliding techniques.

The comparison with the outcome of the random search and the simple BLF-heuristic shows that in many cases (e.g. J1, J2, D1 and D2) it is not necessary to use intelligent search methods, since usually the simple heuristic in connection with sorted input, can achieve equal or even better results. This aspect has so far been neglected by the research focussed on the development of meta-heuristic techniques for rectangular packing problems. For most cases an improvement over the outcome of the GA techniques or the simple heuristic can be obtained with simulated annealing. This again raises the question of the computational effort of meta-heuristic techniques compared to more straightforward approaches (section 6.4.1).

6.7 Comparison with Commercial Nesting Software

Approaching packing problems with meta-heuristic algorithms has so far only been a research topic. Industrial packing tasks are usually solved manually or by use of problem-specific algorithms. In recent years, the number of commercial nesting software especially tailored towards industrial needs has grown tremendously (Appendix C, Table C.3). A commercial package has therefore also been included in the

evaluation of meta-heuristic solution approaches using Nestlib version 9.0 by Geometric Software Services Co. Ltd.

The benchmark problems published in the literature as well as some of the test problems created with the problem generators have been nested using the nesting package. Nestlib can be set to evaluate a maximum of 30 trials, the best of which is stated in Table 6.14 and Table 6.15. It can be seen that the meta-heuristic approaches are more successful for smaller problems (i.e. T1a to T4a). The same is true for the benchmark problems from the literature, which belong to the same problem category in terms of problem size. Even simple heuristics as the BLF-routine perform equally well or better than the commercial packing software with one exception i.e. T2a. For the larger test problems i.e. T5a to T7a Nestlib achieves packing heights that are up to 3% closer to the optimal solution than those obtained with the meta-heuristics. The difference between the Nestlib outcome and the simple BLF-heuristic is smaller.

Table 6.14: Relative difference to optimal packing heights obtained with Nestlib, meta-heuristics, RS and simple heuristics for benchmark problems [%]

	T1a	**T2a**	**T3a**	**T4a**	**T5a**	**T6a**	**T7a**
problem size	17	25	29	49	73	97	199
Nestlib	16.3	8.7	9.9	5.2	4.8	3.4	2.0
GA +BLF	6.2	6.0	7.4	7.3	7.5	6.5	5.1
GA + BLF (seeded)	5.4	5.2	6.3	5.2	5.0	2.9	2.4
SA + BLF	6.2	4.6	5.9	4.7	5.1	4.2	4.0
RS + BLF	8.0	7.7	8.1	7.8	8.4	7.4	6.4
BLF-heuristic	9.1	9.5	7.0	5.2	6.5	2.9	2.4
best sorting	DRA	DRA	DRP	DH	DW, DRA, DRP,	DRA, DRP	DH

Table 6.15: Packing heights obtained with GA, SA and Nestlib for benchmark problems from literature [units]

	J1	**J2**	**D1**	**D2**	**D3**	**D4**	**Kendall**
problem size	25	50	31	21	37	37	13
literature	17.5	17.3	48.0	42.0	118.0	164.8	168.1
Nestlib	17.0	17.0	47.0	45.0	119.0	191.0	168.0
GA +BLF	16.0	16.0	48.1	41.4	117.3	167.5	145.4
GA +BLF (seeded)	16.0	16.0	47.0	41.0	115.9	166.0	143.8
SA +BLF	16.0	16.0	47.4	40.0	114.3	163.9	145.6

This investigation shows that commercial packing software offers an advantage for larger packing problems (>70 items), since it does not only generate better layouts than the meta-heuristics, but also outperforms the extremely efficient simple BLF-heuristic. A further advantage, which probably is even more important in this context, is the computational effort. Whereas the meta-heuristic search processes

have run times in the range of hours for the large problems, a better solution can be obtained almost instantly with the commercial tool. Since timing is an important issue in industrial applications, the commercial packing software certainly has its justification for large packing tasks. For small tasks, a trade-off between solution quality and computational effort is necessary.

6.8 Conclusions

A family of hybrid algorithms for the rectangle packing problem have been implemented consisting of a combination of a meta-heuristic algorithm (GA, NE, SA and SO) and heuristic packing routine to allocate the items on the object. The heuristic packing routines generate the layouts in a bottom-left justified manner. Whereas three of the techniques (BL, BLLT and BLD) are based on a sliding principle, the fourth one (BLF) is able to fill enclosures in the layouts, but at higher computational cost. The meta-heuristic hybrid algorithms have been tested on a number of packing problems including benchmark problems from the literature. For the evaluation of the hybrid algorithms their performance has been compared with heuristic search methods and other approaches from the literature.

In terms of solution quality the meta-heuristic algorithms outperform the heuristic packing routines and the hill-climbing approach with simulated annealing performing best. The combinations with the more sophisticated heuristic (BLF) achieve better layouts than the ones using the decoder based on the sliding technique. The improved BL-heuristic suggested by Liu and Teng (1999), which has been referred to in this study as BLLT-algorithm, has been shown to perform better than the BL-routine compared to the simple heuristics as well as to the combination with the meta-heuristics. The BLD-algorithm developed to transfer some of the adjacency information in the permutation to the layout, has not met these expectations and is outperformed by the BLLT- and the BLF-routine.

For industrial problems the question, which technique in combination with the BLF-routine to choose, is a trade-off between material cost and simulation cost. In this study simulated annealing has achieved the best layout quality over all problem categories. Its computation time becomes larger with increasing problem size. The genetic algorithms are better in terms of computational effort and yield results, which are slightly worse than ones obtained by the simulated annealing. Hence if the time for solving a packing task is limited, genetic algorithms are appropriate. For very small time margins, only heuristic packing algorithms will be able to meet this criterion.

Although meta-heuristics generate very good layouts and outperform simple heuristics in most problem categories, the computation time is limited in industrial applications. In this case a trade-off has to be made between solution quality and computational effort. Table 6.16 summarises which methods are most appropriate under given time constraints. It is assumed that a solution approach is executed several times, before reliable results can be obtained. The time categories in Table 6.16 refer to the elapsed time of 10 runs concerning the meta-heuristic methods and to 1000 runs for the simple BLF-heuristic. The

information in Table 6.16 has to be seen in connection with the computational power of the simulation system. With the processing power constantly increasing, it will be possible to apply meta-heuristics efficiently to even larger packing tasks in the future.

Table 6.16: Method for best results within specified time limit (hybrids are with BLF decoder)

problem	N	< 1min	< 10min	< 1h	< 10h
T1	17	HC	GA*	GA*	GA*
T2	25	BLF	GA*	GA*, SA	GA*, SA
T3	29	BLF	GA*	GA*, SA	GA*, SA
T4	49	BLF	BLF	GA*	SA
T5	73	BLF	BLF	GA*	GA*, SA
T6	97	BLF	BLF	BLF	BLF, GA^+, SA
T7	199	BLF	BLF	BLF	BLF, GA^+

GA*: GA has been seeded with best sorted sequence
GA^+: GA has been seeded with best sorted sequence; final GA outcome is identical to seed

Since the performance difference between the hybrid methods using the decoders based on the sliding technique and the ones using the BLF-decoder is only due to the more sophisticated packing technique, the decoder has a larger effect on the outcome of the hybrid technique than the meta-heuristic technique itself. This seems to suggest approaches, where more layout-specific knowledge is incorporated in the meta-heuristic, rather than the decoder. However, representation schemes studied by other researchers that use more layout information did not achieve higher packing densities than hybrid techniques. The combination between the genetic algorithm and the BLF-routine also outperforms most of the other hybrid techniques in the literature.

Due to its efficiency in terms of solution quality the BLF-routine should be used as a benchmark test to analyse the performance of newly developed algorithms. It is not only simple to implement, but also has a low computation time compared to meta-heuristic and problem-specific methods. Therefore it can be easily applied as a relative measure for new packing algorithms, especially since it has outperformed several algorithms published in the literature.

The impact of the generic design variables on the outcome of the genetic algorithms has been investigated. In particular, an influence for the population size on the final outcome has been analysed. Further to that, the cross-over and mutation rates have an affect on the outcome the combination with the BLF-decoder. In this case, higher values result in better solution quality. Some influence has also been established for the BL-decoder. With respect to the problem-specific variables none of the different methods examined has proven to achieve significantly better layout quality. The decision about the population size depends on the industrial needs since the use of larger populations results in better layouts, but also higher computational effort.

Concerning the methodology, hybrid algorithms are well suited for industrial demands. The layouts achieved are of better quality than other meta-heuristic and heuristic techniques for a certain category of packing tasks, which is defined by the problem size and time to be dedicated to find a solution. For larger problems commercial packing tools have been shown to generate very efficient layouts in terms of solution quality as well as computational effort. The implementation of the hybrid algorithm is easier, as it is based on well-known techniques and does not require development of problem-specific algorithms. With heuristics already being applied in industry, the acceptance of research methods such as meta-heuristics certainly will be higher.

7. Genetic Algorithms and Irregular Packing

7.1 Introduction

Irregular packing tasks are encountered in many industrial applications (chapter 4). Industry as well as the research community have dedicated some effort towards the development of automated packing systems for irregular shapes (section 4.3.3). Furthermore, a range of problem-specific heuristic algorithms has been developed for many different industrial needs over the years. With increasing computational power researchers have started to concentrate on heuristic as well as meta-heuristic search principles to solve packing problems.

Due to encouraging results of other research in this area the main focus of this investigation has been on the application of genetic algorithms to irregular packing problems. Since the hybrid methods achieved a very good outcome in the case of rectangular packing this concept has also been applied to irregular packing tasks. A number of decoding algorithms have been developed for the combination with genetic algorithms. In order to establish the role of the cross-over operator in the search process, naïve evolution has been implemented in the same way as the genetic algorithms and used for the nesting tasks. The implementation of the two evolutionary techniques is described in section 5.7.3.4. The application of meta-heuristic techniques to 2D rectangular packing problems showed that simulated annealing outperformed the other search methods in most problem categories. Consequently, genetic algorithms have also been compared to simulated annealing in irregular packing. Hill-climbing and the downhill simplex method have been applied to establish possible advantages of meta-heuristic search over heuristic search in this area.

Since time can be one of the major issues in industrial applications the solution quality obtained with the various hybrid methods has been examined in the context of computation time. The investigation on the heuristic packing algorithms has considered possible performance enhancement through pre-ordered input, which has proven to be very successful in the case of rectangular packing.

In order to evaluate the quality of the meta-heuristic hybrid approaches developed in this work, their performance has been compared to a number of meta-heuristic and heuristic methods from the literature using a range of benchmark problems. In particular, over the last few years a variety of commercial fully automatic nesting packages have appeared on the market. Two of them have been included in these comparisons.

7.2 Comparison of the Packing Algorithms

In section 5.6.4 a set of simple heuristic algorithms is introduced which have been developed for the decoding stage of the hybrid algorithms. Four out of the six packing routines consist of two stages (i.e. BLi, BLLTi, BLDi and BLFi). During the first stage the enclosing rectangle is placed in the partial layout according to one of the rectangular placement rules described in section 5.5.4. The second stage iterates the item in the partial layout. The other two techniques operate on the polygon moving it to the bottom-left side of the object in a series of movements. In order to evaluate the suitability of these six routines for the packing of polygons, their performance has been tested in the following with respect to solution quality and computation time using a number of different sized packing tasks ranging from 15 to 75 items. The details of the test problems are summarised in Appendix B.2.

Solution Quality

The best packing height, which has been achieved by the various packing techniques after 1000 runs, is illustrated in Figure 7.1. The BLi- and BLLTi-routines show a similar performance. In most problems the BLi-algorithm works slightly better. Table 7.1 shows the relative differences to the result obtained with the BLi-technique. The negative values indicate that a certain technique achieves a lower packing height than the BLi-routine. The BLDi-routine, which also belongs to the two-stage sliding techniques, performs worse than the BLi-technique in all test cases.

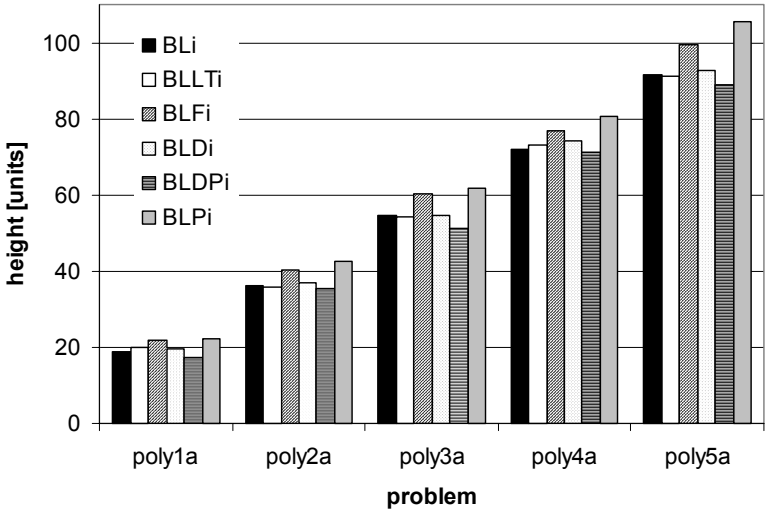


Figure 7.1: Packing heights obtained with the simple placement routines for random input

Among all the routines based on the enclosing rectangle (BLi, BLLTi, BLDi and BLFi), the worst results have been obtained with the BLFi-routine. The outcome can differ by up to 16% from the heights achieved with the BLi-rule (Table 7.1). This difference becomes smaller with increasing problem size. As during the first stage of the BLFi-routine the complete layout is constructed on the basis of the rectangles, the movement steps operating on the polygons in the next step are restricted. Each polygon is iterated

only once into a stable position in the partial layout during the second stage. Gaps in the layout created during the movement of successive polygons could allow previously iterated polygons to move further down or left. The BLFi-algorithm is limited to one iteration step. This explains its poor results compared to the other two-stage techniques.

Table 7.1: Relative difference to BL-routine with respect to the packing height [%]

	BLLTi	**BLFi**	**BLDi**	**BLDPi**	**BLPi**		**BLLTi**	**BLFi**	**BLDi**	**BLDPi**	**BLPi**
poly1a	5.3	15.8	4.1	-7.9	17.6						
poly2a	-0.6	11.9	2.2	-1.8	18.5	**poly2b**	2.8	16.2	4.0	-7.2	8.6
poly3a	-0.6	10.4	0.1	-6.3	13.0	**poly3b**	-0.2	9.8	4.5	-9.5	10.4
poly4a	1.8	6.8	3.2	-0.8	12.2	**poly4b**	-0.8	8.5	1.4	-8.8	9.5
poly5a	-0.6	8.4	1.3	-3.0	15.2	**poly5b**	1.2	1.8	0.4	-9.4	18.9

In order to investigate to what extent the layout generation is restricted by the use of enclosing rectangles during the first sliding step, two routines have been developed which operate directly on the polygon. The first technique (BLPi) works in a similar way to the BL-technique for rectangles and places a polygon by a series of bottom-left movements. Compared to the other packing techniques this method has the worst performance. The relative difference to the BLi-rule is between 8% and 19%. The reason for this is the sliding principle. Since a polygon can only move in the horizontal and vertical directions it can get easily trapped in the partial layout when it collides with another polygon.

Although there might be sufficient space in the layout, the movement process stops if the polygons cannot be moved further to the left or the bottom. This is illustrated in Figure 7.2, which shows a layout that has been generated with the BLPi-algorithm. The enclosed areas in the partial layout cannot be reached by the polygons, which are above, since a vertical movement is restricted by the collision with previously placed polygons.

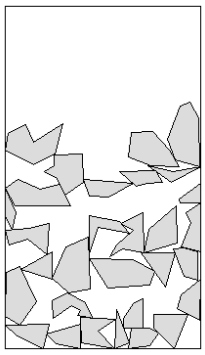


Figure 7.2: Layout generation with the BLPi-algorithm

In order to escape from such an interlocking position, the polygon would need to "reverse" before it can continue with further bottom-left moves. Since these additional movements will result in an increase of the iteration time, a reverse mechanism has not been included in this placement technique. Due to its poor performance BLPi-algorithm has not been considered in the hybrid combinations with the meta-heuristics discussed in section 7.3 and 7.4.

The second packing technique based on the sliding of the polygons rather than the enclosing rectangle, i.e. BLDPi, outperforms all other techniques. Instead of starting the sliding process at the top border of

the object, the polygons are placed into the partial layout in the same way as with the BLDi-algorithm, i.e. positioning it behind and above the proceeding polygon. Unlike for the BLDi-algorithm, a stable position is found translating the polygon rather than the enclosing rectangle in the consecutive sliding step. For some problems, the BLDPi-routine achieves nearly up to 10% better layouts than the BLi-technique (Table 7.1).

Solution Quality with Pre-Ordered Input

The investigation of the rectangular strip packing problem has shown that pre-ordering the input sequence can improve the layout quality achieved with the simple packing algorithms (section 6.2). In particular, sorting the rectangles by decreasing height or width has been successful for most problems. In order to examine to what extent pre-ordered input sequences could improve the packing height, a series of sorting rules have been tested in connection with the six packing techniques. Table 7.2 summarises the outcome obtained by sorting the enclosing rectangles of the polygons by decreasing height. The values in Table 7.2 show the relative difference in packing height between the height-sorted and the random input. For most test problems the height-sorted input sequences have resulted in a better outcome than the random input. In some cases other sorting rules outperform the height-sorted input (Appendix G, Table G.1 to Table G.6). In general, sorting by decreasing height has been found to perform best with most packing routines and for most test problems.

Table 7.2: Relative difference between height-sorted and random input for packing height [%]

	BLi	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**	**BLPi**
poly1a	0.8	-0.2	5.2	-9.0	2.6	-6.6
poly2a	-5.2	-2.9	7.4	-11.0	-5.5	-12.1
poly3a	-4.7	-3.5	-2.1	-6.5	-4.6	-8.7
poly4a	-5.3	-5.8	-0.1	-8.7	-7.1	-8.1
poly5a	-6.4	-4.9	1.0	-9.2	-7.7	-8.7

Influence of the Minimum Enclosing Rectangle Concept on the Solution Quality

In the above experiments the orientation of the polygons has been limited to intervals of 90°. Including reflection, a total of eight orientations of the polygon are possible. For the general test situation using random input, the sequence of the items and their orientation has been generated randomly. In the case of pre-ordered input, both the sequence and the orientation strings are generated randomly before the permutation of items in their respective orientation is sorted according to a certain ordering criteria. Two experiments have been conducted in order to investigate the influence of rotation.

In the first assessment the impact of the starting orientation of the polygons is considered conducting two series of measurements. The first test has used a random orientation of the polygons as a starting orientation. This orientation has been used as orientation '0' out of the eight orientations possible. For the

second test each polygon has been rotated into the position where the area of its enclosing rectangle is smallest. Table 7.3 summarises the relative difference between the first to the second test. The positive values indicate that the packing heights achieved with the use of the random orientation of the polygons are higher. In most cases rotating the items into the minimum enclosing rectangle position results in a better outcome. Depending on the packing method, the difference can be up to 15%.

For the BLDPi-routine the use of the minimum enclosing rectangle concept allows the highest improvements in terms of packing height. This is due to the packing principle of this routine. Rotating the polygons into the minimum position before nesting has a high impact on the solution quality. To a lesser extent this also influences the other packing methods. Depending on the problem type, the influence of the starting orientation generally improves the packing height for the problem sets that contain multiple copies of the polygons, i.e. poly1a to poly5a. For the test problems containing unique polygons usually the random start orientation works better with the exception of the BLDPi-routine.

Table 7.3: Relative difference in height between test with random start orientation and test using the minimum enclosing rectangle of the polygons as start orientation (for random input sequences) [%]

	BLi	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**		**BLi**	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**
poly1a	1.5	-3.8	-8.8	-1.6	15.2						
poly2a	6.8	5.3	-2.2	2.3	11.7	**poly2b**	-0.6	-4.4	-11.4	-4.2	5.1
poly3a	5.5	4.2	1.9	2.5	10.9	**poly3b**	1.9	0.1	-3.7	-3.8	15.0
poly4a	5.5	1.6	1.6	3.4	6.6	**poly4b**	-4.6	-2.2	-9.5	-3.2	7.7
poly5a	3.7	2.3	-2.0	4.1	8.8	**poly5b**	-5.8	-7.3	-3.4	-4.8	5.6

Influence of Rotation Interval on the Solution Quality

In a second series of experiments the impact of the rotation interval has been investigated. The outcome mentioned above has been obtained using a rotation interval of 90°. Smaller intervals introduce more flexibility into the packing process, since the polygons can occur in more orientations. Additional orientations of the polygon increase the search space but will make the search process more complex.

A series of measurements have been carried out with intervals of 90°, 45° and 22.5° allowing 8, 16 and 32 orientations of an item inclusively reflection. Table G.8 to Table G.11 (Appendix G) summarise the outcome of this experiment showing the relative difference to the packing height achieved with a 90° interval. The positive values indicate that the packing heights using the higher number of orientations are higher than the ones using an interval of 90°. In 4 out of 80 cases only, the smaller interval has resulted in a better outcome. In particular for the test problems containing multiple copies i.e. poly1a to poly5a, the smallest interval at 22.5° is worse than the 45° interval. Figure 7.3 shows the best layouts that have been obtained with the BLLTi-algorithm for three different rotation intervals using random input sequences.

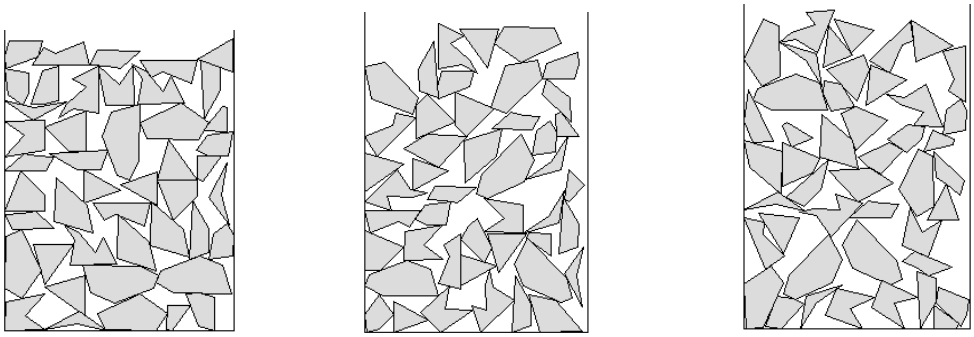


Figure 7.3: Best layout with rotation intervals of 90°, 45° and 22.5° for poly3a

A poorer outcome of smaller rotation intervals is certainly due to the increase in the search space. Despite the higher number of possible combinations, the packing heuristics have been executed over the same number of iterations in all three cases. Another reason for the poorer results is that the polygon can also occur in orientations that do not correspond to the minimum enclosing rectangle orientation. This is beneficial for the heuristic packing task as mentioned above.

The experiments have shown that rotating the irregular items into the position, which describes the smallest area of the enclosing rectangle, achieves denser layouts than using a random orientation. This pre-rotation step is only possible if the packing task does not contain any constraints concerning the orientation. Industrial problems are often restricted in their orientation like in the metal and textile industry. Rotation constraints are due to inhomogeneous material properties such as grain orientation or pattern of the fabric.

Allowing more flexibility does not necessarily improve the final layout quality. If the solution space becomes too large, the heuristic is not able to generate a good layout in spite of the additional freedom. The size of the search space has also got an influence on the outcome. This confirms the efficiency of the minimum rectangle concept. A rotation interval of 90° restricts the number of possible orientations to eight, all of which describe the polygon in its minimum enclosing rectangle position. This is not the case when the rotation interval is smaller and if the polygons can occur in other orientations as well. Whether the additional flexibility in terms of the orientation can assist the search process, has to be shown with the application of meta-heuristics since these methods have been designed for larger solution spaces.

Time Performance

The efficiency of the application of meta-heuristics particularly to industrial problems does not only depend on the solution quality, but also the computation time. Table G.7 (Appendix G) compares the elapsed time for the six packing techniques for 1000 runs. Whereas three techniques (BLi, BLLTi and BLDi) have similar time performance, the BLDPi-algorithm has the highest run time for all the tested problem sizes.

The BLi-, BLLTi- and BLDi-routines require similar time to place one item (Table 7.4) because they use the same underlying packing technique. After finding a stable position for the enclosing rectangle the polygon is iterated in the partial layout in the next step. The elapsed time for the BLFi-algorithm becomes much larger with increasing problem size compared to the other three techniques that use the enclosing rectangle. The BLFi-routine requires more time to place the rectangle into the partial layout because of its hole-filling feature as shown in section 6.2.

The other two techniques are not based on the enclosing rectangle. Whereas the BLDPi-routine has the highest run time of all packing routines, the BLPi-algorithm has the lowest one. In the case of the BLPi-algorithm, this is due to the collisions with other polygons that stop the sliding process at an early stage. That does not only explain the short elapsed times, but also the poor outcome in terms of the packing height obtained with this method. The BLDPi-algorithm needs more time to place an item because the sliding technique is based entirely on polygons. A higher number of distance calculations between polygons are performed compared to the rectangle-based routines, which only need to calculate the distance between the rectangles in the first stage. Since the computations involving polygons are more time intensive, the BLDPi-algorithm has the longest run times.

Table 7.4: Average elapsed time to place one item per run [ms]

	BLi	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**	**BLPi**
poly1a	2.1	2.3	2.4	2.3	3.9	0.5
poly2a	2.5	2.5	2.7	2.6	4.4	0.7
poly3a	2.7	2.8	3.0	2.8	4.6	0.7
poly4a	2.8	2.8	3.1	2.7	4.8	0.8
poly5a	2.7	2.8	3.3	2.8	4.8	0.9

Sample Layouts

Some of the simple packing heuristics tested in this section have obtained dense layouts. In particular pre-ordering the polygons by decreasing height or area has resulted in good outcomes. Figure 7.4 and Figure 7.5 illustrate the best layouts that have been achieved with the various packing algorithms. Whether the gain in performance of the BLDPi-routine in terms of layout quality can justify its application in the decoding stage of a meta-heuristic hybrid algorithm for irregular problems depends on the final packing heights achieved in comparison with other hybrid or heuristic methods. As in the case of rectangle strip packing a trade-off between solution quality and computation time will have to be made for industrial applications.

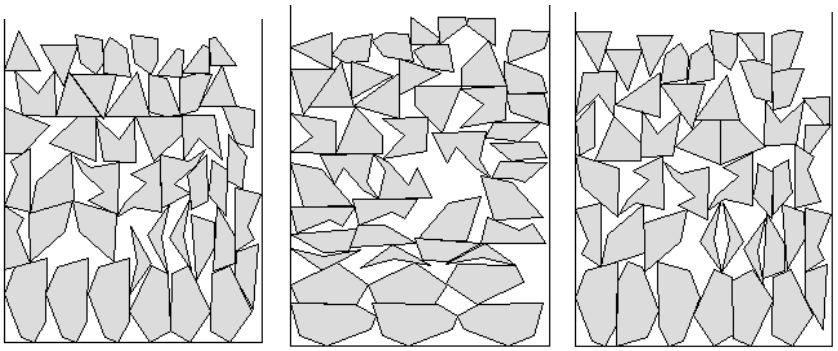


Figure 7.4: Best layouts for problem poly3a with BLi (DH), BLLTi (DW) and BLDi (DH)

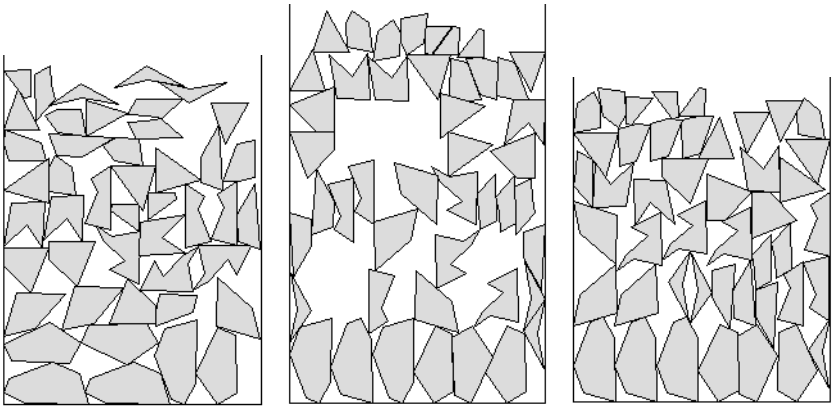


Figure 7.5: Best layouts for problem poly3a with BLFi (DA), BLPi (DH) and BLDPi (DH)

7.3 Comparison of the Evolutionary Approaches

The heuristic packing techniques developed for the decoding stage of the hybrid algorithms have been combined with the genetic algorithm as described in section 5.6.3.4. As already mentioned in the previous section, the BLPi-algorithm does not achieve high quality layouts. Since it has been outperformed by the other packing methods in every single test case (Appendix G, Table G.1 to Table G.6), the following investigation has concentrated on the other five decoders in context with the meta-heuristic as well as heuristic search algorithms. All the experiments summarised in section 7.3 and 7.4 use the minimum enclosing rectangle concept unless otherwise stated. In order to test the efficiency of the genetic algorithms their performance has been compared with random search and naïve evolution.

7.3.1 Genetic Algorithms versus Random Search

The efficiency of the search processes has been assessed by iterating the heuristic decoding algorithms over the same number of function evaluations as the genetic algorithms (i.e. 50,000). The relative difference in packing height of the random process to the outcome obtained with the genetic algorithms is

summarised in Table 7.5. The packing heights achieved with the genetic algorithm can be up to 9% better than the ones based on a random input. The relative difference gets smaller with increasing problem size. This means that the genetic algorithms become less efficient for larger problems and hence for larger search spaces. In order to facilitate the comparison between the different packing methods, the absolute packing heights are stated in Appendix G (Table G.12 and Table G.17).

Table 7.5: Relative difference between RS and GA with respect to the average packing height [%]

	BLi	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**		**BLi**	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**
poly1a	7.0	4.0	3.8	4.1	9.1						
poly2a	1.9	0.9	0.6	3.8	4.8	**poly2b**	3.0	3.2	3.0	1.6	5.5
poly3a	2.2	2.2	0.6	3.1	3.4	**poly3b**	2.9	0.9	0.5	1.8	4.4
poly4a	1.4	1.9	1.3	2.0	1.7	**poly4b**	1.3	-0.1	0.4	-0.8	1.9
poly5a	1.2	1.9	2.4	2.6	1.3	**poly5b**	2.6	1.3	1.2	0.9	2.9

The fact that the genetic algorithms outperform the random process for nearly every single test case shows that they can search the solution space in a more efficient manner. This task becomes more difficult for larger search spaces. Figure 7.6 illustrates the course of the genetic and the random search. Whereas the random process finds better solutions more rapidly at the beginning of the search, the genetic algorithm manages to outperform the random search during the first fifth of the total function evaluations. This is due to the fact that the genetic algorithm maintains a population of solutions, whereas the random search only explores the solution space and progresses more rapidly at the start. During the later stage of the search, the genetic algorithm is capable of elaborating good solutions found with the aid of the cross-over and mutation operators. The contribution of the two genetic operators to the search process is investigated in the next section. The search processes for the BLLTi- and BLDPi-decoder are illustrated in Figure G.1 (Appendix G).

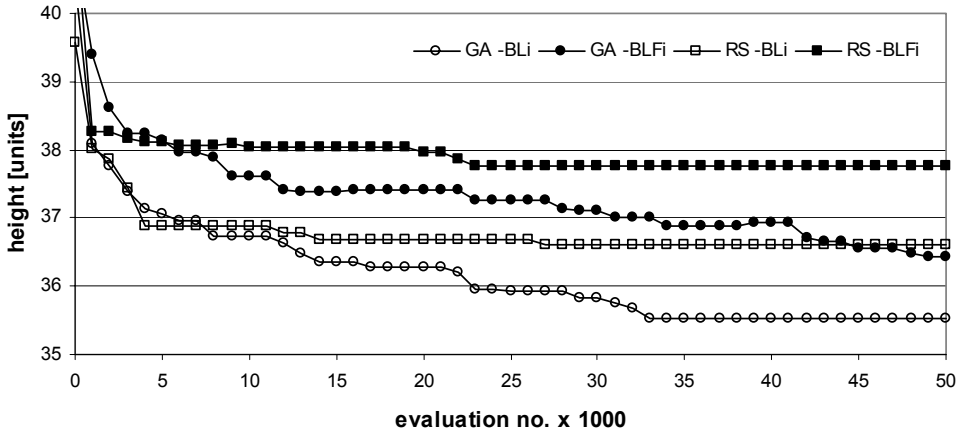


Figure 7.6: Comparison between GA and RS with BLi- and BLFi- decoder for problem poly2b

The random search process, which uses the same number of function evaluations as the genetic algorithm achieves better results than the simple heuristics applying only 1000 iterations. Since the performance of the random search lies between those two methods, some of the performance gain of the genetic search is simply due to the higher number of iterations compared to the simple packing heuristics.

7.3.2 Genetic Algorithms versus Naïve Evolution

The comparison between the genetic algorithms and the random search in the previous section has shown that the genetic search process works better than the random one. Features other than pure exploration of the solution space influence the outcome of the search process. One of the aspects that distinguish genetic algorithms from other meta-heuristics is the cross-over operator. In order to examine its contribution to the search process, naïve evolution has been implemented using the same parameters for the problem-specific and generic design variables (section 5.6.3.4). The comparison with the genetic search process shows that the outcome of the naïve evolution algorithm is slightly worse for most test problems. The relative difference in the packing heights is usually below 2% (Table 7.6). In some cases naïve evolution outperforms the genetic algorithms or performs equally well. The search process conducted using the five decoders is shown in Figure G.2 (Appendix G).

Table 7.6: Relative difference between GA and NE with respect to the average packing height [%]

	BLi	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**		**BLi**	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**
poly1a	0.9	1.8	4.1	0.7	4.0						
poly2a	0.1	-3.8	-4.2	1.2	-2.8	**poly2b**	1.7	0.1	-1.9	-3.2	3.8
poly3a	0.3	-0.3	-4.6	-2.1	-1.8	**poly3b**	-0.8	0.1	-3.2	-1.4	1.0
poly4a	-1.7	-2.6	-1.2	-1.8	0.5	**poly4b**	-1.8	-1.7	-3.1	-2.0	-1.1
poly5a	-2.8	-1.4	-3.1	-0.8	-0.2	**poly5b**	0.7	-1.6	-3.0	-2.1	1.1

The rather small performance difference together with the fact that naïve evolution sometimes outperforms genetic algorithms shows that the cross-over operator used in this implementation is not the main contributor to the success of the search process. In order to illustrate how similar both meta-heuristics operate, their progress over the number of function evaluations is shown in Figure 7.7. Generally, genetic algorithms make faster progress at the beginning of the search. Starting off with a random population the cross-over operation allows the genetic algorithms to explore a larger part of the search space. The mutation operator cannot change the permutations to such a large extent as the cross-over operator and it is applied with a lower probability. During the run of the search the naïve evolution usually manages to catch up with the genetic search and sometimes even overtakes it before both processes converge to a final solution.

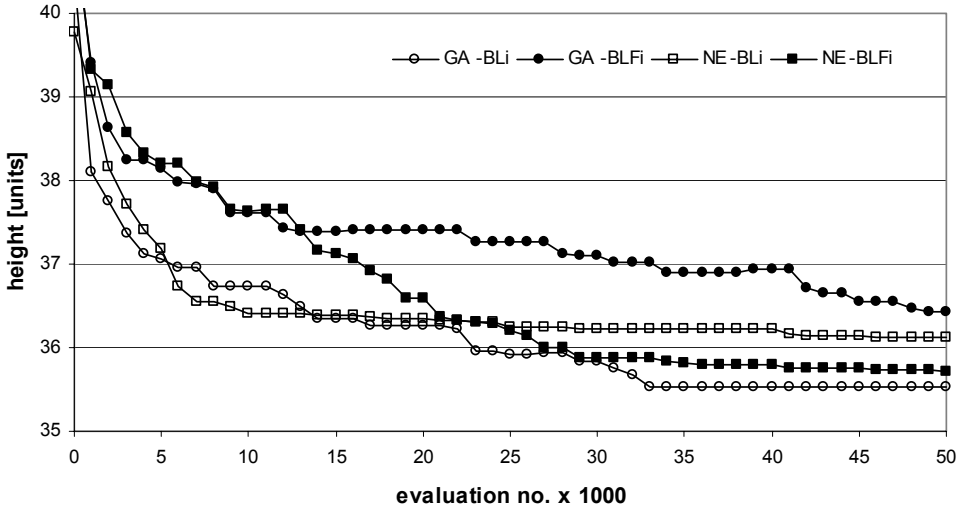


Figure 7.7: Comparison between GA and NE with BLi- and BLFi-decoder for problem poly2b

The fact that the two graphs cross each other several times shows how marginal the differences between genetic algorithm and naïve evolution are during the course of the search. The cross-over operation hardly has any influence on the outcome of the meta-heuristic algorithm. With respect to irregular strip packing, the mutation operation is sufficient for the search process and both methods can be seen as equally suitable. In the rectangular case the genetic algorithm could outperform naïve evolution for nearly all test cases. Although the performance difference has been small (Table 6.3), this reveals some influence of the cross-over operation.

Unlike the cross-over operation, mutation only introduces minor changes into the permutation. It is applied less frequently so that a lower percentage of the population changes. In order to see how successful a search process is, which applies only neighbourhood moves on one solution as opposed to a population, a comparison with simulated annealing has been carried out (section 7.4.1).

7.3.3 Comparison of the Hybrid Combinations for the Genetic Algorithm

This section compares the five packing techniques that are used as decoders in the hybrid algorithms with respect to solution quality and run time for the application with genetic algorithms. Referring to the case of rectangular strip packing, one of the four packing algorithms has outperformed the other methods to a large extent due to its efficient packing technique. In order to examine the quality of the irregular packing methods, the results of the respective combinations with the genetic algorithm are compared.

Solution Quality with the Various Decoding Routines

Table 7.7 shows the relative difference in packing height between the hybrid algorithms and the genetic algorithm using the BLi-decoder. It can be seen that the BLFi-algorithm performs worst of all hybrid techniques. Its packing heights differ by up to 5% from the BLi-decoder. As already indicated in section

7.2 the principle of first generating the complete layout on basis of the enclosing rectangles and applying a sliding step afterwards is very wasteful, since many empty areas in the layout cannot be reached in that way.

The other techniques show a similar performance and sometimes achieve better results than the BLi-combination. The difference in height usually remains below 1%. The case of the BLDi-decoder is an exception in this respect. Whereas it performs equally well for packing problems with multiple copies of polygons, the difference to the other decoders is larger for the test series 'b', which contains unique items only.

Table 7.7: Relative difference to the GA with the BLi-decoder with respect to the packing height [%]

	BLLTi	**BLFi**	**BLDi**	**BLDPi**		**BLLTi**	**BLFi**	**BLDi**	**BLDPi**
poly1a	0.4	5.0	1.0	-1.2					
poly2a	1.0	4.2	-0.2	-0.4	**poly2b**	-0.4	2.5	1.4	-1.9
poly3a	0.1	3.4	0.1	-0.6	**poly3b**	1.8	4.1	1.3	-0.4
poly4a	-0.2	2.5	0.5	0.7	**poly4b**	-0.1	2.2	1.5	0.5
poly5a	-0.4	3.3	0.6	1.8	**poly5b**	-0.4	2.6	1.2	1.2

The experiments with the genetic algorithms have shown that the BLi-, BLLTi, BLDPi and the BLDi-decoders perform equally well for most problem instances tested. Only the combination of the BLFi-routine has a different performance. This ranking of the various packing algorithms is different from the one obtained in section 7.2, when the packing rules have been tested as simple heuristics over fewer iterations. In that case, the BLDPi-algorithm outperformed the BLi- and the BLLTi-algorithm, which showed a similar performance. They were followed by the BLDi-routine. Compared to the other methods the BLFi-technique achieved the worst packing heights, which differed up to 15% from the BLi-technique. The genetic search process is illustrated for the combinations with the five decoders in Figure G.2 (Appendix G).

Comparison with Random Search

The differences in packing height within the group of hybrid genetic algorithms (Table 7.7) are smaller than the ones within the simple heuristics (Table 7.1). To some extent this is due to the higher number of iterations used in the hybrid algorithms. Some of the simple algorithms find good solutions within less than 1000 iterations, whereas others which generate the layouts in a more wasteful way need to evaluate a higher number of input sequences before they find solution of similar quality.

Figure 7.8 shows the progress the five heuristic decoding algorithms over the number of iterations for a random input. Although their progress is different in the beginning they converge on similar final packing heights with the exception of the worst decoder, i.e. the BLFi-algorithm. This indicates that even without the mechanisms of an intelligent search process a good layout can be found purely by exploration of the solution space. As the comparison between the random search and the genetic algorithms has shown, the

final packing heights usually differ by less than 3% for most cases tested. To achieve a minor improvement, meta-heuristic features are required in the search process as highlighted in Table 7.5.

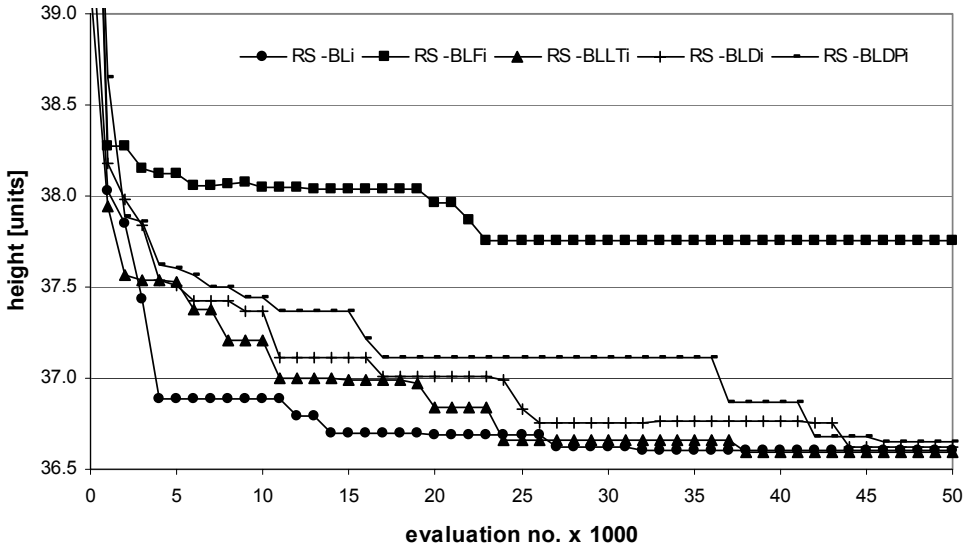


Figure 7.8: Comparison between the packing routines for problem poly2b using random search

Influence of the Rotation Interval on the Solution Quality

The experiments described above have been carried out using a rotation interval of 90°, which restricts the number of possible orientations per item to eight. In the sequel it has been examined to what extent this restriction of the solution space influences the outcome of the hybrid algorithms. Since the performance of the decoding algorithms is very similar, the following experiment has only been carried out with the BLi-routine. Table 7.8 shows the relative difference to the outcome based on the 90° rotation interval. As already seen in section 7.2 for the heuristic methods, the results at smaller rotation intervals are not as good as to the ones at 90°.

Table 7.8: Relative difference in height to experiment using with a rotation interval of 90° for BLi-decoder [%]

	GA		**RS**	
rotation interval	**45°**	**22.5°**	**45°**	**22.5°**
poly1a	3.5	-0.8	-3.3	-7.2
poly2a	6.3	7.7	4.3	5.7
poly3a	5.1	7.0	2.8	4.6
poly4a	4.2	7.6	2.6	4.5

The poorer outcome of smaller rotation intervals is due to the increase in the search space. Figure 7.9 illustrates the progress of the genetic search using the three rotation intervals. Although all three methods make rapid progress in the beginning, the graphs do not cross each other. This shows that none of the

algorithms using the larger search space finds a solution that is equivalent to the solution with the 90° interval at any stage of the search. It can be seen that the initial population of the standard configuration is up to 9% better than the ones of the other two hybrids. This lead is not only maintained throughout the search, but in fact increased.

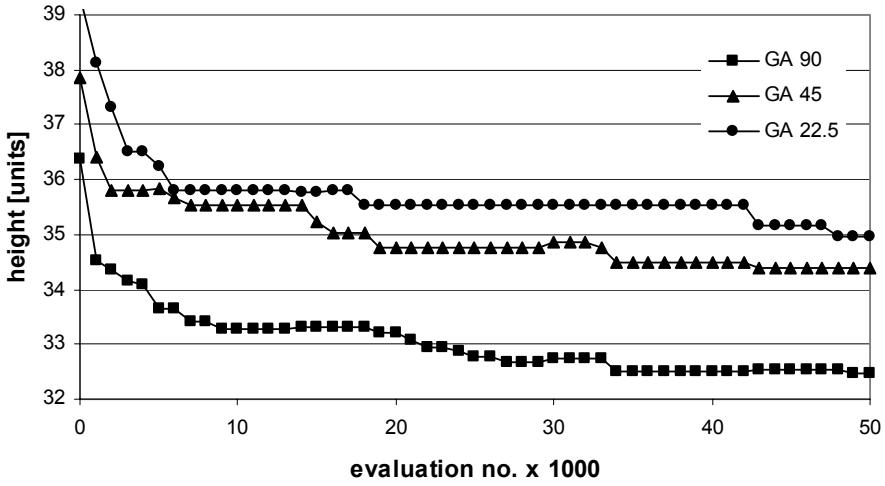


Figure 7.9: Genetic algorithm with BLi-decoder for different rotation intervals [°] for problem poly2a

Despite the larger search space, the genetic algorithms using the smaller rotation angles were executed over the same number of iterations as the standard configuration. It can be argued whether a longer duration of the search allowed the other two variants to catch up. Since the gap at the end of the search is larger than 5%, this probably will prove to be difficult.

A reason for the better starting solution of the standard configuration is certainly the fact that the polygons only occur in the position where their enclosing rectangle has minimum area. Generating the layout on this basis has been shown to be better than using a random starting orientation for the polygons during the testing of the heuristic packing algorithms in section 7.2 (Table 7.3). A smaller rotation interval increases the search space as the polygons can occur in other orientation than the minimum orientations. This does not seem to have any advantage throughout the entire search process in the example shown in Figure 7.9. The best layouts achieved with the various intervals are shown in Figure 7.10.

Using the minimum rectangle concept is only possible in packing tasks that do not contain any constraints concerning the orientation. Industrial problems are usually restricted to 90° or 180° intervals due to inhomogeneous material properties such as grain orientation or fabric pattern, which limits the choice of the search space in the first place. Apart from that, the heuristics as well as the genetic algorithms have been shown to perform better using the smaller solution spaces. The additional flexibility in terms of the orientation could not assist the meta-heuristic search process, at least not in the case of the genetic algorithms.

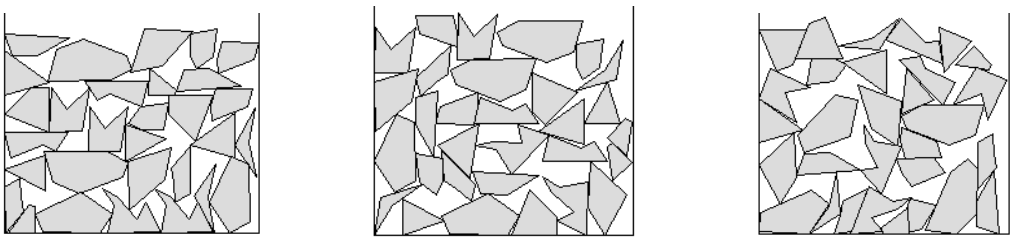


Figure 7.10: Best layout with GA + BLi with rotation intervals of 90°, 45° and 22.5° for poly2a

Time Performance

The comparison of the various decoders has shown that the BLi- and the BLLTi-routine achieve the best layouts for the series of the problems tested in this section. Apart from the solution quality, the time performance also plays an important role in the choice of the decoder for hybrid meta-heuristic algorithms. Table G.15 (Appendix G) summarises the elapsed times for all hybrid genetic algorithms. It can be seen that they have the lowest run times in this group. The other two techniques based on enclosing rectangles, i.e. BLFi and BLDi, have similar elapsed times. Since the BLDPi-algorithm requires a higher number of "expensive" distance calculations between polygons, it has the highest run time.

It has been shown that genetic algorithms obtain the best results using the BLi- and BLLTi-decoders, which also perform best in terms of run time (Table G.7, Appendix G). Whether these hybrid genetic algorithms can compete with other meta-heuristic techniques and commercial software in terms of solution quality and time performance will be investigated in sections 7.4 to 7.6.

7.3.4 Performance with Seeding

As demonstrated for the rectangular strip packing problem, seeding the initial population can improve the final outcome of the genetic algorithms (section 6.3.5). Since pre-ordered input sequences mostly outperform random input when applied to the simple packing algorithms in irregular packing tasks (Appendix G, Table G.1 to Table G.6), sorted sequences have been used as seeding in the genetic algorithms. The improvement achieved with the seeded genetic algorithms compared with the standard configuration is summarised in Table 7.9 in form of the relative distance to the final packing height.

Table 7.9: Relative difference between seeded and unseeded GAs [%]

	BLi	**BLLTi**	**BLFi**	**BLDi**
poly1a	-1.2	-6.6	-9.6	-5.8
poly2a	-0.9	-0.1	-0.9	-0.1
poly3a	-0.0	-0.0	-0.5	-0.8
poly4a	-0.4	-0.8	-2.9	-3.1
poly5a	-1.3	-1.7	-0.9	-4.3

Table 7.10: Relative difference between seeded GAs and the seeded solution [%]

	BLi	**BLLTi**	**BLFi**	**BLDi**
poly1a	-17.5	-22.5	-25.5	-17.1
poly2a	-5.7	-3.9	-14.6	-1.1
poly3a	-4.6	-3.2	-11.2	-3.9
poly4a	-2.3	-4.1	-12.2	-4.1
poly5a	-3.8	-5.5	-10.2	-4.5

Apart from the smallest test problem, i.e. poly1a, the improvement over the unseeded genetic algorithms remains usually below 1%. This result is similar to the case of rectangular strip packing where improvements below 2% have been obtained for problems of this size (Figure 6.10). The performance gain for larger rectangular problems has been up to 10%. As shown in Table 6.6 this has been mainly due to the seeded solution which could only be improved marginally by the genetic algorithm for the two largest problems. In some cases no improvement has been obtained which shows the high quality of the pre-ordered solution in connection with the respective packing technique.

In order to analyse how close the solution of the genetic algorithms is to the seed in the case of irregular packing, the relative difference between the two is stated in Table 7.10. The negative numbers indicate that the seeded genetic algorithm was capable of improving the seed for every test problem (and in combination with every decoding algorithm). The difference to the seeded solution decreases with increasing problem size.

The pre-ordered input sequences are successful when only a small number of iterations are needed (Appendix G, Table G.1 to Table G.6). Random search and especially meta-heuristic search manage to outperform pre-ordered sequences easily as shown in Table G.22 (Appendix G). The relative difference between the best heuristic pre-ordered input sequence and the standard genetic algorithms can be up to 20%. This difference diminishes with increasing problem size. The composition of the test problem seems to have an influence as well. For the problems containing multiple copies of items, i.e. series 'a', the improvement over the best heuristic result is smaller than for the problems with unique polygons. This shows that pre-ordering is more successful for problems with multiple copies, since the genetic algorithm only manages to improve the sorted input to a smaller extent.

In summary, the experiments with seeded genetic algorithms have shown that a seeded population generated by pre-ordered input sequences can improve the outcome of the hybrid algorithm by a few percent. Since not only the seeded algorithm but also the standard genetic algorithm manage to outperform the sorted sequences to a large extent, using pre-ordering in connection with simple heuristics is no guarantee for good layouts. Search processes, which use a larger number of iterations and apply intelligent features to navigate through the solution space, such as such genetic algorithms are more successful for this type of irregular packing task.

7.4 Comparison of the Meta-Heuristic and Heuristic Approaches

The investigation of rectangular strip packing problems in section 6.4 has shown that the implementation of the genetic algorithms used in this work can compete with other meta-heuristic approaches. In order to evaluate the performance of the genetic algorithms for irregular packing tasks, they are compared with simulated annealing in this chapter. Two heuristic search methods have been applied, i.e. hill-climbing and the downhill simplex method. The meta-heuristic search is "expensive" in terms of function

evaluations. The comparison with heuristic search techniques is expected to reveal to what extent the application of meta-heuristic algorithms in the area of irregular packing tasks can be justified when a balance needs to be found between solution quality and computation time.

7.4.1 Genetic Algorithms versus Simulated Annealing

In the case of rectangular strip packing problems a comparison with simulated annealing has shown that this method is capable of outperforming genetic algorithms (Table 6.9). In order to analyse whether simulated annealing in the implementation used in this work is capable of finding better results than the genetic search both methods have been applied to a series of test problems. Table 7.11 shows that simulated annealing outperforms the genetic algorithms for most tests which is indicated by negative values. In the other cases the performance gain of simulated annealing is small and mostly is less than 2%. Only for the smallest problem, i.e. poly1a, the genetic algorithms have found a solution that is much better than the one obtained by simulated annealing.

Table 7.11: Relative difference between SA and GA with respect to the average packing height [%]

	BLi	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**		**BLi**	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**
poly1a	2.8	6.9	-3.5	3.5	9.5						
poly2a	-0.3	1.3	-0.4	1.1	3.5	**poly2b**	2.0	0.6	1.7	-1.4	5.9
poly3a	0.8	0.5	0.4	-0.2	1.5	**poly3b**	1.4	-2.8	-0.6	-3.0	4.9
poly4a	-0.2	-1.4	-1.3	-0.8	0.4	**poly4b**	-1.2	-1.3	-1.2	-3.0	0.7
poly5a	0.5	-2.3	-1.2	-0.6	0.5	**poly5b**	-1.3	0.5	-1.0	-2.7	1.7

Since the stopping criteria for the simulated annealing algorithm is dependent on the problem size, the duration of the search differs for the various test problems. In particular, for the smallest packing problem the algorithm uses less than 20,000 iterations (Table G.21, Appendix G) which is low compared to the 50,000 function evaluations carried out by the genetic algorithm. This could explain why the outcome is worse in this case. For most of the larger problems simulated annealing outperforms genetic algorithms with the number of iterations becoming higher than 50,000 for problems with 30 items and more.

As Figure 7.11 shows, the progress of the search process conducted by simulated annealing is slower than the one made by genetic algorithms. At the stage where genetic algorithms converge on final solutions, simulated annealing continues and eventually achieves better heights. One of the distinctive features of simulated annealing is that the current state in the search space is only manipulated by a mutation like operator, which disrupts a packing sequence less than the cross-over operation in genetic algorithms. Although inferior solutions can be accepted by simulated annealing, the frequency of acceptance becomes less towards the end of the search process allowing only better states to proceed.

The population-based approach also allows inferior solutions to be transferred into the next generation. The difference is that simulated annealing operates on one solution at a time, continuously modifies it and

will hardly accept any inferior solutions towards the later stages of the search. This technique seems to be more successful in the context of packing problems since many layouts can be improved performing only small changes in the sequence such as a single rotation of one item. If the manipulation is larger as in the case of a cross-over operator, this can be too disruptive for the corresponding layout.

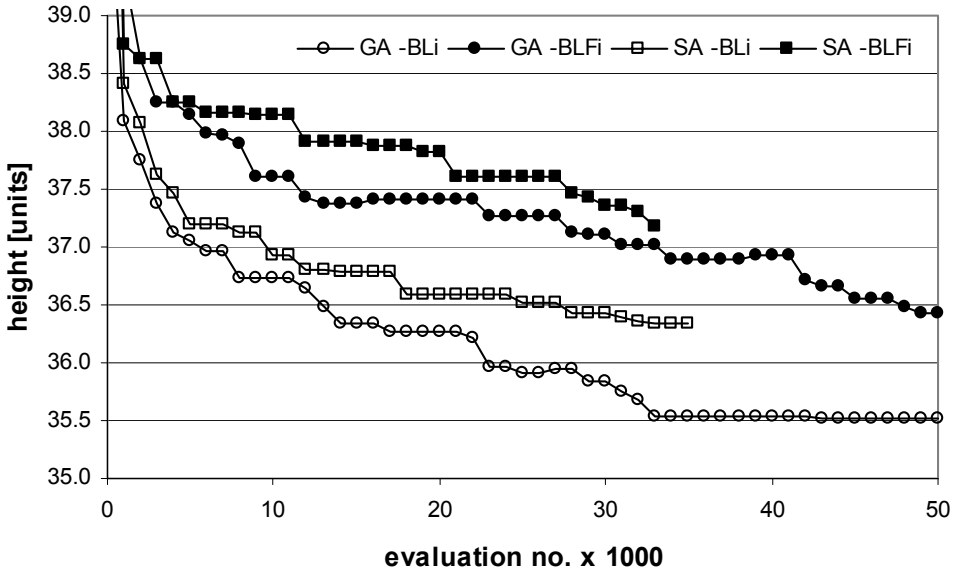


Figure 7.11: Packing height with GA and SA for BL- and BLF-algorithm for problem poly2b

Although simulated annealing can outperform genetic algorithms, the performance gain is only small. Run times of the irregular decoding algorithms are extremely long compared to the rectangular packing tasks. A possible application of simulated annealing in industrial applications has to be seen in context with the time performance. This also applies to the other meta-heuristic algorithms.

7.4.2 Meta-Heuristics and Local Search

Since the duration of the meta-heuristic search process is long compared to simple heuristic packing methods, the run times of the meta-heuristic methods can be extremely large for irregular packing tasks. Simple heuristic packing techniques do not obtain a satisfactory result and are easily outperformed by genetic algorithms and simulated annealing. In order to investigate how satisfactory other heuristic optimisation techniques can be in this context, hill-climbing has been combined with the various decoding algorithms and applied to the same test problems. Table 7.12 summarises the relative difference to the final packing heights achieved with the genetic algorithms. As the positive values indicate, genetic algorithms outperform the local search technique for every single test case. The difference is quite large for smaller problems.

Table 7.12: Relative difference between HC and GA with respect to the average packing height [%]

	BLi	BLLTi	BLFi	BLDi	BLDPi		BLi	BLLTi	BLFi	BLDi	BLDPi
poly1a	23.4	25.2	19.1	22.4	32.4						
poly2a	14.1	10.2	8.0	10.8	11.7	**poly2b**	10.9	13.0	11.2	9.3	15.5
poly3a	7.6	6.9	5.3	9.6	11.1	**poly3b**	9.6	5.4	2.6	4.8	10.8
poly4a	4.8	8.3	4.7	5.1	5.2	**poly4b**	6.1	5.4	3.1	2.4	10.3
poly5a	6.2	5.6	2.4	5.9	5.8	**poly5b**	6.0	5.9	3.1	7.4	7.4

The comparison of the search processes in Figure 7.12 clearly shows that the local search process gets trapped in a local optimum representing an inferior solution. The simulated annealing algorithm already has a large lead at the beginning of the search process since it is capable of exploring the area around the starting solution, whereas hill-climbing can only operate from that solution. Since the performance of hill-climbing is poor in terms of solution quality compared to genetic algorithms, it cannot be considered as an alternative to the meta-heuristic approach despite the lower number of function evaluations (Table G.21, Appendix G).

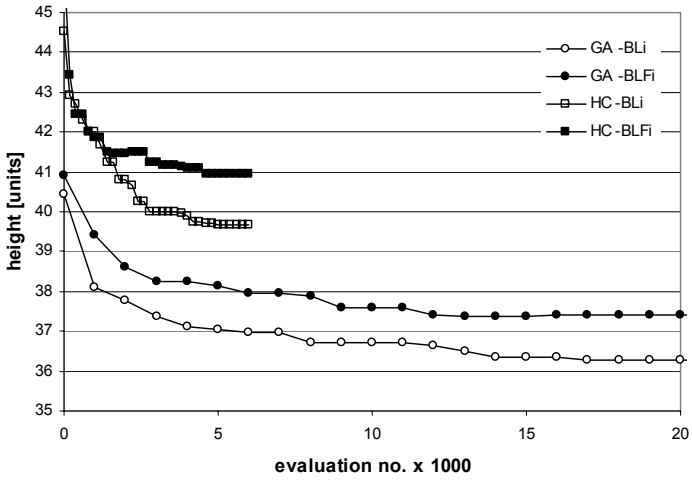


Figure 7.12: Packing heights with HC and GA for BL- and BLF-algorithm for problem poly2b

7.4.3 Meta-Heuristics and the Downhill Simplex Method

As a second heuristic optimisation technique, the downhill simplex method has been applied. The method operates sequentially by attempting to position each item at its local optimum position in the layout. Starting from a random position on the object, new locations are computed by the algorithm and evaluated until a local optimum has been found. As criterion for the evaluation of the quality of a position, the y-value of the highest vertex of the polygon has been used. Two different implementations of the downhill simplex method for the irregular packing tasks are described in section 5.6.3.6.

In the first one only the position of the polygon to be placed is manipulated, which is determined by the x- and y-co-ordinate of its reference point. The polygons are presented to the algorithm in a random or pre-sorted sequence. In the first implementation the orientation of the items is generated randomly and can vary by 90°.

In the second approach the orientation is part of the manipulation process of the downhill simplex method. The rotation interval has been limited to 12.25°, which results in 64 possible orientations. The algorithm determines the x- and y-co-ordinate of the polygon position and its orientation for every optimisation step. Although the number of orientations is higher than in the application of the meta-heuristics, this step has been necessary, because a larger rotation interval (i.e. 90°) has resulted in a very poor performance of the algorithm.

The operation of the downhill simplex method is different from the hybrid algorithms since it performs the optimisation directly on the 2D layout rather than the compound between a data structure and the generation of the complete layout. Whereas the hybrid algorithms evaluate the quality of the complete layout after the placement of all items, the downhill simplex method performs a series of optimisation steps by gradually improving the position of a single polygon in the partial layout. A minimum orientation of 90° would be extremely abrupt in this context and can easily result in overlapping configurations with other polygons. Since an overlap is treated with a severe penalty it becomes almost impossible to nest a polygon along with other items of the partial layout. A small rotation angle ensures that only gradual orientation changes are made. Due to the restriction regarding the orientation of the items, this implementation of the downhill simplex method is only appropriate for a limited amount of industrial packing tasks.

Table 7.13: Relative difference between the downhill simplex method and the GA with respect to the packing height for both implementations [%]

	without rotation	**with rotation**		**without rotation**	**with rotation**
poly1a	22.5	8.9			
poly2a	13.3	5.0	**poly2b**	10.4	1.7
poly3a	8.4	1.2	**poly3b**	11.5	4.4
poly4a	11.8	2.1	**poly4b**	10.5	1.9
poly5a	8.5	3.6	**poly5b**	10.2	9.8

Table 7.13 summarises the outcome obtained for the two implementations of the downhill simplex method. Both implementations perform worse than the best hybrid genetic algorithm with respect to the final packing height. The relative difference to the outcome of the genetic algorithm is higher for the method that does not make use of the rotation of the items. As mentioned above, the task of nesting an item into the partial layout becomes more difficult when the difference between two optimisation steps is too abrupt. In many cases, a small rotation of the polygon could contribute to the allocation of the

polygon at a better position. If the rotation is excluded from the optimisation process, the allocation of the polygon will result in larger empty areas between the items. The algorithm, which also manipulates the orientation of the item, can explore a larger range of the local area. Small changes in the orientation along with a position change can avoid overlapping configurations easier, which allow finding better solutions.

Regarding the overall result, the downhill simplex method in this implementation cannot be considered as an alternative to the meta-heuristic search algorithms. In general, the heuristic search methods are outperformed by meta-heuristic search processes that are capable of better exploring the solution space.

7.4.4 Computation Time of Meta-Heuristic and Heuristic Search Methods

The experiments with the meta-heuristics on a number of different sized packing tasks have shown that the meta-heuristic search algorithms can generate solutions that outperform other methods. One of the major drawbacks of meta-heuristic search is the computation time. Since the search process is based on a large number of function evaluations they have longer run times than heuristic algorithms. This causes particular concern in irregular packing tasks, which require a large amount of complex and expensive geometric computations. Unlike in the case of their rectangular counterparts, computation times can be high even for small packing problems.

The run times of the algorithms play an important role for industrial applications. In many cases the time performance decides whether a certain method will be used for a packing task. In order to judge the suitability of meta-heuristic methods, their computation times are summarised for one run in Table G.18 to Table G.20 in Appendix G. The time performance depends on the number of items and can easily reach several hours for larger problems. Since the number of iterations carried out by the simulated annealing method are higher (Table G.21, Appendix G), its computation times exceed those of the genetic algorithms. The performance gain achieved by simulated annealing compared with genetic algorithms usually is less than 2% (Table 7.11). This probably does not justify its application for most applications, especially since the elapsed times are up to five times longer and will increase even further for larger problems.

Genetic algorithms and naïve evolution have shorter run times. Since both methods are based on a population, only new members need to be computed when the next generation is evaluated, whereas every new step in simulated annealing requires a function evaluation. Due to the lack of the cross-over operation, fewer function evaluations are performed by the naïve evolution algorithm that results in even shorter run times than genetic algorithms. As stated in Table 7.6 the solution quality achieved by the evolutionary techniques is similar. Since the genetic algorithm does not necessarily perform better, naïve evolution is a good alternative of this type of packing problems requiring only half the time for the largest problem in this test.

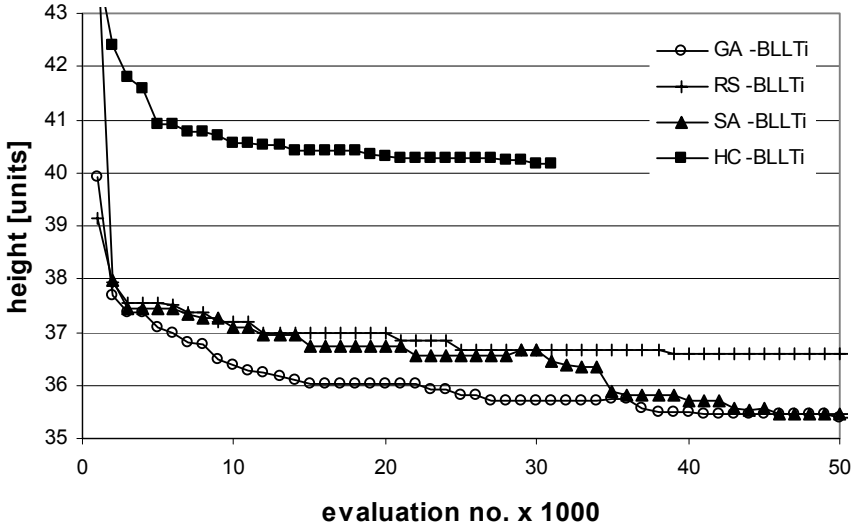


Figure 7.13: Comparison of meta-heuristics and heuristics for BLLTi-algorithm for problem poly4b

Hill-climbing has lower computation times. As the layout qualities are satisfactory, hill-climbing cannot compete with meta-heuristics or random search. Although the packing routines used as decoders in the hybrid methods are very powerful techniques, they cannot reach the performance of the meta-heuristics only applying a small number of evaluations. Pre-ordering input sequences can result in an improvement over random input sequences. This does not result in better outcomes than the meta-heuristics techniques for the problem sizes tested. As the number of iterations gets larger the solutions found by random input sequences are close to the meta-heuristic ones (Figure 7.13). The meta-heuristic search techniques outperform the random search towards the end of the search process. The typical course of the search processes is illustrated in Figure 7.13 for various heuristic and meta-heuristic methods. Figure 7.14 and Figure 7.15 show some of the best layouts achieved with meta-heuristic and heuristic techniques.

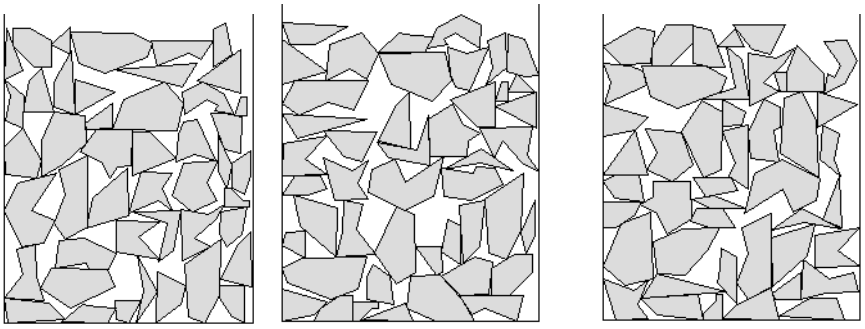


Figure 7.14: Best layouts for poly3b with the following hybrids: GA + BLi, NE + BLi and SA + BLi

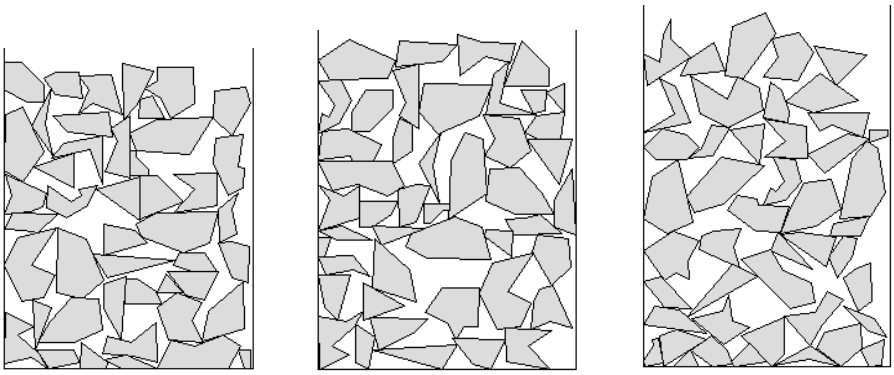


Figure 7.15: Best layouts for poly3b with the following hybrids: RS + BLi, HC + BLi and DHS + BLi

7.5 Comparison with Approaches in Literature

The series of meta-heuristic techniques developed in this investigation for irregular packing problems have shown to generate good layouts in comparison with other techniques. As stated in section 7.3 and 7.4, the three meta-heuristic approaches genetic algorithms, naïve evolution and simulated annealing could easily outperform simple packing routines as well as heuristic optimisation techniques. They performed better than the pure random exploration of the search space. This confirms the successful operation of the meta-heuristic search process.

One of the major advantages of hybrid meta-heuristic algorithms lies in their fairly simple implementation. In particular, the order-based construction allows the use of well known search space manipulation techniques rather than developing problem-specific operators for the application to a special problem representation. The run times of meta-heuristic methods are comparatively high especially when applied to irregular packing tasks. In order to evaluate their overall performance in the context of irregular strip packing, a comparison with some meta-heuristic and heuristic approaches from the literature has been made.

To avoid unnecessary re-implementation of published work, algorithms are usually compared on the basis of commonly recognised benchmark problems. In the area of irregular packing, in particular, such benchmarks do not yet exist (section 5.3). Authors usually publish their test problems in the form of sample layouts rather than a list of vertices with one exception (Oliveira et al., 2000). A comparison with other approaches is a difficult task. In order to be able to proceed with the performance evaluation in this investigation, a vital part of algorithm development, some of the benchmark problems have been obtained from sample layouts in the work published by scanning and digitising the graphic images (section 5.3.1.2).

Generally three types of problems can be distinguished regarding the packing problems used for demonstration purposed in the literature. Some of the test problems originate from the textile or metal

industry, the majority, however, are not connected to any specific application, but have been created artificially for the purpose of testing as done in sections 7.3 and 7.4. The benchmark problems used in the sequel describe packing tasks from the textile industry or have been generated by the various authors.

As the experiments in section 7.3 and 7.4 have demonstrated, the BLi- and the BLLTi-routines generally have performed best as decoders for the meta-heuristic algorithms using the test problems of Table B.4 (Appendix B). For some benchmark problems from the literature other decoders performed better. The best combination of the meta-heuristic and the decoding algorithm has been used for this comparison. Since some of the algorithm development in the literature targeted specific industries, the respective packing tasks are usually constrained in terms of the item rotation. These orientation constraints have been considered in the following comparison. The rotation into the minimum rectangle position has not been applied (section 7.2).

7.5.1 Test Problems from the Literature without Specific Application

Many researchers have developed meta-heuristic approaches for the irregular packing problem. Most work in this area has concentrated on genetic algorithms and simulated annealing (sections 4.3.3 and 4.4.1). Whereas most genetic algorithms are based on hybrid techniques, the approaches with simulated annealing and tabu search operate directly on the layout. Some benchmark problems could be obtained from the published work for a performance comparison with the set of meta-heuristic methods developed in this work. Table 7.14 summarises the results for a series of non-specific benchmark problems and compares them to the outcome obtained by the respective approach described in the literature.

Comparison with Genetic Algorithm Approaches

Jakobs (1996) extended the genetic algorithm developed for rectangles to polygons by introducing a compaction algorithm that attempts to compress the layout at first created on the basis of the enclosing rectangles with the aid of the BL-decoder. As Table 7.14 shows this approach is less effective than the ones which place the polygon along with the polygons of the partial layout in the first place. The meta-heuristic algorithms outperform Jakobs' approach by slightly more than one unit in height on average for the problem Jakobs1. Since the vertices of the polygons in the second test problem (Jakobs2, Appendix B, Figure B.13) have been multiplied by a factor of two for reasons stated in section 5.3.1.2, the performance difference lies in the same range. Random search achieves a height that is extremely close to the outcome of the meta-heuristics, but still better than the more wasteful technique used by Jakobs (1996). This demonstrates the efficiency of the packing principle for finding a good position for the polygon as opposed to the compaction of the complete layout in a consecutive step. This has been confirmed in the experiments with the BLFi-decoder, which always resulted in a poorer outcome than the combinations with the other placement routines (section 7.3).

Dighe and Jakiela (1996) experimented with a hybrid genetic algorithm that slides the irregular item into the partial layout in vertical direction. For the determination of the horizontal position and the orientation of the item an additional genetic algorithm is applied. The major drawback of this technique is that it consists of a high-level and a low-level genetic algorithm. This is extremely wasteful in terms of computation time. The test problems used in their work are especially constructed jigsaw problems, which contain clearance between the parts, such that the packing density of the optimum solution is less than 100%. Since this clearance is of no importance to the packing process (as sample layouts show) a comparison on basis of the achieved packing densities is made. Two test problems have been constructed which have the same item numbers and similar shapes as the original ones (Appendix B, Figure B.20 and Figure B.21). As Table 7.14 shows, both approaches achieve similar packing densities with the newly developed meta-heuristics performing slightly better.

A second hybrid genetic algorithm was developed by Fujita et al. (1993) which allows a free rotation of the items. A local optimisation algorithm (Quasi-Newton) is applied to find the position and orientation of the polygon. This approach slightly outperforms the meta-heuristics with the sliding decoders. One of the reasons for this is that the orientation in this configuration is limited to intervals of 90°, whereas free rotation could be especially beneficial in the context of this test problem (Appendix B, Figure B.14) which consists mainly of triangles and rectangles.

Comparison with a Simulated Annealing Approach

Unlike the genetic approaches mentioned above, Han and Na (1996) used the meta-heuristic search process only to improve an existing layout generated with the aid of a neural network. The simulated annealing algorithm manipulates the search space by a series of translation, rotation and swap operations on the geometric objects penalising overlap. The difference in the outcome is fairly large (Table 7.14). The algorithm developed by Han and Na operates faster and possibly could improve the final outcome using a higher number of iterations.

Comparison with a Tabu Search Approach

The work by Blazewicz et al. (1993) is one of the few examples that used tabu search in irregular packing problems. The search process operates directly on the layout manipulating it by a series of geometric operations. The initial layout is created by an external algorithm. Since the test problems contain arcs and circles, the original items have been approximated by a series of vertices circumscribing the respective shapes (Appendix B, Figure B.16 and Figure B.17). The packing densities achieved with the genetic algorithms and the sliding decoders have been slightly higher for both test problems.

Table 7.14: Packing heights and densities obtained with GA, SA, RS and simple packing heuristic for benchmark problems from the literature

	height [units]				packing density [%]			
problem	**Jakobs1**	**Jakobs2**	**Fu**	**Han**	**Blaz1**	**Blaz2**	**Dighe1**	**Dighe2**
problem size	25	25	12	23	28	20	16	10
literature	15.0	32.0	34.0	62.4	69.1	68.6	69.0	72.0
GA	13.8	29.6	34.2	50.7	72.5	65.4	68.0	74.6
NE	13.7	28.8	35.3	50.6	70.5	66.3	64.5	73.3
SA	13.9	28.2	34.6	49.8	73.4	67.0	68.0	73.6
RS	14.0	29.8	34.8	51.2	69.7	64.9	64.3	68.9
simple heuristic	14.9	31.6	40.6	57.0	69.7	67.3	55.7	58.2

7.5.2 Test Problems from the Textile Industry

Unlike the approaches described above, some researchers tested their algorithms using industrial data. The series of benchmark problems listed in Table 7.15 originate from the textile industry. Although the orientation is usually limited to 0° or 180° in the industrial application, most approaches described in the literature nevertheless operate with a rotation interval of 90°.

Comparison with a Simulated Annealing Approach

Marques et al. (1991) manipulated the search space with the aid of translation, rotation and swapping operations. Overlapping configurations are rejected. The comparison in Table 7.15 shows that the meta-heuristic algorithms with sliding decoders achieve better layouts. Even the simple packing routine applied for a large number of iterations, i.e. random search, outperforms this result. Approaches that operate on the layout rather than according to hybrid principle have to deal with the problem of overlap. The large performance difference indicates that the generation of dense layouts in that manner is more difficult since even small moves and in particular rotation can easily lead to invalid configurations. In this respect packing techniques based on the sliding principle avoiding overlap have a major advantage.

Comparison with Genetic Algorithm Approaches

So far, Ratanapan and Dagli (1997b, 1998) developed the only evolutionary approach that does not operate on a data structure, but directly on the layout. The genetic algorithm developed for the packing of rectangles was extended to the irregular packing task. The layout is manipulated by a translation and rotation of individual polygons as well as parts of the layout. The operations are implemented as mutation and hill-climbing operators. A cross-over operator performs changes between several layouts. As Table 7.15 shows, this technique achieves a good outcome in comparison with the hybrid algorithms. Compared to the hybrid techniques the major drawback of this approach lies in the complexity, since a number of routines have to be implemented to perform complex geometric operations.

The genetic algorithm by Bounsaythip and Maouche (1997) uses a binary tree as the data structure to determine the way in which two items are placed next to each other and orientated. In order to fix the position, the nesting operation aims to find the minimum enclosing rectangle of the two shapes. This technique was developed specifically for the textile industry where a tree does not represent the complete layout but only a strip of it. This approach obtains a layout quality in the range of the hybrid techniques developed in this work (Table 7.15).

Comparison with Heuristic Search

Albano and Sapuppo (1980) used a heuristic search technique for a marker layout problem (section 4.2.2.2). The search is guided by layout information calculating the current waste and estimating the final one obtained if this node in the search tree was fully expanded. A series of rules are introduced to avoid the combinatorial explosion by reducing the increasing number of expanded nodes. This approach achieves layouts with good packing densities compared to the meta-heuristic methods considering that it finds a good solution in less iteration time.

Table 7.15: Packing heights and densities obtained with GA, SA, RS and simple packing heuristic for benchmark problems from the textile industry

	height [units]		packing density [%]	
problem	**Dagli**	**Marques**	**Mao**	**Albano**
problem size	30	24	20	24
literature	69.7	92.0	68.9	83.6
GA	69.9	83.7	71.6	85.0
NE	69.3	83.6	67.4	85.5
SA	69.1	84.6	69.8	86.0
RS	71.4	84.3	70.2	84.3
simple heuristic	75.8	99.1	61.7	82.5

7.5.3 Comparison with Problem-Specific Heuristic Search Method

Oliveira et al. (2000) developed a heuristic search method using the No Fit Polygon to determine a set of the feasible positions in the layout (section 4.2.2.2). Since the authors published the data of their test problems, a direct comparison with the meta-heuristic algorithms developed in this work is possible. Table 7.16 summarises the results for problems from the textile industry as well as problems with no specific application, which have been simulated using several rotation intervals.

The heuristic search algorithm achieves better results for all test problems. Whereas the difference in packing height for the textile problems is below 8%, the outcome for the artificial problems is up to 20% better. One of the major advantages of this approach is the nesting technique itself. Since it is based on the No Fit Polygon two items can be nested close to each other. The fact that several positions are

evaluated and the best one is chosen guarantees that the allocation takes place in a local optimum. This can be very beneficial in nesting problems containing many concave features (Appendix B, Figure B.18).

Table 7.16: Packing heights obtained with GA, RS and simple packing heuristic for benchmark problems by Oliveira et al. (2000) [units]

	SHAPES0	**SHAPES1**	**SHAPES**	**SHIRTS**	**TROUSERS**	**SWIM**
rotation interval	0°	180°	90°	180°	180°	180°
problem size	43	43	43	99	64	48
literature	66.8	61.0	-	66.4	263.2	-
GA	74.6	73.0	65.4	67.8	284.0	7778
NE	74.9	72.2	65.4	68.6	286.8	7671
RS	75.2	73.6	65.2	68.0	286.2	7891
simple heuristic	76.0	76.0	63.0	68.0	286.0	7825

7.5.4 Summary

Apart from the results in Table 7.16 most benchmark problems have been obtained with the aid of a scanning process and are approximations of the ones used by the various researchers. Therefore the comparison should not purely focus on numbers, but rather on the range of an outcome to judge the performance of two methods. In order to enable a strict comparison the original test problems will have to be used.

This comparison has shown that most of the meta-heuristic approaches are capable of obtaining layouts of an equal quality to the hybrid techniques developed in this work. The two simulated annealing approaches published in the literature perform worse as they apply the search space manipulation directly on the layout and consider invalid configurations. Although the hybrid simulated annealing achieves better results, this approach is less successful in the context of irregular packing. It has been shown that heuristic search can be a powerful tool to tackle this kind of problem in terms of solution quality as well as computation time.

7.6 Comparison with Commercial Nesting Software

Since packing always has played an important role in many manufacturing processes, the software industry has directed a lot of effort towards the development of automatic nesting systems (Appendix C, Table C.1 to Table C.3). An overall evaluation of newly developed algorithms should also include commercial products. As described in section 5.4 the nesting packages Nestlib and SigmaNest have been used in the following comparison.

The test problems described in Table 5.13 as well as the benchmark problems obtained from the literature have been nested with both packages considering the respective orientation constraints. The rotation

interval is supplied by the user for each nesting task. Since the commercial nesting process does not use the minimum enclosing rectangle concept used throughout the demonstration of the various hybrid algorithms in sections 7.2 to 7.4, the meta-heuristic algorithms have been applied to the original problems for this comparison using a rotation interval of 90°.

In both nesting packages the user supplies a sorting criterion for the items by choosing between decreasing area and height for SigmaNest and between decreasing area and perimeter in the case of Nestlib. Nestlib can be set to evaluate a maximum of 30 trials; the best ones are stated in Table 7.17 to Table 7.19.

7.6.1 Polygon Test Problems

The comparison in Table 7.17 shows that both commercial packages outperform the meta-heuristic solutions. Whereas the relative difference in packing height is around 10% for the problems with up to 30 items, it increases for the larger test cases. One of the reasons for this is that the commercial algorithms are capable of filling enclosed areas in the partial layout. Since the number of gaps in the layout increases during the nesting process it is beneficial to allocate smaller items towards the end. Both algorithms only operate on input sequences that are sorted according to certain parameters describing the size of the items.

In addition to that, the commercial nesting algorithm uses more time to search for a locally optimum position. The decoders used in the hybrid algorithms are based on a sliding principle accept an allocation if no further movements in either to the left or to the bottom are possible. They do not permit displacements in the reverse direction. Figure 7.16 and Figure 7.17 show the layouts generated with Nestlib and the best hybrid genetic algorithm for two test problems.

Table 7.17: Packing heights obtained with commercial software, meta-heuristics, RS and simple packing heuristic for benchmark problems [units]

poly	**1a**	**2a**	**3a**	**4a**	**5a**	**2b**	**3b**	**4b**	**5b**
problem size	15	30	45	60	75	30	45	60	75
SigmaNest	17.1	30.4	45.0	59.3	73.9	33.1	42.5	54.0	64.3
Nestlib	14.7	30.1	40.4	56.9	71.6	33.6	41.8	52.9	63.4
GA	17.2	33.4	50.9	68.1	86.4	39.2	53.3	57.0	70.0
GA (seeded)	16.0	33.1	50.9	67.7	84.7	36.6	49.8	56.8	77.7
RS	18.1	34.4	52.0	70.1	87.6	38.8	54.9	65.7	77.2
packing heuristic	18.8	36.7	55.6	74.2	85.7	41.7	51.9	66.8	80.3

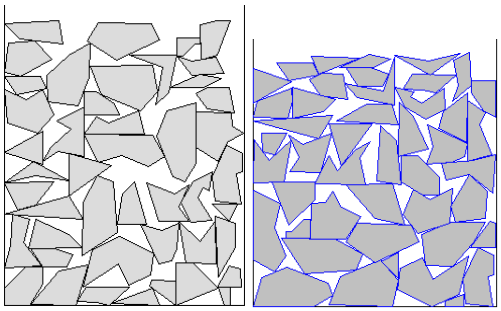


Figure 7.16: Best layouts for poly3b with GA+ BLi (left) and Nestlib

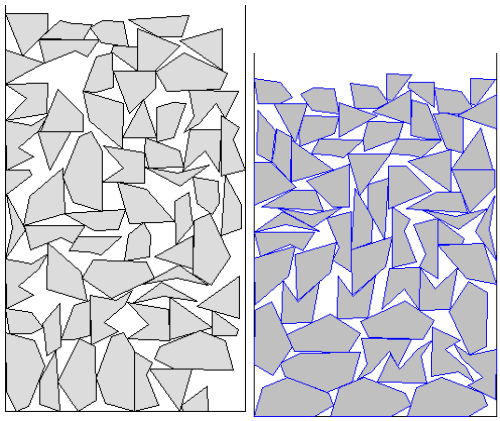


Figure 7.17: Best layouts for poly4a with GA+ BLi (left) and Nestlib

Whereas the comparison with the outcome stated in the literature can only be a general one due to the approximation of the test problem, a direct performance evaluation between the hybrid algorithms and the commercial software packages is possible because the same test problems have been used. Since the test problems stated in Table 7.17 consist of polygons which contain concave features, enclosed empty areas are likely to be created during the nesting process conducted by the sliding routines. In order to see if the commercial algorithms can maintain their lead owing to a certain extent to the gap filling capability, a series of different benchmark problems from the literature have been nested.

7.6.2 Test Problems from the Literature

Table 7.18 and Table 7.19 show that the commercial techniques can only achieve better outcomes for some test problems. In many cases the hybrid algorithms and the approaches from the literature produce better results. This is especially the case for some textile problems, which contain fewer concave features, but mainly rectangle like shapes (e.g. Marques, Mao, Albano and SHIRTS). In problems with more irregular and concave features (e.g. Jakobs1, Jakobs2, SHAPES0 and SHAPES1) the commercial software obtains better results than the hybrid algorithms.

Another reason why the difference between the commercial and the hybrid algorithms is smaller using these examples is that they usually have a smaller problem size than the test problems in section 7.6.1. For problems with less than 30 items the difference has also been smaller for the examples in Table 7.17. Higher item numbers result in more enclosed areas in the case of the hybrid algorithms and packing heights which are higher than the ones obtained with hole filling techniques (e.g. TROUSERS, SWIM). In order to demonstrate the different nesting techniques used by the commercial package and the meta-heuristic the layouts generated with Nestlib and the genetic algorithms for three textile problems are shown in Figure 7.18 to Figure 7.20. It is evident that the commercial technique places the largest items first and then tries to fill the gaps with the smaller ones.

Table 7.18: Packing heights and densities obtained with GA and commercial software for benchmark problems from the literature [units]

	packing height [units]						density [%]			
	Jakobs 1	**Jakobs 2**	**Fu**	**Han**	**SHAPES 0**	**SHAPES 1**	**Blaz 1**	**Blaz 2**	**Dighe 1**	**Dighe 2**
size	25	25	12	23	43	43	28	20	16	10
literature	15.0	32.0	34.0	62.4	66.8	61.0	61.9	68.6	69.0	72.4
SigmaNest	13.2	28.7	38.1	51.2	69.8	69.8	65.3	60.7	69.1	73.1
Nestlib	13.4	29.2	39.3	50.9	69.0	64.1	69.9	67.6	72.4	67.7
GA	13.8	29.6	34.2	49.3	74.6	73.0	72.0	67.3	68.0	74.6

Table 7.19: Packing heights obtained with GA and commercial software for textile problems from the literature [units]

	packing height [units]					density [%]	
	Dagli	**Marques**	**SHIRTS**	**TROUSERS**	**SWIM**	**Mao**	**Albano**
size	30	24	99	64	48	20	24
literature	69.7	92.0	66.4	263.2		68.9	83.6
SigmaNest	67.1	84.1	74.8	275.3	7587	68.2	83.2
Nestlib	65.6	86.7	69.1	283.5	6568	70.6	83.5
GA	69.9	83.7	67.8	284.0	7778	71.6	85.0

The comparison has shown that the commercial nesting algorithms are capable of achieving better or comparable outcomes for most of the test problems used. Since these test problems are very different in their origin and geometric features, this work demonstrates the flexibility of the nesting software. Another important advantage of the commercial packages over the meta-heuristic methods is their low computation time. Even the largest packing tasks such as SHIRTS or TROUSERS have been nested within less than one minute. Only heuristic search techniques are able to compete in this respect. Oliveira et al. (2000) reported an execution time of less than four minutes on the largest problem i.e. SWIM. Apart from the computation time the heuristic search method can also compete with the commercial packages in terms of the solution quality. The technique developed by Oliveira et al. (2000) obtained a better result than both packages in the respective test cases.

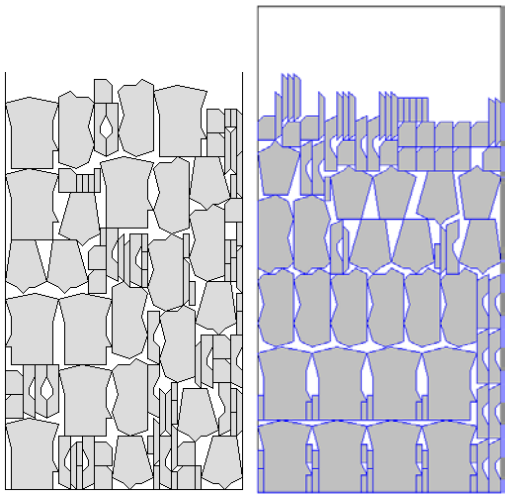


Figure 7.18: Best layouts for SHIRTS with GA+BLFi (left) and Nestlib

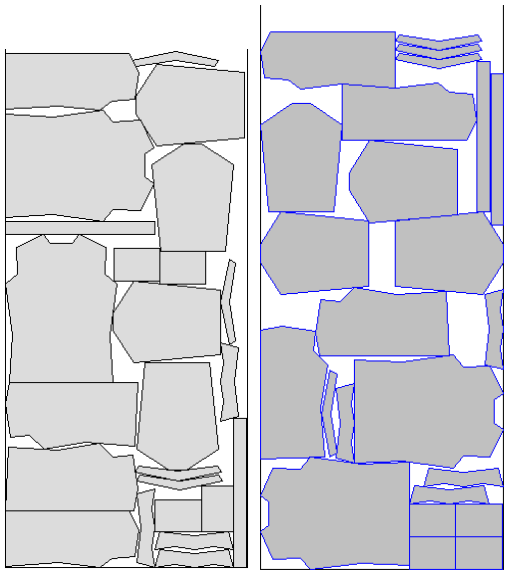


Figure 7.19: Best layouts for Albano with GA+BLFi (left) and Nestlib

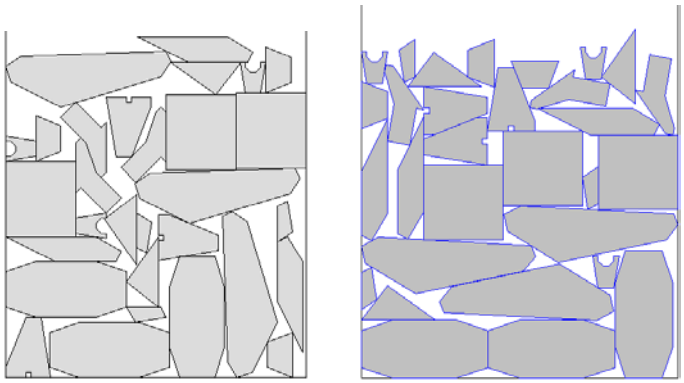


Figure 7.20: Best layouts for Dagli with GA+BLFi (left) and Nestlib

7.7 Conclusions

A series of algorithms have been developed for the irregular strip packing problem consisting of the combination between a meta-heuristic and a packing routine. Several different placement techniques have been implemented and tested for the second stage of the hybrid approach. These routines are based on two principles. The first category generates the layout by placing the enclosing rectangle and then iterating the polygon in the partial layout. The second one generates the complete layout on basis of the enclosing rectangle and applies a compaction step at the end. The first group outperforms the second one.

The meta-heuristic methods work better than the simple packing algorithms executed over a far lower number of iterations. Using pre-ordered input improves the layout quality achieved with the packing routines. Pre-ordering does not reach the solution quality obtained by the meta-heuristics. Seeding the

initial population with pre-ordered input sequences only improves the results of the genetic algorithms slightly. When the simple packing routines are executed over a large number of iterations as in the case of random search their outcome can be close to one of the meta-heuristics. The meta-heuristics are capable of searching the solution space more efficiently and generate layouts that are usually 3% better in terms of packing height than the random search process. It has been shown that genetic algorithms generate layouts of lower quality if the solution space is larger using smaller rotation intervals for the items.

Regarding the meta-heuristic methods, no major performance difference has been observed between genetic algorithms and naïve evolution, which leads to the conclusion that the cross-over operator does not have an important impact on the final outcome of the search using this implementation. The mutation operator has shown to be sufficient to navigate the search process through the solution space. This is also confirmed by the outcome of simulated annealing, which manages to outperform the packing height achieved with genetic algorithms in most cases. The performance difference remains below 1% considering the additional function evaluations carried out by simulated annealing.

The comparison with meta-heuristic approaches in the literature has shown that the hybrid techniques developed in this work obtain a similar solution quality. The principle of avoiding overlap in the layout generation has been more successful in this respect than simulated annealing approaches from the literature that can result in invalid configurations during the search process. As in the case of rectangular strip packing benchmark problems and algorithms could facilitate the performance analysis of newly developed algorithms in the area of irregular cutting and packing.

In terms of computation time meta-heuristic approaches have a major drawback over heuristic search. Heuristic search has been found to be a powerful method, which achieves very good layout quality and has extremely short run times compared to the meta-heuristics. An efficient implementation of the geometric algorithms is of paramount importance for the efficient operation of the meta-heuristics.

The nesting technique has a large influence on the outcome of the search process. Packing strategies, which find a sub-optimal position for a polygon in every iteration step, have been shown to be extremely successful. The nesting techniques applied in commercial software point in this direction. Keeping track of enclosed areas allows the generation of dense layouts. Both nesting packages apply this strategy.

The best nesting methods available to-date which meet industrial requirements are the algorithms used in commercial packages as well as heuristic search approaches. Meta-heuristics, as found in the literature and developed in this work, are not sufficiently competitive in terms of the computation time and to a smaller extent weak in solution quality - at least for some nesting tasks investigated in this study.

8. Genetic Algorithms and 2D Bin Packing

8.1 Introduction

The hybrid genetic algorithms developed for the 2D rectangular strip packing problems generate layouts of higher quality than the ones obtained with simple packing algorithms. In addition to that, it has been shown that the outcome is comparable or even better than some of the approaches used in the literature. The two commercial packages investigated in section 6.7 and 7.6 for regular and irregular packing tasks do not make an attempt to find the best set of objects. The hybrid algorithms developed for the 2D rectangular strip packing problems have therefore been extended for use with multiple objects.

As mentioned in the previous chapters, the performance of the genetic algorithms is compared to random search. In order to establish the contribution of the cross-over operator to the search process a naïve evolution algorithm has been implemented. Since simulated annealing usually has achieved the best layouts in the case of rectangular strip packing, it also has been applied to the 2D bin packing problem. The comparison of the various hybrid algorithms also includes a local search heuristic, i.e. hill-climbing.

Due to the lack of a comprehensive set of benchmark problems in the area of 2D bin packing, a direct performance evaluation of the hybrid algorithms has only been possible in respect to commercial software. For this purpose three problem categories have been created with each of them containing five test problems. The test problems of one category have the same number of items and have been created such that the total area of the items is similar (section 5.3.3.3).

8.2 Comparison of the Packing Algorithms

Before the performance of the hybrid algorithms is compared, the simple packing algorithms have been tested on their own applying them to random and pre-ordered input sequences. Since the objective of the algorithm is to find the set of objects with the least total area, the outcome of the algorithms is evaluated on the basis of the average object utilisation.

Solution Quality

Figure 8.1 shows the object utilisation for the three test categories using the four packing routines with random input. As in the case of the strip packing problem the BLF-algorithm achieves the best layouts. This is due to its packing principle that allows nesting into enclosed areas of the partial layout. The utilisation achieved with this routine is up to 5% higher than the results of the next best algorithm. Among the sliding techniques the BL- and the BLLT-algorithm work equally well, with the BLLT-

routine working better for the larger problems (i.e. M2 and M3). The BLD-routine shows the worst performance.

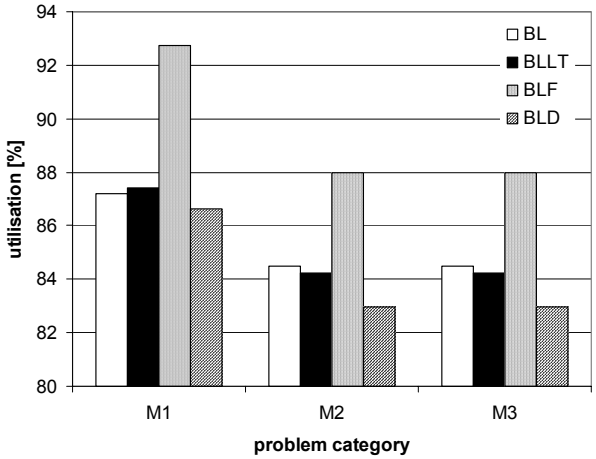


Figure 8.1: Average object utilisation for packing routines using random input

Table 8.1: Relative difference in average utilisation between area-sorted and random input [%]

	M1	M2	M3
BL	6.4	5.5	6.3
BLLT	6.0	5.9	7.0
BLF	5.5	6.0	7.5
BLD	6.5	6.9	7.6

The outcome of all packing routines can be improved by applying pre-ordered input sequences. Sorting the rectangles by decreasing area and perimeter works better than other sorting criteria (Appendix H, Table H.1 and Table H.2). The relative difference in the average object utilisation between the area-sorted and the random input has been between 5 and 7.6% for the problems tested (Table 8.1). Figure 8.2 shows the best layouts generated with the BLF- and the BLLT-algorithm input sequences that have been sorted by decreasing rectangle area. It should be noted that not all the unused objects are shown in Figure 8.2.

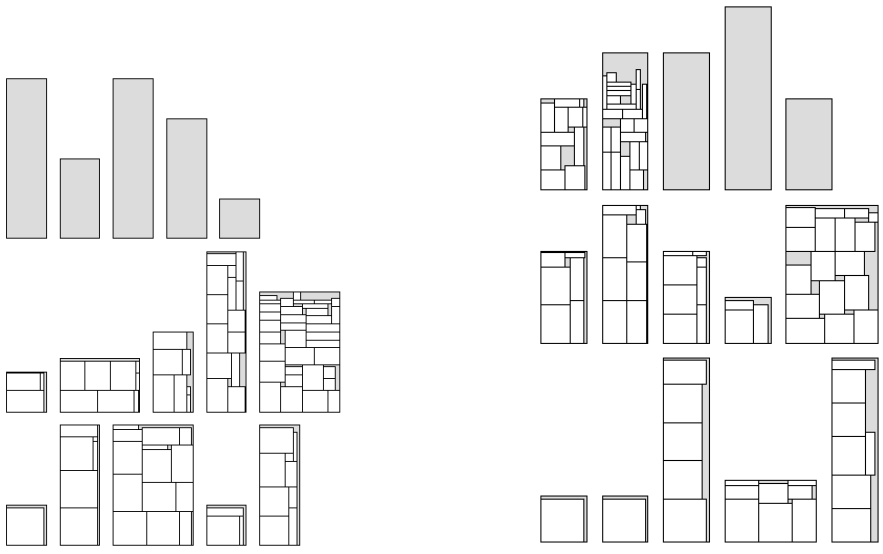


Figure 8.2: Best layouts for problem M2a using BLF (left) and BL (right) with area-sorted input (not all the unused objects are shown)

Time Performance

The computation time of the four algorithms and consequently the relative performance are different from the one observed in the strip packing problem (Figure 6.3). Table 8.2 summarises the elapsed times of the routines for 1000 runs. Unlike in the case of the strip packing problems used in chapter 6 the BLF-algorithm does not have the longest run time. As expected, the two sliding techniques (BL and BLLT) have similar elapsed times, which are lower than the one of the BLF-algorithm.

The BLF-routine operates faster than the third sliding technique, i.e. BLD-routine. Since overlap calculations are required in the BLD-routine, its time performance is poorer than that of the performance of the other two sliding techniques. Although the BLF-algorithm keeps track of the enclosed areas in the partial layout, this operation is still more efficient than the intersection checking carried out by the BLD-routine. This is mainly due to this type of 2D bin packing problem. Since the rectangle size is fairly large compared to the object size, the number of enclosed areas in the layout is usually small. In particular, since layout information is stored when a new item is positioned, intersection testing is still required but reduced to a minimum (section 5.5.4.5). A similar principle could be implemented in the BLD-routine in order to improve its time performance.

Table 8.2: Average elapsed time of simple packing routines for 1000 runs [s]

	BL	**BLLT**	**BLF**	**BLD**
M1	3.7	3.5	3.9	6.0
M2	3.6	4.0	4.4	6.2
M3	14.6	14.2	15.3	16.0

8.3 Comparison of the Evolutionary Approaches

In this section the performance of the evolutionary approaches is analysed. In order to investigate whether the implementation of the search process conducted by meta-heuristic methods works successfully, a comparison with random search is undertaken. Since the data structure for the bin packing problem consists of two chromosomes, which are manipulated separately by the operators of the meta-heuristic method, a series of experiments is conducted to establish the contribution of the cross-over and mutation operators to the search process.

8.3.1 Genetic Algorithms versus Random Search

The representation of the 2D bin packing problem in the hybrid algorithms consists of two permutations describing the order of the items and the objects. Whereas cross-over and mutation operators manipulate both sequences, the rotation operator is only applied to the items. In order to analyse if this implementation of genetic algorithms can search the solution space successfully, a random search

algorithm has been used to generate the permutations as well as the orientation of the items randomly. Both search processes have been simulated over the same number of iterations. Since the objective of this bin packing problem is to minimise the total area of the sets of objects used, the performance of the two methods is compared on the basis of the average object utilisation.

Table 8.3 summarises the relative difference between the outcome of the random search and the genetic algorithms for all decoder combinations. The negative numbers indicate that the average utilisation of the objects is up to 2.3% lower than with random search. This difference is smallest for the combination with the BLF-algorithm. Since this routine is based on a very efficient nesting technique that fills the enclosed areas in the partial layout, it achieves very good results when applied to random input (section 8.2).

Table 8.3: Relative difference between RS and GA result for average object utilisation [%]

	M1	**M2**	**M3**
BL	-1.9	-1.8	-1.1
BLLT	-2.3	-1.7	-1.1
BLF	-1.1	-1.3	-0.9
BLD	-2.1	-1.8	-1.0

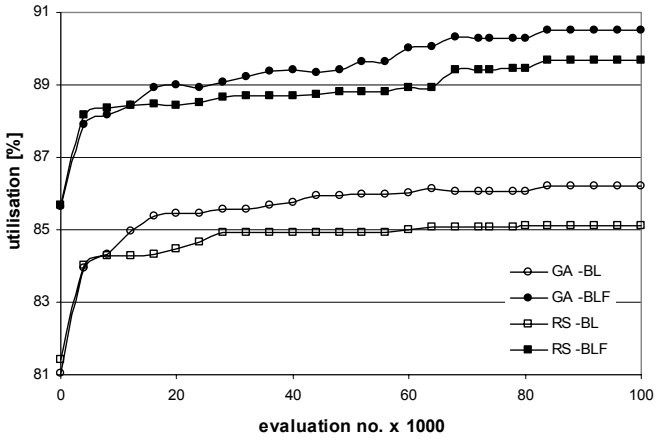


Figure 8.3: Average object utilisation with genetic algorithms and random search for problem M2a

Figure 8.3 illustrates the search process conducted by genetic algorithms and random search in combination with two different packing algorithms for one of the five test problems in the category M2. Similar utilisation values were obtained at the start of the search. The genetic algorithm outperforms the random process after roughly 10,000 iterations. Although the quality difference of the final solutions is very small, the genetic algorithms work consistently better than random search. This confirms that the genetic operators guide the search process towards better areas of the solution space than purely random exploration.

8.3.2 Genetic Algorithms versus Naïve Evolution

Unlike in the case of the strip packing two permutations are used in this implementation to describe the 2D bin packing problem. Three operators used by the genetic algorithms manipulate the item and object sequences. The cross-over operators are applied to both the item and object permutations, whereas the second mutation operator that flips the orientation by 90° is only used on the item string. In order to

analyse the impact of the cross-over operator on the final outcome; a naïve evolution algorithm has been applied which only uses the two mutation operators. The relative difference between the two approaches concerning the average object utilisation is summarised in Table 8.4. As the negative values indicate, the cross-over operators have a beneficial influence on the search process. The improvement of the average utilisation achieved with the genetic algorithms is usually less than 1%.

The similar performance of genetic algorithms and naïve evolution is reflected in the course of the search process (Figure 8.4). At the beginning genetic algorithms make faster progress since the cross-over operator allows them to explore the solution space better. Usually, within less than 50,000 iterations this lead becomes smaller and naïve evolution might even outperform genetic algorithms temporarily. All hybrid genetic algorithms achieve slightly better solutions when both methods converge. Since the genetic algorithms obtain better results for all test problems, this confirms a certain positive influence of the cross-over operators on the search process.

Table 8.4: Relative difference between NE and GA for average utilisation [%]

	M1	**M2**	**M3**
BL	-0.5	-0.9	0.1
BLLT	-0.7	-1.3	0.4
BLF	-0.4	-0.7	-0.1
BLD	-0.6	-0.8	0.4

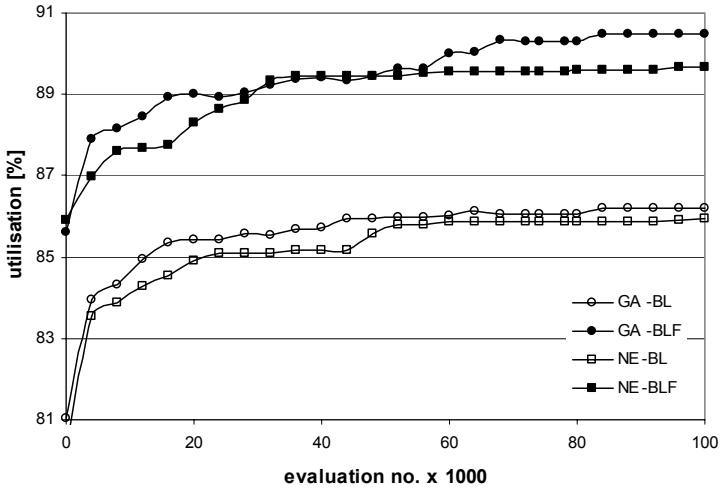


Figure 8.4: Average object utilisation with genetic algorithms and naïve evolution for problem M2a

In the above experiments, the genetic algorithms used the cross-over and mutation operators on both permutations. Since the genetic algorithms in this configuration achieve a better final outcome than naïve evolution, it is interesting to see to what extent the operators working on the object string contribute to this result. For this purpose the following two variations of the standard genetic algorithm have been tested. The probabilities for cross-over and mutation operation on the object string have been changed as described in the sequel leaving the ones for the item string unchanged (i.e. 60% and 3%).

- GA variant A: object string: probability of cross-over: 0%; probability of mutation: 0%;
- GA variant B: object string: probability of cross-over: 0%; probability of mutation: 3%;

The outcome of the two experiments is summarised in Table 8.5 with respect to the relative difference to the standard implementation of the genetic algorithm. The genetic algorithm, which neither uses cross-

over nor mutation on the object part of the chromosome (variant A), performs equally well or slightly worse than the standard version in most test cases. Although the difference is very small, the application of the genetic operators has an influence during the search process. In order to evaluate which of the two operators has the stronger impact, the mutation operation has been permitted in the second experiment (variant B). As Table 8.5 shows, this configuration performs equally well or slightly better than the standard one in most test cases.

Table 8.5: Relative difference between GA-variants and standard GA for average object utilisation [%]

	variant A				**variant B**			
	BL	**BLLT**	**BLF**	**BLD**	**BL**	**BLLT**	**BLF**	**BLD**
M1	0.0	-0.1	-0.1	-0.2	0.5	-0.3	0.0	0.1
M2	-0.2	0.1	0.0	-0.1	0.2	0.1	0.0	0.2
M3	-0.1	0.1	0.0	0.2	-0.1	0.4	0.1	-0.2

Varying the probabilities applied to cross-over and mutation on the object string has shown that the implementation without cross-over outperforms the standard one slightly. Applying only mutation has less a disruptive impact on the object sequence and is therefore better to guide the search process in this problem context. The overall influence of the application of the genetic operators on the object permutation is very small. Not using a cross-over operator on the item string has more impact on the outcome as the comparison with naïve evolution has shown (Table 8.4).

8.3.3 Comparison of the Hybrid Combinations for the Genetic Algorithms

The investigation of the rectangular strip packing problem has shown that the four packing routines used as decoders can be distinguished according to the final solution quality achieved in connection in the hybrid algorithms. As already established in the analysis of the simple packing algorithms (section 8.2), the BLF-algorithm generates better layouts than the sliding algorithms. The same ranking as in the case of strip packing has been found for bin packing. In order to compare the layout quality achieved with the various decoders against each other, the relative difference to the hybrid combination with the BL-decoder has been calculated for the other methods (Table 8.6). The ranking of the decoders has been the same for all test cases with the BLF-algorithm performing best. Among the sliding techniques the performance of the BLLT-routine is slightly better than the BL-decoder. The BLD-routine has achieved layouts whose quality has been more than 1% lower.

Table 8.6: Relative difference to GA + BL result for average utilisation [%]

	BLLT	BLF	BLD
M1	0.6	4.3	-1.2
M2	0.4	4.1	-1.2
M3	0.2	5.1	-1.4

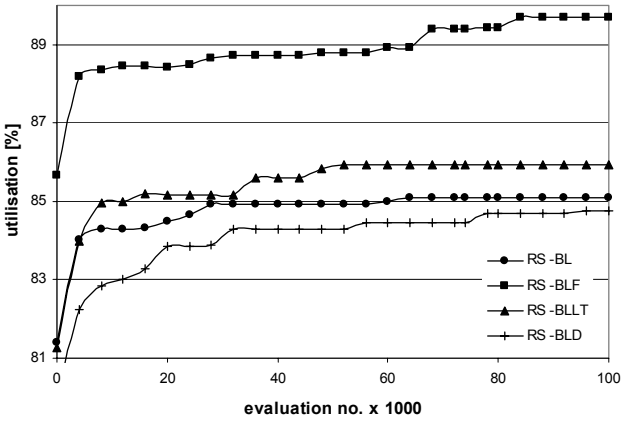


Figure 8.5: Average object utilisation with random search using various packing routines for problem M2a

Figure 8.5 reflects the close performance of the BL- and BLLT-algorithm during the course of the search for random inputs. Although both perform similarly at the start of the search, the BLLT-routine outperforms the BL-algorithm as the number of iterations increases. The overall performance of the BLD-technique remains low. The ranking of the decoders is the same as the one obtained with the genetic algorithms (Appendix H, Figure H.1). Even the search for a good sequence as performed by the genetic algorithm cannot change this. This confirms that the decoder is of paramount importance for the efficient operation of the hybrid algorithm. Searching for a better sequence in the case of the poorer decoder will therefore not result in outcomes better than those achieved with a more efficient packing algorithm.

The simple packing algorithms can outperform the hybrid genetic algorithms in this type of 2D bin packing problem under certain circumstances. It has been shown that sorting the rectangles by decreasing area or perimeter results in very good layouts (Appendix H, Table H.1 and H2). In Table 8.7 the average object utilisation is compared to the outcome of the respective hybrid genetic algorithms using the best of the four ordering criteria as stated. In all test cases the simple packing algorithms in combination with pre-ordered input performed better than the genetic algorithms. The object utilisation has been up to 6% higher. A similar observation concerning the performance of the simple packing algorithms with pre-ordered input sequences has been made for rectangular strip packing problems of comparable size (Table 6.3).

Table 8.7: Relative difference between simple packing algorithm (for best pre-ordered input) and GA with respect to average object utilisation [%]

	BL	BLLT	BLF	BLD
M1	3.0 (DRA)	2.6 (DRA)	3.2 (DRA)	3.9 (DRA)
M2	3.0 (DRA)	2.6 (DRA)	3.0 (DRA)	3.6 (DRA)
M3	4.7 (DH)	4.7 (DRA)	2.8 (DRA)	6.0 (DH)

The experiments above have shown that the combination between an efficient packing technique such as the BLF-algorithm and pre-ordered input can generate layouts of very high quality. Although the genetic algorithms in their current implementation outperform easily random sampling of the solution space, they cannot reach the performance of the simple packing algorithms with the sorted input. Since the simple routines achieve better solution quality and have shorter run times, the application of genetic algorithms is not appropriate for this type of 2D bin packing. It remains to be investigated whether other meta-heuristic methods may be more successful in this context.

8.4 Comparison of the Meta-Heuristic and Heuristic Approaches

The previous investigation of strip packing problems has shown that other meta-heuristic methods are powerful tools to approach packing tasks. In particular, simulated annealing has shown very good performance for rectangular problems by achieving better results than genetic algorithms. In order to evaluate the genetic algorithms in comparison with other meta-heuristic and heuristic methods for the 2D bin packing problem, simulated annealing and stochastic optimisation (Pargas and Jain, 1993) have been applied to the same benchmark problems. The performance of hill-climbing has also been investigated in this context.

8.4.1 Genetic Algorithms versus Simulated Annealing

As in the case of rectangular strip packing problems, the comparison between the two meta-heuristic methods shows that simulated annealing achieves better results than genetic algorithms. The relative difference in the average object utilisation can be up to 6.5% (Table 8.8). The outcome obtained with three sliding techniques is improved by a similar amount whereas the performance gain is lower for the more efficient BLF-decoder. The absolute values for the average object utilisation with the various meta-heuristic methods are summarised in Table H.3 and Table H.4 (Appendix H). The performance gain achieved by simulated annealing lies in the same range as in the case of rectangular strip packing. Again, the outcome of the sliding techniques could be improved by a higher percentage than the that of the combination with the BLF-routine (Table 6.9).

Table 8.8: Relative difference between SA and GA result for average utilisation [%]

	M1	M2	M3
BL	5.3	5.2	4.6
BLLT	4.3	5.3	4.8
BLF	2.2	3.3	2.3
BLD	6.2	6.5	5.0

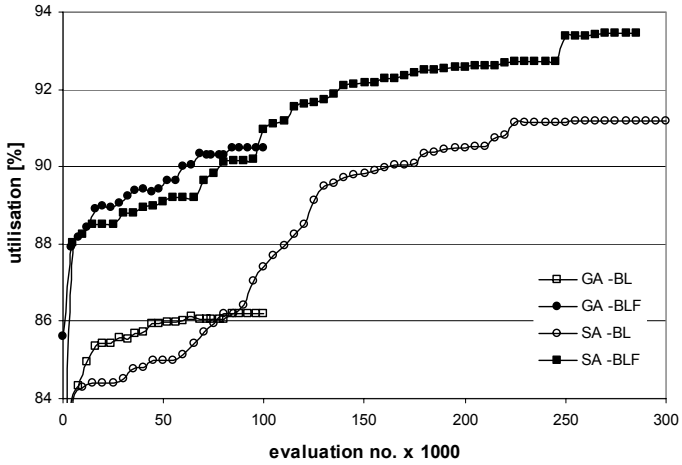


Figure 8.6: Average object utilisation with genetic algorithms and simulated annealing for problem M2a

The course of the search process conducted by both meta-heuristics is illustrated in Figure 8.6 for the combination with two decoders. Whereas the genetic algorithm progresses rapidly in the beginning of the search process, it gets outperformed by simulated annealing when it starts to converge on the final solution. The genetic algorithm has a certain advantage at the early stage of the search process. Since it is based on a population it can explore many distinct areas within the search space. The cross-over operator introduces more abrupt changes into the permutation than the neighbourhood operator of simulated annealing and contributes to the wide exploration of the solution space.

At the point where the convergence starts, the genetic algorithm cannot reach better solutions anymore and the population has a high average solution quality. The cross-over operator that applies larger changes in the permutation leads the search away from the good solutions rather than contributing to the progress. The neighbourhood operator in simulated annealing introduces the smoother changes manipulating only one feature at a time, for instance the rotation of a single rectangle. Therefore progress is still possible at this stage because good layouts can often be improved by minor manipulations such as the ones performed by the simulated annealing algorithm. Especially, at the later stage of the search the algorithms accept almost only better states in the search space. As a consequence of this, the current solution is operated upon until a better or equal one is found. Rarely a solution of minor quality is accepted.

Although genetic algorithms also contain a neighbourhood operator in the form of the mutation operator, it is not constantly applied on a certain solution and only with a low probability. If a good individual is not selected into the mating pool or manipulated by the cross-over operator, it may get lost in the next generation. This is not the case in simulated annealing which works on one solution only.

The comparison between simulated annealing and genetic algorithms refers to their standard implementation. Since the number of iterations in simulated annealing is higher (Appendix H, Table H.5),

it can be argued whether genetic algorithms may be more successful using more iterations. To investigate this the genetic algorithm has been used in combination with a population size of 200 over 1500 generations resulting in the same number of iterations as applied in simulated annealing. The relative difference between the outcome by the simulated annealing and genetic algorithm is presented in Table 8.9.

Despite the same number of iterations, simulated annealing achieves higher packing densities than the genetic algorithm. The difference is lower than when the standard implementation of the genetic algorithm was used (Table 8.8). The same experiment has been carried out with random search using 300,000 iterations. Table 8.10 shows that genetic algorithms achieve layouts, which are up to 2.7% denser. As this figure is higher than in the case of the standard implementation (Table 8.3), genetic algorithms search the solution space more fully than random search at higher iteration numbers.

Table 8.9: Relative difference between SA and GA result for average utilisation [%]; GA with population size of 200

	M1	M2	M3
BL	3.6	3.8	3.1
BLLT	3.0	3.9	3.0
BLF	1.2	2.2	1.3

Table 8.10: Relative difference between RS and GA result for average utilisation [%]; GA with population size of 200;
RS with 300,000 iterations

	M1	M2	M3
BL	-2.7	-2.4	-1.9
BLLT	-2.6	-2.5	-2.0
BLF	-1.5	-1.9	-1.6

8.4.2 Genetic Algorithms versus Stochastic Optimisation

The comparison with genetic algorithms has shown that simulated annealing has a very slow convergence rate. Especially at the beginning of the search, when the algorithm explores the search space simulated annealing has a certain disadvantage since it only operates on one solution at a time. The cross-over operator rather limits the search process conducted by the genetic algorithms at a certain stage, when good solutions have been found. Therefore stochastic optimisation has been applied to the 2D bin packing algorithm as a fourth meta-heuristic approach. This method applies features from genetic algorithms and simulated annealing (Pargas and Jain, 1993).

The relative difference in the average object utilisation between genetic algorithms and stochastic optimisation is stated in Table 8.11 for the three problem categories and the various decoder combinations. The stochastic algorithm slightly outperforms the genetic algorithm for two out of three problems, although the difference is less than 1% for most cases investigated.

Table 8.11: Relative difference between SO and GA result for average utilisation [%]

	M1	M2	M3
BL	-0.6	0.7	1.1
BLLT	-1.2	0.4	0.9
BLF	-0.4	0.2	0.6
BLD	-0.6	1.1	1.2

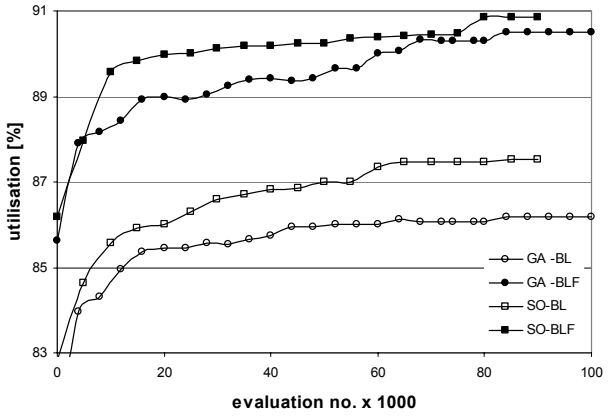


Figure 8.7: Average object utilisation with genetic algorithms and stochastic optimisation for problem M2a

Figure 8.7 shows the search process conducted by the two meta-heuristics for one problem of category M2. The stochastic optimisation algorithm makes better progress in the beginning of the search and usually can maintain the lead. The performance difference for the hybrid method, which achieves the best absolute results, i.e. the combination with the BLF-routine, is very low and only around 0.5% better. Therefore the stochastic optimisation method in this implementation cannot be considered as a real alternative to the genetic algorithms.

8.4.3 Meta-Heuristics and Local Search

In order to establish the advantages of meta-heuristic search over a local search algorithm in the context of 2D bin packing, hill-climbing has been applied to the same test problems. As the relative difference to the genetic algorithms shows, the object utilisation obtained by the local search method is lower. The difference can be up to 6.5% and is slightly smaller for larger problems (Table 8.12).

Table 8.12: Relative difference between HC and GA result for average utilisation [%]

	M1	M2	M3
BL	-4.9	-4.2	-1.9
BLLT	-6.4	-4.3	-3.2
BLF	-3.2	-3.9	-2.2
BLD	-5.2	-4.3	-2.8

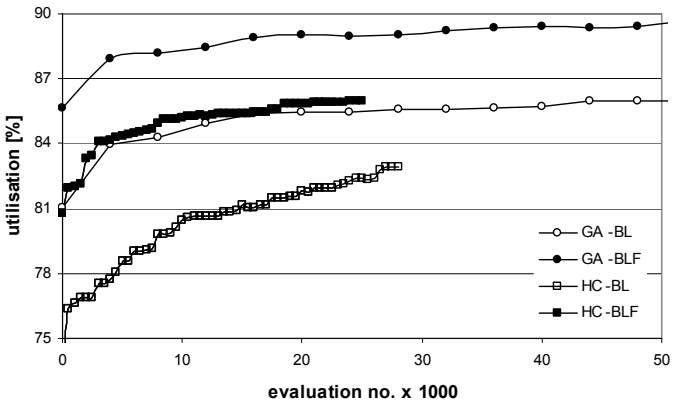


Figure 8.8: Average object utilisation with genetic algorithms and hill-climbing for problem M2a

Hill-climbing cannot compete with the outcome of random search (Appendix H in Table H.3 and Table H.4.) in any of the three test categories. This indicates how uneven the search space is. Since the hill-climbing algorithm can only progress when a better solution is found, it gets easily trapped in local optimum solutions. The fact that most of these local optima are worse than solutions obtained by pure random sampling of the search space shows that the solution space of this packing problem has a very "rough surface". Therefore only meta-heuristics with means to escape local minima can be successful in this type of environment (Figure 8.8).

8.4.4 Comparison of the Meta-Heuristic and Heuristic Search Methods

The comparison between the various meta-heuristic and heuristic approaches to the 2D bin packing problem has shown that simulated annealing performs best. The major advantage of simulated annealing is the neighbourhood operator that only introduces moderate changes into layouts when moving to the next state in the search space. This technique is more successful in the context of 2D packing than search space manipulation techniques used by other meta-heuristics. In order to base the performance comparison on the same conditions, genetic algorithms and random search have been simulated over a higher number of iterations (Table 8.9 and Table 8.10). Although this could improve the outcome obtained with the standard implementation, the layout quality achieved with simulated annealing could not be reached. This confirms the success of the neighbourhood operation for this implementation. Figure 8.9 highlights the course of the search process for the meta-heuristic, heuristic and random search techniques.

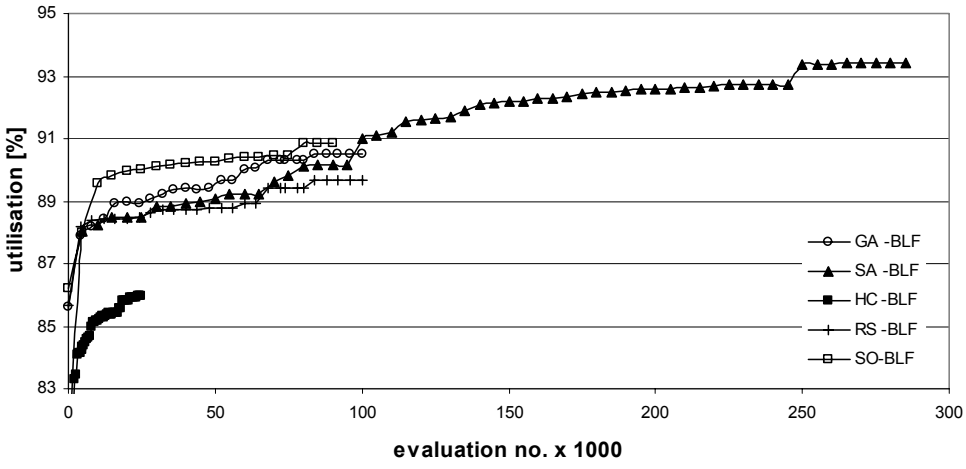


Figure 8.9: Average object utilisation with meta-heuristics, hill-climbing and random search for problem M2a

Simulated annealing outperforms the other meta-heuristic and heuristic methods, and also simple packing routines that achieve very good layouts using pre-ordered input (section 8.3.3). The relative difference to the best outcome that has been obtained with pre-ordered input as stated in Table 8.13. Depending on the

problem the simulated annealing algorithm achieves an average increase in object utilisation by around 4%. Even though the BLF-routine is the most efficient packing technique concerning the total layout quality, only its performance is considered in the following.

Table 8.13: Relative difference between SA and best result by simple packing routine (with pre-ordered input) for average utilisation [%]

	BL	**BLLT**	**BLF**	**BLD**
M1	4.2	4.0	0.1	4.4
M2	4.0	4.5	1.6	4.7
M3	1.0	1.2	0.4	0.1

Compared to the simple BLF-routine, simulated annealing performs only around 1% better. This improvement is comparatively low regarding the higher computational effort (Appendix H, Table H.6 and Table H.7). Whereas the outcome of the simple routine has been achieved with only 1000 iterations, the meta-heuristic search process used 300 times more to achieve an outcome which is less than 2% better regarding the average object utilisation. Whether this marginal improvement in terms of layout quality can justify the higher computational effort depends on the industrial application. Under certain circumstances where time is not a limiting factor and layout computation can be done off-line, it may be well worth the effort to invest additional simulation time in order to achieve a better layout. Figure 8.10 and Figure 8.11 show the best layouts achieved with genetic algorithms and simulated annealing using two different decoders.

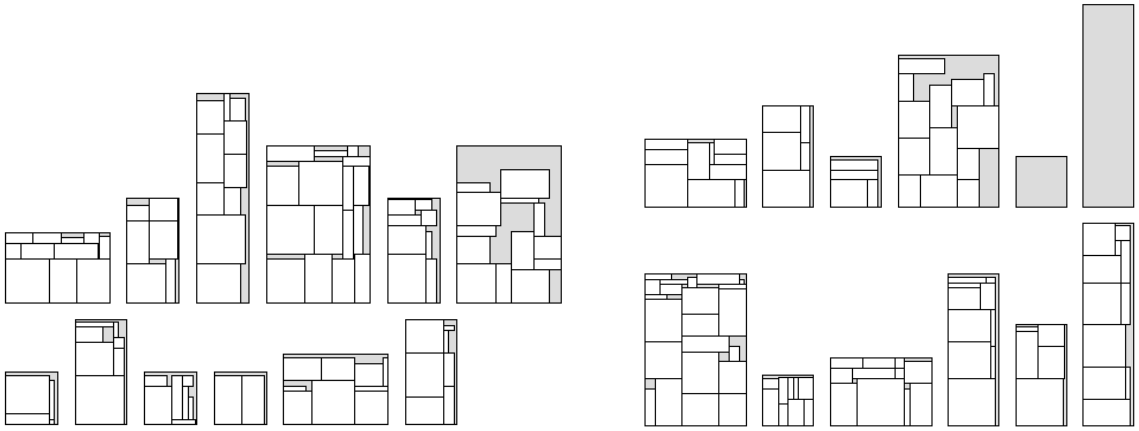


Figure 8.10: Best layouts obtained with GA+BL and SA+BL for problem M2a (not all empty objects are shown)

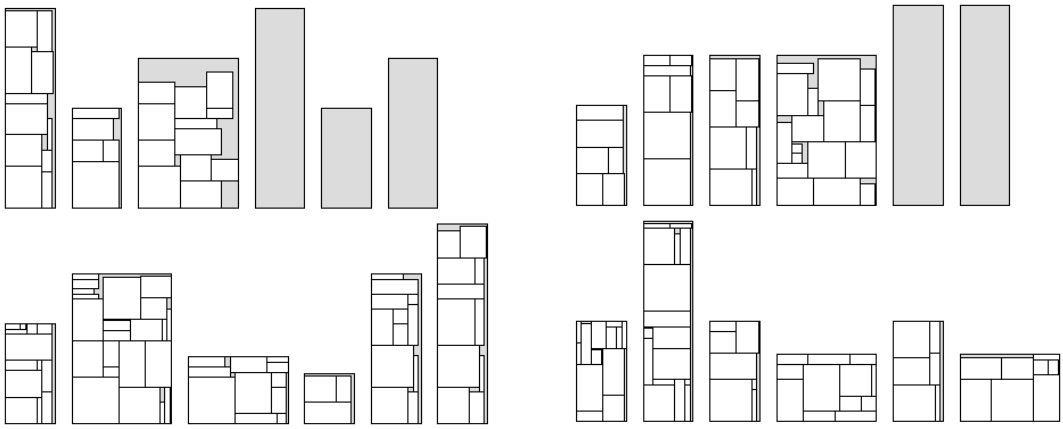


Figure 8.11: Best layouts obtained with GA+BLF and SA+BLF for problem M2a (not all empty objects are shown)

8.5 Comparison with Commercial Nesting Software

As in the case of rectangular strip packing, the packing software Nestlib (section 4.5) has been applied to the benchmark problems used in the previous sections. Table 8.14 summarises the average object utilisation obtained with Nestlib as well as with the simple and hybrid algorithms developed in this work.

Table 8.14: Object utilisation obtained with GA, SA and Nestlib [%]

	M1	**M2**	**M3**
Nestlib	94.4	90.8	90.6
GA	95.8	91.7	92.9
SA	97.9	94.8	95.0
RS	94.7	90.5	92.1
BLF- algorithm	97.8	93.3	94.6
sorting	DRA	DRA	DRA

The comparison shows that the meta-heuristic and the simple techniques achieve better layouts than the commercial product. Although Nestlib has been shown to nest individual sheets extremely efficiently (Table 6.14) and sometimes to optimality as shown in the sequel, it does not optimise for the best set of objects. The package uses the objects according to the order in which they are fed into the system. The meta-heuristics as well as the simple heuristics have this additional flexibility.

Therefore in a layout obtained with Nestlib usually the last object has the lowest utilisation because it contains the items which could not be packed into any of the previous ones. Depending on the size of the last sheet, this approach can be very wasteful. This can be seen in Table H.8 in Appendix H, which lists the object utilisation that has been obtained with Nestlib, the BLF-routine with area-sorted input and the genetic algorithm for the individual objects in two packing tasks. In the Nestlib layout, the utilisation of

the last object has the smallest value and contains the rectangles, which have not fit into the other bins. Some objects have a utilisation of 100%, which shows that the packing algorithm implemented in Nestlib is very efficient, especially for the second problem (M2d). Many of the other objects have an utilisation rate of above 95%. The simple BLF-routine in connection with area-sorted input and randomly generated object sequences is capable of nesting a number of bins to optimality. Dense layouts are achieved because it can nest into enclosed areas in the partial layout.

Sorting according to decreasing rectangle area guarantees that a set of small rectangles is left for filling these holes towards the end of the packing process. Regarding the first problem, i.e. M2b, fewer objects have a utilisation above 95%, which shows that the BLF-algorithm is not so efficient as the packing algorithm used by the commercial package. The genetic algorithm achieves less optimally nested objects. Compared to the simple BLF-routine this is due to the fact this it is virtually impossible for the genetic algorithm to encounter area-sorted sequences. On average the object utilisation achieved by the genetic algorithm is higher because searching for a set of objects to fit all items reduces the total object size. The commercial algorithm is restricted in this respect and nests the objects in the input order.

The remaining area of the last object is not wasted in a strict sense, since it can be used for future nesting processes. The processing of the remainder adds additional cost with respect to handling and stock. The implementation of the meta-heuristic hybrid algorithms attempts to find a set of objects with every object being densely packed and therefore chooses an object set which usually does not contain large remaining areas. Table H.8 (Appendix H) shows that in the first problem M2b all objects have a utilisation of 90% or higher with one exception, i.e. the layout generated by Nestlib.

This comparison has shown that the application of meta-heuristic produces efficient layouts for the 2D bin packing problem where the objective is to reduce the total area of the objects used. Out of the three different approaches including commercial software, simple packing algorithms and meta-heuristic methods, the best packing strategy for the 2D bin packing problem is simulated annealing. Since the run times are comparatively long (Table 8.2 and Appendix H, Table H.7), the application of the BLF-routine with pre-ordered input can be regarded as the best compromise.

8.6 Conclusions

The concept behind the hybrid algorithms developed for the rectangular strip packing problem have been extended to the 2D bin packing task involving multiple objects. Experiments on a number of different sized nesting tasks have shown that simple packing algorithms are a powerful tool for the 2D bin packing problem when they are applied in combination with pre-sorted input. Among the various packing algorithms, the BLF-routine achieves the best results due to its capability of nesting into enclosed areas in the partial layout. Sorting the items according to decreasing rectangle area or perimeter has been shown to generally perform best.

The layouts generated by the BLF-routine using pre-ordered input outperforms the hybrid genetic algorithms regarding the test problems used. The only meta-heuristic method, which is capable of achieving layouts of higher quality, is simulated annealing. It has been shown that the performance of local search is extremely poor because the algorithm easily gets trapped in local optima because of the very uneven search space. Especially since even simple packing algorithms generate layouts of much higher quality the application of a local search technique is not appropriate in the context of 2D bin packing. Although all meta-heuristic search methods have means to escape local optima, the neighbourhood operator used by simulated annealing introduces rather small layout changes and has demonstrated to work more successfully in the context of an order-based data structure applied to rectangular packing.

The major drawback of simulated annealing is the long simulation time. Depending on the major objective of the industrial nesting task, the application of meta-heuristics may be justified in order to obtain a marginal improvement over a simple approach. It is inevitable to consider commercial packages as well in this area since they use nesting algorithms that are efficient in terms of solution quality and computation time. With respect to the 2D bin packing problem, the commercial product tested has been found to be very limited since it does not optimise the object set to be used in the nesting task.

9. Conclusions and Future Work

9.1 Conclusions

In this work a series of hybrid approaches have been developed for the solution of cutting and packing problems including 2D rectangular and irregular strip packing and 2D bin packing. Since cutting and packing tasks are combinatorial problems with very large search spaces the recent literature encourages the use of genetic algorithms in particular, along with other meta-heuristic methods for their solution. The focus of this investigation has been on the application of genetic algorithms.

Despite the substantial effort in algorithm development, many new approaches in the literature are not compared with existing methods. Usually comparisons are only made between different algorithm implementations or meta-heuristic and heuristic search methods that use the same concept. Whereas a few comparisons have been made recently for rectangular strip packing, for the majority of the irregular packing approaches in the literature this is not the case. This is mainly due to the lack of suitable benchmark problems in the area. Therefore, it is difficult to evaluate the overall performance of existing as well as new solution methods.

In order to fill this gap as well as to allow the testing of the methods developed in this work, new problem generators have been implemented for the creation of rectangular strip and bin packing problems. For the analysis of the performance some test problems from the literature have been used. In addition to that, an efficient heuristic packing method has been proposed (i.e. BLF-routine) which is suitable as a benchmark algorithm for rectangular strip packing. It achieves layouts of very high density and is not only simple to implement, but also has a low computation time. The application as a benchmark test would facilitate the performance comparison with newly developed algorithms. Although a large number of algorithms in the area are demonstrated using industrial data, they are not compared with commercial nesting packages which are especially designed for industrial requirements. The performance analysis has also included commercial nesting software.

With respect to the performance analysis of meta-heuristic algorithms, two search methods have been suggested and applied throughout this project. The application of random search allows identifying how well the manipulation operators used in heuristic and meta-heuristic search guide the search process. In order to analyse the performance of the cross-over operator, naïve evolution is used. A consequent application of these two methods contributes to establish their performance assessment of newly developed meta-heuristic algorithms better than done in the literature to-date.

The packing methods developed in this work are based on a hybrid concept that uses an order-based meta-heuristic and a simple packing algorithm in the decoding stage. In the three packing tasks

investigated simple packing algorithms have been found to generate very good layouts using pre-ordered input. In particular for rectangular packing problems, they can outperform the meta-heuristic methods for nesting tasks above a certain problem size. For smaller packing tasks genetic algorithms achieve better results than the simple routines. With regard to the three different packing categories investigated, the cross-over operation is of minor importance to the search process. Mutation like operators as used in naïve evolution and stochastic optimisation are able to guide the algorithm through the solution space successfully and find solutions with a similar quality to genetic algorithms.

Among the meta-heuristic methods simulated annealing achieves better packing densities than genetic algorithms. Whereas the improvement has been small for irregular strip packing problems, the performance gain for both rectangular packing tasks lies in the range of a few percent with respect to packing heights. The neighbourhood operation has been analysed as a powerful feature in the simulated annealing algorithm with respect to packing. Since it introduces only small alterations in the layout at each change of state, slow but continuous improvement of the current state are possible once good solutions have been discovered. The suitability of meta-heuristics for packing problems has been investigated with respect to local search techniques. Despite the shorter run time local search cannot be considered as an alternative to meta-heuristics. The fact that pure random sampling can achieve better final outcomes than heuristic search techniques shows how uneven the search space is. A means to escape local minima is required during the search process as provided by meta-heuristic methods.

As the comparison with approaches in the literature demonstrated, the set of hybrid algorithms is capable of obtaining layouts of similar or better solution quality. This is particularly the case for the rectangular strip packing problems where a direct comparison has been possible due to the provision of some benchmark problems. A similar outcome has been obtained for the performance evaluation of the irregular nesting tasks. Since the testing has not been based on the original problems but on an approximation, the evaluation is best made in relative terms. It was shown that the meta-heuristic hybrid algorithms usually achieve layouts of similar or better quality for the problems tested, which originate from a variety of applications. The hybrid concept of this method can be regarded as successful compared with other approaches that operate directly on the layout rather than using an encoding technique to represent the problem.

Comparing the meta-heuristic solution approaches of this work as well as the ones from the literature with the nesting packages shows that the commercial algorithms operate extremely efficiently in terms of solution quality and computation time. For most rectangular strip and bin packing problems the hybrid algorithms could outperform the commercial algorithms. In particular, the additional flexibility implemented in the meta-heuristics proved successful maximising the object usage for the 2D bin packing problem. Commercial software has a clear advantage in certain irregular nesting tasks with items that are likely to cause larger enclosed areas in the partial layout. Since commercial packages include features that allow nesting into these gaps, they outperform the hybrid techniques whose decoding algorithms are

based on sliding principles. This confirms the paramount importance of an efficient nesting algorithm in a hybrid solution approach. For other problems composed of different shapes such as nesting tasks from the textile industry, the hybrid algorithms have achieved similar layout quality as the commercial software.

The major disadvantage of meta-heuristics is their comparatively long computation time. Whereas the application of meta-heuristics can often be justified for smaller rectangle problems where they achieve better solutions than simple techniques, the situation is different for larger item numbers. In this case simple packing heuristics not only achieve comparable or better layouts, but also require far less computation time. They also become competitive when time is an important issue in industrial packing tasks and their solution quality is only marginally lower than the one for the meta-heuristics. In the decision making process, a careful evaluation of the major objectives of the nesting task is important, which entirely depends on the industrial application. With respect to irregular packing problems, the hybrid algorithms have extremely long computation times and do not necessarily achieve better results than heuristic search techniques or the algorithms applied in commercial nesting packages. Hence meta-heuristics cannot be regarded as an alternative in irregular strip packing at the moment. At least with computational power constantly increasing computation time should become less of an issue in the future.

9.2 Future Work

One of the major obstacles in algorithm development in the area of cutting and packing is the lack of benchmark problems. Unlike many other areas of computer science and operational research, benchmark problems for 2D packing problems are generally not available and agreed upon. Whereas it is possible to obtain some test data from publications for rectangular packing tasks, there is a clear need for benchmarks for the irregular cases. Since they originate from many different applications, their special characteristics need to be elaborated and classified. A comprehensive classification is a prerequisite for the design of problem generators. Automatic problem generators facilitate the generation of test problems to a great extent allowing the use of a number of parameters that describe the packing task. Benchmark problems will make performance comparisons between different packing methods easier and contribute to algorithm development. They will assist the user of packing algorithms to select the most appropriate one for a certain nesting task.

This investigation has shown that meta-heuristic search techniques are capable of generating layouts of better quality than the ones computed by other heuristic nesting algorithms. The large number of function evaluations required during the search process is one of the major drawbacks of meta-heuristic methods. In particular, their application to packing problems leads to long computation time, since complex and expensive geometric calculations are required during the evaluation stage. An efficient implementation of the geometric algorithms can reduce the overall run times to a large extent. Techniques like hashing applied to limit the number of re-calculations in the layout generation point in this direction. So far, for

many industrial applications a trade-off between solution quality and computation time is necessary. Lower computation time will certainly make meta-heuristics more competitive when compared to other algorithms and therefore more attractive for industrial applications.

A considerable quantity of work remains to be done in the area of tabu search. Only a few researchers have used tabu search in cutting and packing so far. At the same time its performance in comparison with other meta-heuristic techniques needs to be established. Where this has been done in the literature, the conclusions vary.

Computation time is not the only issue for algorithm development in the area of packing. The quality of the layouts is also decisive for selecting certain packing techniques. Especially in irregular tasks, the quality can be improved with the application of more efficient nesting algorithms. The sliding algorithms used as decoders in the meta-heuristic hybrid algorithms are limited concerning the nesting into empty areas in the partial layout, or in the items themselves. More sophisticated packing techniques can overcome these problems. As the example of the rectangular strip packing task demonstrated, the routine that accesses enclosed areas in a layout easily outperformed the other methods. Techniques such as in-shape-nesting, in-hole-nesting, grouping and pairing of identical or similar items offer advantages for efficient nesting and have long been utilised for commercial products. Similar techniques need to be implemented in the decoding algorithms used by the meta-heuristics to achieve better solution quality.

10. References and Bibliography

Adamowicz M. and Albano A., 1976a. Nesting two-dimensional shapes in rectangular modules. Computer Aided Design 8, 27-33.

Adamowicz M. and Albano A., 1976b. A solution of the rectangular cutting-stock problem. IEEE Transactions on Systems, Man and Cybernetics, SMC-6, 302-310.

Albano A., 1977. A method to improve two-dimensional layout. Computer Aided Design 9, 48-52.

Albano A. and Orsini R., 1980. Heuristic solution for the rectangular stock cutting problem. Computer Journal 23, 338-343.

Albano A. and Sappupo G., 1980. Optimal Allocation of two-dimensional irregular shapes using heuristic search methods. IEEE Transactions on Systems, Man and Cybernetics, SMC-10, 242-248.

Amaral C., Bernardo J. and Jorge J., 1991. Marker-making using automatic placement of irregular shapes for the garment industry, Computers & Graphics 14, 41-46.

András P., András, A. and Zsuzsa, S., 1996. A genetic solution for the cutting stock problem. In: Proceedings of the First On-line Workshop on Soft Computing, Aug. 1996, Nagoya University, pp. 87-92.

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

Bäck T., Hammel U. and Schwefel H. P., 1997. Evolutionary computation: Comments on the history and current state. IEEE Transactions on Evolutionary Computation 1, 3-17.

Baker, J. E., 1985. Adaptive selection methods for genetic algorithms. In: Proceedings of the 1st International Conference on Genetic Algorithms, pp. 101-111.

Barequet, G., Bidgeman, S. S., Duncan C. A., Goorich, M. T. and Tanassia R., 1997. Classical computational geometry in GeomNet. In: Proceedings of the 13th Annual Symposium on Computational Geometry, pp. 412 - 414.

Batchelor B. G., 1991. Intelligent Image Processing in Prolog, Springer-Verlag, London, 195-205.

Beasley, D., Bull D. R. and Martin R. R., 1993. An overview of genetic algorithms: part 1, fundamentals. University Computing, 15, 1-16.

Beasley, D., Bull D. R. and Martin R. R., 1993. An overview of genetic algorithms: part 2, research topics, fundamentals. University Computing 15, 1-15.

Beasley J. E., 1985. An algorithm for the two-dimensional assortment problem. European Journal of Operational Research 19, 253-261.

Beasley J. E., 1990. OR-Library: distributing test problems by electronic mail, Journal of the Operational Research Society 41, 1069-1072.

Bengtsson B. E., 1982. Packing rectangular pieces – a heuristic approach. Computer Journal 25, 353-357.

Berkey J. O. and Wang P. Y., 1987. Two-dimensional finite bin-packing algorithms. Journal of the Operational Research Society 38, 423-429.

Bischoff E. and Mariott M. D., 1990. A comparative evaluation of heuristics for container loading. European Journal of Operational Research 44, 267-276.

Bischoff E. E., Janetz F. and Ratcliff M. S. W., 1995. Loading pallets with non-identical items. European Journal of Operational Research 84, 681-692.

Blazewicz J., Hawryluk P. and Walkowiak R., 1993. Using a tabu search approach for solving the two-dimensional irregular cutting problem. Annals of Operations Research 41, 313-327.

Bounsaythip C., Maouche S. and Neus M., 1995. Evolutionary search techniques application in automated lay-planning optimization problem. In: Proceedings of the IEEE Conference on SMC, pp. 4497-4502.

Bounsaythip C. and Maouche S., 1997. Irregular shape nesting and placing with evolutionary approach, In: Proceedings of the IEEE International Conference On Systems, Man and Cybernetics, vol. 4, pp. 3425-3430.

Bortfeldt A., 1994. A genetic algorithm for the container loading problem. In: Rayward-Smith (ed.), Proceedings of the Unicom Seminar on Adaptive Computing and Information Processing, pp. 749-757.

Burke E. and Kendall G., 1999a. Comparison of Meta-Heuristic Algorithms for Clustering Rectangles. Computers in Engineering 37, 383-386.

Burke E. and Kendall G., 1999b. Applying Simulated Annealing and the No Fit Polygon to the Nesting Problem. Proceedings of the World Manufacturing Congress, Durham, UK, pp. 27-30.

Carpenter H. and Dowsland W. B., 1985. Practical considerations of the pallet-loading problem, Journal of the Operational Research Society 36, 489-497.

Catastini A., Cavagna C., Cugini U. and Moro P., 1976. A computer-aided design system for pattern grading and marker making in the garment industry. In: CAD76 Proceedings of the Second International Conference on Computers in Engineering and Building Design, Imperial College London, March 76, IPC Science and Technology Press, Guildford, pp.159 - 165.

Chazelle B., 1983. The Bottom-Left Bin-Packing Heuristic: An Efficient Implementation. IEEE Transactions on Computers c32/8, 697-707.

Cheok B. T. and Nee A. Y. C., 1991. Algorithms for nesting of ship/ offshore structural plates, ASME, DE-Vol. 32-2, Advances in Design Automation 2, 221-226.

Christofides N. and Hadjiconstantinou E., 1995. An exact algorithm for orthogonal 2D cutting problems using guillotine cuts. European Journal of Operational Research 83, 21-38.

Coffman, E. G., Garey, M. R. and Johnson, D. S., 1984. Approximation algorithms for bin-packing - an updated survey. In: Ausiello, G., Lucertini, M. and Serafini, P. (eds.), Algorithms Design for Computer Systems Design, Springer, Vienna, pp. 49-106.

Coffman E. G. and Shor P. W., 1990. Average-case analysis of cutting and packing in two dimensions. European Journal of Operational Research 44, 134-144.

Corcoran A. L. and Wainwright R. L., 1992. Genetic algorithm for packing in three dimensions. In: Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing SAC '92, Kansas City, pp. 1021-1030.

Corno F., Prinetto P., Rebaudengo M. and Sonza Reorda M., 1997. Optimising area loss in flat glass cutting. In: Proceedings of Second International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA '97, University of Strathclyde, Glasgow, UK, pp. 450-455.

Dagli C. H. and Poshyanonda P., 1997. New approaches to nesting rectangular patterns. Journal of Intelligent Manufacturing 8, 177-190.

Dagli C. H. and Tatoglu M. Y., 1987. An approach to two-dimensional cutting stock problems. International Journal of Production Research 25, 175-190.

Dantzig, G. B., 1951. Maximisation of a linear function of variables subject to linear inequalities. Activity analysis of production allocation. In: T. C. Koopmans (Ed.), Cowles Commission Monograph 13. Wiley, New York, pp.339-347.

Dighe R. and Jakiela M. J., 1996. Solving Pattern Nesting Problems with Genetic Algorithms Employing Task Decomposition and Contact Detection. Evolutionary Computation 3, 239-266.

Dori D. and Ben-Bassat M., 1984. Efficient nesting of congruent convex figures. Communications of the ACM 27, 228-235.

Davis L., 1985. Applying adaptive search algorithms to epistatic domains. In: Proceedings of the 9th International Joint Conference on Artificial Intelligence, Los Angeles, pp. 162-164.

Davis L., 1991. Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York.

Dietrich R. D. and Yakowitz S. J., 1991. Rule-based approach to the trim-loss problem, International Journal of Production Research. 29, 401-415.

Dowsland K. A., 1984. The three-dimensional pallet chart: an analysis of the factors affecting the set of feasible layouts for a class of two-dimensional packing problems. Journal of the Operational Research Society 35, 895-905.

Dowsland K. A., 1987. An exact algorithm for the pallet loading problem. European Journal of Operational Research 31, 78-84.

Dowsland, K.A., 1993. Some experiments with simulated annealing techniques for packing problems. European Journal of Operational Research 68, 389-399.

Dowsland W. B., 1985. Two and three dimensional packing problems and solution methods. New Zealand Journal of Operational Research 13, 1-18.

Dowsland W. B., 1991. Three-dimensional packing-solution approaches and heuristic development. International Journal of Production Research 29, 1673-1685.

Dowsland K. A. and Dowsland W. B., 1992. Packing problems. European Journal of Operational Research 56, 2-14.

Dowsland K. A. and Dowsland W. B., 1995. Solution approaches to irregular nesting problems. European Journal of Operational Research 84, 506-521.

Dowsland K. A., Dowsland W. B. and Bennell J. A., 1998. Jostling for position: local improvement for irregular cutting patterns. Journal of the Operational Research Society 49, 647-658.

Dyckhoff H., 1981. A new linear programming approach to the cutting stock problem. Operations Research 29, 1092-1104.

Dyckhoff H., 1990. Typology of cutting and packing problems. European Journal of Operational Research 44, 145-159.

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

Eglese R. W., 1990. Simulated annealing. A tool for operational research. European Journal of Operational Research 46, 271-281.

Epstein, P., Kavanagh, J., Knight A., May, J., Nguyen T. and Sack J. R., 1994. A workbench for computational geometry. Algorithmica 11, 404-428.

Fabri A., Giezeman G. J. Kettner L., Schirra S. and Schoenherr S., 1996. The CGAL Kernel: A Basis for Geometric Computation. In: Proceedings Workshop on Applied Computational Geometry, May 27-28, 1996, Philadelphia, Pennsylvania.

Faina L., 1999. Application of simulated annealing to the cutting stock problem. European Journal of Operational Research 114, 542-556.

Falkenauer E., 1996. Hybrid grouping genetic algorithm for bin packing. Journal of Heuristics 2, 5-30.

Falkenauer E., 1998. Genetic algorithms and grouping problems. John Wiley & Sons, Chichester.

Falkenauer E. and Delchambre A., 1992. A genetic algorithm for bin-packing and line balancing. In: Proceedings of the 1992 IEEE International Conference on Robotics and Automation, Nice, Fr, vol. 2, pp. 1186-1192.

Fogel, D., 1998. Evolutionary computation: The fossil record. IEEE Press, New York.

Fogel, L. J., Owens A. J. and Walsh M. J., 1966. Artificial intelligence through simulated evolution, Wiley, New York.

Fowler, R. J., Paterson, M. S. and Tatimoto, S. L., 1981. Optimal packing and covering in the plane are NP-complete. Information Processing Letters 12, 133-137.

Freeman H. and Shapira R., 1975. Determining the minimum-area encasing rectangle for an arbitrary closed curve. ACM, Management Science/ Operations Research 18, 409-413.

Frenk J. B.G. and Galambos G., 1987. Hybrid next-fit algorithm for the two-dimensional rectangle bin-packing problem. Computing 39, 201-217.

Fujita K., Akagji, S. and Kirokawa, N. 1993. Hybrid approach for optimal nesting using a genetic algorithm and a local minimisation algorithm. Proceedings of the 19th Annual ASME Design Automation Conference, Part 1 (of 2), Albuquerque, NM, USA, vol. 65, part 1, pp. 477-484.

Garey, M. R. and Johnson D. S., 1979. Computers and Interactability: A guide to the theory of NP-completeness. W.H. Freeman and Company, San Francisco.

Gary Parker R., 1995. Deterministic Scheduling Theory. Chapman Hall.

George J. A., 1992. Method for solving container packing for a single size of box. Journal of the Operational Research Society 43, 307-312.

George J. A., George, J. M. and Lamar, B. W., 1995. Packing different-sized circles into a rectangular container. European Journal of Operational Research 84, 693-712.

Giezeman G.-J., 1994. PlaGeo, a library for planar geometry, and SpaGeo, a library for spatial geometry, http://www.cs.uu.nl/people/geert/DOC/

Gilmore P. C. and Gomory R. E., 1961. A linear programming approach to the cutting stock problem. Operations Research 9, 849-859.

Gilmore P. C. and Gomory R. E., 1963. A linear programming approach to the cutting stock problem: part II. Operations Research 11, 863-888.

Gilmore P. C. and Gomory R. E., 1965. Multistage cutting stock problems of two and more dimensions. Operations Research 13, 94-120.

Glover F. and Laguna M., 1993. Tabu search. In Reeves (Ed.), Modern Heuristics for Computational Problems. Basil Blackwell, Oxford.

Glover F. and Laguna M., 1997. Tabu search. Kluwer Academic Publishers, Boston.

Goldberg D. E., 1989. Genetic Algorithms in Search, Optimisation and Machine Learning. Addison-Wesley Publishing Company, Reading.

Goldberg D. E. and Lingle, R, 1985. Alleles, loci and the travelling salesman problem. In: Proceedings of an International Conference on Genetic Algorithms and their Applications, pp. 154–159.

Goldberg D. E. and Deb K., 1991. A comparative analysis of selection schemes used in genetic algorithms. In: B. M. Spatz (Ed.), Foundations of Genetic Algorithms, Morgan Kaufmann Publishers, San Mateo, California, pp. 69-93.

Golden, B., 1976. Approaches to the cutting stock problem. AIIE Transactions 8, 265-274.

Grefenstette J., Gopal, R., Rosmaita B. and van Gucht, D., 1985. Genetic Algorithms for the Travelling Salesman Problem. In: Proceedings of an International Conference on Genetic Algorithms and their Applications, pp. 160-168.

Grinde R. B. and Cavalier T. M., 1995. A new algorithm for the minimal-area convex enclosure problem. European Journal of Operational Research 84, 522-538.

Gwee B. H. and Lim M. H., 1996. Polyominoes tiling by a genetic algorithm. Computational Optimisation and Applications 6, 273-291.

Hadjiconstantinou E. and Christofides N., 1995. An exact algorithm for general, orthogonal, two-dimensional knapsack problems. European Journal of Operational Research 83, 39-56.

Haessler R. W. and Sweeney, P. E., 1991. Cutting stock problems and solution procedures. European Journal of Operational Research 54, 141-150.

Hahn S. G., 1968. On the optimal cutting of defective sheets, Operations Research 16, 1100-1114.

Haims M. J. and Freeman H., 1970. A multistage solution of the template-layout problem. IEEE Transactions on Systems Science and Cybernetics SSC-6, 145-151.

Han G. C. and Na S. J., 1996. Two-stage approach for nesting in two-dimensional cutting problems using neural network and simulated annealing. In: Proceedings of the Institute of Mechanical Engineers, Part B, Journal of Engineering Manufacture 210, B6, pp. 509-519.

Han G. C., Kim D. I., Kim S. K. and Na S. J., 1997. A new approach for nesting problem using part decomposition technology. In: Proceedings of the IECON'97 23rd International Conference on Industrial Electronics, Control and Instrumentation, IEEE, New York, NY, USA, 1234-1239.

Healy P. and Moll R., 1996. A local optimisation-based solution to the rectangle layout problem. Journal of the Operational Research Society 47, 523-537.

Heistermann J. and Lengauer T., 1993. Efficient automatic part nesting on irregular and inhomogeneous surfaces. In: Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, Austin, TX, USA, pp. 251-259.

Herbert E. A. and Dowsland K. A., 1996. A family of genetic algorithms for the pallet loading problem. Annals of Operations Research 63, 415-436.

Hinxman A. I., 1980. The trim loss and assortment problems. European Journal of Operational Research 5, 8-18.

Holland, J. H., 1975. Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor.

Hopper E. and Turton B. C. H., 1997. Application of Genetic Algorithms to Packing Problems - A Review. In: P. K. Chawdry, R. Roy and R. K. Kant (eds.), Proceedings of the 2nd On-line World Conference on Soft Computing in Engineering Design and Manufacturing, Springer Verlag, London, 279-288.

Hopper E. and Turton B. C. H., 1999. A genetic algorithms for a 2D industrial packing problem. Computers in Engineering 37, 375-378.

Hopper E. and Turton B. C. H., expected 2000. An Empirical Investigation of Meta-heuristic and Heuristic Algorithms for a 2D Packing Problem. Accepted for publication in European Journal of Operations Research.

House R. L and Dagli C. H., 1993. Genetic three dimensional packer. Proceedings for Artificial Neural Networks in Engineering Conference (ANNIE '93), vol. 3, ASME Press, New York, 837-843.

Hussain S. and Sastry V., 1997. Application of genetic algorithms for bin packing. International Journal of Computer Mathematics 63, 203-214.

Hwang I., 1997. An efficient processor allocation algorithm using two-dimensional packing. Journal of Parallel and Distributed Computing 42, 75-81.

Hwang S. M., Cheng Y. K. and Horng J. T., 1994. On solving rectangle bin packing problems using genetic algorithms. In: Proceedings of the 1994 IEEE International Conference on Systems, Man and Cybernetics. Part 2 (of 3), San Antonio, TX, USA, 1583-1590.

Ikonen I., Biles W. E., Kumar A. and Ragade R. K., 1996. Concept for a genetic algorithm for packing 3D objects of complex shape. In: Proceedings of the 1st Online Workshop on Soft Computing, Nagoya University, pp. 211-215.

Ikonen I. and Kumar A., 1997. A genetic algorithm for packing three-dimensional non-convex objects having cavities and holes. In: Proceedings of the 7th International Conference on Genetic Algorithms, pp. 591 - 598.

Ismail H. S., Hon K. K. B., 1992. New approaches for the nesting of two-dimensional shapes for press tool design. International Journal of Production Research 30, 825-837.

Ismail H. S., Hon K. K. B., 1995. Nesting of two-dimensional shapes using genetic algorithms. In: Proceedings of the Institution of Mechanical Engineers 209, Part B, 115-124.

Jakobs S., 1996. On genetic algorithms for the packing of polygons. European Journal of Operations Research 88, 165-181.

Jain S. and Chang Gea H., 1998. Two dimensional packing problems using genetic algorithms. Engineering with Computers 14, 206-213.

Jain P., Fenyes P. and Richter R., 1988. Optimal blank nesting using simulated annealing. Journal of Mechanical Design - Transactions of the ASME 114, 160-165.

Kämpke T., 1988. Simulated annealing: use of a new tool in bin-packing. Annals of Operations Research 16, 327-332.

Kantorovich, L. V. and Zagaller, V. A., 1951. Optimal Calculation for subdivision in the material industry. Lenzidat, Leningrad, p.18.

Knight, A., May, J., McAffer, J., Nguyen, T. and Sack J. R., 1990. A computational geometry workbench. Proceedings of the sixth Annual Symposium on Computational Geometry, Berkeley, California, June 6-8, ACM Press, Baltimore, MD, p. 370.

Koza J. R., 1992. Genetic programming. Cambridge, MA, MIT Press.

Kröger B., 1995. Guillontineable bin-packing: A genetic approach. European Journal of Operations Research 84, 645-661.

Kröger B., Schwenderling P., Vornberger O., 1991a. Parallel genetic packing of rectangles. In: Parallel Problem Solving from Nature 1st Workshop, Springer Verlag, pp. 160-164.

Kröger B., Schwenderling P. and Vornberger O., 1991b. Genetic packing of rectangles on transputers. In: P. Welch (Ed.), Transputing 2, IOS Press, Amsterdam, pp. 593-608.

Kröger B., Schwenderling P. and Vornberger O., 1993. Parallel genetic packing on transputers. In: J. Stender (Ed.), Parallel Genetic Algorithms: Theory and Applications, IOS Press, Amsterdam, pp. 151-185.

Koroupi F. and Loftus M., 1991. Accommodating diverse shapes within hexagonal pavers. International Journal of Production Research 29, 1507-1519.

van Laarhoeven P. J. M., Aarts, E. H., 1987. Simulated Annealing: Theory and Applications. D. Reidel Publishing Company, Dordrecht.

Lai K. K. and Chan W. M., 1997. An evolutionary algorithm for the rectangular cutting stock problem. International Journal of Industrial Engineering 4, 130-139.

Lamousin H .J., Waggenspack W. N. and Dobson G. T., 1996. Nesting of complex 2D parts within irregular boundaries. Journal of Manufacturing Science and Engineering 118, 615-622.

Laszlo, M. J., 1996. Computational geometry and computer graphics in C++. Prentics Hall.Upper Saddle River, N. J.

Leung T. W., Yung C. H. and Chan C .K., 1999. Applications of genetic algorithm and simulated annealing to the 2-dimensional non-guillotine cutting stock problem. Presented at IFORS '99, Beijing China, August 16-20.

Li Z. and Milenkovic V., 1995. Compaction and separation algorithms for non-convex polygons and their applications. European Journal of Operational Research. 84, 539-561.

Lin J. L., Foote B., Pulat S., Chang C. H. and Cheung, J. Y., 1993. Hybrid genetic algorithm for container packing in three dimensions. Proceedings of the 9th Conference on Artificial Intelligence for Applications, pp. 353-359.

Liu D. and Teng H., 1999. An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles. European Journal of Operational Research 112, 413-419.

Lodi A., Martello S. and Vigo D., 1999a. Approximation algorithms for the oriented two-dimensional bin packing problem. European Journal of Operational Research 112, 158-166.

Lodi A., Martello S. and Vigo D., 1999b. Neighborhood search algorithm for the guillotine non-oriented two-dimensional bin packing problem. In: Voss, S., Martello, S., Osman I. H. and Roucairol, C. (eds). Meta-Heuristics. Advances and Trends in Local Search Paradigms for Optimization. Kluwer Academic Publishers, pp. 125-139.

Marques V. M. M., Bispo C. F .G. and Sentieiro J. J. S., 1991. A system for the compaction of two-dimensional irregular shapes based on simulated annealing. Proceedings of the 1991 International Conference on Industrial Electronics, Control and Instrumentation - IECON '91, Kobe, Japan, Oct. 1991, pp. 1911-1916.

Martello S. and Toth P., 1990. Knapsack problems: algorithms and computer implementations. John Wiley & Sons Ltd., Chichester.

Martin R. R. and Stephenson P. C., 1988. Putting objects into boxes. Computer Aided Design 20, 506-514.

Mehlhorn K. and Näher S., 1995. LEDA: A platform for combinatorial and geometric computing. Communications of the ACM 38, 96-102.

Mehlhorn K. and Näher S., 2000. LEDA: a platform for combinatorial and geometric computing. Cambridge University Press, New York.

Mehlhorn, K., Müller, M., Näher, S., Schirra, S., Seel M., Uhrig, C. and Ziegler J., 1997. A computational basis for higher-dimensional computational geometry and applications. Proceedings of the 13th Annual Symposium on Computational Geometry, pp. 254 - 263.

Mitchell M., 1996. An introduction to genetic algorithms. MIT Press, Massachusetts.

Mohanty B. B., Mathur K. and Ivancic N. J., 1994. Value considerations in three-dimensional packing - a heuristic procedure using the fractional knapsack problem. European Journal of Operational Research 74, 143-151.

Munzner T., Levy S. and Philips M., 1995. Geomview: A system for geometric visualisation. Proceedings of the 11th Annual ACM Symposium on Computational Geometry, C12-C13.

Nee A. Y. C., 1984a. Computer aided layout of metal stamping blanks. Institute of Mechanical Engineers 198B, 187-194.

Nee A. Y. C., 1984b. A heuristic algorithm for optimum layout of metal stamping blanks. Annals of CIRP 33/1, 317-320.

Nee A. Y. C., Seow S. W. and Long S. L., 1986. Designing Algorithm for nesting irregular shapes with and without boundary constraints. CIRP Annals 35, 107-110.

Nievergelt J., Schorn P., de Lorenzi M., Ammann C. and Brüngger A., 1991. Computational Geometry: Methods, Algorithms and Applications. In: H. Bieri and H.Noltemeier (eds.), In: Proceedings of, International Workshop on Computational Geometry, CG'91, Bern, March 1991, Springer LNCS, pp. 171-186.

Noether, G. E., 1990. Introduction to statistics - The non-parametric way. Springer-Verlag, New York.

Oliveira J. F., Gomes A. M. and Ferreira S., expected 2000. A new constructive algorithm for nesting problems. Accepted for OR Spektrum.

Oliver, I. M., Smith, D. J., Holland, J. R. C., 1987. A study of permutation crossover operators on the travelling salesman problem, In: Proceedings of the Second International Conference on Genetic Algorithms and their Applications, pp. 224 – 230.

O'Rourke, 1998. Computational geometry in C. Cambridge University Press, Cambridge.

Overmars M. H., 1996. Designing the Computational Geometry Algorithms Library CGAL. Proceedings Workshop on Applied Computational Geometry, May 27-28, 1996, Philadelphia, Pennsylvania.

Pargas R. P. and Jain R., 1993. A parallel stochastic optimisation algorithm for solving 2D bin packing problems. In: Proceedings of the 9th Conference on Artificial Intelligence for Applications, pp. 18-25.

Petridis V. and Kazarlis S., 1994. Varying quality function in genetic algorithms and the cutting problem. In: Proceedings of the IEEE Conference on Evolutionary Computation, pp. 166- 169.

Pirlot M., 1996. General local search methods. European Journal of Operational Research 92, 493-511.

Poshyanonda P. and Dagli C. H., 1993. Genetic neuro-nester for irregular patterns. In: Proceedings for Artificial Neural Networks in Engineering Conference (ANNIE '93), vol. 3, ASME Press, New York, 825-830.

Prasad and Somasundaram, 1991. CASNS - A heuristic algorithm for the nesting of irregular shaped sheet-metal blanks. Computer Aided Engineering Journal 8, 69-73.

Prasad Y. K. D. V., Somasundarum S. and Rao K. P., 1995. A sliding algorithm for optimal nesting of arbitrarily shaped sheet metal blanks. International Journal of Production Research 33, 1505-1520.

Press W. H., Teuckolsky S. A., Vetterling W. T. and Flannery B. P., 1995. Numerical Recipes in C, The Art Scientific Computing, 2nd edition, Cambridge University Press, Cambridge.

Prosser P., 1988. A hybrid genetic algorithm for pallet loading. In: B. Radig (Ed.), ECAI 88 Proceedings of the 8th European Conference on Artificial Intelligence, Pitman, London, pp. 159-164.

Qu W. and Sanders J. L., 1987. A nesting algorithm for irregular parts and factor affection trim losses. International Journal of Production Research 25, 381-397.

Rahmani A. T. and Ono N., 1995. An evolutionary approach to two-dimensional guillotine cutting problem. In: Proceedings of the IEEE Conference on Evolutionary Computation, pp. 148-151.

Ratanapan K. and Dagli C. H., 1997a. An object-based evolutionary algorithm for solving rectangular piece nesting problems. In: IEEE (eds.), Proceedings of the IEEE Conference on Evolutionary Computation, ICEC, IEEE, Piscataway, NJ, USA, pp. 989-994.

Ratanapan K. and Dagli C. H., 1997b. An object-based evolutionary algorithm for solving irregular nesting problems. In: Proceedings for Artificial Neural Networks in Engineering Conference (ANNIE '97), vol. 7, ASME Press, New York, pp. 383-388.

Ratanapan K. and Dagli C. H., 1998. An object-based evolutionary algorithm: the nesting solution. In: IEEE (eds.) Proceedings of the International Conference on Evolutionary Computation 1998, ICEC '98, IEEE, Piscataway, NJ, USA, pp. 581-586.

Rayward-Smith V. J. and Shing M. T., 1983. Bin packing. Bulletin of the Institute of Mathematics and its Applications 19, 142-146.

Rechenberg I., 1973. Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der Evolution. Frommann-Holzboog Verlag, Stuttgart, 1973.

Reeves C., 1993. Modern Heuristics for Computational Problems. Basil Blackwell, Oxford.

Reeves C., 1996. Modern heuristic techniques. In: V. J. Rayward-Smith, I. H., Osman, C. R. Reeves and G. D. Smith (eds.), Modern Heuristic Search Methods, John Wiley & Sons Ltd., New York.

Riehme J., Scheithauer G. and Terno J., 1996. Solution of two-stage guillotine cutting stock problems having extremely varying order demands. European Journal of Operational Research 91, 543-552.

Rode M. and Rosenberg O., 1987. An analysis of heuristic trim loss algorithms, Engineering Cost and Production Economics 12, 71-78.

Rönnqvist M., 1995. A method for the cutting stock problem with different qualities, European Journal of Operational Research 83, 57-68.

Roussel G. and Maouche S., 1993. Improvements about automatic lay-panning for irregular shapes on plain fabric. In: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics 3, IEEE, Piscataway, NJ, USA, 90-97.

Runarsson T.P., Jonsson M. T. and Jensson P., 1996. Dynamic dual bin packing using fuzzy objectives. In: The 1996 IEEE International Conference on Evolutionary Computation, ICEC'96, Nagoya, Japan, May 20-22 1996, pp. 219-222.

Sarin S. C., 1983. Two-dimensional stock cutting problems and solution methodologies. ASME Transactions, Journal of Engineering for Industry 104, 155-160.

Scheithauer G. and Terno J., 1996. G4-heuristic for the pallet loading problem. Journal of the Operational Research Society 47, 511-522.

Schirra S., 1996. Designing a computational geometry library. In: Lecture Notes for Advanced School on Algorithmic Foundations of Geographic Information Systems, CISM, Udine, September 16-20.

Schorn P., 1990. An object oriented workbench for experimental geometric computation. Proceedings of the 2nd Canadian Conference in Computational Geometry, Ottawa, pp. 172- 175.

Sedgewick R., 1992. Algorithms in C++, Addison-Wesley, Reading.

Smith D., 1985. Bin-packing with adaptive search. In: Grefenstette (Ed.), Proceedings of an International Conference on Genetic Algorithms and their Applications, Lawrence Erlbaum, pp. 202-206.

Smith A. and DeCani P., 1980. Algorithms to optimise the layout of boxes in pallets, Journal of the Operational Research Society 31, 573-578.

Sweeney P. E. and Paternoster E., 1992. Cutting and packing problems: a categorised, application-orientated research bibliography. Journal of the Operational Research Society 43, 691-706.

Syswerda D., 1991. Schedule optimisation using genetic algorithms. In: Davis (Ed.), Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York.

Szykman S. and Cagan J., 1995. A simulated annealing-base approach to three-dimensional component packing. Transactions of ASME, Journal of Mechanical Design 117, 308-314.

Theodoracatos V. E. and Grimsley J. L., 1995. Optimal packing of arbitrarily-shaped polygons using simulated annealing and polynomial-time cooling schedules. Computer Methods in Applied Mechanics and Engineering 125, 53-70.

Veltkamp R. C., 1997. Generic Programming in CGAL, the Computational Geometry Algorithms Library. Proceedings of the 6th Eurographics Workshop on Programming Paradigms in Graphics.

Voss, S., Martello, S., Osman I. H. and Roucairol, C., 1999. Meta-Heuristics. Advances and Trends in Local Search Paradigms for Optimization. Kluwer Academic Publishers.

Wang P. Y., 1983. Two algorithms for constrained two-dimensional cutting stock problems. Operations Research 31, 573-586.

Whelan P. F., 1993. Automated packing of arbitrary shapes - a systems engineering approach. PhD Thesis, University of Wales, Cardiff.

Whelan P. F. and Batchelor B. G., 1991. Automated packing of arbitrary shapes, SPIE, Machine Vision Architectures, Integration and Applications 1615, 77-86.

Whelan P. F. and Batchelor B. G., 1992. Development of a vision system for the flexible packing of random shapes. In: Proceedings of SPIE - The International Society for Optical Engineering, Machine Vision Applications, Architectures, and Systems Integration, Boston, MA, USA, Nov 17-18 1992, vol. 1823, pp. 223-232.

Whelan P. F. and Batchelor B. G., 1993. Automated packing systems: Review of industrial implementations, SPIE, Machine Vision Architectures, Integration and Applications 2064, 358-369.

Whelan P. F., 1996. A practical packing strategy for the automated handling of irregular shapes. In: IEE Colloquium (Digest), Proceedings of the 1996 Colloquium on Mechatronics in Automated Handling, London, vol. 153, pp. 3/1-3/5.

Whitley, D., 1989. The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. In: Third International Conference on Genetic Algorithms, ICGA'89, pp. 116-121.

Winston, P. H., 1984. Artificial Intelligence, 2nd edition, Addison Wesley, London.

Yanasse H. H., Zinobper A. S. I. and Harris R. G., 1991. Two-dimensional cutting stock with multiple stock sizes. Journal of the Operational Research Society 42, 673-683.

Øian J., Hasselknippe B. and Lillehagen F., 1976. An interactive computer graphics approach to the problem of nesting of plane parts on a raw steel format. In: CAD76 In: CAD76 Proceedings Second International Conference on Computers in Engineering and Building Design, Imperial College London, March 76, IPC Science and Technology Press, Guildford, pp. 166-170.

11. Glossary

assortment problem	assignment of the *order list* to a supply of *objects*, such that the best selection of *objects* is used
automatic nesting	computer generated *layout*, no interference by the user
bin	*object (rectangular in the general sense)*
bin-packing problem	assignment of a set of *items* to a set of *bins*, either minimising the number of the *bins* or minimising the size of the *bins*
blank	*item* in the metal cutting industry, which is small compared to the *object* and has repetitive occurrences in the *layout*
constraints	pattern restrictions, e.g. distances between *items*, orientation of *figures* relative to each other, type and number of cuts etc.
cutting problem	*objects* (i.e. solid materials) are to be cut into *items*
cutting process	process that realises *patterns* by cutting *objects* into *items*
cutting stock problem	allocation of set of *items* to supply of *objects*; can be split into *trim loss problem* and *assortment problem*
dimensionality	minimum number of dimensions necessary to describe the geometry of a *pattern*
distributor's pallet loading problem	*loading* of pallet with non-identical *items*
exhaustive search	search process that evaluates every state in the *solution space*
figure	of an *object*: geometric representation in the space of relevant dimensions; a *figure* is uniquely defined by its *form*, *size* and orientation
form	shape of *figure*; can be *regular* or *irregular*
guillotine-cut	obtained by cutting along a straight line form one edge of the *object* of another
heuristic information	problem-specific information used by an algorithm to guide the search process towards the best route
heuristic search	search process the makes use of problem-specific information to guide search process in contrast to *exhaustive search*
hide	*object* in the leather industry
interactive nesting	*layout* partially generated by the computer; assistance of the user needed for the *packing process*
irregular form	*form* that is not *regular*
item	that to be cut or packed; part of *order list*

knapsack problem	*bin-packing* problem under additional consideration of value of the *items*; the *packing process* has to maximise the value of the packed *bins*
loading	packing in three-dimensional problems
layout	geometric combination of *items; pattern*
manual layout	*layout* completely generated by the user
manufacturer's pallet loading problem	*loading* of pallet with identical *items*
marker layout problem	*irregular packing problem* in cloth industry
nest	*layout*
nesting	or parts nesting: *irregular packing problem* in ship-building industry
non-guillotineable	it is not possible to obtain ALL the items in a layout by cutting along a straight line form one edge of the *object* to the opposite one
object	stock, consisting of one or more large containers to which *items* will be assigned
objective	criterion to be optimised
order list	a specific set of *items* ordered by the customer
packing problem	*objects* (e.g. containers, vehicles), which are defined as empty, are to be packed with *items*
packing process	process which realises *patterns* by packing *items* into *objects*
pallet loading problem	can be split into *manufacturer's* and *distributor's pallet loading problem*
part	*item*
partial layout	existing *layout,* which has not been completed yet
paving	tessellation, tiling repeated placement of an *item* in the infinite *object* without leaving any gaps or overlaps
pattern	geometric combination of *items*; *layout*
regular forms	*forms* that can be described by a few parameters (approximately 4)
remnant	continuous piece of *object* remaining after the *packing* of the *set of items*
search space	*solution* space
set of items	collection of *items* which has to be packed; *order*

sheet	*object* in 2D *packing*
size of figure	may be measured by its length, area or volume
solution	to the *packing problem*, i.e. one *layout*
solution space	collection of all possible solutions to a problem
stock sheet	*object*
strip	*object* in 2D *strip packing;* i.e. reel/ coil of material to be cut
strip packing	*packing* of *strip* material with the goal of minimising height of the *layout*
template layout problem	*marker layout problem*
trim loss	residual material that remains in the *pattern* but does not belong to order list
trim loss problem	assignment of the *order list* to a supply of *objects*, such that the *trim loss* is minimised
waste	*trim loss;* region(s) of *object(s)* which are of no further use for the current *packing problem*

A. Test Problems for Regular Packing

A.1 Strip Packing

A.1.1 Benchmark Problems in Literature

A.1.1.1 Overview of Regular Test Problems

Description of table entries:

reference: publication in which test problem has been used
name: name which the problem is referred to in the current work
size: number of items
shapes: geometric shape type which the problem consists of
source: source where the co-ordinates used for the experiments in this work have been obtained from; i.e. stated in publication, extracted from sample layout in publication or extracted from scanned sample layout in publication

Table A.1: Rectangular test problems from literature; optimum known

reference	name	size	source
Jakobs (1996)	J1	25	extracted from a sample layout in paper
Jakobs (1996)	J2	50	extracted from a sample layout in paper
Ratanapan and Dagli (1997b)	D2	21	dimensions are stated in paper
Kendall and Burke (1999)	Kendall	13	dimensions are stated in paper

Table A.2: Rectangular test problems from literature; optimum not known

reference	name	size	source
Ratanapan and Dagli (1997b)	D1	31	dimensions are stated in paper
Ratanapan and Dagli (1998)	D3	37	dimensions are stated in paper
Dagli and Poshyanonda (1997)	D4	37	dimensions are stated in paper

A.1.1.2 Dimensions

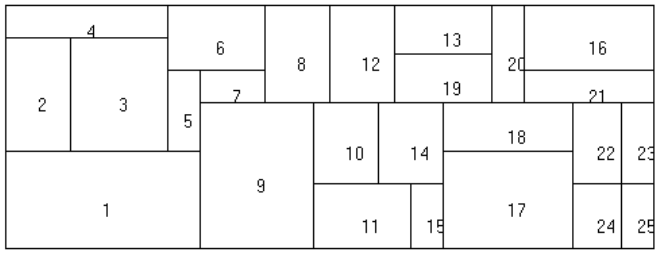
Rectangular test problems from literature; optimum known

name: **J1**
size: 25
object: width = 40

no.	width	height
1	12	6
2	4	7
3	6	7
4	10	2
5	2	5
6	6	4
7	4	2
8	4	6
9	7	9
10	4	5
11	6	4
12	4	6
13	6	3
14	4	5
15	2	4
16	8	4
17	8	6
18	8	3
19	6	3
20	2	6
21	8	2
22	3	5
23	2	5
24	3	4
25	2	4

name: **J2**
size: 50
object: width = 40

no.	width	height
1	5	6
2	7	6
3	4	3
4	4	4
5	6	4
6	6	3
7	4	2
8	6	2
9	3	4
10	3	4
11	2	5
12	4	2
13	3	3
14	3	6
15	4	3
16	4	6
17	4	3
18	4	3
19	4	2
20	4	4
21	4	2
22	4	3
23	3	4
24	3	4
25	2	5
26	2	5
27	2	4
28	3	6
29	5	2
30	5	4
31	3	3
32	5	3
33	2	3
34	4	3
35	2	3
36	4	3
37	2	2
38	2	4
39	3	4
40	3	4
41	2	4
42	3	2
43	3	2
44	2	2
45	3	2
46	2	2
47	3	3
48	2	3
49	3	4
50	2	4



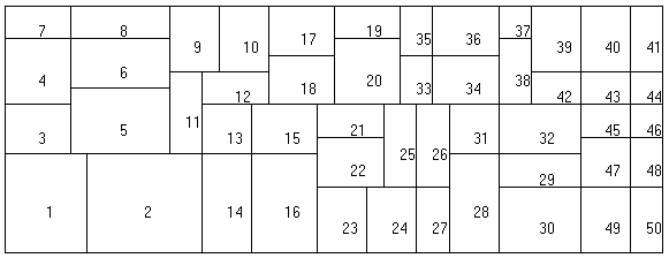


Figure A.1: Data set and optimal solution for test problems J1 and J2

name: **Dagli**
size: 21
object: width = 60

no.	width	height	quantity
1	12	12	4
2	12	10	5
3	12	9	6
4	12	8	6

name: **Kendall**
size: 13
object: width = 80

no.	width	height	quantity
1	24	16	1
2	28	16	2
3	60	14	2
4	20	28	1
5	22	26	2
6	42	44	1
7	18	70	1
8	62	26	1
9	18	48	2

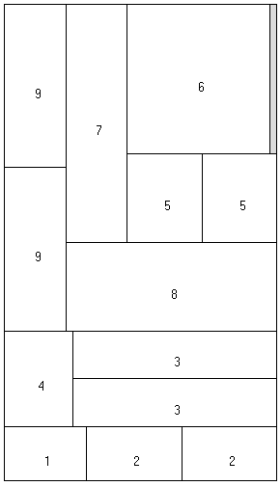


Figure A.2: Data sets for problems D2 and Kendall and optimal solution for Kendall

Rectangular test problems from literature; optimum not known

name: **D1**
size: 31
object: width = 60

width	height	no
12	10	5
10	11	7
9	13	4
4	5	4
9	10	6
6	8	5

name: **D3**
size: 37
object: width = 30

width	height	no
12	10	3
8	11	4
9	13	6
4	5	4
9	10	3
6	8	6
10	12	9
15	7	2

name: **D4**
size: 37
object: width = 20

width	height	no
10	12	3
11	8	4
13	9	6
5	4	4
10	9	3
8	6	6
12	10	9
7	5	2

A.1.2 Constructed Test Problems

A.1.2.1 Overview over Regular Test Problems

Table A.3: Rectangular problems: guillotineable

name	size	optimum object
T1a, T1b, T1c, T1d, T1e	17	200x200
T2a, T2b, T2c, T2d, T2e	25	200x200
T3a, T3b, T3c, T3d, T3e	29	200x200
T4a, T4b, T4c, T4d, T4e	49	200x200
T5a, T5b, T5c, T5d, T5e	73	200x200
T6a, T6b, T6c, T6d, T6e	97	200x200
T7a, T7b, T7c, T7d, T7e	199	200x200

Table A.4: Rectangular problems: non-guillotineable

name	size	optimum object
N1a, N1b, N1c, N1d, N1e	17	200x200
N2a, N2b, N2c, N2d, N2e	25	200x200
N3a, N3b, N3c, N3d, N3e	29	200x200
N4a, N4b, N4c, N4d, N4e	49	200x200
N5a, N5b, N5c, N5d, N5e	73	200x200
N6a, N6b, N6c, N6d, N6e	97	200x200
N7a, N7b, N7c, N7d, N7e	197	200x200

A.1.2.2 Dimensions

T-series:

	name: **T1a** size: 17 optimum: 200 x 200		**name:** **T1b** size: 17 optimum: 200 x 200		**name:** **T1c** size: 17 optimum: 200 x 200		**name:** **T1d** size: 17 optimum: 200 x 200		**name:** **T1e** size: 17 optimum: 200 x 200	
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
1	82	38	41	12	167	40	178	58	28	139
2	37	13	25	34	33	125	22	153	151	25
3	81	22	19	44	20	70	46	142	21	44
4	16	25	115	22	111	84	59	62	44	60
5	21	64	25	51	7	20	73	44	107	19
6	29	55	16	22	29	18	20	18	121	21
7	52	178	71	22	8	2	32	20	7	32
8	98	39	44	109	21	67	21	34	95	20
9	102	37	41	29	15	65	74	11	26	11
10	46	103	90	87	13	90	5	19	33	9
11	29	86	35	137	7	14	22	31	78	54
12	10	37	31	68	118	76	10	14	31	78
13	63	26	129	44	37	6	46	22	63	83
14	50	11	36	15	32	11	28	8	106	61
15	13	40	17	29	18	69	31	17	11	37
16	60	29	19	54	19	5	33	14	20	5
17	119	20	146	25	51	64	154	47	83	32

	name:	**T2a**	**name:**	**T2b**	**name:**	**T2c**	**name:**	**T2d**	**name:**	**T2e**
	size:	25	size:	25	size:	25	size:	25	size:	25
	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
1	114	36	82	39	11	76	14	50	37	71
2	22	75	45	56	58	71	67	13	61	28
3	64	150	73	20	42	34	51	57	90	32
4	13	55	21	35	89	21	68	44	12	48
5	81	31	17	9	14	13	38	37	7	43
6	16	3	35	180	75	44	29	96	37	10
7	4	24	8	26	56	31	22	13	17	33
8	12	36	9	44	20	45	46	75	64	94
9	4	21	22	34	29	6	52	59	26	16
10	8	15	60	17	11	21	33	40	26	112
11	44	24	29	18	17	112	40	22	11	23
12	37	46	105	17	54	69	21	18	38	78
13	21	31	86	33	24	5	19	40	28	89
14	21	6	28	17	34	98	54	22	44	79
15	5	25	51	84	11	27	47	24	50	24
16	16	32	11	16	18	15	21	67	43	44
17	57	22	17	41	35	93	13	10	9	50
18	26	7	97	25	29	12	97	36	46	18
19	56	87	15	49	18	12	35	67	102	32
20	80	37	30	43	22	71	17	67	29	32
21	27	34	69	26	38	59	30	43	17	14
22	117	17	120	43	40	43	81	45	26	30
23	55	17	23	6	14	66	16	31	17	6
24	62	33	7	26	129	31	51	24	26	24
25	82	16	38	20	57	23	51	14	119	18

	name:	**T3a**	**name:**	**T3b**	**name:**	**T3c**	**name:**	**T3d**	**name:**	**T3e**
	size:	29	size:	29	size:	29	size:	29	size:	29
	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
1	58	12	27	154	112	22	37	71	8	47
2	22	20	159	23	18	47	61	28	143	26
3	20	101	14	35	61	25	90	32	16	107
4	54	14	78	32	9	38	12	48	33	95
5	14	40	81	12	41	8	7	43	5	21
6	32	63	95	20	50	12	37	10	54	40
7	30	22	85	39	21	25	17	33	84	15
8	28	8	32	53	26	57	64	94	32	25
9	33	54	10	59	35	13	26	16	52	45
10	21	26	23	52	35	33	26	112	13	39
11	50	14	23	95	6	4	11	23	86	20
12	20	132	17	60	56	29	38	78	50	77
13	16	48	38	29	44	44	28	89	77	31
14	44	16	30	18	39	16	44	79	24	74
15	35	28	15	93	24	94	50	24	11	12
16	33	32	8	43	42	60	43	44	22	65
17	11	51	10	25	64	19	9	50	27	53
18	19	5	22	6	86	60	46	18	14	46
19	13	48	17	11	17	11	102	32	17	9
20	87	43	13	21	31	5	29	32	15	21
21	49	19	32	19	21	6	17	14	31	6
22	64	21	55	10	10	41	7	15	8	38
23	16	33	110	21	38	35	19	7	23	77
24	65	55	10	41	14	34	17	6	12	35
25	35	89	21	22	28	77	26	24	5	12
26	17	44	154	46	25	14	8	8	20	23
27	47	12	7	19	23	72	11	23	73	40
28	63	32	14	28	111	58	119	18	40	39
29	165	34	17	9	38	43	15	15	64	37

	name:	**T4a**	**name:**	**T4b**	**name:**	**T4c**	**name:**	**T4d**	**name:**	**T4e**
	size:	49	size:	49	size:	49	size:	49	size:	49
	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
1	26	25	54	27	25	105	37	57	110	32
2	9	28	109	16	57	24	16	63	12	33
3	47	23	37	102	27	18	48	23	78	15
4	18	42	33	11	13	25	50	41	62	18
5	13	52	76	35	20	15	49	86	16	53
6	45	94	46	15	58	20	27	53	37	76
7	8	40	41	6	14	10	21	18	14	97
8	34	12	14	9	6	28	64	14	28	43
9	11	28	27	18	14	59	7	18	16	9
10	23	63	60	9	13	25	38	21	15	22
11	28	5	35	46	17	57	26	4	64	19
12	19	17	56	18	41	82	25	77	10	13
13	7	33	22	92	32	81	12	6	4	27
14	19	15	45	19	25	102	33	17	12	13
15	37	12	5	30	27	18	28	71	3	6
16	75	18	20	28	46	34	93	20	7	22
17	19	35	36	56	77	25	5	10	67	16
18	6	37	15	32	8	15	54	10	27	14
19	12	10	30	11	25	18	23	18	13	41
20	25	27	17	13	64	13	10	30	78	35
21	56	41	18	11	7	19	58	29	30	115
22	4	14	5	2	14	74	14	41	8	54
23	22	6	13	10	57	21	24	13	20	91
24	8	8	22	8	58	11	18	74	6	6
25	14	15	55	28	6	6	15	72	72	12
26	12	7	16	51	4	9	9	61	15	92
27	32	19	71	38	4	3	18	20	22	58
28	10	68	51	75	29	6	5	12	19	6
29	57	20	40	18	13	5	27	23	38	48
30	52	35	32	13	105	22	31	12	21	61
31	25	26	7	42	10	45	15	8	10	38
32	34	101	27	23	39	74	20	20	22	47
33	5	29	5	26	6	31	32	9	22	37
34	36	9	50	37	26	21	34	51	5	30
35	38	20	12	45	13	10	45	11	5	9
36	6	20	11	37	13	29	6	11	3	15
37	30	22	37	32	51	52	26	31	24	10
38	20	11	10	8	54	23	55	52	20	37
39	18	24	17	3	19	19	17	33	18	13
40	10	9	22	5	44	8	26	20	23	19
41	15	23	25	11	11	11	16	3	41	27
42	11	2	7	29	9	9	11	7	18	5
43	62	14	31	20	46	24	4	8	6	11
44	61	13	19	8	36	21	12	4	39	24
45	104	18	32	18	8	8	29	26	21	6
46	62	20	25	5	7	2	23	11	62	17
47	91	39	12	15	2	7	23	4	20	15
48	13	2	31	12	18	5	59	19	3	8
49	75	37	36	10	28	13	57	15	44	7

	name: **T5a**		**name:** **T5b**		**name:** **T5c**		**name:** **T5d**		**name:** **T5e**	
	size: 73		size: 73		size: 73		size: 73		size: 73	
	optimum: 200 x 200		optimum: 200 x 200		optimum: 200 x 200		optimum: 200 x 200		optimum: 200 x 200	
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
1	54	12	36	24	112	22	18	43	66	25
2	39	23	26	4	18	47	55	12	14	30
3	17	58	14	3	61	25	16	41	18	14
4	90	30	31	26	9	38	11	23	43	34
5	12	32	23	83	41	8	29	20	59	30
6	42	11	52	10	50	12	15	81	10	16
7	81	21	18	37	21	25	30	27	8	33
8	58	20	6	1	26	57	17	23	22	47
9	4	23	8	11	35	13	9	26	18	16
10	25	9	32	10	35	33	15	31	26	13
11	3	13	9	38	6	4	40	73	24	17
12	14	35	43	27	56	29	1	3	26	4
13	11	4	4	10	12	13	28	14	7	28
14	9	15	36	103	32	9	12	11	26	14
15	5	23	23	29	39	16	15	7	30	51
16	15	21	17	35	10	4	2	3	39	25
17	18	10	18	100	22	35	11	4	7	3
18	13	88	13	75	22	31	18	22	19	9
19	5	19	61	11	24	94	12	3	25	6
20	42	10	7	35	13	11	38	19	8	14
21	49	8	32	23	29	8	36	7	18	78
22	9	3	11	4	64	19	4	21	14	17
23	13	5	20	7	5	3	52	14	62	12
24	4	11	9	9	24	52	6	10	15	64
25	14	60	2	3	18	49	7	11	16	5
26	37	9	14	64	34	23	20	8	46	13
27	5	20	9	6	52	43	32	32	27	26
28	79	16	22	6	5	7	24	76	12	63
29	5	16	26	58	12	2	12	15	30	8
30	4	8	10	10	31	5	8	34	36	91
31	42	11	11	17	8	5	2	13	23	31
32	19	49	10	35	4	9	4	1	8	13
33	9	8	6	8	21	6	11	12	22	9
34	33	22	36	21	10	41	9	30	9	50
35	34	6	4	23	13	4	13	39	14	4
36	27	12	7	18	38	35	34	9	8	16
37	7	23	4	22	29	37	20	30	22	12
38	4	21	2	13	5	20	14	35	57	37
39	26	7	30	18	14	34	25	19	7	6
40	23	4	38	9	8	38	31	17	12	36
41	9	3	17	5	20	28	16	30	11	29
42	14	59	106	25	57	17	82	14	30	56
43	18	16	14	83	25	14	33	5	50	62
44	16	24	62	9	23	72	4	27	61	13
45	27	49	88	26	51	17	27	13	4	7
46	8	47	8	35	60	28	97	16	7	33
47	8	23	90	30	2	10	20	56	16	26
48	19	14	60	21	8	17	12	26	12	46
49	3	2	28	9	10	36	43	14	21	27
50	1	5	5	17	19	25	64	17	28	65
51	51	8	28	22	19	10	25	74	4	4
52	10	3	57	11	19	41	8	40	9	19
53	30	9	36	12	32	11	24	10	40	13
54	28	96	30	11	10	39	7	45	6	19
55	9	31	11	9	6	6	52	17	15	22
56	30	19	16	28	7	2	47	68	19	31
57	18	55	23	4	6	33	17	57	21	17
58	20	28	20	7	5	32	11	53	5	6
59	23	86	58	11	3	19	13	35	5	15
60	10	18	5	24	4	4	25	4	30	7
61	9	12	6	19	3	13	7	31	6	19
62	21	41	17	10	24	21	20	58	8	12
63	4	2	6	3	68	11	9	42	11	8
64	4	9	17	12	10	9	16	27	19	21
65	31	7	41	9	24	10	17	11	18	14
66	23	68	26	7	44	19	11	23	3	20
67	18	29	10	3	26	7	10	34	18	3
68	69	27	48	16	6	11	7	12	83	16
69	91	14	53	13	13	13	20	18	16	13
70	35	23	8	3	2	11	23	15	2	7
71	61	22	33	13	24	4	18	22	6	10
72	30	9	25	10	48	9	63	12	37	6
73	65	13	22	5	30	7	42	11	8	3

	name: **T6a** size: 97 optimum: 200 x 200		**name:** **T6b** size: 97 optimum: 200 x 200		**name:** **T6c** size: 97 optimum: 200 x 200		**name:** **T6d** size: 97 optimum: 200 x 200		**name:** **T6e** size: 97 optimum: 200 x 200	
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
1	101	37	28	7	18	90	13	9	15	72
2	99	25	9	8	15	63	22	7	15	47
3	10	12	8	6	23	89	22	64	67	11
4	14	80	59	22	108	19	11	12	56	39
5	55	14	23	10	17	45	35	11	6	35
6	20	19	7	5	19	106	36	14	11	17
7	50	10	27	8	12	30	21	55	8	20
8	9	11	10	45	22	14	18	116	22	8
9	15	18	29	32	74	26	22	56	8	12
10	14	17	3	14	5	16	10	2	3	16
11	23	14	4	3	17	41	12	27	11	9
12	31	19	4	2	9	16	23	25	9	61
13	24	5	4	14	82	12	5	1	47	7
14	8	10	8	21	17	25	30	42	11	11
15	36	9	20	13	3	4	16	41	5	18
16	22	59	13	12	79	17	16	68	6	22
17	28	14	31	11	12	13	7	20	5	7
18	2	13	18	41	7	27	13	6	6	14
19	7	7	5	9	8	32	4	14	25	21
20	9	3	26	8	33	5	9	35	22	4
21	14	45	13	18	4	17	27	94	16	11
22	4	1	37	8	20	5	8	46	33	17
23	32	5	20	6	48	13	11	21	8	7
24	23	42	30	10	10	20	7	11	4	8
25	12	4	9	31	10	32	31	20	26	110
26	22	6	19	24	25	22	8	2	11	4
27	15	23	7	10	8	12	4	18	58	12
28	16	16	65	29	9	12	4	25	56	21
29	16	40	24	25	3	20	41	27	21	102
30	10	56	6	21	8	36	10	60	6	25
31	18	94	4	13	14	9	12	75	9	51
32	15	40	25	57	26	7	29	16	16	11
33	37	35	20	14	8	7	35	7	30	5
34	7	7	11	18	9	6	12	7	12	9
35	9	31	3	19	14	73	7	11	4	6
36	22	24	12	59	25	5	4	24	26	16
37	1	5	23	70	12	10	47	12	68	12
38	36	10	22	38	5	10	6	20	20	10
39	22	10	15	4	4	8	34	102	21	26
40	15	104	23	70	9	2	23	49	20	12
41	12	21	19	57	17	11	8	11	49	16
42	4	16	13	3	18	11	4	4	6	28
43	16	5	29	8	35	11	11	7	11	3
44	23	42	20	18	7	29	14	10	37	12
45	22	49	6	12	23	9	6	5	9	18
46	25	11	7	5	7	2	27	8	2	9
47	44	12	7	26	49	14	4	5	5	14
48	16	40	4	1	30	10	2	10	15	4
49	11	48	7	25	28	40	5	21	39	9
50	14	5	36	7	21	48	14	6	64	10

continued

	name:	T6a	name:	T6b	name:	T6c	name:	T6d	name:	T6e
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
51	17	6	23	59	11	44	33	7	32	7
52	2	10	18	50	3	18	18	5	16	26
53	6	1	1	3	13	13	18	15	99	21
54	50	11	14	40	16	16	5	9	38	19
55	15	81	7	46	87	30	28	5	23	32
56	11	33	20	40	5	26	20	21	6	37
57	8	6	37	7	8	3	13	4	22	54
58	9	4	4	23	24	23	15	7	25	36
59	11	2	32	27	10	38	18	3	50	27
60	22	65	5	16	22	6	28	15	27	22
61	13	35	5	30	23	9	56	9	8	41
62	18	47	7	11	67	16	14	4	66	23
63	16	54	9	11	32	24	4	25	8	5
64	10	37	5	13	1	4	56	19	15	3
65	6	8	14	86	10	12	16	14	18	12
66	20	53	8	25	6	11	13	56	9	4
67	3	7	6	6	16	3	13	34	6	2
68	2	10	9	7	22	8	50	24	9	6
69	19	4	4	14	39	8	7	33	5	8
70	8	35	16	9	8	20	7	21	4	18
71	17	29	14	5	4	26	7	2	15	19
72	25	46	2	9	55	8	2	4	8	5
73	8	6	16	18	63	18	7	22	14	4
74	11	12	15	6	18	5	5	3	48	13
75	10	6	26	17	20	4	2	2	14	14
76	5	16	8	4	4	1	4	1	23	10
77	8	12	6	14	6	13	65	17	5	18
78	8	32	28	73	10	12	11	12	10	23
79	13	19	7	12	22	22	13	10	51	11
80	26	4	2	7	49	15	37	29	58	14
81	4	14	35	11	35	11	6	1	7	12
82	16	3	24	10	3	13	55	15	10	9
83	11	12	25	9	1	1	8	5	16	12
84	3	17	6	5	9	11	3	15	13	3
85	13	9	62	22	7	10	24	35	35	9
86	34	6	49	8	35	20	14	11	34	12
87	9	5	47	19	28	9	12	8	17	25
88	21	13	7	14	31	11	6	22	3	3
89	5	1	24	20	4	2	2	10	7	23
90	4	12	18	25	7	19	8	3	10	20
91	39	11	6	21	1	6	4	11	109	18
92	24	8	41	41	7	9	22	8	7	7
93	6	4	69	18	47	16	19	21	6	6
94	20	3	2	12	2	7	36	12	44	13
95	5	1	22	5	5	13	5	12	3	1
96	15	3	40	7	66	11	110	16	3	12
97	11	2	117	20	40	9	47	9	10	11

	name:	**T7a**	**name:**	**T7b**	**name:**	**T7c**	**name:**	**T7d**	**name:**	**T7e**
	size:	199	size:	199	size:	199	size:	199	size:	199
	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
1	18	11	19	50	11	3	14	46	15	69
2	23	11	8	50	4	3	28	46	13	28
3	17	35	8	8	8	11	29	7	6	28
4	21	35	9	8	44	9	34	7	11	27
5	25	10	8	2	3	9	10	33	5	27
6	5	1	10	2	19	20	23	49	59	17
7	7	1	10	29	6	20	20	86	29	9
8	76	12	17	22	50	8	19	31	11	9
9	8	12	32	22	2	2	23	31	9	4
10	5	9	9	22	2	2	3	6	15	4
11	7	9	6	9	51	16	26	6	4	24
12	9	13	21	9	2	6	21	19	5	24
13	16	13	3	15	2	6	4	19	18	51
14	12	22	10	5	11	8	9	22	9	58
15	5	4	30	5	4	8	3	20	15	58
16	13	4	8	11	50	8	21	14	29	8
17	16	19	10	11	4	8	5	14	11	8
18	7	19	10	10	44	8	21	3	30	14
19	46	8	30	10	3	8	4	3	29	14
20	30	8	8	5	15	6	21	6	10	66
21	8	20	9	5	8	6	5	6	2	4
22	5	31	6	13	54	8	25	4	9	4
23	13	31	21	13	51	8	9	4	3	10
24	23	7	17	64	5	22	8	14	6	19
25	23	7	10	26	6	22	9	6	10	19
26	30	12	8	26	12	46	2	6	2	6
27	9	9	3	18	24	8	11	20	9	6
28	16	9	40	18	23	8	8	11	4	27
29	23	5	17	7	19	4	4	11	5	27
30	23	5	32	7	6	4	63	16	11	8
31	16	16	9	11	8	17	7	9	5	8
32	7	16	27	11	17	17	3	9	11	20
33	21	12	4	23	4	11	9	8	3	20
34	10	12	6	23	19	11	2	8	13	7
35	6	19	14	7	10	14	7	7	6	7
36	32	39	35	7	20	11	3	7	9	44
37	9	39	10	6	52	11	8	9	3	21
38	43	45	3	6	6	21	4	9	9	21
39	17	11	5	11	7	12	8	41	9	35
40	5	7	14	15	11	12	11	41	12	38
41	16	7	4	15	7	13	10	12	7	38
42	5	4	12	74	2	13	4	12	10	53
43	16	4	12	29	4	16	21	15	9	3
44	21	7	19	29	10	23	7	15	10	3
45	10	7	14	41	4	3	73	18	7	15
46	13	19	22	8	19	3	23	18	9	15
47	11	11	13	8	20	9	11	35	6	11
48	17	11	10	38	52	9	12	35	10	11
49	6	18	8	38	7	9	10	9	9	31
50	32	18	10	5	11	9	4	9	10	31
51	19	28	3	5	7	3	21	6	4	21
52	9	9	22	33	2	3	7	6	10	21
53	3	9	5	13	23	6	5	10	5	8
54	6	35	8	13	10	6	16	10	11	8
55	11	8	8	10	5	24	6	6	7	19
56	17	8	3	9	6	24	15	6	9	19
57	9	19	2	9	9	7	42	6	9	11
58	3	19	5	31	4	7	28	6	18	11
59	6	22	14	27	8	42	10	8	3	14
60	19	5	4	27	11	18	7	3	9	14
61	13	5	10	41	6	18	9	3	5	28
62	13	21	9	41	20	5	7	5	11	28
63	6	13	8	52	13	5	9	5	16	24
64	5	6	4	25	12	12	6	9	8	24
65	5	4	6	25	18	12	15	9	6	31
66	12	4	3	1	26	14	20	5	4	3
67	5	2	2	1	9	11	22	5	17	3

continued

	name: T7a		name: T7b		name: T7c		name: T7d		name: T7e	
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
68	12	2	8	21	7	11	7	5	4	28
69	19	17	5	21	6	37	21	5	9	13
70	13	17	5	20	18	37	10	11	8	13
71	5	7	8	20	13	35	7	7	12	9
72	17	7	12	45	10	35	9	7	9	9
73	32	6	19	45	20	34	5	5	4	15
74	9	6	18	41	13	34	16	5	10	15
75	41	9	18	41	9	3	20	8	15	71
76	43	9	7	12	7	3	22	8	9	37
77	6	8	28	12	12	9	7	8	26	37
78	22	8	6	13	18	9	21	8	12	15
79	19	7	25	9	26	7	5	19	7	15
80	12	7	28	9	16	7	16	19	9	26
81	60	17	25	4	11	24	21	49	9	9
82	19	17	28	4	6	24	7	4	5	9
83	10	50	3	4	8	10	9	4	7	14
84	24	30	4	4	3	10	26	35	9	15
85	5	30	7	9	12	20	31	11	8	15
86	28	34	21	9	12	13	13	11	10	18
87	27	8	6	12	18	13	4	15	30	18
88	27	8	5	3	42	18	3	15	9	5
89	9	4	17	3	8	10	10	5	5	5
90	18	4	31	7	3	10	9	5	19	17
91	7	18	10	11	12	5	10	13	10	17
92	3	10	9	11	18	5	10	13	16	15
93	12	10	3	9	44	11	15	6	8	15
94	5	12	4	9	11	11	27	6	14	12
95	9	62	5	4	12	20	10	15	7	12
96	9	47	17	4	3	20	9	15	6	8
97	9	47	22	5	8	21	7	16	21	8
98	11	7	31	5	6	21	8	16	24	10
99	29	7	7	4	18	13	10	10	17	10
100	4	22	21	4	49	13	5	5	50	8
101	2	11	27	14	13	26	12	5	8	8
102	5	11	94	14	27	26	31	24	14	16
103	9	16	12	9	9	36	7	11	3	5
104	19	97	31	9	13	13	6	11	14	5
105	3	2	4	4	31	13	5	5	9	13
106	12	2	8	4	11	18	12	5	2	13
107	15	6	17	9	18	11	10	32	7	12
108	5	6	25	12	49	11	5	12	2	12
109	11	15	24	12	12	9	5	12	9	34
110	29	15	4	10	3	9	5	30	11	21
111	7	48	20	10	8	3	16	30	15	21
112	8	41	19	16	6	3	4	5	3	11
113	12	41	29	45	13	5	3	5	14	11
114	2	5	28	5	31	5	4	18	21	8
115	5	5	15	5	14	21	6	18	29	8
116	24	4	7	32	7	7	17	29	8	21
117	5	4	4	5	35	7	7	25	24	19
118	3	6	8	5	11	8	5	14	12	17
119	3	3	5	15	14	8	14	14	5	17
120	1	3	23	15	13	10	4	5	7	4
121	9	14	6	18	27	10	3	5	2	4
122	22	4	9	18	10	9	6	13	9	3
123	7	4	12	10	29	9	7	23	2	3
124	19	3	17	10	6	31	8	23	11	10
125	9	3	4	6	10	31	5	20	10	10
126	3	3	20	6	15	88	5	20	29	13
127	1	3	25	7	7	8	4	8	14	9
128	17	4	24	7	35	8	3	8	17	9
129	2	4	6	19	11	7	5	11	11	9
130	9	13	9	19	14	7	14	11	9	9
131	18	4	9	21	40	9	4	11	6	6
132	4	4	19	38	9	9	6	11	25	6
133	7	12	7	19	10	40	26	10	20	14
134	40	8	10	3	29	40	19	4	4	7

continued

	name: T7a		name: T7b		name: T7c		name: T7d		name: T7e	
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
135	4	8	12	3	42	6	25	4	7	7
136	3	8	28	27	25	6	19	6	15	13
137	4	8	9	18	22	38	25	6	11	3
138	12	4	12	18	28	15	12	24	10	3
139	5	4	5	12	5	6	22	10	12	2
140	2	9	23	12	26	6	35	10	5	2
141	18	8	10	16	7	2	6	20	41	10
142	4	8	12	16	6	2	5	20	58	10
143	12	5	6	9	2	4	15	8	6	8
144	5	5	9	9	13	8	22	8	25	8
145	13	44	43	13	21	8	21	16	4	6
146	13	37	7	13	7	2	19	15	7	6
147	7	11	6	2	6	2	16	15	37	7
148	11	11	9	2	13	8	8	10	13	7
149	16	67	9	9	2	8	4	4	60	11
150	4	14	12	9	5	9	15	4	15	11
151	6	14	15	17	26	9	4	6	11	20
152	39	6	9	17	13	4	15	6	18	9
153	18	6	7	16	21	4	15	8	9	9
154	39	18	6	13	5	11	22	8	28	5
155	18	18	8	3	2	11	22	14	9	5
156	7	26	8	3	8	15	35	14	28	13
157	11	26	8	10	8	41	8	8	9	13
158	9	15	8	10	10	19	19	8	21	36
159	9	15	4	27	16	19	19	3	10	26
160	4	10	25	27	6	18	16	3	6	26
161	6	10	4	15	7	3	37	35	13	53
162	8	7	3	15	3	3	11	18	18	9
163	12	7	20	24	28	23	3	16	9	9
164	23	7	9	30	31	23	7	16	60	9
165	17	7	4	24	7	15	17	24	15	9
166	27	14	10	24	3	15	18	24	27	19
167	21	21	28	8	5	4	13	35	33	14
168	28	16	20	4	2	4	14	35	4	14
169	5	16	1	4	7	45	6	21	75	17
170	23	7	20	4	8	45	5	21	11	17
171	17	7	1	4	10	22	8	15	33	5
172	13	7	6	3	16	22	4	15	4	5
173	18	7	16	3	30	6	57	17	10	10
174	33	6	3	19	9	6	3	2	6	10
175	7	6	27	8	12	22	7	2	62	10
176	27	26	48	8	4	22	11	17	24	10
177	28	5	17	17	15	28	10	17	12	12
178	5	5	26	17	7	28	8	2	19	12
179	5	10	4	9	10	6	4	2	17	7
180	8	10	3	9	7	6	69	10	5	3
181	31	23	27	11	42	19	9	4	11	3
182	6	7	48	11	15	15	2	4	5	4
183	3	7	7	10	15	15	17	27	11	4
184	24	8	20	10	9	33	11	17	21	17
185	7	20	4	6	10	13	7	17	16	17
186	16	14	10	6	7	13	9	6	17	8
187	5	14	17	13	8	19	2	6	12	2
188	21	3	26	13	26	19	80	18	4	2
189	12	3	4	12	15	18	37	8	12	6
190	21	16	13	8	15	18	21	8	4	6
191	12	16	12	8	12	17	13	16	62	13
192	5	13	3	11	4	17	14	16	24	13
193	8	13	75	11	17	15	18	4	12	3
194	6	1	9	4	40	7	19	4	19	3
195	3	1	14	4	2	7	21	10	15	4
196	9	12	27	5	40	8	11	10	16	4
197	24	12	23	5	2	8	7	10	33	8
198	16	5	13	4	15	6	18	6	15	4
199	5	5	12	4	7	6	19	6	16	4

N-series:

	name:	**N1a**	**name:**	**N1b**	**name:**	**N1c**	**name:**	**N1d**	**name:**	**N1e**
	size:	17	size:	17	size:	17	size:	17	size:	17
	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
1	33	132	29	137	35	86	41	12	21	84
2	64	41	11	31	17	54	25	34	56	67
3	103	32	75	22	36	81	19	44	123	86
4	59	9	22	107	70	88	115	22	30	17
5	44	38	63	132	42	117	25	51	26	61
6	123	29	44	9	5	32	16	22	51	44
7	57	62	31	27	12	42	71	22	25	42
8	32	8	55	18	25	15	44	109	79	32
9	25	26	28	74	11	55	41	29	19	66
10	53	39	58	58	40	10	90	87	36	82
11	15	81	80	16	26	78	35	137	43	34
12	17	18	70	14	44	29	31	68	66	27
13	42	63	38	36	77	40	129	44	36	8
14	9	50	24	27	86	49	36	15	23	19
15	44	91	39	68	79	64	17	29	13	64
16	90	68	99	22	9	30	19	54	62	48
17	66	41	161	41	121	34	146	25	89	45

	name:	**N2a**	**name:**	**N2b**	**name:**	**N2c**	**name:**	**N2d**	**name:**	**N2e**
	size:	25	size:	25	size:	25	size:	25	size:	25
	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
1	41	12	37	71	34	79	34	70	14	50
2	25	34	61	28	79	30	21	53	67	13
3	19	44	90	32	24	112	26	101	51	57
4	115	22	12	48	63	71	29	74	68	44
5	25	51	7	43	47	49	90	26	38	37
6	16	22	37	10	32	86	34	48	29	96
7	71	22	17	33	52	41	15	59	22	13
8	35	16	64	94	11	69	41	24	46	75
9	9	33	26	16	81	37	8	35	52	59
10	41	29	26	112	76	28	33	73	33	40
11	17	93	11	23	75	47	9	17	40	22
12	18	17	38	78	14	38	12	34	21	18
13	90	87	28	89	24	24	43	17	19	40
14	27	76	44	79	22	37	51	27	54	22
15	35	137	50	24	89	13	12	41	47	24
16	31	68	43	44	77	24	23	38	21	67
17	59	21	9	50	12	47	37	60	13	10
18	70	28	46	18	4	9	18	14	97	36
19	36	15	102	32	10	46	44	46	35	67
20	17	29	29	32	30	7	42	22	17	67
21	19	54	17	14	49	27	9	14	30	43
22	50	23	26	30	26	30	21	8	81	45
23	9	7	17	6	4	20	41	24	16	31
24	79	16	26	24	99	23	78	77	51	24
25	146	25	119	18	53	10	122	53	51	14

	name:	**N3a**	**name:**	**N3b**	**name:**	**N3c**	**name:**	**N3d**	**name:**	**N3e**
	size:	29	size:	29	size:	29	size:	29	size:	29
	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
1	89	37	8	47	35	55	58	12	38	11
2	56	28	143	26	65	50	22	20	20	103
3	12	66	16	107	54	39	20	101	68	49
4	13	24	33	95	46	80	54	14	74	11
5	30	25	5	21	33	108	14	40	21	119
6	8	10	54	40	21	41	32	63	17	92
7	5	1	84	15	34	5	30	22	27	38
8	35	9	32	25	22	29	28	8	47	89
9	36	9	52	45	9	43	33	54	95	51
10	20	78	13	39	40	53	21	26	71	30
11	26	32	86	20	29	48	50	14	27	82
12	17	104	50	77	19	41	20	132	44	21
13	52	48	77	31	3	14	16	48	37	27
14	73	33	24	74	67	67	44	16	6	33
15	14	55	11	12	12	27	35	28	15	21
16	24	45	22	65	25	5	33	32	23	79
17	55	15	27	53	4	17	11	51	58	36
18	18	36	14	46	65	12	19	5	71	29
19	107	21	17	9	59	80	13	48	6	12
20	54	45	15	21	41	27	87	43	9	40
21	81	33	31	6	39	15	49	19	12	28
22	10	32	8	38	50	16	64	21	12	7
23	8	10	23	77	52	14	16	33	37	9
24	16	27	12	35	5	2	65	55	22	29
25	22	17	5	12	47	16	35	89	70	34
26	65	62	20	23	27	38	17	44	10	32
27	66	12	73	40	12	15	47	12	27	20
28	15	61	40	39	55	14	63	32	48	18
29	120	49	64	37	114	23	165	34	49	12

	name: **N4a**		**name:** **N4b**		**name:** **N4c**		**name:** **N4d**		**name:** **N4e**	
	size: 49		size: 49		size: 49		size: 49		size: 49	
	optimum: 200 x 200		optimum: 200 x 200		optimum: 200 x 200		optimum: 200 x 200		optimum: 200 x 200	
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
1	37	57	110	32	54	27	25	105	28	99
2	16	63	12	33	109	16	57	24	57	27
3	48	23	78	15	37	102	27	18	13	25
4	50	41	62	18	33	11	13	25	21	12
5	49	86	16	53	76	35	20	15	17	46
6	27	53	37	76	46	15	58	20	8	10
7	21	18	14	97	41	6	14	10	24	5
8	64	14	28	43	14	9	6	28	32	69
9	7	18	16	9	27	18	14	59	13	5
10	38	21	15	22	60	9	13	25	11	18
11	26	4	64	19	35	46	17	57	21	13
12	25	77	10	13	56	18	41	82	10	13
13	12	6	4	27	22	92	32	81	11	48
14	33	17	12	13	45	19	25	102	11	23
15	28	71	3	6	5	30	27	18	21	30
16	93	20	7	22	20	28	46	34	23	35
17	5	10	67	16	36	56	77	25	48	43
18	54	10	27	14	15	32	8	15	9	33
19	23	18	13	41	30	11	25	18	28	7
20	10	30	78	35	17	13	64	13	45	39
21	58	29	30	115	18	11	7	19	4	16
22	14	41	8	54	5	2	14	74	43	10
23	24	13	20	91	13	10	57	21	36	23
24	18	74	6	6	22	8	58	11	4	27
25	15	72	72	12	55	28	6	6	24	15
26	9	61	15	92	16	51	4	9	46	12
27	18	20	22	58	71	38	4	3	17	4
28	5	12	19	6	51	75	29	6	7	8
29	27	23	38	48	40	18	13	5	10	36
30	31	12	21	61	32	13	105	22	53	28
31	15	8	10	38	7	42	10	45	2	12
32	20	20	22	47	27	23	39	74	22	14
33	32	9	22	37	5	26	6	31	57	18
34	34	51	5	30	50	37	26	21	24	108
35	45	11	5	9	12	45	13	10	6	2
36	6	11	3	15	11	37	13	29	56	11
37	26	31	24	10	37	32	51	52	9	51
38	55	52	20	37	10	8	54	23	36	48
39	17	33	18	13	17	3	19	19	131	23
40	26	20	23	19	22	5	44	8	49	38
41	16	3	41	27	25	11	11	11	17	37
42	11	7	18	5	7	29	9	9	65	19
43	4	8	6	11	31	20	46	24	40	18
44	12	4	39	24	19	8	36	21	25	48
45	29	26	21	6	32	18	8	8	1	3
46	23	11	62	17	25	5	7	2	35	13
47	23	4	20	15	12	15	2	7	10	10
48	59	19	3	8	31	12	18	5	57	30
49	57	15	44	7	36	10	28	13	94	29

	name:	**N5a**	**name:**	**N5b**	**name:**	**N5c**	**name:**	**N5d**	**name:**	**N5e**
	size:	73	size:	73	size:	73	size:	73	size:	73
	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
1	112	22	66	25	54	12	18	43	89	17
2	18	47	14	30	39	23	55	12	70	29
3	61	25	18	14	17	58	16	41	12	60
4	9	38	43	34	90	30	11	23	10	33
5	41	8	59	30	12	32	29	20	17	6
6	50	12	10	16	42	11	15	81	2	9
7	21	25	8	33	81	21	30	27	8	17
8	26	57	22	47	58	20	17	23	9	3
9	35	13	18	16	4	23	9	26	11	14
10	35	33	26	13	25	9	15	31	36	31
11	6	4	24	17	3	13	40	73	53	12
12	56	29	26	4	14	35	1	3	10	10
13	12	13	7	28	11	4	28	14	9	26
14	32	9	26	14	9	15	12	11	123	19
15	39	16	30	51	5	23	15	7	20	16
16	10	4	39	25	15	21	2	3	79	28
17	22	35	7	3	18	10	11	4	13	46
18	22	31	19	9	13	88	18	22	18	70
19	24	94	25	6	5	19	12	3	44	9
20	13	11	8	14	42	10	38	19	5	30
21	29	8	18	78	49	8	36	7	5	11
22	64	19	14	17	9	3	4	21	24	39
23	5	3	62	12	13	5	52	14	8	31
24	24	52	15	64	4	11	6	10	36	21
25	18	49	16	5	14	60	7	11	17	28
26	34	23	46	13	37	9	20	8	35	6
27	52	43	27	26	5	20	32	32	30	12
28	5	7	12	63	79	16	24	76	14	18
29	12	2	30	8	5	16	12	15	41	10
30	31	5	36	91	4	8	8	34	21	17
31	8	5	23	31	42	11	2	13	14	6
32	4	9	8	13	19	49	4	1	44	11
33	21	6	22	9	9	8	11	12	80	17
34	10	41	9	50	33	22	9	30	10	56
35	13	4	14	4	34	6	13	39	27	5
36	38	35	8	16	27	12	34	9	17	101
37	29	37	22	12	7	23	20	30	41	7
38	5	20	57	37	4	21	14	35	34	10
39	14	34	7	6	26	7	25	19	19	68
40	8	38	12	36	23	4	31	17	55	50
41	20	28	11	29	9	3	16	30	6	39
42	57	17	30	56	14	59	82	14	40	12
43	25	14	50	62	18	16	33	5	1	3
44	23	72	61	13	16	24	4	27	35	9
45	51	17	4	7	27	49	27	13	21	36
46	60	28	7	33	8	47	97	16	63	15
47	2	10	16	26	8	23	20	56	9	21
48	8	17	12	46	19	14	12	26	50	15
49	10	36	21	27	3	2	43	14	13	6
50	19	25	28	65	1	5	64	17	22	9
51	19	10	4	4	51	8	25	74	7	30
52	19	41	9	19	10	3	8	40	9	37
53	32	11	40	13	30	9	24	10	1	6
54	10	39	6	19	28	96	7	45	55	16
55	6	6	15	22	9	31	52	17	7	25
56	7	2	19	31	30	19	47	68	9	17
57	6	33	21	17	18	55	17	57	22	46
58	5	32	5	6	20	28	11	53	29	18
59	3	19	5	15	23	86	13	35	26	45
60	4	4	30	7	10	18	25	4	33	36
61	3	13	6	19	9	12	7	31	22	11
62	24	21	8	12	21	41	20	58	4	8
63	68	11	11	8	4	2	9	42	5	10
64	10	9	19	21	4	9	16	27	11	2
65	24	10	18	14	31	7	17	11	48	27
66	44	19	3	20	23	68	11	23	4	26
67	26	7	18	3	18	29	10	34	3	7
68	6	11	83	16	69	27	7	12	22	7
69	13	13	16	13	91	14	20	18	16	13
70	2	11	2	7	35	23	23	15	12	19
71	24	4	6	10	61	22	18	22	6	18
72	48	9	37	6	30	9	63	12	16	6
73	30	7	8	3	65	13	42	11	32	12

	name:	**N6a**	**name:**	**N6b**	**name:**	**N6c**	**name:**	**N6d**	**name:**	**N6e**
	size:	97	size:	97	size:	97	size:	97	size:	97
	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
1	14	78	13	9	28	7	52	34	18	90
2	27	27	22	7	9	8	31	14	15	63
3	17	35	22	64	8	6	29	7	23	89
4	39	21	11	12	59	22	16	79	108	19
5	66	16	35	11	23	10	7	48	17	45
6	37	49	36	14	7	5	7	24	19	106
7	5	19	21	55	27	8	21	12	12	30
8	9	16	18	116	10	45	22	37	22	14
9	19	16	22	56	29	32	15	13	74	26
10	26	12	10	2	3	14	11	7	5	16
11	7	26	12	27	4	3	4	15	17	41
12	31	14	23	25	4	2	14	10	9	16
13	8	25	5	1	4	14	3	6	82	12
14	15	35	30	42	8	21	3	10	17	25
15	12	19	16	41	20	13	15	26	3	4
16	21	21	16	68	13	12	5	24	79	17
17	5	14	7	20	31	11	10	35	12	13
18	1	3	13	6	18	41	42	22	7	27
19	8	19	4	14	5	9	7	5	8	32
20	5	19	9	35	26	8	7	19	33	5
21	14	37	27	94	13	18	1	6	4	17
22	48	11	8	46	37	8	2	4	20	5
23	6	16	11	21	20	6	11	14	48	13
24	12	7	7	11	30	10	5	2	10	20
25	68	16	31	20	9	31	13	14	10	32
26	70	20	8	2	19	24	50	11	25	22
27	19	18	4	18	7	10	2	2	8	12
28	50	16	4	25	65	29	62	9	9	12
29	11	62	41	27	24	25	27	11	3	20
30	22	40	10	60	6	21	13	10	8	36
31	15	80	12	75	4	13	15	36	14	9
32	2	12	29	16	25	57	33	11	26	7
33	25	4	35	7	20	14	18	18	8	7
34	20	22	12	7	11	18	42	32	9	6
35	12	44	7	11	3	19	19	34	14	73
36	29	37	4	24	12	59	20	26	25	5
37	18	8	47	12	23	70	14	26	12	10
38	7	16	6	20	22	38	23	98	5	10
39	11	32	34	102	15	4	25	11	4	8
40	36	6	23	49	23	70	8	7	9	2
41	8	34	8	11	19	57	26	4	17	11
42	9	29	4	4	13	3	21	26	18	11
43	20	8	11	7	29	8	30	10	35	11
44	10	20	14	10	20	18	24	52	7	29
45	11	11	6	5	6	12	11	36	23	9
46	15	40	27	8	7	5	14	28	7	2
47	17	76	4	5	7	26	72	16	49	14
48	10	2	2	10	4	1	35	14	30	10
49	30	74	5	21	7	25	13	48	28	40
50	2	9	14	6	36	7	76	13	21	48

continued

	name:	N6a	name:	N6b	name:	N6c	name:	N6d	name:	N6e
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
51	9	15	33	7	23	59	7	17	11	44
52	10	22	18	5	18	50	7	18	3	18
53	12	47	18	15	1	3	25	8	13	13
54	12	6	5	9	14	40	12	22	16	16
55	25	7	28	5	7	46	13	45	87	30
56	4	18	20	21	20	40	2	8	5	26
57	2	5	13	4	37	7	12	29	8	3
58	7	18	15	7	4	23	37	20	24	23
59	13	2	18	3	32	27	13	29	10	38
60	19	4	28	15	5	16	26	94	22	6
61	12	12	56	9	5	30	6	13	23	9
62	1	2	14	4	7	11	1	1	67	16
63	10	13	4	25	9	11	13	21	32	24
64	37	11	56	19	5	13	8	12	1	4
65	20	10	16	14	14	86	26	23	10	12
66	5	22	13	56	8	25	28	15	6	11
67	36	7	13	34	6	6	9	9	16	3
68	6	1	50	24	9	7	21	20	22	8
69	9	16	7	33	4	14	3	5	39	8
70	12	18	7	21	16	9	28	15	8	20
71	7	32	7	2	14	5	22	6	4	26
72	17	16	2	4	2	9	6	14	55	8
73	17	76	7	22	16	18	9	5	63	18
74	23	24	5	3	15	6	23	9	18	5
75	14	15	2	2	26	17	7	14	20	4
76	22	18	4	1	8	4	18	17	4	1
77	3	9	65	17	6	14	6	9	6	13
78	6	2	11	12	28	73	3	16	10	12
79	2	13	13	10	7	12	42	26	22	22
80	10	8	37	29	2	7	9	26	49	15
81	5	25	6	1	35	11	60	14	35	11
82	18	7	55	15	24	10	5	12	3	13
83	19	3	8	5	25	9	18	41	1	1
84	2	1	3	15	6	5	12	7	9	11
85	3	4	24	35	62	22	5	34	7	10
86	5	8	14	11	49	8	2	3	35	20
87	105	32	12	8	47	19	20	31	28	9
88	7	40	6	22	7	14	37	12	31	11
89	1	4	2	10	24	20	23	19	4	2
90	1	3	8	3	18	25	20	29	7	19
91	4	1	4	11	6	21	14	28	1	6
92	7	3	22	8	41	41	28	7	7	9
93	19	9	19	21	69	18	46	7	47	16
94	71	35	36	12	2	12	97	21	2	7
95	101	19	5	12	22	5	10	2	5	13
96	4	8	110	16	40	7	15	11	66	11
97	11	11	47	9	117	20	48	9	40	9

	name:	**N7a**	**name:**	**N7b**	**name:**	**N7c**	**name:**	**N7d**	**name:**	**N7e**
	size:	197	size:	197	size:	197	size:	197	size:	197
	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200	optimum:	200 x 200
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
1	19	10	10	25	10	48	54	16	17	57
2	5	14	10	8	37	8	38	11	21	19
3	16	12	5	17	32	16	18	29	36	8
4	6	7	37	6	8	37	22	75	24	9
5	2	9	40	17	23	4	11	16	5	24
6	46	9	24	75	43	16	6	23	20	20
7	48	9	43	13	20	7	8	27	35	19
8	16	23	6	10	3	8	43	12	42	15
9	35	8	25	68	24	19	9	56	11	11
10	7	13	25	8	5	33	29	18	25	26
11	2	6	9	10	12	25	7	15	10	31
12	4	2	3	11	6	30	21	10	14	19
13	26	37	3	7	18	6	15	6	2	4
14	9	5	5	4	2	1	11	49	22	12
15	6	4	2	9	8	27	7	29	18	4
16	26	18	3	3	29	8	11	51	32	15
17	20	29	3	17	5	5	11	63	37	13
18	4	7	1	3	7	15	14	51	3	8
19	15	9	4	11	12	3	9	45	15	13
20	24	8	21	4	4	4	2	7	10	4
21	5	14	25	8	61	19	3	4	10	8
22	16	37	22	12	8	24	6	17	15	4
23	3	4	3	2	8	24	6	4	25	5
24	14	15	4	8	8	16	10	5	39	12
25	2	1	12	10	16	4	14	4	11	8
26	10	14	7	6	3	8	2	13	16	8
27	16	32	43	9	10	3	4	23	6	22
28	8	33	10	10	2	1	8	38	40	10
29	2	6	11	4	6	2	5	1	4	19
30	2	2	36	6	40	9	9	19	6	14
31	8	5	3	2	13	2	2	11	13	5
32	13	4	12	66	2	5	13	9	11	2
33	3	8	13	64	1	4	15	18	15	5
34	17	4	14	16	12	6	26	6	12	42
35	9	17	5	18	1	3	21	24	8	22
36	4	4	7	16	4	4	7	32	3	3
37	27	16	18	5	3	3	16	3	5	9
38	11	5	6	27	61	18	3	16	8	18
39	10	8	22	28	5	12	8	10	7	6
40	6	3	5	32	1	6	2	2	11	11
41	1	4	19	74	6	5	11	9	18	29
42	50	11	12	5	5	5	11	15	42	12
43	5	10	9	12	6	8	5	13	6	25
44	1	1	9	41	2	12	4	7	10	14
45	5	10	8	35	3	1	3	20	19	15
46	6	3	10	22	3	13	4	23	21	9
47	2	9	9	37	11	13	8	54	2	13
48	16	7	3	7	20	13	16	85	5	5
49	21	9	12	30	49	19	14	45	11	9
50	9	27	8	29	4	12	13	54	16	8
51	14	33	6	5	13	10	8	2	25	6
52	16	29	3	2	6	20	40	14	3	10
53	10	56	4	5	2	4	5	14	11	7
54	11	5	8	3	14	4	5	31	28	30
55	8	2	9	4	2	6	7	36	16	11
56	11	14	9	19	3	20	14	3	30	10
57	34	7	4	20	11	10	6	34	6	16
58	1	3	5	15	7	8	5	12	31	8
59	7	11	16	13	63	14	9	26	7	4
60	12	8	21	21	5	1	13	27	6	13
61	76	23	1	4	1	6	34	7	15	47
62	7	5	4	4	11	2	14	38	11	18
63	7	11	2	8	4	5	13	54	9	3
64	3	6	2	13	10	22	5	33	2	10
65	6	2	2	4	30	6	7	37	5	21
66	22	10	14	5	4	6	8	18	2	9

continued

	name:	N7a	name:	N7b	name:	N7c	name:	N7d	name:	N7e
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
67	2	4	16	5	1	5	15	8	12	7
68	4	14	8	18	8	6	4	10	53	17
69	3	10	4	9	3	3	27	24	6	38
70	10	11	14	46	7	3	5	19	24	6
71	6	1	20	35	4	2	3	1	32	13
72	6	36	7	32	3	8	8	3	8	12
73	1	6	14	79	2	1	1	5	8	17
74	17	16	42	13	25	12	2	2	22	22
75	5	10	51	9	60	11	10	3	9	20
76	9	6	10	2	15	6	5	8	14	18
77	13	11	4	1	4	21	10	2	64	23
78	8	30	36	19	7	4	14	6	14	37
79	2	7	51	9	4	1	2	13	8	10
80	1	1	2	1	1	3	9	16	4	10
81	11	6	2	4	3	13	5	2	18	4
82	18	5	12	3	8	10	14	15	18	102
83	7	47	25	12	28	16	9	13	5	6
84	2	6	8	4	11	69	5	9	13	8
85	12	48	6	12	21	62	13	13	2	9
86	4	27	4	12	18	50	3	7	2	11
87	9	20	4	8	12	71	2	5	4	5
88	5	31	1	4	3	1	7	19	3	7
89	17	18	4	2	57	11	2	1	5	2
90	6	41	46	28	28	10	15	18	9	2
91	20	37	1	2	21	7	9	2	12	2
92	27	61	3	8	44	13	7	3	2	11
93	53	28	2	6	60	9	36	11	27	5
94	16	41	18	20	29	21	11	51	26	10
95	6	13	7	4	5	30	18	4	15	6
96	11	38	10	4	22	4	3	8	6	5
97	18	10	2	5	26	8	15	10	1	6
98	5	32	15	3	12	6	9	7	1	2
99	14	21	3	1	66	13	8	27	3	20
100	25	7	6	3	8	2	6	28	27	27
101	2	12	1	4	3	1	13	27	3	4
102	12	54	2	2	1	5	41	6	1	1
103	14	53	11	14	2	10	2	9	5	4
104	10	26	30	10	1	4	3	6	16	3
105	8	22	11	11	34	9	4	4	73	15
106	11	25	9	45	2	6	2	8	11	27
107	17	26	8	2	21	23	1	2	4	15
108	8	5	7	6	8	9	5	29	7	24
109	11	4	8	2	4	26	7	13	6	15
110	9	19	21	16	24	18	6	15	6	10
111	10	21	10	4	17	14	18	40	16	11
112	17	15	2	4	20	4	16	6	10	9
113	7	9	7	42	21	41	4	1	5	8
114	9	2	10	34	12	3	44	11	3	1
115	2	3	9	8	6	9	1	2	2	12
116	5	3	8	4	10	15	3	10	13	11
117	3	3	22	8	2	6	26	10	1	5
118	6	1	25	34	12	10	4	8	5	16
119	8	2	19	4	8	32	20	5	4	8
120	31	9	4	26	13	14	14	67	10	2
121	6	4	5	30	14	74	30	29	2	9
122	5	7	24	33	4	21	6	20	5	6
123	13	4	38	12	7	3	5	30	5	15
124	13	3	55	8	1	6	9	10	17	7
125	27	9	9	19	6	6	30	6	12	16
126	93	18	18	9	1	3	16	16	43	7
127	16	44	22	14	14	14	4	8	18	12
128	38	22	8	4	15	18	3	2	2	13
129	12	67	7	8	2	3	9	6	2	9
130	15	9	8	3	11	8	21	4	7	11
131	93	14	4	17	13	35	9	11	6	1
132	15	26	15	18	6	6	12	21	3	5
133	3	1	11	11	15	9	12	27	2	8
134	11	51	2	7	11	30	4	1	4	4

continued

	name: N7a		name: N7b		name: N7c		name: N7d		name: N7e	
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
135	15	50	6	4	10	27	9	35	17	8
136	10	16	4	18	10	28	3	8	17	4
137	7	6	4	14	8	7	10	34	3	11
138	5	7	13	3	13	47	10	34	6	41
139	16	49	10	10	3	9	30	7	18	58
140	21	11	8	5	3	3	3	1	31	9
141	5	17	6	5	15	27	13	27	12	16
142	42	21	18	3	18	6	32	12	9	4
143	25	12	30	5	7	21	5	14	7	4
144	5	11	13	2	12	6	5	7	8	18
145	2	1	5	5	12	8	8	26	9	7
146	7	10	2	7	2	4	4	25	8	2
147	6	12	9	56	16	53	2	10	4	18
148	9	8	14	4	17	4	2	7	5	5
149	6	6	19	3	13	52	3	14	24	7
150	4	6	59	14	8	15	29	15	13	2
151	6	1	21	11	7	15	14	25	6	4
152	18	5	45	12	5	34	9	5	20	16
153	12	10	5	17	11	47	7	12	7	17
154	10	29	8	2	26	5	16	11	4	18
155	18	5	2	3	30	11	5	5	8	14
156	11	14	6	2	3	15	11	9	5	2
157	7	3	4	15	14	8	3	14	43	7
158	6	6	4	20	4	25	9	6	3	6
159	5	5	7	40	6	1	7	15	2	2
160	20	3	10	19	8	6	1	4	11	12
161	2	8	1	1	3	4	6	3	12	11
162	16	19	5	2	5	12	3	11	26	4
163	7	4	3	1	19	37	6	7	22	32
164	2	12	9	11	14	19	2	6	12	43
165	6	5	27	15	6	7	3	9	16	13
166	1	3	4	3	8	11	2	1	34	13
167	7	10	17	19	1	2	4	4	14	52
168	13	5	9	16	2	5	3	3	3	8
169	19	5	3	20	16	31	21	8	3	13
170	6	7	33	10	14	8	21	5	43	7
171	11	18	63	16	9	3	18	3	4	6
172	6	20	8	35	20	45	13	4	3	1
173	17	3	1	4	8	30	65	22	7	5
174	13	8	9	5	3	15	50	10	9	30
175	7	2	26	8	8	3	30	16	7	38
176	3	16	2	10	9	4	27	9	8	37
177	24	9	5	21	13	12	49	11	80	16
178	3	16	5	30	20	9	1	6	2	11
179	15	5	46	14	4	11	17	4	1	5
180	17	5	8	8	7	19	2	5	4	6
181	2	2	1	4	25	31	3	3	12	33
182	8	3	19	23	7	15	13	16	15	21
183	12	14	7	2	14	5	6	18	57	10
184	10	6	9	15	6	2	2	2	8	13
185	29	8	46	7	10	15	1	6	14	25
186	22	4	25	6	10	28	2	9	8	19
187	8	7	9	21	10	3	15	5	47	18
188	16	10	4	4	22	9	4	4	10	3
189	3	4	83	16	2	5	31	5	15	7
190	19	9	32	8	9	4	20	4	21	5
191	26	8	8	2	17	4	6	1	18	15
192	32	5	6	11	20	9	74	14	21	8
193	4	5	4	13	66	15	71	13	5	2
194	7	2	4	9	42	13	36	11	16	11
195	13	3	12	9	14	12	11	8	20	9
196	11	3	41	7	3	5	2	2	23	7
197	13	3	10	4	23	7	8	6	36	6

A.2 Bin Packing

Table A.5: Constructed test problems for 2D rectangular bin packing

problem name	item number	object number
M1a, M1b, M1c, M1d, M1e	100	16
M2a, M2b, M2c, M2d, M2e	100	18
M3a, M3b, M3c, M3d, M3e	150	20

objects for problem category: **M1**

size: 16

width	height	number
20	20	2
10	30	3
10	10	3
20	30	2
10	20	3
10	40	3

objects for problem category: **M2**

size: 18

width	height	number
60	40	2
30	90	3
30	30	4
60	90	2
30	60	4
30	120	3

objects for problem category: **M3**

size: 20

width	height	number
60	50	3
30	100	3
40	30	3
60	100	4
30	60	4
30	120	3

	name:	**M1a**	**name:**	**M1b**	**name:**	**M1c**	**name:**	**M1d**	**name:**	**M1e**
	size:	100	size:	100	size:	100	size:	100	size:	100
	item area:	2520	item area:	2597	item area:	2554	item area:	2846	item area:	2665
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
1	1	6	7	9	4	9	8	9	6	9
2	2	8	2	4	4	4	6	9	5	5
3	6	5	2	9	9	8	1	6	5	9
4	4	9	5	9	5	8	8	4	9	2
5	8	7	1	6	7	4	3	3	3	2
6	2	8	1	4	3	1	1	6	7	9
7	7	5	9	3	5	9	4	1	6	1
8	3	1	3	7	4	3	1	5	6	1
9	1	4	4	2	6	3	8	9	6	5
10	2	2	6	6	7	9	4	7	5	1
11	9	5	5	5	4	9	6	8	9	2
12	2	1	6	6	6	8	8	6	6	1
13	1	4	8	8	6	2	6	1	2	6
14	5	6	6	7	3	4	8	5	3	3
15	6	6	6	4	1	4	6	6	6	4
16	2	6	2	7	1	6	6	8	4	2
17	5	4	5	9	8	6	7	4	7	2
18	1	6	3	2	7	8	1	3	5	5
19	8	8	6	7	5	7	4	3	8	4
20	5	3	6	4	2	7	7	4	1	4
21	8	7	5	1	7	7	8	2	4	2
22	9	9	7	6	6	7	7	6	8	7
23	5	2	6	7	1	5	9	2	9	6
24	5	3	8	2	3	2	5	9	6	5
25	8	2	6	8	9	6	8	9	7	7
26	8	8	9	6	3	9	2	9	9	5
27	9	9	7	1	4	7	8	9	4	2
28	6	4	7	9	2	2	8	9	9	8
29	3	3	3	6	3	5	6	2	5	1
30	4	1	3	2	9	4	6	3	8	7
31	7	1	8	7	2	9	3	7	1	4
32	3	3	2	3	7	8	8	2	2	2
33	6	7	1	1	2	5	7	5	2	6
34	8	7	6	8	1	7	8	2	2	6
35	5	2	5	5	2	4	3	6	6	7
36	7	5	4	3	3	9	5	7	7	9
37	5	9	5	5	1	3	2	7	7	3
38	7	1	2	3	2	3	6	7	8	4
39	6	4	7	3	2	6	2	1	9	7
40	7	6	8	9	8	9	3	7	2	2
41	6	4	8	7	2	8	8	6	4	2
42	2	3	3	9	6	6	4	8	6	2
43	4	8	2	1	9	5	3	6	9	2
44	5	9	8	5	9	6	3	2	4	3
45	7	4	6	9	2	6	9	3	2	2
46	2	6	6	3	6	4	1	7	1	7
47	5	1	1	5	9	5	2	1	7	8
48	7	5	3	3	6	5	1	4	1	5
49	2	9	9	5	3	4	6	4	4	8
50	2	9	6	2	4	1	4	4	5	5

continued

	name:	M1a	name:	M1b	name:	M1c	name:	M1d	name:	M1e
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
51	7	3	5	4	8	2	1	5	7	3
52	4	1	5	8	1	4	7	4	6	1
53	9	7	5	4	6	1	6	8	9	9
54	2	8	7	5	2	2	5	6	5	6
55	8	6	6	4	6	5	1	5	7	7
56	2	2	1	7	6	7	3	4	3	9
57	8	5	1	6	4	7	8	5	8	2
58	2	5	8	5	4	8	1	6	5	1
59	7	4	7	5	6	2	7	8	5	8
60	3	6	4	8	9	8	8	8	7	1
61	7	7	5	4	4	9	3	8	7	5
62	7	5	9	6	2	9	6	7	9	8
63	2	4	2	4	6	4	8	7	8	3
64	5	6	2	7	6	7	8	9	4	5
65	4	1	9	6	1	3	9	5	1	6
66	9	9	1	3	3	4	4	1	1	7
67	5	4	1	6	9	7	4	8	7	2
68	5	4	2	8	7	9	4	2	2	1
69	8	3	8	6	4	4	1	6	7	1
70	5	3	7	8	9	3	7	1	4	7
71	9	3	1	2	4	6	1	4	8	4
72	7	6	7	6	4	7	7	8	9	6
73	2	7	2	1	7	7	2	2	3	5
74	8	4	2	6	7	2	1	4	1	2
75	5	9	4	3	2	3	6	1	2	9
76	6	1	4	6	7	9	9	6	3	4
77	5	2	8	8	4	5	4	1	8	3
78	8	6	5	6	2	5	3	4	9	3
79	6	2	8	9	6	4	6	8	3	2
80	8	2	2	6	7	1	2	6	3	8
81	1	7	4	3	1	1	5	9	8	8
82	3	9	2	8	1	4	5	2	8	6
83	3	4	1	4	3	7	3	8	1	2
84	2	7	6	3	2	4	4	5	2	4
85	8	7	5	3	1	7	6	1	5	2
86	6	7	3	1	7	2	9	7	9	6
87	3	2	8	2	7	6	2	5	5	9
88	1	1	3	8	6	5	7	9	5	7
89	8	8	9	8	1	5	7	6	9	9
90	2	2	7	6	5	9	6	9	8	8
91	5	1	4	8	7	5	7	3	2	4
92	2	5	3	8	7	7	6	3	3	2
93	7	7	3	3	4	4	5	7	9	7
94	5	1	5	9	2	1	3	5	7	1
95	4	2	1	1	3	8	5	9	6	8
96	7	3	5	4	2	3	7	7	7	8
97	4	3	1	3	5	1	6	5	2	7
98	6	5	9	3	4	5	8	7	4	6
99	6	2	2	2	6	7	9	2	9	9
100	5	9	1	1	4	4	6	9	1	7

	name:	**M2a**	**name:**	**M2b**	**name:**	**M2c**	**name:**	**M2d**	**name:**	**M2e**
	size:	100	size:	100	size:	100	size:	100	size:	100
	item area:	24753	item area:	25283	item area:	25200	item area:	25614	item area:	24811
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
1	3	19	22	28	13	28	24	27	18	28
2	6	25	6	12	13	13	18	27	15	15
3	19	16	6	28	28	25	3	18	15	28
4	13	28	16	28	16	25	24	12	28	6
5	25	22	3	19	22	13	9	9	9	6
6	6	25	3	12	9	3	3	18	21	28
7	22	16	28	9	16	28	12	3	18	3
8	9	3	9	22	13	9	3	15	18	3
9	3	13	12	6	19	9	24	27	18	15
10	6	6	19	19	22	28	12	21	15	3
11	28	16	16	16	13	28	18	24	28	6
12	6	3	19	19	19	25	24	18	18	3
13	3	13	25	25	19	6	18	3	6	18
14	16	19	19	22	9	13	24	15	9	9
15	19	19	19	12	3	13	18	18	18	12
16	6	19	6	22	3	19	18	24	12	6
17	16	13	16	28	25	19	21	12	21	6
18	3	19	9	6	22	25	3	9	15	15
19	25	25	19	22	16	22	12	9	25	12
20	16	9	19	12	6	22	21	12	3	12
21	25	22	16	3	22	22	24	6	12	6
22	28	28	22	19	19	22	21	18	25	21
23	16	6	19	22	3	16	27	6	28	18
24	16	9	25	6	9	6	15	27	18	15
25	25	6	19	25	28	19	24	27	21	21
26	25	25	28	19	9	28	6	27	28	15
27	28	28	22	3	13	22	24	27	12	6
28	19	13	22	28	6	6	24	27	28	25
29	9	9	9	19	9	16	18	6	15	3
30	13	3	9	6	28	13	18	9	25	21
31	22	3	25	22	6	28	9	21	3	12
32	9	9	6	9	22	25	24	6	6	6
33	19	22	3	3	6	16	21	15	6	18
34	25	22	19	25	3	22	24	6	6	18
35	16	6	16	16	6	13	9	18	18	21
36	22	16	12	9	9	28	15	21	21	28
37	16	28	16	16	3	9	6	21	21	9
38	22	3	6	9	6	9	18	21	25	12
39	19	13	22	9	6	19	6	3	28	21
40	22	19	25	28	25	28	9	21	6	6
41	19	13	25	22	6	25	24	18	12	6
42	6	9	9	28	19	19	12	24	18	6
43	13	25	6	3	28	16	9	18	28	6
44	16	28	25	16	28	19	9	6	12	9
45	22	13	19	28	6	19	27	9	6	6
46	6	19	19	9	19	13	3	21	3	21
47	16	3	3	16	28	16	6	3	21	25
48	22	16	9	9	19	16	3	12	3	15
49	6	28	28	16	9	13	18	12	12	25
50	6	28	19	6	13	3	12	12	15	15

continued

	name:	M2a	name:	M2b	name:	M2c	name:	M2d	name:	M2e
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
51	22	9	16	12	25	6	3	15	21	9
52	13	3	16	25	3	13	21	12	18	3
53	28	22	16	12	19	3	18	24	28	28
54	6	25	22	16	6	6	15	18	15	18
55	25	19	19	12	19	16	3	15	21	21
56	6	6	3	22	19	22	9	12	9	28
57	25	16	3	19	13	22	24	15	25	6
58	6	16	25	16	13	25	3	18	15	3
59	22	13	22	16	19	6	21	24	15	25
60	9	19	12	25	28	25	24	24	21	3
61	22	22	16	12	13	28	9	24	21	15
62	22	16	28	19	6	28	18	21	28	25
63	6	13	6	12	19	13	24	21	25	9
64	16	19	6	22	19	22	24	27	12	15
65	13	3	28	19	3	9	27	15	3	18
66	28	28	3	9	9	13	12	3	3	21
67	16	13	3	19	28	22	12	24	21	6
68	16	13	6	25	22	28	12	6	6	3
69	25	9	25	19	13	13	3	18	21	3
70	16	9	22	25	28	9	21	3	12	21
71	28	9	3	6	13	19	3	12	25	12
72	22	19	22	19	13	22	21	24	28	18
73	6	22	6	3	22	22	6	6	9	15
74	25	13	6	19	22	6	3	12	3	6
75	16	28	12	9	6	9	18	3	6	28
76	19	3	12	19	22	28	27	18	9	12
77	16	6	25	25	13	16	12	3	25	9
78	25	19	16	19	6	16	9	12	28	9
79	19	6	25	28	19	13	18	24	9	6
80	25	6	6	19	22	3	6	18	9	25
81	3	22	12	9	3	3	15	27	25	25
82	9	28	6	25	3	13	15	6	25	18
83	9	13	3	12	9	22	9	24	3	6
84	6	22	19	9	6	13	12	15	6	12
85	25	22	16	9	3	22	18	3	15	6
86	19	22	9	3	22	6	27	21	28	18
87	9	6	25	6	22	19	6	15	15	28
88	3	3	9	25	19	16	21	27	15	21
89	25	25	28	25	3	16	21	18	28	28
90	6	6	22	19	16	28	18	27	25	25
91	16	3	12	25	22	16	21	9	6	12
92	6	16	9	25	22	22	18	9	9	6
93	22	22	9	9	13	13	15	21	28	21
94	16	3	16	28	6	3	9	15	21	3
95	13	6	3	3	9	25	15	27	18	25
96	22	9	16	12	6	9	21	21	21	25
97	13	9	3	9	16	3	18	15	6	21
98	19	16	28	9	13	16	24	21	12	18
99	19	6	6	6	19	22	27	6	28	28
100	16	28	3	3	13	13	18	27	3	21

	name:	**M3a**	**name:**	**M3b**	**name:**	**M3c**	**name:**	**M3d**	**name:**	**M3e**
	size:	150	size:	150	size:	150	size:	150	size:	150
	item area:	25703	item area:	27414	item area:	27953	item area:	27430	item area:	29530
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
1	3	19	16	13	25	28	22	10	7	30
2	6	26	16	26	19	28	19	3	7	3
3	19	16	16	13	3	19	29	29	7	27
4	13	29	23	16	25	13	16	19	10	20
5	26	22	20	13	9	9	22	22	13	17
6	6	26	3	23	3	19	10	29	20	27
7	22	16	3	20	13	3	25	6	30	30
8	10	3	26	16	3	16	16	3	3	7
9	3	13	23	16	25	28	16	25	20	27
10	6	6	13	26	13	22	22	3	27	27
11	29	16	16	13	19	25	22	16	13	17
12	6	3	29	20	25	19	29	25	23	17
13	3	13	7	13	19	3	25	10	10	30
14	16	19	7	23	25	16	13	16	30	23
15	19	19	29	20	19	19	3	19	20	10
16	6	19	3	10	19	25	3	22	7	7
17	16	13	3	20	22	13	22	6	20	17
18	3	19	7	26	3	9	6	3	13	10
19	26	26	26	20	13	9	22	3	17	23
20	16	10	23	26	22	13	13	22	30	23
21	26	22	3	7	25	6	25	13	30	10
22	29	29	23	20	22	19	29	19	17	10
23	16	6	7	3	28	6	10	16	30	17
24	16	10	7	20	16	28	3	6	10	17
25	26	6	13	10	25	28	6	29	20	13
26	26	26	13	20	6	28	10	13	13	7
27	29	29	26	26	25	28	25	10	10	7
28	19	13	16	20	25	28	29	10	30	23
29	10	10	26	29	19	6	10	6	27	20
30	13	3	7	20	19	9	10	25	23	10
31	22	3	13	10	9	22	25	25	23	3
32	10	10	7	26	25	6	25	19	23	27
33	19	22	3	13	22	16	3	6	13	3
34	26	22	20	10	25	6	6	13	17	3
35	16	6	16	10	9	19	16	6	27	20
36	22	16	10	3	16	22	29	19	17	20
37	16	29	26	7	6	22	16	29	20	23
38	22	3	10	26	19	22	16	22	20	23
39	19	13	29	26	6	3	29	29	23	27
40	22	19	23	20	9	22	25	25	7	13
41	19	13	13	26	25	19	6	13	13	3
42	6	10	10	26	13	25	10	6	17	10
43	13	26	10	10	9	19	29	22	3	10
44	16	29	16	29	9	6	22	3	30	17
45	22	13	3	3	28	9	19	25	20	3
46	6	19	16	13	3	22	22	25	13	30
47	16	3	3	10	6	3	6	22	27	13
48	22	16	29	10	3	13	13	19	10	10
49	6	29	7	7	19	13	29	29	7	3
50	6	29	3	3	13	13	3	22	7	10

continued

	name:	M3a	name:	M3b	name:	M3c	name:	M3d	name:	M3e
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
51	22	10	13	29	3	16	16	3	27	20
52	13	3	13	13	22	13	10	13	3	10
53	29	22	29	26	19	25	6	29	7	7
54	6	26	16	26	16	19	16	10	20	7
55	26	19	23	13	3	16	22	29	20	3
56	6	6	10	3	9	13	6	3	17	23
57	26	16	16	29	25	16	22	10	13	3
58	6	16	13	10	3	19	19	25	10	23
59	22	13	20	10	22	25	25	10	27	20
60	10	19	23	29	25	25	16	29	27	30
61	22	22	13	29	9	25	25	19	20	20
62	22	16	20	26	19	22	10	25	20	20
63	6	13	20	7	25	22	13	13	7	30
64	16	19	10	13	25	28	3	19	10	3
65	13	3	3	13	28	16	13	6	10	10
66	29	29	3	20	13	3	10	29	13	27
67	16	13	26	20	13	25	6	29	10	30
68	16	13	23	26	13	6	6	25	13	23
69	26	10	16	23	3	19	29	16	7	3
70	16	10	7	23	22	3	6	19	20	13
71	29	10	23	23	3	13	19	19	13	23
72	22	19	20	23	22	25	13	3	30	10
73	6	22	3	16	6	6	25	6	30	13
74	26	13	10	7	3	13	10	10	30	20
75	16	29	29	20	19	3	22	19	7	10
76	19	3	10	29	28	19	29	19	13	17
77	16	6	13	23	13	3	25	13	10	7
78	26	19	7	7	9	13	25	22	3	3
79	19	6	10	16	19	25	6	16	23	27
80	26	6	29	13	6	19	3	3	23	27
81	3	22	7	29	16	28	22	13	3	17
82	10	29	23	26	16	6	29	16	20	30
83	10	13	7	16	9	25	22	3	10	20
84	6	22	3	23	13	16	19	3	23	27
85	26	22	7	13	19	3	22	29	30	27
86	19	22	10	29	28	22	16	22	20	10
87	10	6	3	10	6	16	10	25	27	7
88	3	3	7	10	22	28	19	19	17	23
89	26	26	7	20	22	19	10	25	13	30
90	6	6	26	29	19	28	16	22	3	13
91	16	3	7	26	22	9	13	3	3	23
92	6	16	20	20	19	9	16	19	10	7
93	22	22	29	16	16	22	13	13	10	7
94	16	3	29	20	9	16	3	10	27	23
95	13	6	7	20	16	28	19	25	3	17
96	22	10	20	13	22	22	13	29	3	20
97	13	10	29	16	19	16	13	16	30	20
98	19	16	20	16	25	22	25	13	27	13
99	19	6	10	13	28	6	22	10	13	13
100	16	29	13	3	19	28	19	16	20	27

continued

	name:	M3a	name:	M3b	name:	M3c	name:	M3d	name:	M3e
no	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**	**width**	**height**
101	22	29	26	7	19	28	19	3	30	10
102	6	13	3	13	16	16	13	10	3	20
103	6	29	20	3	16	28	29	25	3	20
104	16	29	7	7	28	6	29	19	13	30
105	3	19	20	16	9	6	19	3	20	7
106	3	13	20	23	22	28	13	10	13	13
107	29	10	13	23	19	3	16	3	20	27
108	10	22	13	26	19	3	25	10	3	3
109	13	6	20	7	19	16	25	29	13	10
110	19	19	29	26	16	3	19	6	10	27
111	16	16	13	29	28	6	10	19	27	10
112	19	19	7	29	19	3	25	10	7	20
113	26	26	20	13	6	19	10	22	17	30
114	19	22	20	23	9	9	25	10	13	13
115	19	13	3	10	19	13	6	16	13	13
116	6	22	10	13	13	6	29	19	20	27
117	16	29	29	23	22	6	3	19	13	13
118	10	6	23	29	16	16	10	6	30	17
119	19	22	13	13	25	13	6	10	13	20
120	19	13	29	10	3	13	16	29	23	3
121	16	3	13	20	13	6	3	3	7	7
122	22	19	13	23	25	22	13	10	3	10
123	19	22	23	23	28	19	16	16	17	20
124	26	6	23	7	19	16	13	29	10	3
125	19	26	7	10	22	22	19	10	20	7
126	29	19	23	29	28	16	16	13	10	27
127	22	3	13	16	13	6	29	16	13	30
128	22	29	7	16	28	25	25	25	7	3
129	10	19	20	13	16	3	25	19	17	10
130	10	6	23	3	25	22	25	10	3	20
131	26	22	3	3	3	13	29	10	23	3
132	6	10	3	13	6	6	29	22	30	13
133	3	3	10	23	6	19	6	22	30	17
134	19	26	7	13	6	19	16	6	10	10
135	16	16	3	23	19	22	3	3	17	10
136	13	10	23	7	22	28	16	25	13	23
137	16	16	23	20	22	9	6	3	27	13
138	6	10	20	16	25	13	6	25	13	23
139	22	10	3	16	28	22	25	25	23	3
140	26	29	16	29	6	6	19	10	3	23
141	26	22	23	16	13	6	25	29	27	20
142	10	29	23	23	19	6	25	10	13	17
143	6	3	13	13	28	6	16	3	3	13
144	26	16	7	3	13	9	3	3	7	13
145	19	29	10	26	6	6	13	22	30	10
146	19	10	7	10	3	22	13	6	10	7
147	3	16	16	3	22	25	10	6	13	7
148	10	10	13	16	3	16	22	6	10	7
149	29	16	20	23	13	25	3	16	7	13
150	19	6	13	13	16	16	13	25	3	17

B. Test Problems for Irregular Packing

B.1 Benchmark Problems in Literature

B.1.1 Overview over Benchmark Problems in Literature

Description of table entries:

reference: publication in which test problem has been used
name: name which the problem is referred to in this work
size: number of items
shapes: geometric shape type which the problem consists of
source: source where the co-ordinates used for the experiments in this work have been obtained from; i.e. supplied by authors, stated in publication, extracted from sample layout in publication or extracted from scanned sample layout in publication
factor: scaling factor between problem instance used in the current work and the problem used in the publication; only stated if dimensions are used in publication

Table B.1: Irregular test problems from literature: textile industry

reference	name	size	problem type	shapes	source	factor
Ratanapan and Dagli (1997b)	Dagli	30	textile	polygons, non-polygonal pieces with arcs	scanned from sample layout in paper; approximated by polygons	
Bounsaythip and Maouche (1997)	Mao	20	textile	polygons, non-polygonal pieces with arcs	scanned from sample layout in paper; approximated by polygons	5
Marques et al. (1991)	Marques	24	textile	polygons, non-polygonal pieces with arcs	scanned from sample layout in paper; approximated by polygons	
Albano and Sapuppo (1980)	Albano	24	textile	polygons, non-polygonal pieces with arcs	scanned from sample layout in paper; approximated by polygons	
Oliveira et al. (2000)	SHIRTS	99	textile	polygons	co-ordinates stated in paper	1
Oliveira et al. (2000)	TROUSERS	64	textile	polygons	co-ordinates stated in paper	1
Oliveira et al. (2000)	SWIM	48	textile	polygons	co-ordinates stated in paper; rounded to the nearest integer	1

Table B.2: Irregular test problems from literature: artificially created problems

reference	name	size	problem type	shapes	source	factor
Jakobs (1996)	Jakobs1	25	artificial	polygons	constructed from sample layout in paper	1
Jakobs (1996)	Jakobs2	25	artificial	polygons	constructed from sample layout in paper	2
Fujita et al. (1993)	Fu	12	artificial	convex polygons	scanned from sample layout in paper	
Han and Na (1996)	Han	23	artificial	polygons	scanned from sample layout in paper	
Blazewicz et al. (1993)	Blaz1	28	artificial	polygons, non-polygonal pieces with arcs	Oliveira et al. (2000)	
Blazewicz et al. (1993)	Blaz2	20	artificial	polygons, non-polygonal pieces with arcs	Oliveira et al. (2000)	
Oliveira et al. (2000)	SHAPES0	43	artificial	polygons, recti-linear shapes	co-ordinates stated in paper; rotation interval 0°	1
Oliveira et al. (2000)	SHAPES1	43	artificial		equal to SHAPES0, rotation interval 180°	1
Oliveira et al. (2000)	SHAPES	43	artificial		equal to SHAPES0, rotation interval 90°	1
Oliveira et al. (2000)	SHAPES2	28	artificial	polygons	co-ordinates stated in paper	1

Table B.3: Irregular test problems from literature with known optimum

reference	name	size	problem type	shapes	source
Dighe and Jakiela (1996)	Dighe1	16	jigsaw	polygons	constructed according to a sample layout in the paper
Dighe and Jakiela (1996)	Dighe2	10	jigsaw	polygons	constructed according to a sample layout in the paper

B.1.2 Test Problems

B.1.2.1 Problems from Textile Industry

name: **Dagli**
size: 30
object: width: 60

no.	quantity											
1	3	x	1	0	1	7	12	15	10	11	1	
		y	1	8	7	5	0	3	8	9	9	
2	3	x	2	2	0	4	4	5	5	9	9	5	2
		y	2	12	0	0	1	1	0	0	2	12	12
3	3	x	3	2	0	0	2	9	11	11	9		
		y	3	24	18	6	0	0	6	18	24		
4	3	x	4	0	0	14	14						
		y	4	15	0	0	15						
5	3	x	5	0	9	7	2						
		y	5	0	0	5	5						
6	3	x	6	0	1	3	6	11	5	2			
		y	6	32	3	0	1	11	31	33			
7	3	x	7	0	5	23	13	1					
		y	7	3	0	0	5	5					
8	3	x	8	1	4	5	4	4	3	2	1	1	0
		y	8	0	0	6	6	5	4	4	5	6	6
9	3	x	9	0	1	8	6						
		y	9	2	0	0	3						
10	3	x	10	0	14	5							
		y	10	0	0	6.4							

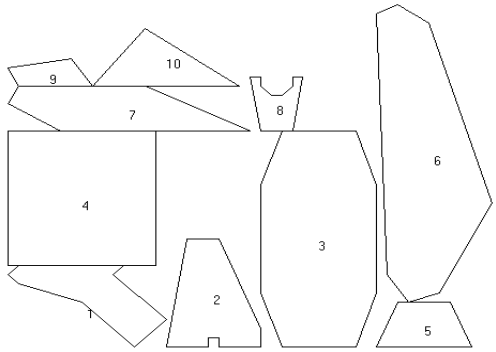


Figure B.1: Data set for test problem Dagli

name: **Mao**
size: 20
object: width: 2550

no.	quantity																			
1	2	x	10	478	727	856	930	1049	1097	1078	0									
		y	115	113	21	44	47	0	295	396	407									
2	2	x	0	316	316	0														
		y	0	2	274	275														
3	4	x	0	239	242	0														
		y	1	0	80	78														
4	2	x	0	0	384	384	759	758												
		y	310	119	0	70	69	310												
5	2	x	0	0	150	148														
		y	252	75	0	175														
6	2	x	852	503	387	355	300	45	0	0	55	145	329	407	539	597	668	736	784	849
		y	412	414	451	505	451	325	221	154	53	16	25	35	38	51	20	0	3	194
7	2	x	0	827	894	975	1097	1003	884	845	761	732	694	616	539	0				
		y	232	0	20	258	260	500	463	453	490	547	544	477	414	426				
8	2	x	151	97	0	75	126	191	271	316	359	436	442	694	726	1091	1094	1033	987	926
		y	427	266	259	28	21	61	50	64	10	6	43	75	42	0	211	208	346	393
9	2	x	0	745	619	590	522	6												
		y	0	24	196	276	260	282												

Figure B.2: Data set for test problem Mao

name: **Marques**
size: 24
object: width: 104

no.	quantity															
1	4	x	0	21	21	14	7	0								
		y	0	0	22	28	28	22								
2	4	x	0	2	11	19	21	17	11	4	0					
		y	5	0	2	0	5	4	6	4	5					
3	2	x	0	33	33	30	26	13	0							
		y	0	0	14	14	17	15	17							
4	2	x	0	4	4	0										
		y	0	0	39	39										
5	4	x	0	10	10	0										
		y	0	0	11	11										
6	4	x	2	0	2	0	2	4	2							
		y	20	18	10	2	0	10	20							
7	2	x	0	29	27	29	25	25	18	16	14	12	4	4	0	2
		y	0	0	12	22	26	33	37	35	35	37	34	26	22	12
8	2	x	0	33	37	35	28	25	14	0						
		y	0	0	6	13	13	15	13	15						

Figure B.3: Data set for test problem Marques

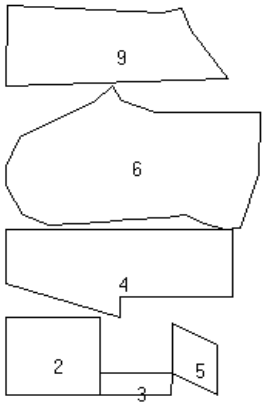


Figure B.4: Problem: Mao

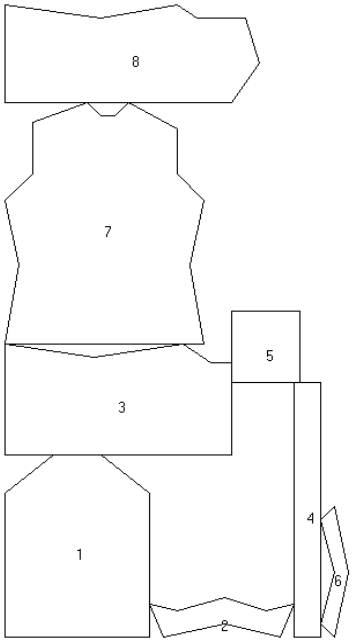


Figure B.5: Problem: Marques

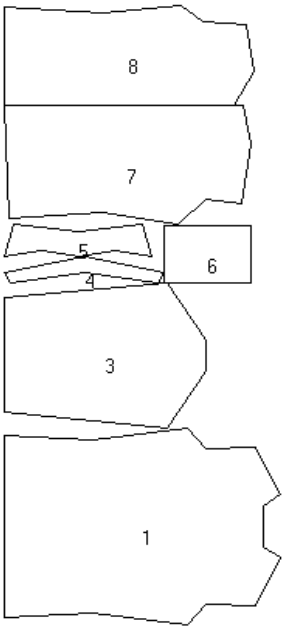


Figure B.6: Problem: Albano

name:		**Albano**														
size:		24														
object:		width:	4900													
no.	quantity															
1	2	x	0	966	1983	2185	2734	3000	2819	2819	3000	2734	2185	1983	966	0
		y	86	142	0	238	217	767	900	1360	1493	2043	2022	2260	2118	2174
2	2	x	0	3034	3034	0										
		y	0	0	261	261										
3	4	x	0	1761	2183	2183	1761	0								
		y	173	0	650	1010	1660	1487								
4	4	x	74	870	1666	1740	870	0								
		y	0	119	0	125	305	125								
5	4	x	0	411	800	1189	1600	1500	800	100						
		y	0	65	0	65	0	368	286	368						
6	4	x	0	936	936	0										
		y	0	0	659	659										
7	2	x	56	1066	1891	2186	2573	2676	2594	0						
		y	73	143	0	288	241	926	1366	1366						
8	2	x	0	2499	2705	2622	2148	1920	1061	0						
		y	0	0	387	934	967	1152	1059	1125						

Figure B.7: Data set for test problem: Albano

name:		**SHIRTS**									
size:		99									
object:		width:	40								
no.	quantity										
1	8	x	2	9	9	2	1	1	0	1	
		y	0	1	5	7	5	4	3	2	
2	8	x	1	12	13	12	12	11	7	4	1	0
		y	0	0	2	3	5	6	5	6	5	3
3	8	x	0	3	3	5	8	11	12	11	7	0
		y	1	1	0	1	1	0	5	9	8	8
4	15	x	0	4	4	3	0					
		y	0	0	2	3	3					
5	15	x	0	8	7	1						
		y	0	0	1	1						
6	15	x	0	4	4	0						
		y	0	0	1	1						
7	15	x	0	3	3	0						
		y	0	0	1	1						
8	15	x	1	9	9	7	5	4	3	0		
		y	0	0	2	2	1	1	2	2		

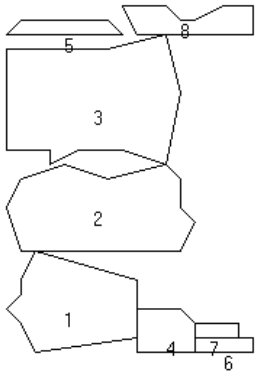


Figure B.8: Data set for test problem: SHIRTS

name:		**TROUSERS**								
size:		64								
object:		width:		79						
no.	quantity									
1	8	x	0	12	14	24	33	59	59	4
		y	3	2	0	3	4	5	16	16
2	8	x	0	0	27	37	41	47	56	56
		y	22	8	5	2	0	3	6	22
3	1	x	0	57	57	0				
		y	0	0	5	5				
4	1	x	0	52	52	0				
		y	0	0	5	5				
5	1	x	0	44	44	0				
		y	0	0	5	5				
6	1	x	0	42	42	0				
		y	0	0	5	5				
7	8	x	0	21	21	0				
		y	0	0	2	2				
8	8	x	0	15	13	0				
		y	0	4	7	6				
9	1	x	0	15	15	0				
		y	0	1	4	4				
10	1	x	0	14	14	0				
		y	0	0	5	5				
11	1	x	0	14	14	0				
		y	0	1	4	4				
12	1	x	0	13	13	0				
		y	0	0	5	5				
13	2	x	0	13	13	0				
		y	0	1	4	4				
14	2	x	0	12	12	0				
		y	0	0	5	5				
15	8	x	6	10	12	10	2	0	2	
		y	0	2	10	11	11	10	2	
16	8	x	0	3	8	8	2			
		y	2	0	2	10	10			
17	4	x	3	5	6	5	1	0	1	
		y	0	1	6	7	7	6	1	

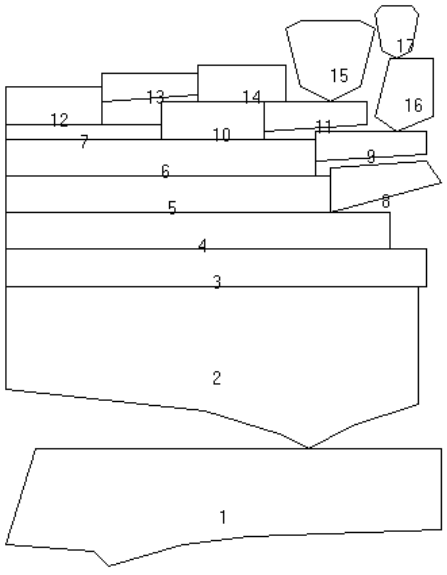


Figure B.9: Data set for test problem: TROUSERS

name: **SWIMi**
size: 48

item no.	**1**	**item no.**	**2**	**item no.**	**3**	**item no.**	**4**	**item no.**	**5**
quantity	3	quantity	6	quantity	6	quantity	6	quantity	6
x	**y**	**x**	**y**	**x**	**y**	**x**	**y**	**x**	**y**
5	837	0	546	648	435	0	248	826	940
0	516	8	398	479	392	8	76	860	797
6	195	27	246	333	358	309	82	903	672
156	177	68	83	112	312	484	83	915	552
264	146	179	97	0	261	616	81	0	501
428	99	288	100	275	236	718	76	2	342
552	67	401	83	399	220	876	54	865	333
685	36	548	49	592	177	1031	0	995	290
863	0	764	0	730	119	1351	137	1035	140
993	155	805	196	937	0	1209	191	1077	0
1080	217	887	350	1071	31	1049	241	1197	16
1216	284	755	370	1285	76	909	264	1180	214
1324	310	655	391	1459	135	785	274	1174	330
1486	311	523	425	1573	189	668	277	1176	528
1602	297	272	494	1700	258	511	274	1186	676
1742	273	117	534	1922	362	213	262	1208	846
1724	397			1932	489			1315	1172
1713	520			1940	613			1178	994
1723	643			1747	643			1043	940
1741	767			1599	665				
1601	742			1487	679				
1485	728			1331	690				
1323	728			1164	690				
1215	754			988	664				
1079	820			808	624				
991	882								
861	1036								
683	999								
550	968								
426	935								
262	887								
154	856								

name: **SWIMi** continued

item no.	**6**	**item no.**	**7**	**item no.**	**8**	**item no.**	**9**	**item no.**	**10**
quantity	6	quantity	3	quantity	6	quantity	6	quantity	3
x	**y**	**x**	**y**	**x**	**y**	**x**	**y**	**x**	**y**
0	29	40	620	882	623	105	545	856	1496
130	0	36	497	780	508	3	423	727	1418
307	59	32	329	688	402	0	259	624	1376
495	146	23	166	546	334	44	140	521	1361
616	208	0	0	0	325	127	0	358	1377
737	274	159	1	4	165	208	71	241	1400
855	326	286	82	592	148	299	181	41	1445
959	349	392	119	689	49	359	320	2	1277
1071	342	532	162	868	0	350	475	271	1216
1275	291	687	204	954	169	248	639	500	1165
1260	430	806	251	998	265			618	1139
1250	535	953	318	1052	396			743	1059
1260	642	1101	396	1098	518			779	869
1275	781	1086	520	1147	666			779	707
1071	730	1077	622	963	713			757	554
959	723	1101	844					498	412
855	746	953	922					269	362
737	798	806	989					0	302
616	864	687	1036					38	134
495	926	532	1078					238	178
307	1013	392	1121					356	201
130	1072	286	1158					519	216
0	1043	159	1239					622	201
		0	1240					724	158
		23	1074					851	81
		32	911					1037	0
		36	743					1350	70
								1301	279
								1279	426
								1272	647
								1273	787
								1273	927
								1277	1075
								1288	1211
								1353	1504
								1047	1577

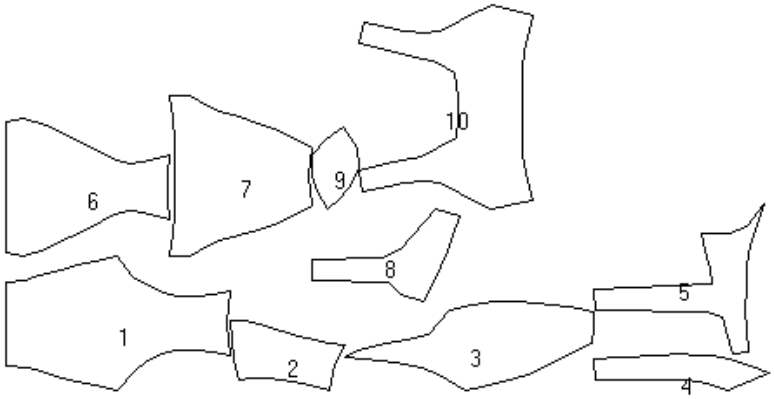


Figure B.10b: Data set for test problem: SWIM

B.1.2.2 Artificial Problems

name:		**Jakobs1**												
size:		25												
object:		width:		40										
no														
1	x	0	2	0										
	y	0	0	2										
2	x	0	3	0										
	y	0	0	3										
3	x	0	4	0										
	y	0	0	4										
4	x	5	5	0										
	y	0	5	5										
5	x	0	6	6										
	y	3	0	3										
6	x	0	7	7										
	y	4	0	4										
7	x	0	5	5	3	3	0							
	y	0	0	3	3	5	5							
8	x	0	4	4	2	2	0							
	y	0	0	1	1	4	4							
9	x	0	6	6	4	4	0							
	y	0	0	3	3	6	6							
10	x	0	5	5	3	3	0							
	y	0	0	2	2	1	1							
11	x	0	4	4	3	3	0							
	y	0	0	2	2	1	1							
12	x	0	6	6	4	4	0							
	y	0	0	3	3	6	6							
13	x	0	6	6	0									
	y	0	0	6	6									
14	x	0	5	5	0									
	y	0	0	5	5									
15	x	0	4	4	0									
	y	0	0	4	4									
16	x	2	4	4	6	6	4	4	2	2	0	0	2	
	y	0	0	2	2	4	4	6	6	4	4	2	2	
17	x	1	2	2	3	3	2	2	1	1	0	0	1	
	y	0	0	1	1	2	2	3	3	2	2	1	1	
18	x	2	4	4	6	6	4	4	2	2	0	0	2	
	y	0	0	2	2	4	4	6	6	4	4	2	2	
19	x	1	2	2	3	3	2	2	1	1	0	0	1	
	y	0	0	1	1	2	2	3	3	2	2	1	1	
20	x	0	6	6	0									
	y	0	0	3	3									
21	x	0	1	1	0									
	y	0	0	4	4									
22	x	0	5	5	0									
	y	0	0	2	2									
23	x	2	4	6	6	4	2	0	0					
	y	0	0	2	4	6	6	4	2					
24	x	3	6	8	8	6	3	0	0					
	y	0	0	2	4	6	6	4	2					
25	x	0	2	4	6	6	4	2	0					
	y	1	0	0	1	2	3	3	2					

name:		**Jakobs2**							
size:		25							
object:		width:		70					
no									
1	x	0	6	12	6				
	y	4	0	4	8				
2	x	5	10	10	0	0	5		
	y	0	0	10	10	5	5		
3	x	0	6	4	4	6	0	2	2
	y	0	0	2	4	6	6	4	2
4	x	0	8	6	6	8	0	2	2
	y	0	0	2	6	8	8	6	2
5	x	0	10	8	8	10	0	2	2
	y	0	0	2	8	10	10	8	2
6	x	0	6	6	4	4	2	2	0
	y	0	0	6	6	2	2	6	6
7	x	0	8	8	6	6	2	2	0
	y	0	0	8	8	2	2	8	8
8	x	0	10	10	8	8	2	2	0
	y	0	0	10	10	2	2	10	10
9	x	0	12	6					
	y	0	0	12					
10	x	0	8	4					
	y	0	0	8					
11	x	0	10	10					
	y	10	0	10					
12	x	6	12	6	0				
	y	0	8	16	8				
13	x	6	12	0					
	y	0	12	12					
14	x	0	8	4					
	y	0	0	8					
15	x	5	10	0					
	y	0	10	10					
16	x	4	8	4	0				
	y	0	5	10	5				
17	x	2	4	6	6	3	0	0	
	y	0	0	2	4	6	4	2	
18	x	2	6	8	8	4	0	0	
	y	0	0	2	6	8	6	2	
19	x	0	6	6	3	3	0		
	y	0	0	3	3	6	6		
20	x	4	8	8	0	0	4		
	y	0	0	8	8	4	4		
21	x	0	6	6	3	3	0		
	y	0	0	3	3	6	6		
22	x	4	8	8	0	0	4		
	y	0	0	8	8	4	4		
23	x	0	12	12	0				
	y	0	0	12	12				
24	x	0	8	8	0				
	y	0	0	8	8				
25	x	0	10	10	0				
	y	0	0	10	10				

Figure B.11: Data set for test problem Jakobs1 and Jakobs2

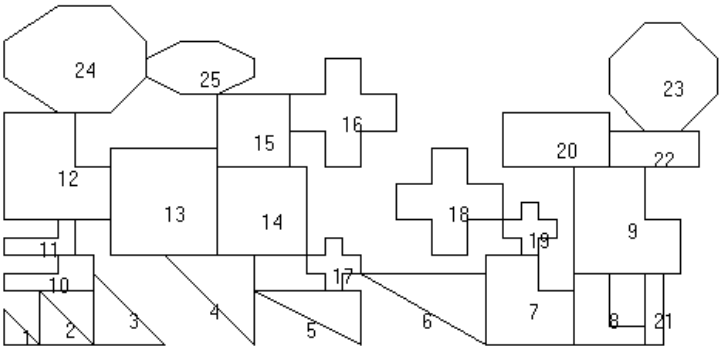


Figure B.12: Data set for test problem Jakobs1

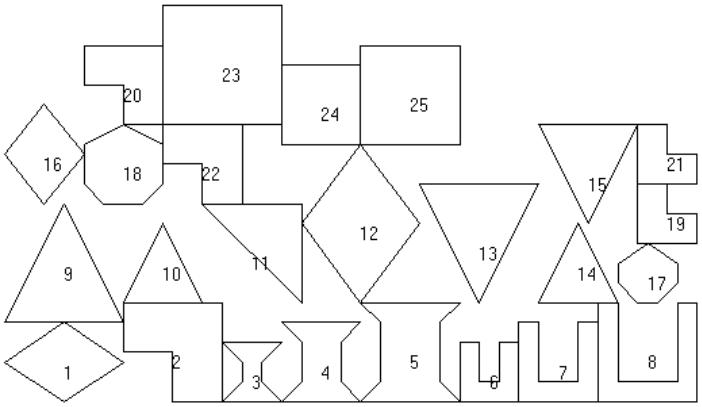


Figure B.13: Data set for test problem Jakobs2

name:		**Fu**			
size:		12			
object:		width:		38	
no.					
1	x	0	10	10	0
	y	0	0	10	10
2	x	0	10	10	0
	y	0	0	10	10
3	x	0	14	14	0
	y	0	0	9	9
4	x	0	14	7	
	y	0	0	7	
5	x	0	0	14	
	y	9	0	9	
6	x	0	14	14	0
	y	0	0	14	14
7	x	0	10	10	0
	y	0	4	9	9
8	x	0	5	5	0
	y	0	0	9	9
9	x	0	14	14	
	y	0	0	14	
10	x	0	10	10	0
	y	0	0	10	14
11	x	0	4	8	
	y	8	0	8	
12	x	0	14	7	
	y	0	0	12	

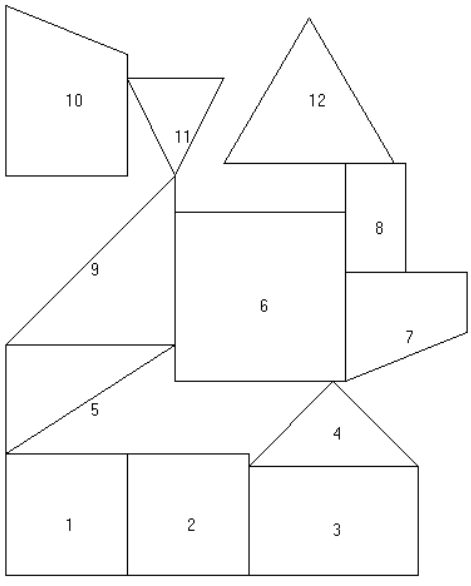


Figure B.14: Data set for test problem Fu

name:		**Han**											
size:			23										
object:		width:	58										
no.	quantity												
1	1	x	0	5	5								
		y	0	0	13								
2	1	x	9	13	17	18	17	13	9	5	1	0	1
		y	0	1	5	9	13	17	18	17	13	9	5
3	1	x	2	6	8	8	6	2	0				
		y	0	0	2	9	11	11	9				
4	1	x	0	5	5								
		y	0	0	6								
5	1	x	5	7	4	2	2	4	7	5	2	0	0
		y	0	0	3	6	9	12	15	15	12	9	6
6	1	x	0	8	8								
		y	0	0	17								
7	3	x	0	5	6	6	3	3	8	8			
		y	0	0	2	3	3	6	6	11			
8	1	x	0	1	3	3	1	0	0	2	2		
		y	0	0	5	8	13	13	9	9	4		
9	1	x	0	11									
		y	0	0									
10	1	x	6	6	11	11	23	20	20	8	8	5	
		y	0	2	2	0	0	3	5	10	15	15	
11	1	x	6	15	15	0	0						
		y	0	0	12	12	8						
12	1	x	0	6	6	2	2						
		y	0	0	16	16	7						
13	1	x	6	11	15	12	12	0	0	2			
		y	2	0	5	10	13	13	11	10			
14	1	x	0	0	3	5	8	11	11				
		y	8	2	0	3	6	6	8				
15	1	x	0	4	6	2	6						
		y	5	0	0	5	10						
16	1	x	11	17	17	15	15	17	17	14	3	0	2	3
		y	0	0	5	5	8	8	10	10	15	12	11	8
17	1	x	0	0	19	19	6	6	16	16	2	2		
		y	13	0	0	13	13	11	11	3	3	13		
18	2	x	0	0	5	12	12							
		y	8	0	0	7	8							
19	1	x	0	0	10									
		y	13	0	0									
20	1	x	0	0	7									
		y	5	0	0									

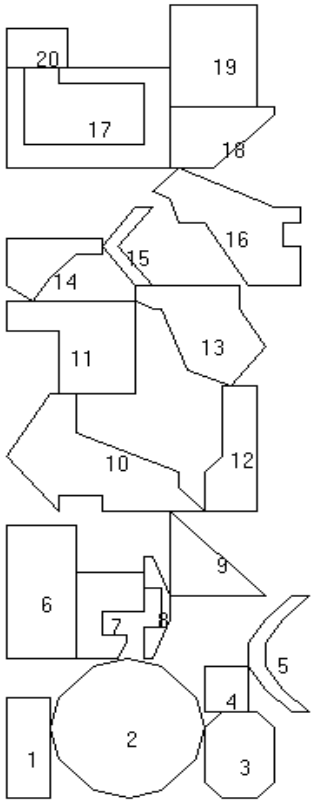


Figure B.15: Data set for test problem Han

name:		**Blaz1**								
size:			28							
object:		width:	15							
no.	quantity									
1	4	x	0	2	4	4	2	0		
		y	1	0	1	4	5	4		
2	4	x	1	4	3	4	4	2	0	0
		y	0	0	2	4	5	5	3	1
3	4	x	1	3	4	4	3	1	0	0
		y	0	0	1	3	4	4	3	1
4	4	x	0	2	4	3	4	2	0	1
		y	0	1	0	2	5	4	5	3
5	4	x	0	5	5	4	3	2	0	
		y	0	0	5	5	3	2	1	
6	4	x	2	4	0					
		y	0	3	3					
7	4	x	0	2	2	0				
		y	0	0	2	2				

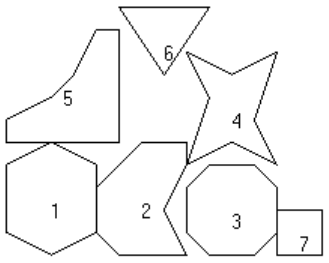


Figure B.16: Data set for test problem Blasz1

name:		**Blaz2**								
size:			20							
object:		width:	15							
no.	quantity									
1	5	x	0	2	4	4	2	0		
		y	1	0	1	4	5	4		
2	5	x	1	4	3	4	4	2	0	0
		y	0	0	2	4	5	5	3	1
3	5	x	1	3	4	4	3	1	0	0
		y	0	0	1	3	4	4	3	1
4	5	x	0	2	4	3	4	2	0	1
		y	0	1	0	2	5	4	5	3

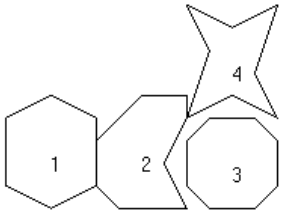


Figure B.17: Data set for test problem Blas2

name:		**SHAPES0**											
size:			43										
object:		width:	40										
no.	quantity												
1	15	x	0	2	2	12	12	14	14	0			
		y	0	0	3	3	0	0	5	5			
2	7	x	0	6	12	6							
		y	6	0	6	12							
3	9	x	0	6	6	7	11	11	8	8	2	2	0
		y	2	2	0	0	4	6	6	3	3	6	6
4	12	x	0	2	2	4	4	6	6	4	4	2	2	0
		y	2	2	0	0	2	2	4	4	6	6	4	4

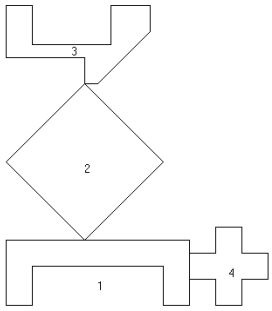


Figure B.18: Data set for test problem SHAPES0

name:		**SHAPES2**							
size:			28						
object:		width:	15						
no.	quantity								
1	4	x	0	2	4	4	2	0	
		y	1	0	1	4	5	4	
2	4	x	1	4	3	4	4	2	0	0
		y	0	0	2	4	5	5	3	1
3	4	x	1	3	4	4	3	1	0	0
		y	0	0	1	3	4	4	3	1
4	4	x	0	2	4	3	4	2	0	1
		y	0	1	0	2	5	4	5	3
5	4	x	0	5	5	4	3	2	0	
		y	0	0	5	5	3	2	1	
6	4	x	2	4	0					
		y	0	3	3					
7	4	x	0	2	2	0				
		y	0	0	2	2				

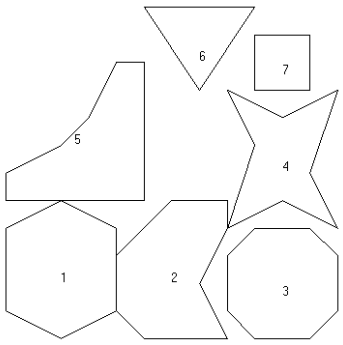


Figure B.19: Data set for test problem SHAPES2

B.1.2.3 Jigsaw Problems

The jigsaw problems Dighe1 and Dighe2 which are presented below are not identical to the problems used by Dighe and Jakiela (1996), however similar in terms of number and shape of the items and the object. Dighe and Jakiela allow clearance between the parts, so that the optimal packing density is less than 100%.

name:		**Dighe1**					
size:		16					
object:		width:	100				
no							
1	x	0	35	0			
	y	0	0	34			
2	x	0	33	18			
	y	0	0	15			
3	x	0	32	32	14		
	y	0	0	41	33		
4	x	35	53	51	48	39	0
	y	0	15	38	62	57	34
5	x	2	17	31	25	0	
	y	15	0	33	34	38	
6	x	0	39	20	0		
	y	0	23	26	66		
7	x	3	28	30	32	22	0
	y	4	0	18	39	36	28
8	x	0	6	24	2		
	y	1	0	8	19		
9	x	0	22	2			
	y	11	0	32			
10	x	0	20	20			
	y	32	0	37			
11	x	1	26	0			
	y	0	9	15			
12	x	0	19	28	27	24	
	y	3	0	5	20	43	
13	x	20	44	0			
	y	0	40	40			
14	x	3	29	36	0		
	y	6	0	2	29		
15	x	0	36	56			
	y	27	0	27			
16	x	0	20	20			
	y	0	5	27			

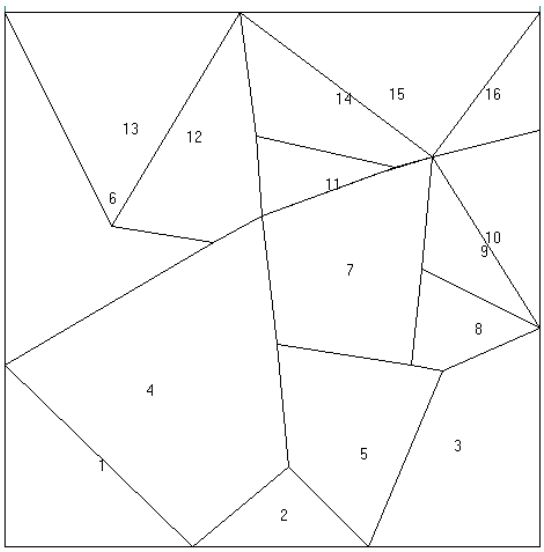


Figure B.20: Data set for test problem Dighe1

name:		**Dighe2**						
size:		10						
object:		width:	100					
no								
1	x	0	33	33	3			
	y	0	0	19	11			
2	x	0	42	37	0			
	y	0	0	30	19			
3	x	5	30	30	0			
	y	0	0	51	30			
4	x	0	3	7	8	0		
	y	0	11	33	38	36		
5	x	0	30	67	56	4		
	y	0	8	19	29	22		
6	x	23	53	53	19	7	0	12
	y	0	21	70	70	42	23	10
7	x	0	52	40	47	3	1	
	y	0	7	20	39	30	5	
8	x	0	8	10	12	0		
	y	0	2	27	64	64		
9	x	0	44	16	2			
	y	0	9	37	37			
10	x	0	28	40				
	y	28	0	28				

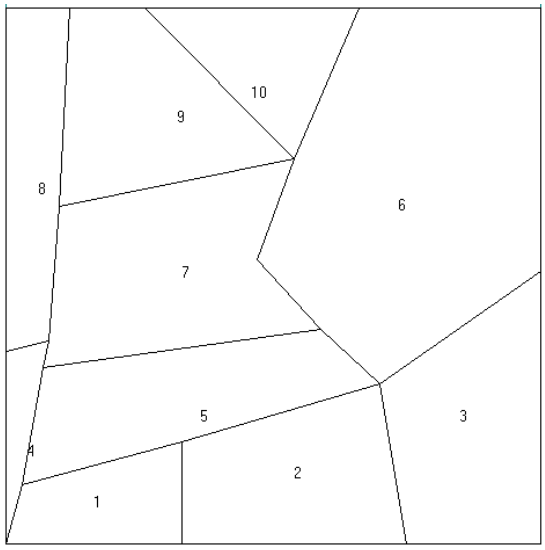


Figure B.21: Data set for test problem Dighe2

B.2 Constructed Test Problems

B.2.1 Overview over Constructed Test Problems

Table B.4: Irregular test problems from literature: artificially created problems

name	size	problem type	shapes	object width
poly1a	15	artificial	polygons	40
poly2a	30	artificial	multiple of poly1a	40
poly3a	45	artificial	multiple of poly1a	40
poly4a	60	artificial	multiple of poly1a	40
poly5a	75	artificial	multiple of poly1a	40
poly2b	30	artificial	polygons	40
poly3b	45	artificial	polygons	40
poly4b	60	artificial	polygons	40
poly5b	75	artificial	polygons	40

B.2.2 Test Problems

name: **poly1a**
size: 15

no								
1	x	0	10	8	4			
	y	0	2	5	4			
2	x	0	7	5				
	y	0	0	7				
3	x	0	3	7	4	1		
	y	1	0	2	5	3		
4	x	2	6	7	0			
	y	3	0	7	10			
5	x	0	10	13	12	5	1	
	y	0	1	3	5	7	5	
6	x	0	5	4	0			
	y	2	0	4	3			
7	x	4	6	0				
	y	0	7	4				
8	x	2	4	2	1	0		
	y	0	7	9	6	4		
9	x	0	6	8	3			
	y	0	0	2	7			
10	x	3	11	11	3	0		
	y	0	0	3	7	5		
11	x	1	8	4	7	0		
	y	0	0	3	6	5		
12	x	2	6	9	7	3	5	0
	y	0	2	1	3	5	7	8
13	x	0	3	9	4			
	y	6	0	0	2			
14	x	0	10	9	4	2		
	y	0	0	4	2	3		
15	x	0	5	7	8	2		
	y	3	0	2	4	5		

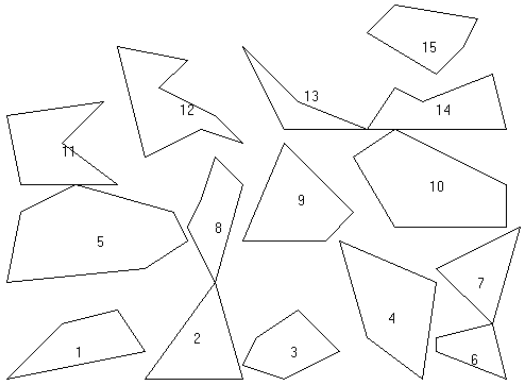


Figure B.22: Data set for test problem poly1a

name: **poly2b**
size: 30 consists of 15 polygons of problem poly1aN and 15 additional ones

no								
16	x	0	3	7	8	8	3	
	y	4	0	2	7	11	8	
17	x	0	4	7	9	5	0	
	y	4	4	0	6	6	7	
18	x	0	6	9	1			
	y	0	0	1	5			
19	x	0	3	7	11	12	1	
	y	1	0	3	3	8	5	
20	x	0	4	9	13	6	6	2
	y	5	0	0	5	4	8	8
21	x	0	4	7	8			
	y	3	0	0	5			
22	x	0	2	9	12	2		
	y	4	0	1	5	8		
23	x	0	3	11	10	2		
	y	8	4	0	7	10		
24	x	0	2	4	4	0		
	y	4	1	0	6	6		
25	x	0	2	3				
	y	7	0	8				
26	x	0	4	5	2			
	y	0	4	7	5			
27	x	0	5	4	2			
	y	0	0	5	7			
28	x	0	5	9	9	5		
	y	2	5	0	3	8		
29	x	0	3	7	9	7	5	3	3
	y	2	0	0	3	6	2	2	3
30	x	0	2	2	4	6	9	4	
	y	6	3	0	1	0	5	8	

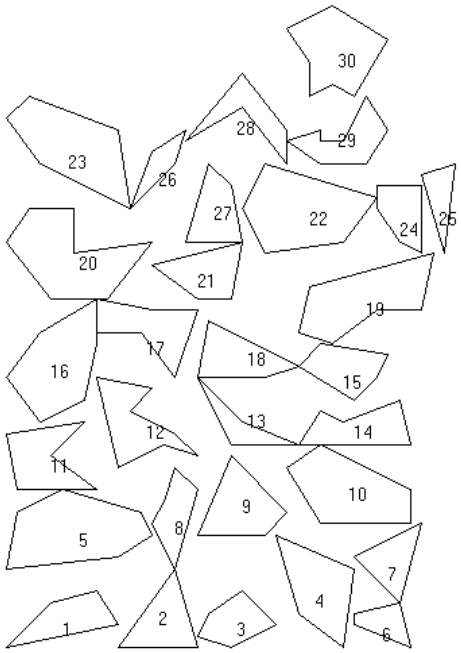


Figure B.23: Data set for test problem poly2b

name: **poly3b**
size: 45 consists of 30 polygons of problem poly2bN and 15 additional ones

no								
31	x	0	6	5	7	3	1	
	y	3	0	5	7	6	8	
32	x	0	3	0				
	y	0	3	4				
33	x	0	13	2				
	y	0	2	4				
34	x	0	6	8	5	1		
	y	0	0	2	6	4		
35	x	0	3	7	5	1		
	y	1	0	0	3	3		
36	x	0	4	6	5	0		
	y	3	2	0	4	6		
37	x	0	3	7	9	7	2	
	y	3	0	1	3	5	5	
38	x	0	2	2	3	3	0	
	y	3	1	0	0	4	4	
39	x	0	3	3	0			
	y	0	1	2	2			
40	x	0	5	5	8	10	6	2
	y	3	2	1	0	2	6	7
41	x	0	7	9	0			
	y	0	2	5	4			
42	x	0	4	5	9	9	4	
	y	4	3	0	2	6	8	
43	x	0	3	2	0			
	y	0	2	7	6			
44	x	1	3	4	0			
	y	0	0	4	3			
45	x	0	2	4	7	4	2	
	y	7	0	3	4	7	8	

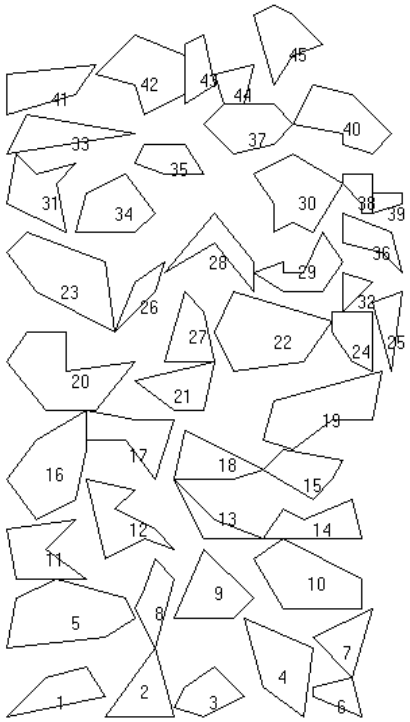


Figure B.24: Data set for test problem poly3b

name: **poly4b**
size: 60 consists of 45 polygons of problem poly3bN and 15 additional ones

no								
46	x	0	6	6	4			
	y	6	0	5	9			
47	x	0	10	11	7	2		
	y	1	0	2	5	6		
48	x	0	2	4	5	5		
	y	3	0	0	3	4		
49	x	2	3	2	0			
	y	0	3	7	4			
50	x	2	3	2	3	0		
	y	0	2	6	13	7		
51	x	0	2	4	3	0		
	y	0	0	5	8	5		
52	x	0	1	6	6	1		
	y	4	0	4	6	8		
53	x	0	2	5	4	2	2	
	y	4	0	3	7	8	5	
54	x	0	2	4	2			
	y	6	0	4	8			
55	x	0	3	6	5	2	2	
	y	1	0	0	5	6	3	
56	x	0	7	11	6	3		
	y	0	1	3	6	7		
57	x	0	7	1	3			
	y	0	0	4	1			
58	x	0	4	7	11	9	5	1
	y	3	0	2	2	5	4	5
59	x	0	3	6	4	1		
	y	5	0	3	6	8		
60	x	0	3	3	2			
	y	2	0	3	4			

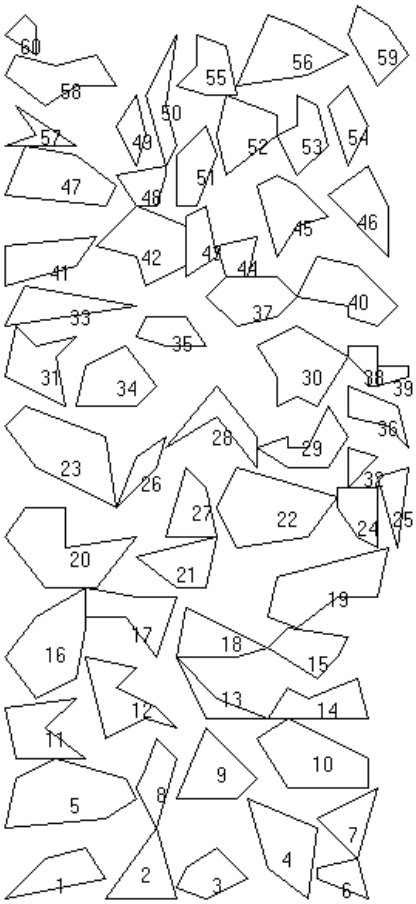


Figure B.25: Data set for test problem poly4b

name: **poly5b**
size: 75 consists of 60 polygons of problem poly4bN and 15 additional ones

no							
61	x	1	8	3	0		
	y	0	2	3	7		
62	x	0	5	3	4	2	0
	y	1	0	2	5	6	3
63	x	0	5	8	2		
	y	3	0	1	7		
64	x	0	10	3			
	y	0	1	2			
65	x	1	4	6	5	0	
	y	2	0	1	4	7	
66	x	0	4	5	2		
	y	2	0	3	6		
67	x	0	2	9	3		
	y	4	0	5	6		
68	x	0	1	9	10	5	
	y	5	0	1	3	3	
69	x	0	1	8	4		
	y	5	0	1	4		
70	x	0	1	5	2		
	y	3	0	1	2		
71	x	0	6	5	2	0	
	y	0	2	5	7	7	
72	x	0	2	4	2		
	y	6	0	4	3		
73	x	0	5	7	4	5	2
	y	2	0	1	4	6	5
74	x	0	1	6	10	2	
	y	3	1	0	3	6	
75	x	0	3	6	5		
	y	1	0	2	5		

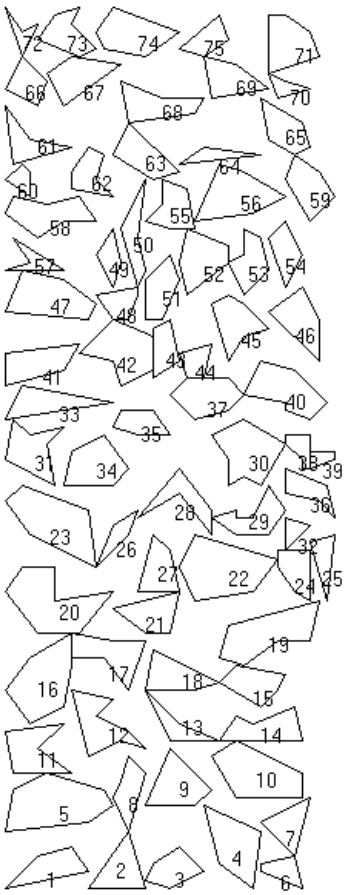


Figure B.26: Data set for test problem poly5b

C. Commercial Nesting Software

Table C.1: Commercial packages for rectangular packing problems

product	company	description	location on the Internet
Cut Planner	Remarkable Software Ltd., USA	cutting optimisation for rectangular parts for multiple sheets	http://www.remarkable.co.nz/optimisers/cutplan.htm
DEC++	Advanced Technology Institute, Romania	packing software for rectangular parts for multiple sheets	http://prodlog.wiwi.uni-halle.de/Sicuphomepage/software.htm
Itemizer	MID-OHIO SYSTEMS	PC-based packing system for rectangular items	http://ourworld.compuserve.com/homepages/midohio/itemizer.htm
Ritmo4	ALMA Manufacturing Software, France	rectangular and linear cutting optimisation software package for woodwork, mirror manufacturing, sheet metal and cutting plastics or composites	http://www.alma.fr/productique/ang/
Shear/ Miser	Fab/Trol Systems, Inc., USA	packing system for rectangular parts for multiple sheets with remnant tracking	http://www.fabtrol.com/

Table C.2: Commercial packages for 3D packing problems

product	company	description	location on the Internet
PowerPack	Remarkable Software Ltd., New Zealand	flexible 3D packing tool for packing boxes on containers, pallets etc.	http://www.remarkable.co.nz/optimisers/powerpak.htm
CubeIQ	Remarkable Software Ltd., New Zealand	loading optimiser for rectangular and irregular shaped containers, i.e. air craft	http://www.remarkable.co.nz/optimisers/qubeiq.htm
CARGO MANAGER	Gower Optimal Algorithms Ltd, UK	3D system for packing boxes into containers, pallets etc.	http://www.goweralg.co.uk
PALLET MANAGER	Gower Optimal Algorithms Ltd, UK	powerful package to pack pallets with rectangular boxes and cylindrical boxes	http://www.goweralg.co.uk

Table C.3: Commercial packages for rectangular and irregular packing problems

product	company	description	location on the Internet
UG/Sheet Metal Nesting	Unigraphics Solutions Inc.	fully automatic and interactive nesting for the sheet metal industry	http://www.ugsolutions.com/products/unigraphics/
PowerNest	ALMA Manufacturing Software, France	fully automatic nesting of system mainly for the mechanical industry; includes a specific function for the nesting of rectangular parts; available as library of nesting algorithms	http://www.alma.fr/productique/ang/
Nestix2	Nestix Oy, Finland	automatic nesting module; part of 3D CAD package for the shipbuilding industry	http://www.nestix.fi
HUMANCAD Optitex	Humantec Industrie-systeme, Germany	template construction and nesting for the apparel, the upholstery and the automotive industry; interactive and automatic nesting is supported	http://www.humantec.de/humantec/software.htm
Materials Manager 2.7	MID-OHIO SYSTEMS, USA	general purpose true shape nesting and optimisation system; runs inside AutoCAD; fully automatic and interactive nesting	http://ourworld.compuserve.com/homepages/midohio/matmgr.htm
OptiNest	Remarkable Software Ltd., New Zealand	general purpose true shape nesting program; automatic and interactive mode	http://www.remarkable.co.nz
SigmaNEST	SigmaTEK Corporation, USA	true shape automatic nesting system; automatic and interactive mode; for the metal and wood cutting industry	http://www.sigmanest.com
SS-NEST	Striker Systems, USA	automatic nesting software for sheet metal	http://www.striker-systems.com/ssnest/
SS-STRIP	Striker Systems, USA	automatic strip layout; also interactive	http://www.striker-systems.com/ssstrip/
Accu FABTM	DynaSys, Inc., USA	multi-purpose fully automatic nesting system; also interactive	http://www.accufab.com/solve1.htm
Nestlib	Geometric Software Solutions Co. Ltd., India	fully automatic true shape automatic nesting; also available as DLL containing the nesting algorithms	http://www.gsslco.com/
SMP/IS	Merry Mechanization, Inc., USA	fully automatic and interactive true shape nesting system sheet metal fabrication; with modules for just-in-time requirements, special cutting processes etc.	http://www.merrymech.com
PINS XL 2000	Generative N/C Technology, USA	automatic nesting system	http://www.gen-c.com/
Helix Manu-facturing	MICROCADAM, Inc., USA	automatic true shape nesting system for the sheet metal industry	http://www.microcadam.com/marketing/pages/helmanuf.html
AutoNest	Virtek, Canada	fully automatic dynamic nesting software specifically designed to optimise yields on leather hides	http://www.virtek.ca/nesting/ nesting.htm
Friendly 2D	DNT, Italy	automatic nesting software for the leather cutting industry; also interactive mode	http://www.fastnet.it/market/dnt/2dengl.htm
Lasernest	Humantec Industriesystem Germany	fully automatic nesting software for the leather industry	http://www.humantec.de/humantec/lasernes.htm

D. Software Packages for Computational Geometry

Table D.1: Overview over software packages for computational geometry

product	description	organisation	location on the Internet
CGAL (Computational Geometry Algorithms Library)	library of geometric and non-geometric data structures and algorithms	joint project between European universities and companies	http://www.mpi-sb.mpg.de/LEDA
LEDA (Library of Efficient Data types and Algorithms)	library of geometric and non-geometric data structures and algorithms	Max-Planck Institut für Informatik, Saarbrücken, Germany	http://www.cs.uu.nl/CGAL/
PlaGeo SpaGeo	library for planar geometry library for spatial geometry	Utrecht University	http://www.cs.uu.nl/people/geert/DOC/
Computational Geometry Workbench	library of geometric and non-geometric data structures and algorithms; also geometrical programming environment with tools for creating, editing and manipulating geometric objects; animating, demonstrating and implementing complex geometric algorithms	Carleton University, Canada University of Passau, Germany	ftp alfred.ccs.carleton.ca; directory: pub/workbench http://fusion.scs.carleton.ca/research/algorithms.shtml
XYZ GeoBench	library of computational geometry algorithms and user interface; focus on implementation of fundamental algorithms	Institut für Theoretische Informatik, ETH, Zürich, Switzerland	http://wwwjn.inf.ethz.ch/geobench/XYZGeoBench.html ftp.inf.ethz.ch in directory /pub/software/xyz
GeomNet	system for performing distributed geometric computing over the Internet	John Hopkins University, Baltimore Brown University, Providence	http://www.cs.brown.edu/cgc/
Geomview	interactive 3D viewer for geometric objects provided by external programs	Geometry Center, University of Minnesota	http://www.geom.umn.edu/

Table D.2: Collections of geometric software and web pages on computational geometry

name	description	location on the Internet
Directory of Computational Geometry Software	collection of computational geometry programs and packages	http://www.geom.umn.edu/software/cglist/
Geometry in Action	real world applications of computational geometry	http://www.ics.uci.edu/~eppstein/geom.html
The Geometry Junkyard	links, lecture notes, research excerpts, papers, abstracts, programs, problems, and other material related to discrete and computational geometry	http://www.ics.uci.edu/~eppstein/junkyard/
Computational Geometry Pages	directory of computational geometry resources on and off the internet	http://compgeom.cs.uiuc.edu/~jeffe/compgeom/
CG Tribune	newsletter on different matters of concern to the computational geometry community including events and announcements	http://www-sop.inria.fr/prisme/personnel/bronnimann/cgt/index.html
Computational Geometry Resources	collection of links to bibliographies, software, newsletters and centres of computational geometry	http://fusion.scs.carleton.ca/~csgs/resources/cg.html

E. Cutting and Packing Resources on the Internet

E.1 Benchmark Problems

Table E.1: Collections of benchmark problems on the Internet

name	description	location on the Internet
OR-Library	OR problems; some for cutting and packing	http://mscmga.ms.ic.ac.uk/library/
SICUP- Library	cutting and packing related problems	http://prodlog.wiwi.uni-halle.de/sicup/

E.2 Comprehensive Databases and Bibliographies for Cutting and Packing

Table E.2: Databases and bibliographies on cutting and packing on the Internet

name	description	location on the Internet
SICUP- Library	cutting and packing related problems; books, collections of papers and bibliographies, articles, 'grey' literature (working papers, dissertations, theses, etc.), papers presented at conferences	http://prodlog.wiwi.uni-halle.de/sicup/

F. Results for 2D Rectangular Strip Packing

Table F.1: Relative difference of best solution to optimum height [%] for simple packing algorithms with random and pre-ordered input sequences

	BL Z	DH	DW	DRA	DRP	BLLT Z	DH	DW	DRA	DRP
T1	15	26	27	20	19	17	29	26	19	19
T2	19	32	27	22	22	17	40	26	21	21
T3	22	24	29	30	32	20	35	27	27	27
T4	23	21	23	25	27	21	29	21	26	28
T5	28	19	18	29	30	22	24	17	33	27
T6	34	14	19	26	27	26	17	18	28	29
T7	36	11	19	27	27	29	10	16	24	24

Table F.2: Relative difference of best solution to optimum height [%] for simple packing algorithms with random and pre-ordered input sequences

	BLF Z	DH	DW	DRA	DRP	BLD Z	DH	DW	DRA	DRP
T1	12	18	17	12	12	18	23	42	25	25
T2	14	15	14	11	11	21	29	29	26	26
T3	12	13	11	10	8	22	20	28	27	27
T4	12	8	8	7	7	25	16	25	28	29
T5	10	6	6	6	6	26	13	23	28	29
T6	10	4	4	4	4	31	11	23	27	29
T7	8	2	4	3	3	33	6	19	26	26

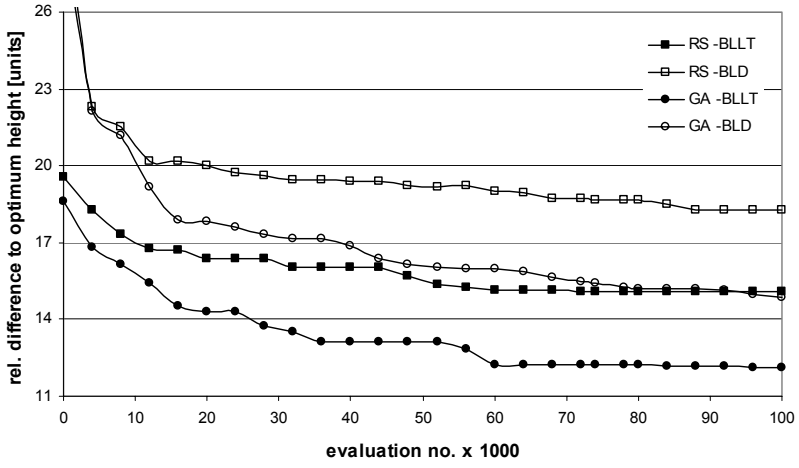


Figure F.1: Comparison between GA and RS with BLLT and BLD decoder for T4a

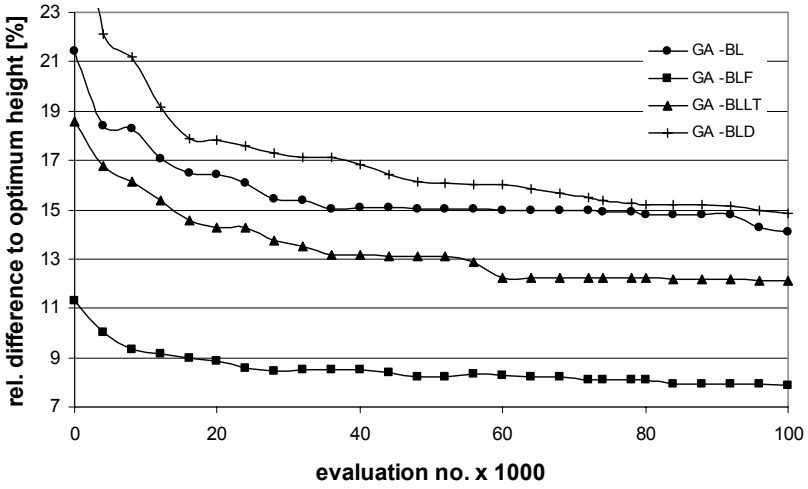


Figure F.2: Comparison between hybrid genetic algorithms for T4a

Table F.3: Relative difference to optimal height for evolutionary methods [%]

	GA				**GA seeded**				**NE**			
	BL	**BLLT**	**BLF**	**BLD**	**BL**	**BLLT**	**BLF**	**BLD**	**BL**	**BLLT**	**BLF**	**BLD**
T1	8	8	6	8	8	7	5	8	11	10	7	10
T2	10	9	7	10	9	9	6	9	12	11	8	12
T3	10	10	7	11	10	9	6	10	12	12	8	13
T4	14	13	8	15	12	12	6	11	15	14	8	15
T5	17	15	8	18	12	14	5	12	19	15	9	19
T6	19	16	7	21	13	14	4	10	21	17	8	20
T7	26	21	6	26	11	9	2	7	27	21	6	26

Table F.4: Relative difference to optimal height for meta-heuristics and random search [%]

	SA				**SO**			**RS**			
	BL	**BLLT**	**BLF**	**BLD**	**BL**	**BLLT**	**BLF**	**BL**	**BLLT**	**BLF**	**BLD**
T1	8	7	6	7	8	7	6	11	10	8	11
T2	8	8	6	8	10	9	7	13	12	9	15
T3	9	8	6	8	11	9	7	15	13	9	16
T4	9	8	5	9	14	11	7	19	15	9	19
T5	10	8	5	12	17	14	7	19	18	9	24
T6	11	8	4	13	20	15	7	25	19	8	26
T7	17	14	4	23	27	21	6	31	24	7	30

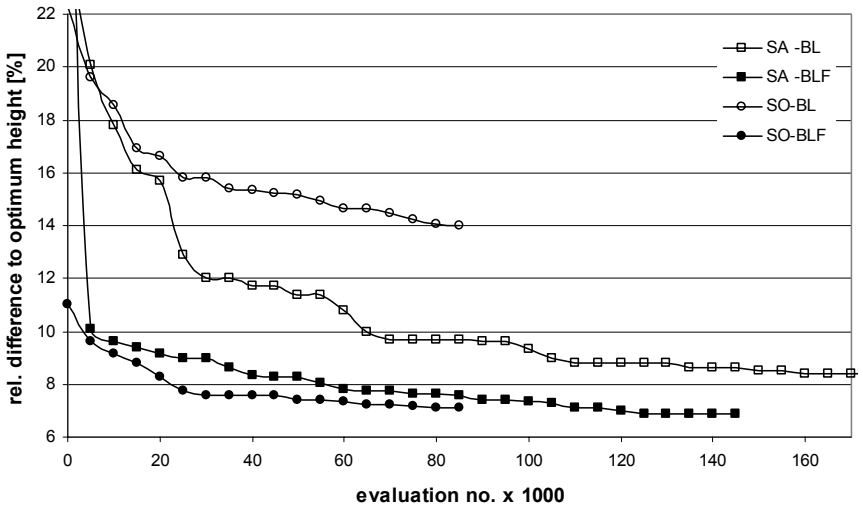


Figure F.3: Comparison between SA and SO for T4a

Table F.5: Relative difference to optimal height [%] for hill-climbing and heuristics with random and height-sorted input

	HC				**HEU random**				**HEU height-sorted**			
	BL	**BLLT**	**BLF**	**BLD**	**BL**	**BLLT**	**BLF**	**BLD**	**BL**	**BLLT**	**BLF**	**BLD**
T1	28	25	14	26	36	34	21	18	27	27	14	23
T2	25	24	12	25	27	28	14	21	37	37	15	29
T3	25	34	13	28	25	28	11	22	30	30	13	20
T4	26	28	10	26	23	30	8	25	25	25	8	16
T5	29	32	9	30	29	29	6	26	24	24	6	13
T6	33	34	8	32	28	23	4	31	28	28	4	11
T7	35	41	7	32	25	44	10	33	23	21	4	6

Table F.6: Average number of iterations per run for HC, SA and SO

	HC				**SA**			**SO**		
	BL	**BLLT**	**BLF**	**BLD**	**BL**	**BLLT**	**BLF**	**BL**	**BLLT**	**BLF**
T1	2970	7164	3045	5724	62931	61895	54190	75578	76270	65055
T2	5230	10237	4925	8638	91032	90785	78322	89930	89711	85416
T3	5094	28643	5997	9344	101032	101785	84693	90502	90239	88271
T4	18257	31603	10865	18554	181655	180196	142022	89893	89893	89893
T5	32477	34457	19948	31052	283849	282102	210031	90551	90551	90551
T6	44119	43210	24839	52796	358127	357850	243604	90399	90323	90399
T7	87977	48343	63345		500000	500000	500000	90339	90294	90378

Table F.7: Average elapsed time per run [s]

	GA				NE				RS			
	BL	**BLLT**	**BLF**	**BLD**	**BL**	**BLLT**	**BLF**	**BLD**	**BL**	**BLLT**	**BLF**	**BLD**
T1	20	22	37	27	12	14	28	19	42	44	79	58
T2	25	27	63	39	16	19	46	28	71	77	147	99
T3	30	37	78	48	20	23	57	34	91	102	186	125
T4	51	69	175	91	37	54	134	67	215	254	469	292
T5	84	154	332	161	62	111	251	122	437	557	923	582
T6	127	243	519	245	95	182	386	188	732	959	1499	964
T7	396	1193	1723	818	312	929	1315	629	2877	4415	5496	3705

Table F.8: Average elapsed time per run [s]

	SA				SO			HC			
	BL	**BLLT**	**BLF**	**BLD**	**BL**	**BLLT**	**BLF**	**BL**	**BLLT**	**BLF**	**BLD**
T1	40	39	62	55	33	23	64	3	3	2	4
T2	70	71	129	112	42	50	110	2	10	10	5
T3	90	118	252	165	49	61	133	4	7	19	20
T4	283	422	849	520	91	129	302	19	43	89	35
T5	484	927	1876	939	148	259	559	26	62	174	152
T6	729	1435	2942	1463	223	441	889	220	142	313	165
T7	2374	7147	10400	4827	714	2194	2980	948	976	3394	1588

Table F.9: Relative difference to the outcome of the BL-algorithm for all hybrid genetic algorithms [%]

		GA			
	N	**BL**	**BLLT**	**BLF**	**BLD**
T2	25	-13.3	-13.9	-15.6	-13.4
T3	29	-11.4	-11.9	-13.9	-11.1
T4	49	-7.7	-8.8	-12.7	-6.9
T5	73	-9.2	-11.2	-16.4	-8.4

Table F.10: Relative difference to the outcome of the BL-algorithm for hybrid genetic algorithms [%]; by Kroeger et al. (1991)

N	**GA**	**N**	**GA**
25	-6.7	45	-2.2
25	-6.2	45	-1.8
25	-3.5	45	-0.3
35	-2.1	65	-1.3
35	-1.1		
35	-2.9		

Table F.11: Rank sums for variation of population size using four problem categories; BL-algorithm

method	pop. size	T2	T3	T4	T5
1	26	25	25	23	22
2	50	19	19	18	18
3	100	13	14	14	16
4	150	12	10	11	12
5	200	6	7	9	7
P-value		<0.01	<0.01	<0.01	<0.01

Table F.12: Rank sums for variation of population size using four problem categories; BLF-algorithm

method	pop. size	T2	T3	T4	T5
1	26	20	20	25	20
2	50	25	25	18	25
3	100	14	13	17	13
4	150	9	10	9	11
5	200	7	7	6	6
P-value		< 0.01	< 0.01	0.05 .. 0.025	0.05 .. 0.025

Table F.13: Relative difference to optimum height for GAs for two population sizes

decoder pop. size	BL 100	BL 200	BLF 100	BLF 200
T2	10.1	8.2	7.1	6.9
T3	10.3	10.0	7.2	6.5
T4	13.9	13.5	7.7	7.3
T5	17.4	16.5	8.1	7.2

Table F.14: Rank sums for variation of cross-over rate using four problem categories; BL-algorithm

method	rate [%]	T2	T3	T4	T5
1	0	26	25	26	25
2	20	17	19	19	15
3	40	14	20	21	16
4	60	13	19	10	15
5	80	19	**11**	11	18
6	100	16	**11**	18	16
P-value		0.5 .. 0.25	0.25 .. 0.1	0.1.. 0.05	0.75 .. 0.5

Table F.15: Rank sums for variation of cross-over rate using four problem categories; BLF-algorithm

method	rate [%]	T2	T3	T4	T5
1	0	28	15	29	20
2	20	20	20	21	15
3	40	16	10	15	25
4	60	15	25	14	30
5	80	19	30	**11**	10
6	100	**7**	**5**	15	**5**
P-value		0.025 .. 0.01	< 0.01	0.05 .. 0.025	< 0.01

Table F.16: Rank sums for variation of mutation rate using four problem categories; BL-algorithm

method	rate [%]	T2	T3	T4	T5
1	0	30	30	30	26
2	1	12	11	7	5
3	3	6	6	9	10
4	5	13	13	15	15
5	10	19	20	22	22
6	20	25	25	22	27
P-value		< 0.01	< 0.01	< 0.01	< 0.01

Table F.17: Rank sums for variation of mutation rate using four problem categories; BLF-algorithm

method	rate [%]	T2	T3	T4	T5
1	0	30	15	30	15
2	1	11	20	19	20
3	3	13	10	13	10
4	5	9	25	12	25
5	10	18	30	21	30
6	20	24	**5**	**10**	**5**
P-value		< 0.01	< 0.01	< 0.01	< 0.01

Table F.18: Rank sums for variation of cross-over type using four problem categories; BL-algorithm

meth.	type	T2	T3	T4	T5
1	PMX1	19	14	15.5	12
2	OX	12	19	15.5	16
3	PBX	16	11	15.5	18
4	PMX2	15	20	15.5	15
5	CX	13	**11**	**13**	**14**
P-value		0.75 .. 0.5	0.25 .. 0.1	0.99 .. 0.975	0.9 .. 0.75

Table F.19: Rank sums for variation of cross-over type using four problem categories; BLF-algorithm

meth.	type	T2	T3	T4	T5
1	PMX1	17	20	15.5	18
2	OX	9	12	15.5	13.5
3	PBX	16	15	13	7
4	PMX2	20	14	15.5	18
5	CX	13	14	15.5	18.5
P-value		0.25 .. 0.1	0.75 .. 0.5	0.99 .. 0.975	0.1 .. 0.05

Table F.20: Rank sums for variation of mutation type using four problem categories; BL-algorithm

method	type	T2	T3	T4	T5
1	OBM	7.5	8	7.5	8
2	PBM	7.5	7	7.5	**7**
P-value		> 0.99	0.75 .. 0.5	> 0.99	0.75 .. 0.5

Table F.21: Rank sums for variation of mutation type using four problem categories; BLF-algorithm

method	type	T2	T3	T4	T5
1	OBM	7.5	8	7.5	8
2	PBM	7.5	7	7.5	**7**
P-value		0.75 .. 0.5	0.25 .. 0.1	0.75 .. 0.5	0.75 .. 0.5

Table F.22: Rank sums for variation of selection type using four problem categories; BL-algorithm

method	type	T2	T3	T4	T5
1	prop	10	6	7	7
2	rank	5	9	8	8
P-value		0.025.. 0.010	0.25 .. 0.1	0.75 .. 0.5	0.75 .. 0.5

Table F.23: Rank sums for variation of selection type using four problem categories; BLF-algorithm

method	type	T2	T3	T4	T5
1	prop	10	7	10	9
2	rank	**5**	8	**5**	**6**
P-value		0.025.. 0.010	0.75 .. 0.5	0.025.. 0.010	0.25 .. 0.1

Table F.24: Relative difference of best solution to optimum height [%] for simple packing algorithms with random and pre-ordered input sequences; test problems of N-series

	BL					BLLT				
	Z	**DH**	**DW**	**DRA**	**DRP**	**Z**	**DH**	**DW**	**DRA**	**DRP**
N1	16	31	35	21	21	17	32	37	19	20
N2	17	36	31	23	25	15	33	33	25	24
N3	21	31	25	25	30	19	35	24	26	26
N4	24	20	24	27	27	20	29	22	27	28
N5	30	18	19	29	29	24	24	18	32	30
N6	33	13	19	29	29	25	14	18	27	27
N7	39	13	18	26	26	28	13	16	25	24

Table F.25: Relative difference of best solution to optimum height [%] for simple packing algorithms with random and pre-ordered input sequences; test problems of N-series

	BLF					BLD				
	Z	**DH**	**DW**	**DRA**	**DRP**	**Z**	**DH**	**DW**	**DRA**	**DRP**
N1	12	18	15	12	12	19	33	38	31	31
N2	11	13	18	11	11	19	25	33	27	27
N3	11	14	11	11	11	24	24	28	25	27
N4	11	9	7	8	8	25	16	25	27	25
N5	11	6	6	6	6	26	14	21	30	30
N6	10	4	5	4	4	31	10	23	25	25
N7	8	3	2	2	2	35	7	18	27	27

Table F.26: Relative difference to optimal height [%] for genetic algorithms and random search; test problems of N-series

	GA				RS			
	BL	**BLLT**	**BLF**	**BLD**	**BL**	**BLLT**	**BLF**	**BLD**
N1	7	7	6	8	10	9	7	12
N2	8	8	6	9	11	10	8	12
N3	9	9	6	11	15	13	7	16
N4	13	12	7	15	18	15	8	19
N5	17	15	7	19	22	18	9	23
N6	19	16	7	20	24	20	8	25
N7	27	20	5	26	32	24	7	32

Table F.27: Difference to GA outcome with respect to the relative difference to the optimal height [%]; test problems of N-series

	RS				**height-sorted + routine**			
	BL	**BLLT**	**BLF**	**BLD**	**BL**	**BLLT**	**BLF**	**BLD**
N1	2.7	2.5	1.6	3.9	24.0	25.0	12.5	25.0
N2	2.4	2.2	1.8	3.0	27.7	24.9	7.1	16.0
N3	5.2	4.0	1.6	4.9	21.7	25.4	8.2	12.5
N4	4.7	3.1	1.5	4.7	6.8	17.3	1.6	0.9
N5	4.7	3.1	1.4	4.2	0.3	9.6	-0.7	-5.3
N6	5.6	4.5	1.0	5.3	-5.9	-1.3	-2.6	-9.7
N7	5.4	3.6	1.8	5.4	-13.8	-7.3	-2.7	-19.2

G. Results for 2D Irregular Strip Packing

Table G.1: Best height [units] for simple packing algorithms with random and pre-ordered input sequences

	BLi					**BLLTi**				
	Z	**DH**	**DW**	**DA**	**DP**	**Z**	**DH**	**DW**	**DA**	**DP**
poly1a	19.0	19.2	19.2	20.0	18.8	20.0	20.0	19.0	19.0	19.1
poly2a	36.0	34.1	35.2	35.2	35.6	35.8	34.7	34.1	35.1	36.4
poly3a	54.6	52.1	51.9	52.4	54.0	54.3	52.4	51.1	53.3	54.8
poly4a	71.8	68.0	69.4	70.9	72.7	73.1	68.9	70.2	72.4	71.4
poly5a	91.7	85.8	88.9	89.8	90.4	91.1	86.6	87.9	90.0	89.0

Table G.2: Best height [units] for simple packing algorithms with random and pre-ordered input sequences

	BLFi					**BLDi**				
	Z	**DH**	**DW**	**DA**	**DP**	**Z**	**DH**	**DW**	**DA**	**DP**
poly1a	22.0	23.2	21.7	20.0	21.0	19.8	18.0	19.2	19.2	20.0
poly2a	40.3	43.2	41.4	40.0	39.3	36.8	32.7	34.9	36.4	36.1
poly3a	60.3	59.0	58.1	57.3	60.3	54.7	51.1	52.5	55.1	54.4
poly4a	76.7	76.7	75.6	77.1	76.5	74.1	67.7	70.8	73.3	74.6
poly5a	99.3	100.3	95.7	96.3	95.3	92.8	84.3	87.5	91.2	92.2

Table G.3: Best height [units] for simple packing algorithms with random and pre-ordered input sequences

	BLDPi					**BLPi**			
	Z	**DH**	**DW**	**DA**	**DP**	**Z**	**DH**	**DA**	**DP**
poly1a	17.5	18.0	16.8	17.9	17.3	22.3	20.9	21.2	21.9
poly2a	35.4	33.4	33.5	35.6	34.3	42.6	37.5	38.3	40.7
poly3a	51.2	48.8	50.4	52.6	53.3	61.7	56.4	60.6	60.2
poly4a	71.3	66.2	66.1	70.9	70.2	80.5	74.0	81.4	78.8
poly5a	88.9	82.1	84.6	87.4	87.0	105.5	96.4	101.0	104.7

Table G.4: Best height [units] for simple packing algorithms with random and pre-ordered input sequences

	BLi					**BLLTi**				
	Z	**DH**	**DW**	**DA**	**DP**	**Z**	**DH**	**DW**	**DA**	**DP**
poly2b	42.2	41.7	41.9	42.3	42.9	43.4	42.7	42.7	42.7	43.3
poly3b	55.9	55.0	57.3	55.8	55.7	55.7	54.3	57.7	55.7	57.8
poly4b	72.8	68.8	73.1	72.3	72.5	72.2	70.3	71.3	72.8	74.4
poly5b	88.4	81.9	88.7	87.8	87.5	89.5	84.3	88.3	88.1	88.3

Table G.5: Best height [units] for simple packing algorithms with random and pre-ordered input sequences

	BLFi					**BLDi**				
	Z	**DH**	**DW**	**DA**	**DP**	**Z**	**DH**	**DW**	**DA**	**DP**
poly2b	49.0	46.6	46.6	46.0	48.7	43.9	40.2	42.7	41.7	44.1
poly3b	61.4	60.5	60.8	62.0	61.4	58.4	55.4	56.7	57.7	58.0
poly4b	79.0	79.5	76.8	78.4	77.0	73.8	68.9	74.0	73.1	74.7
poly5b	90.0	89.0	86.0	93.3	98.0	88.8	83.9	90.3	89.2	88.1

Table G.6: Best height [units] for simple packing algorithms with random and pre-ordered input sequences

	BLDPi					**BLPi**			
	Z	**DH**	**DW**	**DA**	**DP**	**Z**	**DH**	**DA**	**DP**
poly2b	39.2	36.1	37.9	38.8	39.3	45.8	41.8	44.6	46.2
poly3b	50.6	48.9	49.1	51.9	50.3	61.6	59.2	59.5	61.6
poly4b	66.4	62.1	63.0	65.5	66.3	79.7	74.5	81.6	83.5
poly5b	80.1	74.1	76.1	80.2	80.9	105.1	92.0	103.8	102.6

Table G.7: Average elapsed time for simple packing algorithms per 1000 runs [s]

	BLi	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**	**BLPi**
poly1a	32	34	36	34	59	8
poly2a	74	75	82	77	132	20
poly3a	120	124	136	126	205	32
poly4a	167	166	185	162	290	50
poly5a	206	209	248	210	357	65

Table G.8: Relative difference in height to test with a rotation interval of 90° using polygon in minimum enclosing rectangle for orientation at 0° [%]

	BLi		**BLLTi**		**BLFi**		**BLDi**	
rotation interval	**45°**	**22.5°**	**45°**	**22.5°**	**45°**	**22.5°**	**45°**	**22.5°**
poly1a	5.9	4.5	-1.4	-0.7	2.5	-0.5	1.4	5.7
poly2a	6.5	8.3	6.3	12.5	14.7	16.6	6.4	13.0
poly3a	8.1	8.9	10.0	12.3	14.7	17.7	10.1	12.9
poly4a	7.3	10.4	7.4	10.7	18.8	22.4	7.3	11.3
poly5a	5.8	9.1	10.2	10.4	4.9	15.8	7.9	11.6

Table G.9: Relative difference in height to test with a rotation interval of 90° using polygon in minimum enclosing rectangle for orientation at 0° [%]

	BLi		**BLLTi**		**BLFi**		**BLDi**	
rotation interval	**45°**	**22.5°**	**45°**	**22.5°**	**45°**	**22.5°**	**45°**	**22.5°**
poly2b	3.4	-1.9	0.5	1.4	0.0	0.5	1.5	1.9
poly3b	3.5	2.9	2.2	5.3	9.0	11.9	0.4	0.1
poly4b	2.5	1.6	1.8	5.0	9.0	9.8	4.5	4.4
poly5b	0.7	1.1	-2.1	1.4	12.8	9.3	-0.1	3.3

Table G.10: Relative difference in height to test with a rotation interval of 90° using original polygon for orientation at 0° [%]

	BLi		**BLLTi**		**BLFi**		**BLDi**	
rotation interval	**45°**	**22.5°**	**45°**	**22.5°**	**45°**	**22.5°**	**45°**	**22.5°**
poly1a	5.0	2.4	9.8	7.3	-0.8	-2.0	1.9	4.5
poly2a	-0.1	-1.4	1.8	3.1	0.9	0.0	-0.6	2.2
poly3a	-1.0	2.3	4.2	3.4	-0.5	2.6	2.6	0.7
poly4a	2.9	3.1	3.0	2.6	0.1	2.2	-3.0	1.4
poly5a	-1.2	-10.0	-0.6	0.6	-0.9	-0.7	-0.7	-0.1

Table G.11: Relative difference in height to test with a rotation interval of 90° using original polygon for orientation at 0° [%]

	BLi		**BLLTi**		**BLFi**		**BLDi**	
rotation interval	**45°**	**22.5°**	**45°**	**22.5°**	**45°**	**22.5°**	**45°**	**22.5°**
poly2b	3.1	1.3	-3.2	-0.3	3.3	4.3	-0.3	-3.8
poly3b	-0.5	0.8	-0.9	3.2	-0.5	-0.5	1.4	-0.9
poly4b	-4.5	0.0	1.0	1.5	1.5	-1.1	0.6	3.1
poly5b	17.9	14.0	0.4	2.2	0.0	0.5	0.8	-0.4

Table G.12: Packing height for GA and NE [units]

	GA					NE				
	BLi	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**	**BLi**	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**
poly1a	15.7	15.8	16.5	15.8	15.5	15.8	16.0	17.1	15.9	16.1
poly2a	32.5	32.8	33.8	32.4	32.3	32.5	31.6	32.4	32.8	31.4
poly3a	49.5	49.5	51.1	49.5	49.2	49.6	49.4	48.8	48.5	48.3
poly4a	66.7	66.6	68.4	67.0	67.1	65.6	64.8	67.6	65.8	67.4
poly5a	83.6	83.3	86.4	84.1	85.2	81.3	82.1	83.7	83.4	85.0

Table G.13: Packing height for SA and RS [units]

	SA					RS				
	BLi	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**	**BLi**	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**
poly1a	16.1	16.8	15.9	16.4	17.0	16.8	16.4	17.1	16.5	16.9
poly2a	32.4	33.2	33.7	32.8	33.5	33.1	33.1	34.0	33.6	33.9
poly3a	49.9	49.8	51.3	49.4	49.9	50.6	50.6	51.5	51.0	50.8
poly4a	66.6	65.7	67.5	66.5	67.4	67.6	67.9	69.2	68.4	68.3
poly5a	84.1	81.4	85.3	83.6	85.6	84.6	84.9	87.8	86.3	86.3

Table G.14: Packing height for HC and simplex method [units]

	HC					DHS	
	BLi	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**	**with rotation**	**without rotation**
poly1a	19.4	19.7	19.6	19.4	20.5	16.9	19.0
poly2a	37.0	36.1	36.5	35.9	36.1	34.0	36.7
poly3a	53.2	52.9	53.8	54.3	54.6	49.8	53.3
poly4a	69.9	72.1	71.6	70.5	70.7	68.0	74.4
poly5a	88.8	87.9	88.4	89.1	90.1	86.3	90.3

Table G.15: Packing height [units] for GA and NE

	GA					NE				
	BLi	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**	**BLi**	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**
poly2b	35.5	35.4	36.4	36.0	34.9	36.1	35.4	35.7	34.9	36.2
poly3b	48.0	48.8	49.9	48.6	47.8	47.6	48.9	48.3	47.9	48.2
poly4b	62.3	62.2	63.7	63.2	62.6	61.2	61.2	61.7	62.0	61.9
poly5b	73.9	73.6	75.8	74.7	74.7	74.4	72.4	73.5	73.2	75.6

Table G.16: Packing height for SA and RS [units]

	SA					RS				
	BLi	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**	**BLi**	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**
poly2b	36.2	35.6	37.0	35.5	36.9	36.6	36.5	37.5	36.6	36.8
poly3b	48.6	47.4	49.6	47.1	50.1	49.3	49.2	50.2	49.5	49.9
poly4b	61.5	61.4	62.9	61.3	63.0	63.1	62.2	63.9	62.7	63.7
poly5b	72.9	73.9	75.0	72.8	76.0	75.8	74.5	76.7	75.4	76.9

Table G.17: Packing height for HC and downhill simplex method [units]

	HC					DHS	
	BLi	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**	**with rotation**	**without rotation**
poly2b	39.4	40.0	40.5	39.4	40.2	35.4	38.5
poly3b	52.6	51.5	51.2	50.9	52.9	49.9	53.2
poly4b	66.1	65.6	65.6	64.7	69.0	63.4	68.8
poly5b	78.3	77.9	78.1	80.3	80.3	80.7	81.1

Table G.18: Average elapsed time of GA and NE per run [min]

	GA					NE				
	BLi	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**	**BLi**	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**
poly1a	23	23	24	22	70	16	16	18	16	25
poly2a	49	50	53	48	102	35	36	39	36	53
poly3a	75	79	84	79	142	54	55	64	56	83
poly4a	117	118	129	118	198	73	73	84	65	121
poly5a	148	150	164	149	244	85	88	107	84	148

Table G.19: Average elapsed time of SA and HC per run [min]

	SA					HC				
	BLi	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**	**BLi**	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**
poly1a	13	11	15	13	17	2	2	2	2	6
poly2a	62	52	60	69	79	9	11	14	13	30
poly3a	156	136	138	160	198	21	25	26	27	51
poly4a	242	301	279	277	364	59	26	57	55	78
poly5a	459	618	448	432	522	54	63	108	67	91

Table G.20: Average elapsed time of RS per run [min]

	RS				
	BLi	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**
poly1a	27	28	30	28	49
poly2a	62	63	68	64	110
poly3a	100	103	113	105	171
poly4a	139	138	154	135	242
poly5a	172	174	207	175	298

Table G.21: Average number of iterations per run for HC and SA

	SA					HC				
	BLi	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**	**BLi**	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**
poly1a	19499	16092	19208	19580	16022	3760	3660	3540	3880	3530
poly2a	39782	33599	32520	39964	29251	6980	8240	10360	10460	9830
poly3a	56679	55783	47522	62759	51735	10880	13020	12340	13900	7730
poly4a	79529	85972	71316	76722	79719	21400	19460	19260	20660	17560
poly5a	125570	92889	89654	93741	96567	16380	18720	28250	20120	20120

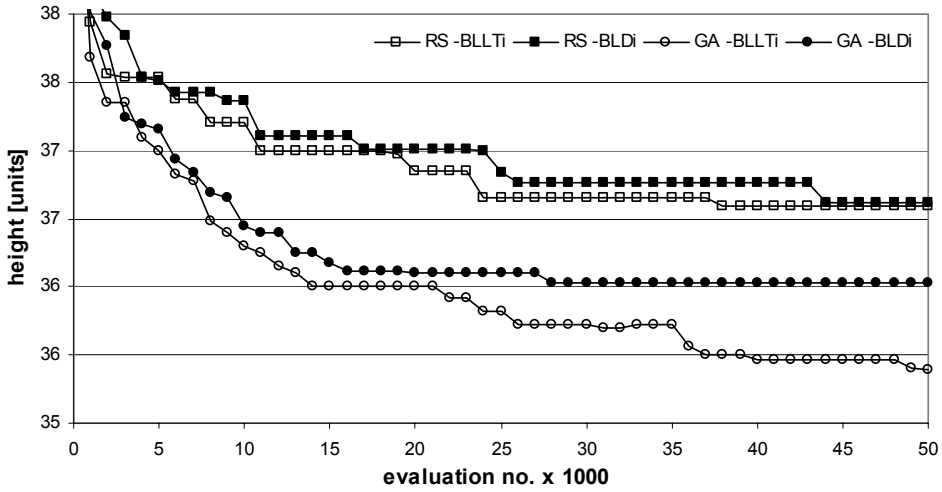


Figure G.1: Comparison between GA and RS with BLLTi and BLDPi decoder for poly2b

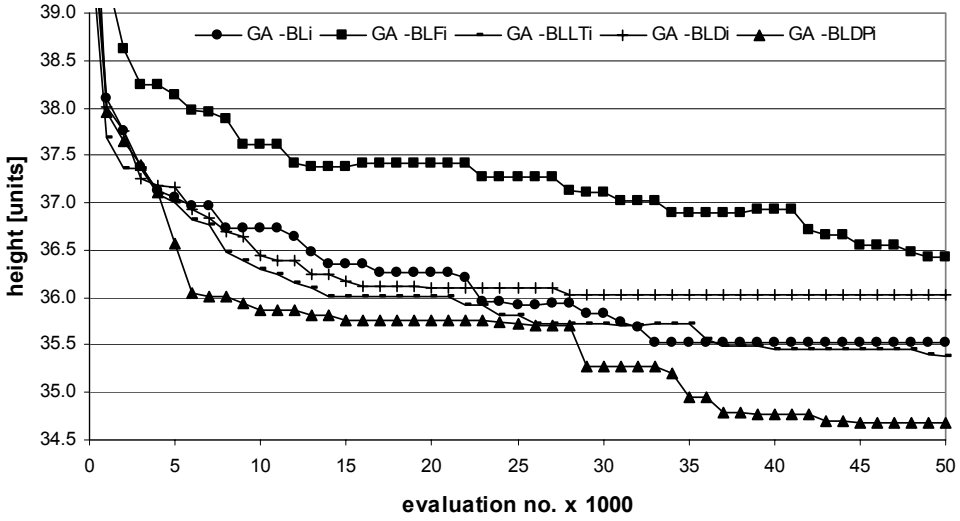


Figure G.2: Comparison between hybrid genetic algorithms for poly2b

Table G.22: Relative difference between GA and best heuristic solution with respect to packing height [%]

	BLi	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**		**BLi**	**BLLTi**	**BLFi**	**BLDi**	**BLDPi**
poly1a	-17	-17	-18	-12	-8						
poly2a	-5	-4	-14	-1	-3	poly2b	-15	-17	-21	-10	-4
poly3a	-5	-3	-11	-3	1	poly3b	-13	-10	-17	-12	-2
poly4a	-2	-3	-10	-1	2	poly4b	-9	-11	-17	-8	1
poly5a	-3	-4	-9	0	4	poly5b	-10	-13	-12	-11	1

H. Results for 2D Rectangular Bin Packing

Table H.1: Best object utilisation [%] for simple packing algorithms with random and pre-ordered input sequences

	BL					BLLT				
	Z	**DH**	**DW**	**DRA**	**DRP**	**Z**	**DH**	**DW**	**DRA**	**DRP**
M1	87.2	91.5	89.9	92.8	92.8	87.4	91.6	90.0	92.7	92.7
M2	84.5	87.9	87.7	89.1	89.1	84.2	88.0	87.5	89.2	89.2
M3	85.7	91.5	90.2	89.8	89.8	85.6	91.7	90.2	90.2	90.2

Table H.2: Best object utilisation [%] for simple packing algorithms with random and pre-ordered input sequences

	BLF					BLD				
	Z	**DH**	**DW**	**DRA**	**DRP**	**Z**	**DH**	**DW**	**DRA**	**DRP**
M1	92.7	94.5	89.2	97.8	97.8	86.6	91.7	88.6	92.3	92.3
M2	88.0	89.8	87.7	93.3	93.3	83.0	87.4	87.2	88.7	88.7
M3	90.7	93.4	90.9	94.6	94.6	84.2	91.4	90.1	89.2	89.2

Table H.3: Average object utilisation [%] for meta-heuristics and random search

	GA				NE				RS			
	BL	**BLLT**	**BLF**	**BLD**	**BL**	**BLLT**	**BLF**	**BLD**	**BL**	**BLLT**	**BLF**	**BLD**
M1	91.9	92.4	95.8	90.7	91.4	91.8	95.4	90.2	90.1	90.3	94.7	88.9
M2	88.2	88.5	91.7	87.1	87.3	87.3	91.1	86.5	86.6	86.9	90.5	85.6
M3	88.4	88.6	92.9	87.1	88.5	88.9	92.7	87.4	87.4	87.6	92.1	86.2

Table H.4: Average object utilisation [%] for meta-heuristics and hill-climbing

	SA				SO				HC			
	BL	**BLLT**	**BLF**	**BLD**	**BL**	**BLLT**	**BLF**	**BLD**	**BL**	**BLLT**	**BLF**	**BLD**
M1	96.7	96.5	97.9	96.4	91.3	91.4	95.4	90.2	87.3	86.5	92.7	86.1
M2	92.7	93.2	94.8	92.8	88.7	88.8	91.9	88.1	84.5	84.7	88.2	83.4
M3	92.4	92.8	95.0	91.5	89.3	89.4	93.4	88.2	86.7	85.8	90.8	84.6

Table H.5: Average number of iterations hill-climbing and for meta-heuristics for one run

	HC				SO				SA
	BL	**BLLT**	**BLF**	**BLD**	**BL**	**BLLT**	**BLF**	**BLD**	**limited to**
M1	42664	35464	42904	37588					300000
M2	44764	38880	38060	36604	89453	89238			300000
M3	68312	58045	62200	70116					300000

Table H.6: Average elapsed time for meta-heuristics and random search for one run [s]

	GA				NE				RS			
	BL	**BLLT**	**BLF**	**BLD**	**BL**	**BLLT**	**BLF**	**BLD**	**BL**	**BLLT**	**BLF**	**BLD**
M1	103	129	125	338	60	75	74	192	387	411	405	617
M2	72	88	142	330	73	90	82	212	406	435	431	695
M3	149	187	187	467	142	178	176	423	895	1057	940	1455

Table H.7: Average elapsed time for meta-heuristics and hill-climbing for one run [s]

	SA				SO				HC			
	BL	**BLLT**	**BLF**	**BLD**	**BL**	**BLLT**	**BLF**	**BLD**	**BL**	**BLLT**	**BLF**	**BLD**
M1	663	839	856	2239	195	242	240	623	124	120	148	295
M2	788	979	936	2376	232	288	290	710	148	153	146	314
M3	1750	2242	1864	5139	499	629	521	1482	445	457	414	1253

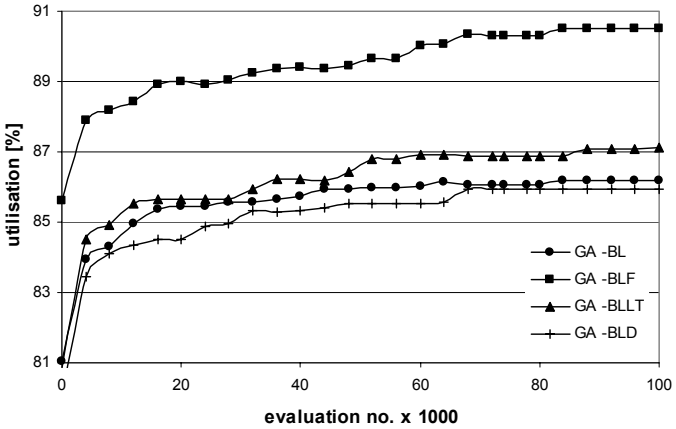


Figure H.1: Average object utilisation with genetic algorithms using four packing routines for problem M2a

Table H.8: Utilisation of the individual objects obtained with Nestlib, BLF-algorithm and GA [%]

	Nestlib		**BLF-algorithm + area-sorted input**		**GA + BLF**	
object number	**problem M2b**	**problem M2d**	**problem M2b**	**problem M2d**	**problem M2b**	**problem M2d**
1	98.5	97.5	94.2	97.5	95.1	94.9
2	98.3	97.5	91.3	96.8	97.0	-
3	91.3	100.0	-	-	92.7	-
4	91.0	99.3	-	99.7	90.0	98.7
5	83.9	100.0	89.3	100.0	-	-
6	77.8	100.0	-	97.0	-	-
7	87.1	100.0	-	100.0	-	100.0
8	87.1	96.0	-	100.0	-	99.0
9	84.1	98.0	-	98.0	-	99.0
10	96.0	98.5	98.5	-	85.1	-
11	91.4	73.8	-	-	85.2	90.0
12	12.7	-	90.7	-	93.3	97.0
13	-	-	91.8	98.5	91.8	-
14	-	-	-	79.5	-	-
15	-	-	-	100.0	-	94.5
16	-	-	91.7	-	-	92.5
17	-	-	88.4	98.5	89.4	95.3
18	-	-	92.6	96.0	-	81.3
19	-	-	92.1	97.0	91.1	94.7
average	**83.3**	**96.4**	**92.6**	**96.7**	**91.3**	**96.9**

I. Papers Produced during this Course of Study

Publications:

Hopper E. and Turton B. C. H., expected 2000.
An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. European Journal of Operational Research.
Accepted in June 99.

Hopper E. and Turton B. C. H., 1999.
A genetic algorithm for a 2D industrial packing problem.
Computers and Industrial Engineering, vol. 37, pp. 375-378.

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.

Submissions:

Hopper E. and Turton B. C. H.
A Review of the Application of Meta-Heuristic Algorithms to 2D Regularand Irregular Strip Packing Problems.
submitted in June 2000 to Artificial Intelligence Review

Hopper E. and Turton B. C. H
Meta-Heuristic Algorithms for 2D Irregular Strip Packing Problems.
submitted in May 2000 to European Journal of Operational Research.