A General Genetic Algorithm for One and Two Dimensional Cutting and Packing Problems

by

Vusisizwe Mancapa, BTech Elec Eng

A dissertation submitted in compliance with the full requirements for the degree of

Magister Technologiae:Engineering: Electrical

in the

Faculty of Engineering, the Built Environment and Information Technology

Nelson Mandela Metropolitan University

Nelson Mandela Metropolitan University

for tomorrow

Promoter: Prof. T. I van Niekerk

Co-Promoter: Mr. M.C. du Plessis

I, **Vusisizwe Mancapa**,

hereby declare that this work is my original work and all sources used or referred to have been documented and recognised.

Further this work has not been submitted in full or partial fulfillment of the requirements for any degree at another recognised educational institute.

Date ....

Author's Signature ....

Abstract

Cutting and packing problems are combinatorial optimisation problems. The major interest in these problems is their practical significance, in manufacturing and other business sectors. In most manufacturing situations a raw material usually in some standard size has to be divided or be cut into smaller items to complete the production of some product. Since the cost of this raw material usually forms a significant portion of the input costs, it is therefore desirable that this resource be used efficiently. A hybrid general genetic algorithm is presented in this work to solve one and two dimensional problems of this nature. The novelties with this algorithm are:

A novel placement heuristic hybridised with a Genetic Algorithm is introduced and a general solution encoding scheme which is used to encode one dimensional and two dimensional problems is also introduced.

Acknowledgements

I wish to express my sincere thanks to both my promoters. Firstly, Prof. T.I van Niekerk who afforded me the opportunity to further my studies, provided much needed support during the course of this study. Secondly, my co-promoter Mr. M.C. du Plessis whose advice and direction, patience I am indebted to. I also wish to express my gratitude to the National Research Foundation, who have funded this study. I wish to thank my colleagues at the Manufacturing Technology Research Center (MTRC), Pule Mulenga, Khotso Majara, Louis von Wielligh, Tao Hua, Vuyo Mjali and Tao Zhang. I wish to thank all the people from the Automotive Components Technology Station (ACTS). I wish to thank my family, especially 'Ma and 'Ta. Last but not least I wish to thank the Lord Jesus Christ, who always makes us triumph over adversity.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cutting and Packing (C&P) problems are combinatorial optimisation problems of practical significance. In most manufacturing situations it is required that a single resource be cut into smaller pieces. This process usually results in waste, it is therefore desirable to reduce the waste that results as much as possible. Examples of this phenomenon can be observed in the following industries: Glass, Paper, Steel, semiconductor, Textile and many other industries.

1.1 Objectives of the study

The objectives of this work are as follows:

- Gain an understanding of what constitutes cutting and packing problems in general.
- The study of those cutting and packing problems that are of manufacturing

significance.

- Conduct a literature survey in this field .
- Gain an understanding of what Genetic Algorithms are.
- Design a general Genetic Algorithm aimed at solving these problems.
- Conduct computational experiments on test problems collected from various literature sources.

1.2 Scope of the research

This work will only be limited to one dimensional and two dimensional problems. All the problems dealt with in this work are listed and defined in section 2.4.

1.3 Overview of the thesis

In chapter 2 a general introduction to cutting and packing is offered and a review of related work is also offered. The description of problems that are targeted in this work is also given and a problem coding scheme that allows the general genetic algorithm proposed in this work to uniquely solve these problems. In chapter 3 a brief introduction to Genetic Algorithms is presented. A review of how genetic algorithms have been used as solution procedures to C&P problems is also offered. The design of the general genetic algorithm and the implementation of the algorithm is dealt with in chapters 4 and 5. The results of computational tests and discussion of the results is offered in chapter 6. The conclusion and suggestions for future research are offered in chapter 7.

Chapter 2

Cutting and Packing

Cutting and Packing problems are optimisation problems whose concern is the optimal allocation of a set of multiple small items into a set of large containing regions (objects), subject to a set of constraints. In disciplines such as Management Science, Information and Computer Science, Engineering and Operations Research, diverse terms are used to refer to problems of this nature (cutting problems, knapsack problem, container and vehicle loading problems, bin packing problems, assembly line balancing, etc). High material utilisation is of particular interest to mass producing industries. Effective utilisation of the material has a financial incentive. If a company is able to minimise waste that results from inefficient use of material, there is a quantifiable saving in the cost of the raw material. This saving can be passed on to the customer or can result in increased profits for the company. In addition, the concerned company may be able to realise further savings in form of reduced stock holding and warehousing capacity. It has always been an objective of decades of academic and industrial research that a means to solve manufacturing problems of this nature be automated. Problems of this nature are common in the sheet metal, lumber, textile and paper industries. In all the above mentioned industries, it is usually more economical to produce large objects in only a few standard sizes at first and later cut them into sizes requested by the customers, than produce the required sizes directly. Other examples of problems of this nature appear in areas that seem unrelated at all to the above stated examples, areas such as land development, facilities layout and electrical circuit layout. Cutting and Packing problems have been shown to be NP-complete (Non-deterministic polynomial time) [Fowler et al. (1981),Garey and Johnson (1980)], therefore it is impossible to solve them in polynomial time.

2.1 Types of Problems

“Cutting and Packing” has now become a term that is used to group subtly different problems into a single field. A few examples of these problems are listed below:

Bin Packing

This problem is concerned with minimising the number of bins into which small items need to be packed in. There are several different versions of this problem appearing in single dimension or multiple dimensional items and bins. The solution to this problem has several industrial applications. Example applications are wood and glass industries, vehicle loading, vehicle routing. For detailed surveys on bin packing (see [Coffman, Jr. et al. (1997), Lodi et al. (2002)]).

Strip Packing

This problem involves packing rectangular or irregular items on to a strip of unlimited height (usually a roll of material is assumed), the objective is to minimise strip height. When packing rectangular items, it is required that the small items edges be parallel to the edges of the strip. In this case the rectangular items may be subjected to orientation constraints, (i.e only 90° rotations are allowed or no rotation allowed). An example of orthogonal strip packing is shown in figure 2.1

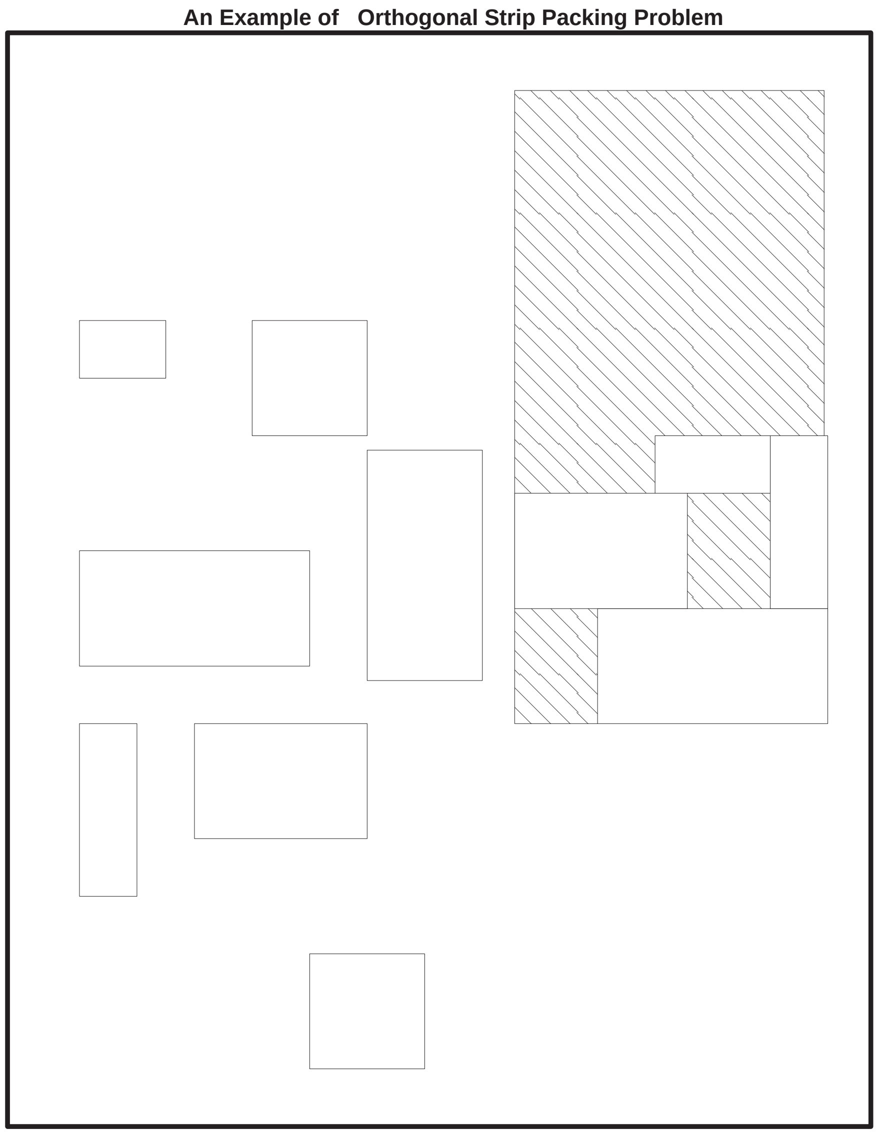


Figure 2.1: A partially packed layout, with those items outside the strip to be packed into the strip.

Knapsack Problem

Given a container of fixed capacity and a set of small items, the requirement is to find the most valuable subset of the small items without violating the capacity constraints of the container. For detailed discussions on knapsack problems (see [Martello and Toth (1990)]).

Nesting

This problem is concerned with packing a set of irregular two dimensional shapes in large two dimensional regions. The complication in this problem arises when the small items are to be packed in irregular sheets (e.g. cow hides). The term nesting is mainly used in the ship building industry.

Loading Problem

The loading of aeroplanes, trucks and containers are all examples of this three dimensional problem, where small boxes have to be loaded to some large three dimensional container efficiently. Additional constraints and objectives can be involved, usually the constraints and objectives vary depending on the industry. An example of the constraints would be to have boxes face a certain direction, because they contain fragile items. An example of the objective would be to order the boxes by the sequence in which they will be offloaded.

Marker Layout Problem

In the textile industries two-dimensional irregular shapes of the pieces of clothing to be cut are packed on textile strips . The templates are used to find optimal material utilisation. The term ”marker” is usually used to refer to the irregular piece of clothing to be cut from the strip of fabric. In academic literature this is sometimes usually referred to as the irregular cutting stock problem. In the leather industries a further complicated version of this problem is encountered, where in addition to irregular small items we have multiple arbitrarily irregular sheets (e.g. cow hides). The quality and strengths of the sheets is not uniform, there might also be defective regions on the sheets.

Assortment Problem

In this problem waist minimisation is approached from a different angle. Instead of trying to minimise waist using available sheets. This problem is concerned with determining what sheet sizes to keep in the warehouse so as to minimise waist.

2.2 Cutting Technology Constraints

When cutting rectangular shapes, another consideration is the cutting technology of the cutting machine. There are two types of cutting achievable for these types of problems i) Guillotine Cutting, (This constraint is particularly important in glass and polystyrene industries for example), ii) Free cutting. Guillotine cuts only allow a cut from one side of the larger rectangular object to the other, parallel to the edges of the larger rectangular object. Figure 2.2 shows an example of two of layouts. One can be cut with guillotine cuts whilst the other cannot.

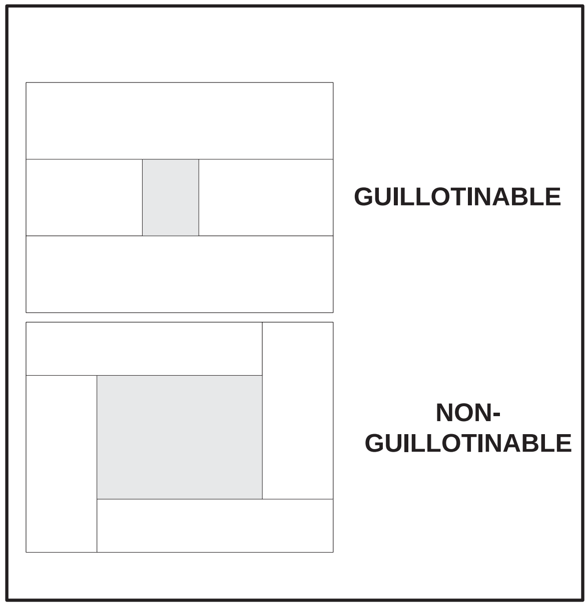


Figure 2.2: Guillotine Cuts vs. Non-Guillotine Cuts

This implies that small rectangular items have to be packed such that this constraint is accommodated. With Free cutting this does not apply.

2.3 Typological Categorisation

In order to provide a comprehensive picture in the field of C&P (Cutting and Packing), Dyckhoff proposed a typology that described problem types based on four characteristics [Dyckhoff and Finke (1992)]. The motivation for Dyckhoff to carry out this task was due to the multitude of problems that exist within the C&P field and the fact that many names are sometimes used to refer to the same problem. Other reasons were to promote cross-fertilisation of research within the academic community and minimise the time spent identifying suitable references. Dyckhoff is credited for highlighting the common underlying structure of cutting problems on one hand and packing problems on the other. Figure 2.3 summarises the main features of Dyckhoff's typology.

1. Dimensionality
   - (1) One-dimensional
   - (2) Two-dimensional
   - (3) Three-dimensional
2. Kind of assignment
   - (B) All objects and a selection of items
   - (V) A selection of objects and all items
3. Assortment of large objects
   - (O) One object
   - (I) Identical dimensions
   - (D) Different dimensions
4. Assortment of small items
   - (F) Few items of different figures
   - (M) Many items of many many different figures
   - (R) Many items of relatively few different figures
   - (C) Items of Congruent (similar) figures

Figure 2.3: Summary of Dyckhoff's Typological Categorisation

Dimensionality

The first characteristic is the identification of the dimensionality of the problem. This criterion deals with the minimal number (1, 2, 3, n>3) of geometric dimensions necessary to describe problems. In one dimensional problems large objects and small items are defined by their length. An example is cutting rods or sewerage pipes, where an object of a given diameter is to be divided into shorter parts. In two-dimensional problems small items and large objects are surfaces. Flat materials (e.g. sheet metal or glass plates) must be cut into smaller sizes of the same material thickness. Three-dimensional problems are typical loading problems (see section 2.1). Multidimensional problems occur mainly in abstract C&P problems. An example would be multi-period capital budgeting in the financial sector. They can however, occur in loading when an item is to be stored within a certain time frame, time is then considered the fourth relevant dimension.

Type of Assignment

The second characteristic is the type of assignment in the particular problem concerned. Dyckhoff separates this criterion to two classes, indicated by B (German for "*Beladeproblem*"). This means all large objects are to be used and a selection of small items is to be assigned to large objects. The second category is V (German for "*Verladeproblem*") characterises a situation in which all small items will be assigned to a selection of large objects.

Assortment of Objects

The third characteristic is the assortment of available objects (e.g. sheets in two-dimensional packing). This characteristic is represented by three options, which are i) O stands for one large object ii) I for several but identical large objects iii)D for several large objects. A perfect example of the first case is when a roll of material is used. With multiple identical objects a new large object has to be initiated once the current object is full, and changing the order in which we use large objects has no effect on the produced solution quality. The situation in which we have an assortment of different multiple objects, the order in which we use large objects has a direct impact on the quality of solution produced, one also need to identify when to initiate a new object if the current object becomes full.

Assortment of Items

The final characteristic is the assortment of small items (shapes in two-dimensional packing). The types of assortment for items take the following form:

- (F) Few items (of different figures)
- (M) Many items of many different figures
- (R) Many items of relatively few figures
- (C)Congruent figures

Below are examples on how Dyckhoff's classification scheme works for a few problem types:

Two-dimensional strip packing problem can be classified as 2/V/O/M

One-dimensional knapsack problem is classified as 1/B/O/M

Two-dimensional Bin Packing Problem is classified as 2/V/I/M

One -dimensional Cutting Stock Problem can be classified as 1/V/I/R

Many inconsistencies and shortcomings of Dyckhoff's typology have been pointed out as a result Dyckhoff's typology has not been well received. A new typology is being proposed that attempts to rectify some of the shortcomings of Dyckhoff's typology (see [Wäscher et al. (2006)]). Another interesting way to characterise typology is that introduced by Lodi et al. (2002) for two-dimensional problems, which they term the three-field typology. For an example a two-dimensional strip packing problem in which rectangular items are to be packed in a strip, rectangular items have to be oriented and free cutting is required, would be written as 2SP|O|F. The diagram in Figure 2.4 gives an overview of C&P problems.

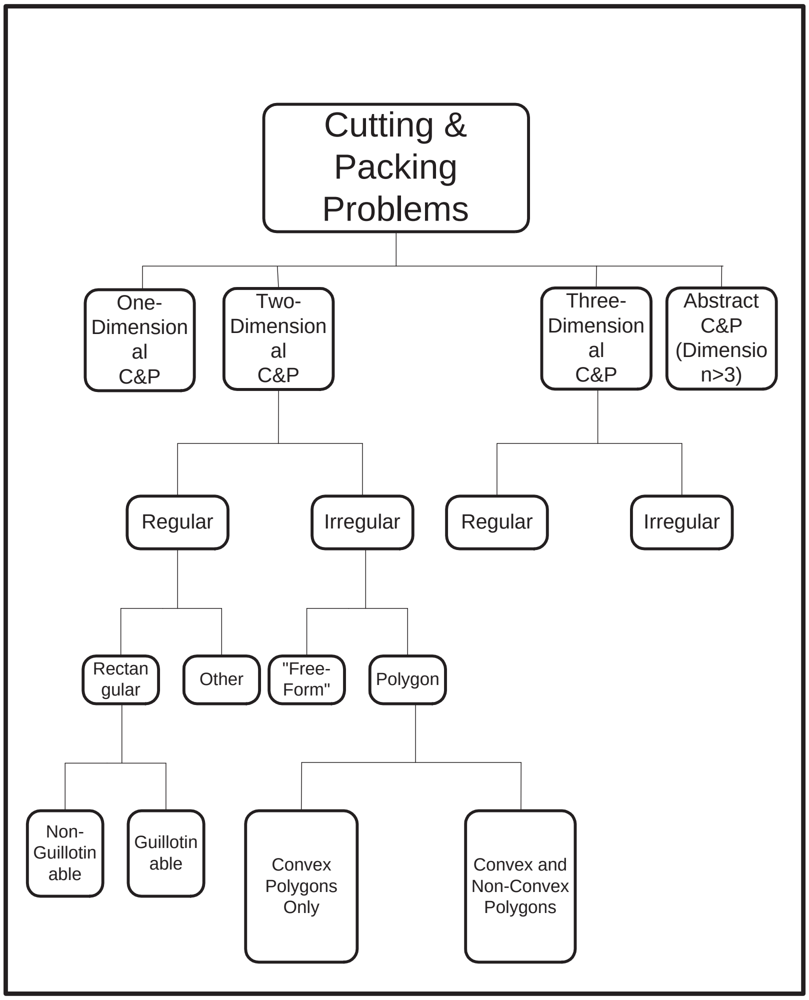


Figure 2.4: Classification of C&P problems adapted from Hopper and Turton (1998)

2.4 Problem Descriptions

In figure 2.4 a diagram that attempts to give an overview of cutting and packing problems is shown. The diagram mainly classifies Cutting and packing problems by the following characteristics: dimension, shape of items, applicable cutting technology constraint. For rectangular items the cutting technology constraint can be further divided into two i.e, guillotine-able and nonguillotine-able. The two dimensional problems can be divided into cutting of regular or irregular shapes. In irregular cutting the pieces to be cut out may take any shape as encountered in clothing, shoe-leather, furniture, automobile and aerospace industries. When cutting out regular shapes, the shapes may be rectangular or any other geometrical shape, i.e. non-rectangular shapes, which are encountered in furniture, paper, and sheet metal industries. This work is limited to only one-dimensional and two-dimensional problems. Table 2.1 contains a lists of problems which will be described formally in the following subsections . The acronyms stand for the following problem types:

- BPP- Bin Packing problem
- CSP- Cutting Stock problem
- SPP-Strip packing problem
- ISPP- Irregular strip packing problem

Consider table 2.1, the following coding scheme: *(**P**roblem type, **D**imension, **O**rientation Constraint, **C**utting Technology Constraint)* is suggested by the table. This problem coding scheme from henceforth would be shortened as (**P**, **D**, **O**, **C**).

Problem Type	Dimension	Orientation Constraints	Cutting Technology Constraint
BPP	1 2	* (1) Fixed Orientation (2) 90^0 rotation permitted	* {Guillotinable (G), Free Cutting (F)}
CSP	1 2	* (1) Fixed Orientation (2) 90^0 rotation permitted	* {Guillotinable (G) , Free Cutting (F)}
SPP	2	(1) Fixed Orientation (2) 90^0 rotation permitted	{Guillotinable (G), Free Cutting (F)}
ISPP	2	(1) Fixed orientation (2) $0^0, 180^0$ Absolute (4) 90^0 Incremental Arbitrary Orientation	*

Table 2.1: Table of Problems

2.4.1 One-Dimensional Problems

Two problems will be dealt with in the one-dimensional category. The one-dimensional cutting stock problem (1-CSP for short) and the one-dimensional bin packing problem (1-BPP). The formal definition for the 1-CSP is, given a set $S = \{1, ..., n\}$ of *items*, each having a length l_i, with each *item* having an associated demand $d_i (i \in S)$. These items have to be cut out of objects with stock length L, $l_i \leq L \quad \forall i \in S$. The objective is to satisfy the demands whilst minimising wate that might result from the cutting process. The definition for the 1-BPP is given a set $J = \{1,...,n\}$ of *items* each having positive weight $w_j (j \in J)$, one has to partition the set J into a minimum number of subsets (*bins*), so that the sum of the weights in each subset does not exceed a given capacity C, $w_j \leq C \quad \forall j \in J$. The two problems stated above are closely related. To illustrate the coding scheme introduced in section 2.4 the problems described above could be coded as follows:

One-dimensional Bin Packing Problem- (BPP,1,*,* [1])

One-dimensional Cutting Stock problem- (CSP,1,*,*)

2.4.2 Two-Dimensional Problems

A formal definition of two-dimensional problems listed in table 2.1 is given in the following subsections and all the problems that arise out of the combination of various constraints.

[1] The * sign stands for a blank or not applicable

2.4.2.1 Two-Dimensional Bin Packing Problem (2BPP)

The formal description of this problem is, We are given a set of items S, where each item $i(i \in S)$ has width w_i and height h_i, and an unlimited number of large objects (rectangular bins) having identical width W and height H. The objective is to place the items into bins without overlap, minimising the number of rectangular bins used to place the items. Taking into account the constraints described above for rectangular packing the following four types of 2BPP problems can be distinguished:

- (BPP,2,2,F):the items may be rotated by 90^0 and no guillotine cutting is required(F);
- (BPP,2,2,G):the items may be rotated by $90^0$and guillotine cutting is required(G);
- (BPP,2,1,F):the orientation of the items should be kept fixed and no guillotine cutting required;
- (BPP,2,1,G):the orientation of the items should be kept fixed and guillotine cutting required;

2.4.2.2 Two-Dimensional Strip Packing Problem (2SP)

We are given n items (small rectangles) each having width w_i and height h_i and one large rectangular object (called a strip) whose width W is fixed, but its height is assumed to be infinite. The objective is to minimise the packing height H of the strip such that all items can be packed into the strip without overlap. Similar to the above stated problem Orientation and Cutting technology constraints have to be taken into account. The resulting problems are as follows :

- (SPP,2,2,F)
- (SPP,2,2,G)
- (SPP,2,1,F)
- (SPP,2,1,G)

2.4.2.3 Two-Dimensional Irregular Strip Packing Problem (2ISP)

We are given n items of arbitrary shapes, and one object (called a strip) with constant width W and a height assumed to be infinite. The objective is to minimise the packing height H of the strip such that all items are contained in the strip without overlap. In this problem the major variant is the orientation constraint of the small arbitrary shapes as shown in table 2.1.

2.5 Related Literature On One-Dimensional Problems

The solution procedures for the solution of one-dimensional cutting and packing problems can be placed in two broad categories, which are i) Exact methods, ii) Heuristic procedures.

The exact methods consists of mathematical programming procedures. Gilmore and Gommory are credited for having been the first to do work in this area for the 1-CSP [Gilmore and Gomory (1961)]. Most of the Linear Programming (LP)-based procedures are inspired by the work of Gilmore and Gommory. This method is based on the following Integer Programming(IP)-model:

$minimise \sum_j X_j$

subject to

$$\sum_j A_{ij} X_j \geq d_i \quad \forall i \in (1, 2, \ldots, n) \tag{2.1}$$

The variable X_j indicates the number of times pattern j will be used. A_{ij} indicates how many times item i appears in pattern j, and d_i represent the demand associated with each item i. In solving the above model Gilmore and Gommory applied a two stage approach . The first stage involved the LP relaxation of the 1-CSP IP model. This is followed by a novel technique that was introduced by Gilmore and Gommory called the column generation technique which is used to generate columns that price out best at every pivot step , to accomplish this an auxiliary problem (a knapsack problem) has to be solved at every step. For simplified example of this approach see [Winston (2004)]. The alternative to LP-based approaches is to use sequential heuristic approaches (SHP). These procedures construct a solution by making one cutting pattern at a time. For more details about this see [Haessler (1992)]. Another set of heuristics to mention is that introduced by Coffman, Garey and Johnson (see [Coffman, Jr. et al. (1997)]) to solve instances of 1-BPP. These are mainly sequential heuristics, i.e a list of items is ordered in some way and items are placed in bins one item at a time. The major difference is in how the items are ordered prior to placement and what criteria is used to place each item in the bin. One of these heuristics is the First Fit Decreasing (FFD) heuristic. In this heuristic the items list is ordered by decreasing weight from largest to smallest first. Items are packed into the first bin that will hold them, If no bin can hold an item a new bin is initialised. See algorithm 1 for the description of this algorithm.

Algorithm 1 *FFD(S,C)*

//S set of items
//C Bin Capacity

1. sort S such that $w_i \geq w_{i+1} \quad \forall i \in S$
2. Place item i in first bin that has enough space. if no bin has enough space, open new bin ($B := B + 1$).
3. Repeat step 2 until all items in S are placed.

The other heuristic in this family of heuristics is the Best Fit Decreasing (BFD) heuristic. With the BFD we sort the items in non-increasing order, The placement criteria for the placement of items is the amount of space left after the placement of an item in the bin, i.e. the bin with the least remaining space after placement. (In the case of the tie we put the item in the lowest numbered bin as labeled from left to right.)

Algorithm 2 *BFD(S,C)*

$//S$ set of items
$//C$ Bin Capacity

1. sort S such that $w_i \geq w_{i+1} \quad \forall i \in S$
2. Place item i in the bin that minimises unused space among those where it fits. If no bin can accommodate i it is placed as in the FFD strategy.
3. Repeat step 2 until every item in S is placed.

See algorithm 2 for summary. Both of the above heuristics have a guaranteed worst case performance of $\frac{11}{9}OPT + 4$, where OPT is the number of bins in the optimal solution to the problem [Coffman, Jr. et al. (1997)].

2.6 Related Literature On Two-Dimensional Rectangular Cutting and Packing Problems

Solution approaches in literature can be broadly categorised into three methods (i) exact methods, (ii) problem specific heuristics, (iii) metaheuristic algorithms.

2.6.1 Exact Methods

Exact methods mainly consist of mathematical programming techniques. The work cited most is that of Gilmore and Gomory as their work is regarded to be seminal in the field of cutting and packing and most LP-based approaches are further modifications of their work.

Gilmore and Gomory [Gilmore and Gomory (1965)] proposed the first model for two-dimensional packing problems , where they extended the approach they used to solve the one-dimensional cutting stock problem see ([Gilmore and Gomory (1961)] and [Gilmore and Gomory (1963)]), They observed that the corresponding number of columns can not be overcome as there was no efficient method for solving the generalised knapsack problem for the two-dimensional problem. Despite this observation they also observed that a wide class of cutting problems in industry have restrictions that permit their knapsack problems to be solved efficiently, i.e. Cutting is done in stages. Beasely [Beasley (1985)] looked at a two-dimensional cutting problem in which profit is associated with each item and the objective is to select a subset of items with maximum profit to be placed into a single bin.

2.6.2 Problem Specific Heuristics

In this section a summary for two-dimensional heuristics is provided.
Two-dimensional heuristics mainly use a permutation coding scheme. These algorithms mainly consist of two phases: (i) Construct a permutation and (ii) Place items one by one onto the larger object(s) using some decoding procedure. For the first phase, items are usually arranged in non-decreasing order based on a certain property e.g. decreasing height, decreasing width or decreasing area. As to which property is best to select is never known apriori. Hence many algorithms generate several permutations with different criteria , and apply a decoding algorithm to all such permutations. The second phase can be further classified to (i) Level-oriented

algorithms, (ii) Non-level oriented algorithms.

2.6.2.1 Level-oriented algorithms

In level based algorithms items are first sorted by some criteria as discussed above. Bin/Strip packing is obtained by placing items , from left to right, in rows forming levels. The bottom of the strip/bin is the first level and subsequent levels are produced by the horizontal line that coincides with the tallest item of the level below. The most popular level algorithms are *next fit, first fit* and *best fit*, which are natural analogues of the one-dimensional bin packing problem. Let $i\,(i = 1,\, 2,\, \ldots, n)$ denote the current item to be placed and s be the level created most recently.

- *Next-fit Decreasing Height* (NFDH) strategy: item i is placed left justified (i.e. placed at the the left-most feasible position) if it fits, else a new level ($s := s+1$) is initialised, and i is packed left justified into it.
- *First-fit Decreasing Height* (FFDH) strategy: item i is placed left-justified on the lowest level (i.e first level) it will fit in. If none of these current levels can accommodate item i, a new level is initialised as in NFDH algorithm.
- *Best-fit Decreasing Height* (BFDH) strategy: we check from level 1 to level s if item i can be accommodated by any of these levels, item i is packed left-justified at a level for which unused horizontal space is a minimum. If no level can accommodate item i, a new level is initialised as in FFDH.

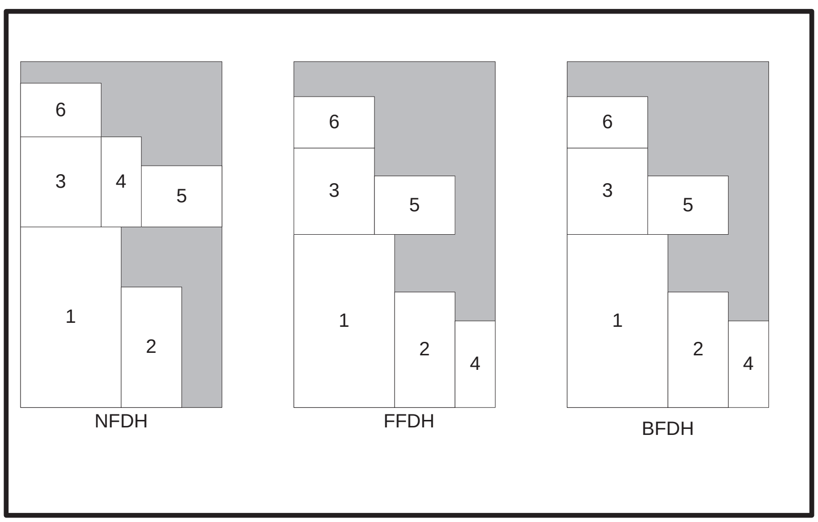


Figure 2.5: Level Oriented Algorithms

The above strategies are illustrated in Figure 2.5 (In this figure items are sorted by non-increasing height and numbered as such). The major difference between the last two strategies compared with the first is that the last two strategies can always turn to previously packed levels for packing a new rectangle, and NFDH always places subsequent rectangles at or above the current level. Level-oriented algorithms were analysed by Coffman, Jr. et al. (1980) for the strip packing problem and determined their worst-case behavior. Given an arbitrary list L of rectangular items and an approximation algorithm A , let $A(L)$ and $OPT(L)$ denote the actual strip packing height for the rectangles in L and minimum height possible respectively. Coffman, Jr. et al. (1980) proved that, if the heights are normalised such that $max_j\{h_j\} = 1$, then

$$NFDH(L) \leq 2 \cdot OPT(L) + 1 \tag{2.2}$$

and

$$FFDH(L) \leq 1.7 \cdot OPT(L) + 1 \tag{2.3}$$

Both bounds are said to be tight (i.e the multiplicative constants on both equations can not be further improved) and if the h_js are not normalised only the additive term is improved. The resulting placements from these algorithms always satisfy the guillotine cut constraint.

2.6.2.2 Non-Level oriented algorithms

The classical algorithm in this category is that proposed by Baker et al. (1980) in 1980 and some variants of this type of algorithm have been proposed the last couple of decades. The characteristic of this algorithm is to place one item at a time, at the lowest feasible position left-justified, this strategy is known as

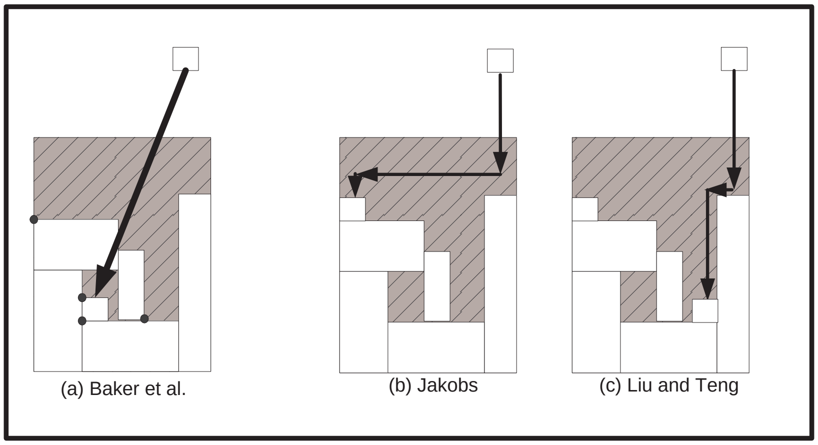


Figure 2.6: Bottom-Left Heuristics

Bottom-Left strategy. Baker et al. (1980) analysed the worst-case performance of this algorithm for the strip packing problem and proved that using a poorly ordered list of rectangular items can perform arbitrarily badly. If rectangular items are ordered by decreasing width then

$$BL(L) \leq 3 \cdot OPT(L) \tag{2.4}$$

The bound can not be improved upon (see Figure 2.6 (a) for example). The other versions of this heuristic are those proposed by [Jakobs (1996), Liu and Teng (1999)]. In Jakobs's algorithm a list of rectangular items L arranged in some order is presented, items are packed into the strip one item at a time. For each item i, first place the item at the top right corner of the strip and slide item i as far down until it collides with either the borders of strip object or another item. Subsequent to this slide the item as far left until it collides with the borders of the object or another rectangular item (see an example in Figure 2.6 (b)). Liu and Teng [Liu and Teng (1999)] algorithm is an improvement on Jakob's work. The observation made by Liu and Teng about Jakob's algorithm was that for small problem instances where optimal solution is known. Jakob's heuristic was unable to find the optimal solution even when all permutations were enumerated. The strategy developed by Liu and Teng was that , the downward movement has priority such that items slide leftwards only if no downward movement is possible (see Figure 2.6 (c)).

2.7 Related Work On Two-Dimensional Irregular Problems

Despite decades of academic research in regular packing problems, the work for the two-dimensional irregular problems is only recent. A major reason for this is the extra dimension of complexity generated by the geometry. However the irregular problem occurs within several important industries, examples include dye-cutting in the engineering sector , parts nesting for shipbuilding, marker layout in the garment industry, furniture and other goods. Published research usually concentrates on a small application areas. These are usually industries in which the raw material forms the large portion of the finished product. Although the number of feasible positions and orientations for a given piece will differ for each application area , the techniques used to solve one problem will be applicable to others. The techniques fall mainly in three categories:

- Items may be nested singly or in groups into a set of enclosing polygons, which are then placed onto the stock sheet.
- Items may be considered one at a time and placed directly onto the stock sheet.
- Items are randomly allocated on the stock sheet initially (which may involve some overlap), then the layout will be improved iteratively.

2.7.1 Nesting

The difficulty encountered when working with problems involving irregular items, has led researchers to devise a strategy that avoids the difficulty altogether. Rather than deal directly with this level of difficulty a number of researchers have considered an alternative strategy in which irregular shapes are nested inside other more regular shapes. The most popular shape for this is the rectangle, which is then packed on the stock sheet using approaches similar to the rectangle packing strategies described in section 2.6. Freeman and Shapira (1975) deal with this problem of finding a minimum area convex polygon that can contain an irregular item, a rectangle of minimum area is then sought which can contain the polygon. This approach was popular in the ship building industry, where many shapes are rectangular and have to be nested with irregular pieces. Adamowicz and Albano (1976) proposed a two staged solution , where the first stage was nesting more than one irregular items together. They placed a limit on the amount of waste that was acceptable in any enclosure. If a shape could be nested on its own with this limit the enclosure is accepted. Otherwise an attempt is made to nest a shape with 180^0 rotation of itself. Another alternative is that taken by Dori and Ben-Bassat (1984), Where they divide the problem in two subproblems, the first is searching for an appropriate set of convex paver polygons, the second subproblem is to find for every irregular shape a paver polygon of optimal (minimal waste) circumscription.

2.7.2 Packing

The advantage of techniques presented above is the simplification of calculations and speeds up computational time. However a better alternative has been shown to be direct methods which base all calculations on suitable representations of the pieces. In this technique items are considered one item at a time and packed directly on the stock sheet according to a given placement policy. One example of these approaches is that by Amaral et al. (1990) whose method select the next piece to be placed dynamically. A sliding process is used to find a suitable position for the next piece on the stock-sheet. Pieces are ordered in non-increasing order of their areas, two different placement policies are used for small and large pieces. Another example is that by Albano and Sapuppo (1980) who attempt to solve a more challenging problem . They use a leftward placement policy and pieces can be placed in a number of different orientations. They restrict pieces to be packed to convex polygons which can be placed in any orientation. Thus the solution space is represented by every permutation of piece types with each one placed at every feasible position in every orientation. This results in a solution space which can not be fully explored in feasible time. To limit the search , they guided the search by two bounds (i) Evaluation of the partial layout obtained so far, (ii) The second is the rough estimate of waste which will be generated by the pieces that are not packed yet. The branch which minimises the sum of the two is chosen next. Milenkovic et al. (1992) observed and interviewed people who design markers in the clothing industry, and sought to design algorithms that emulate skilled workers. They partitioned their approach to three parts: Panel (large pieces) placement , compaction and trim (small pieces) placement . They note that large pieces of similar dimensions are arranged in columns. They also note that the smaller pieces are placed in between the larger pieces. They first identify those pieces thought to be most difficult to place, combining these with other pieces forming columns of four pieces per column. Each column is joined end to end with the previous column so the total length required for the marker can be approximated by the total length of the columns.

2.7.3 Improvement Methods

All the techniques we have considered so far conduct the search in feasible space. Another approach which is increasingly gaining popularity is to produce an initial layout (which may be feasible but suboptimal, or infeasible) and then use small alterations in order to improve it. Such approaches usually seek improvement or incorporate metaheuristic techniques. Penalty functions are usually incorporated to discourage infeasible solutions, e.g. an area of overlap might be proportionally used in the evaluation function as the penalty factor. It is also desirable to reward tight packing and pieces that are nested well. In an attempt to improve the packing neighbourhood moves might include displacing a piece, changing its orientation or swapping two pieces e.t.c (see [Lutfiyya et al. (1992)],[Marques et al. (1991)]).

2.8 Summary

In this chapter cutting and packing was introduced and example industries where cutting and packing problems exist was given. A typological categorisation of C&P problems was presented. It was also pointed out that as far as the typological work is concerned it is still ongoing. A formal introduction to problems that will be looked at in this work was presented. Literature that is related to these problems has also been presented. A general coding scheme has been presented as well. This general coding scheme suggests that a general procedure aimed at solving the problems described above can be realised. A general Genetic Algorithm to achieve this is fully explained in chapter 4.

Chapter 3

Genetic Algorithms Applied to Cutting and Packing Problems

In chapter 2 an introduction to cutting and packing was offered and related work that has been carried out in this field was also presented. In this chapter a brief description is offered of what optimisation is. A section dedicated to a brief description of Genetic Algorithms and a literature survey on how Genetic Algorithms have been used as a solution procedure to tackle cutting and packing problems.

3.1 Optimisation

This section is meant to give a brief explanation of what optimisation is. Optimisation can be loosely described as a process of evaluation of current options, with the intention of finding the best option. In other words it is the minimisation or maximisation of tasks. The nature of optimisation problems can be stated thus for minimisation problems given an objective function f and a search space $\mathcal{S}$ together with its feasible part $\mathcal{F} \subseteq \mathcal{S}$ find $x^* \in \mathcal{F}$ such that

$$f(x^*) \leq f(x) \;\; \forall x \in \mathcal{F}$$

3.2 Genetic Algorithms

Genetic Algorithms (GAs for short) are mathematical procedures based on analogies to the natural evolutionary process. However the evolutionary process simulated by GAs is extremely simplified. Even though recent work reported on GAs focuses on GAs as an optimisation procedure, Dejong cautioned that GAs are not function optimisers but merely procedures that simulate the evolutionary process [Dejong (1993)]. GAs belong to a class of probabilistic algorithms, yet they are different from random algorithms and they combine elements of directed and stochastic search. Algorithm 3 illustrates pseudo code of a simple GA.

Algorithm 3 Simple GA

---

begin
  t← 0
  initialise $P(t)$
   evaluate $P(t)$
 While (!(termination-condition)) **do**
    begin
     t←t+1
       select $P(t)$ from $P(t-1)$
       alter $P(t)$
     evaluate $P(t)$
    end
end

A genetic algorithm is a probabilistic algorithm which maintains a population of individuals, $P(t) = \{x_1^t, \ldots, x_n^t\}$, that are created and selected in an iterative process. Each individual x_i^t consist of a *genome*, a *fitness* and possibly some auxiliary variables such as age and sex. The genome consists of a number of *genes* that altogether *encode* a solution to some optimisation problem. The *encoding* is the internal representation of the problem i.e. the data structure holding the genes. Every member of the population x_i^t is evaluated to measure its fitness. A new population at iteration $t+1$ is formed by selecting those individuals which have more fitness. Some members of the population undergo transformations ("alter" step in the pseudo code), this is achieved by means of some variation operators (These are some times referred to as genetic operators) to form new solutions. The transformations fall into two categories which are unary transformations u_i that create new individuals by a change in a single individual ($m_i : S \rightarrow S$), and higher order transformations c_j that create new individuals by combining parts from several (two or more) individuals ($c_j : S \times \ldots \times S \rightarrow S$). These two transformations are popularly known as mutation and crossover respectively. The algorithm executes until some predefined halting condition is reached, the condition might be the solution quality, number of generations or simply running out of time. During the run of the algorithm the fitness of the best individual (hopefully) improves over time. Ideally at the halting time the best individual found so far should coincide with the discovery of the global optimum, however it is possible for the best individual to converge at a local optimum which is usually the undesirable result. Since GAs are population based search algorithms this means that at any time during the search the fitness function has to evaluate the entire population. This is a serious drawback of GAs as this results in long computational times. For in depth discussions on GAs see [Goldberg (1989), Michalewicz (1996), Mitchell (1998)].

3.2.1 Encoding

Encoding implies representing solutions in a format that will make search operators or genetic operators maintain a functional link between parents and their offspring.

The encoding should make it possible for there to be a useful relationship between parents and offspring. As to which encoding to use differs from from problem to problem , it is fair to say no one encoding technique is best for all problems. Popular examples of encodings are:

- Concatenated binary strings

- Permutations- an example of a problem whose solution is coded using permutations is the Travelling Salesperson Problem (TSP)
- Fixed length vector symbols
- Symbolic expressions

3.2.2 Fitness Evaluation

The fitness evaluation function is the sole means of judging the quality of the evolved solutions. The fitness evaluation function is also necessary in the selection stage, where fitter individuals stand a good chance of being selected as parents and can pass their genetic material on to future generations.

3.2.3 Selection

The basic idea behind selection is that it should be related to the fitness of each individual. The original scheme for its implementation is commonly known as *roulette-wheel* selection, because a common method of accomplishing this procedure can be thought of as a roulette wheel being spun once for each available slot in the next population. Where each solution has a slice of the roulette allocated in proportion to their fitness score (see Figure 3.1 for an example). In this scheme it is possible to choose the best individual more than once, and chances are that the worse individual has a very slim chance of being selected.

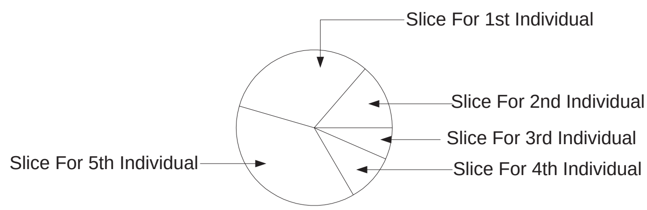


Figure 3.1: A roulette wheel with 5 slices

The other alternative to strict fitness-proportional selection is *tournament selection* in which a set of τ individuals is chosen and compared , the best one being selected for parenthood. It is easy to see that the best solution string will be selected every time it is compared.

Another alternative is rank based selection known as *rank selection.* The fitness assigned to each individual depends only on its position in the individuals rank and not on the actual objective value. With *linear ranking* consider *Nind* the total number of individuals in the population, *Pos* the position of the individual in the population (least fit individual has *Pos*=1, the fittest individual *Pos*=*Nind*) and let *SP* be selective pressure, by selective pressure we mean the ratio of probability the best individual being selected to the probability of the average individual being selected i.e. $SP = \frac{Prob.[selecting\, fittest\, individual]}{Prob.[slecting\, average\, string]}$

The fitness value for the individual is calculated as:

$Fitnes(Pos) = 2 - SP + 2 \cdot (SP - 1) \cdot \frac{(Pos-1)}{(Nind-1)}$

$SP \in [1, 2]$

For all the discussion concerning these see Goldberg (1989).

3.2.4 Variation Operators

Variation operators are means by which to give birth to new solutions or individuals. This is one of the features that make GAs distinct from other search techniques. Not only are GAs evaluating a population of solutions at a time, also these solutions are bred to produce improved solutions. There is usually two types of variation operators i) Crossover Operator , ii) Mutation operator.

3.2.4.1 Crossover Operator

Crossover operator is simply a matter of replacing some genes in one parent by the corresponding genes of the other. Assume we have two individual solutions **a** and **b,** consisting of six variables each, i.e

$(a_1, a_2, a_3, a_4, a_5, a_6)$ and $(b_1, b_2, b_3, b_4, b_5, b_6)$,

which represent two possible solutions to some problem. A one point crossover would be performed by choosing a random crosspoint between positions $1, \ldots, 5$ and a new solution is produced by combining pieces of the original 'parents'. For example if the position 2 is chosen the offspring solutions would be

$(a_1, a_2, b_3, b_4, b_5, b_6)$ and $(b_1, b_2, a_3, a_4, a_5, a_6)$

If we were to choose two cross points randomly between numbers $1, \ldots, 5$ for example if the points were 2 and 4 the offspring solutions would be

$(a_1, a_2, b_3, b_4, b_5, a_6)$ and $(b_1, b_2, a_3, a_4, a_5, b_6)$. In the example presented above we did not use binary strings for the solutions to emphasise that binary representation is not a critical aspect of GAs. Another aspect to crossover operator is that the operation can involve more than two parents.

3.2.4.2 Mutation

Mutation is a one-parent variation operator. The mutation operator is an over simplified analogue from natural evolution. It usually consists of making small random perturbations to one or few genes. One of the major reasons for the mutation operator in GAs is the introduction of population diversity during the genetic search. Originally, with binary encoding, a zero would be changed into a one and vice versa. With alphabets of higher cardinality, there are more options changes can be made at random or following a set of rules.

3.3 Related work on GAs applied to Cutting and Packing Problems

In this section a brief survey of published literature on the application of genetic algorithms to cutting and packing problems is presented. Special emphasis is placed on the encoding of the problem variables and variation operators as these tend to affect the performance of the GA. The survey is by no means exhaustive. Smith (1985) is credited for being the first researcher to apply GAs to packing problems. At roughly the same time Davis (1985) summarised how GAs can be used to solve a two-dimensional bin packing problem.

3.3.1 Literature on GAs and One-dimensional Problems

Falkenauer and Delchambre (1992) attempted to solve the one dimensional bin packing problem (see subsection 2.4.1 for description). They made the following observation about using the classical GA:

The traditional crossover and mutation had a tendency to disrupt good evolved solutions, waisting all the effort in the preceding genetic search.

To overcome this they proposed a grouping encoding scheme that took into account the grouping of items that were to be packed into bins. The scheme was to divide the chromosome into two parts the items part separated by a semi colon from the objects (bins) part i.e.

Items Part: Objects Part

Consider the following example, the encoding

ADBCEB:BECDA

The first part before the semicolon can be interpreted as

Item	1	2	3	4	5	6
In Object (bin)	A	D	B	C	E	B

The second part lists all the objects used to pack items. With this encoding Falkenauer and Delchambre (1992) proposed a special crossover and mutation which only worked with the objects part of the chromosome. The variation operators make use of two heuristic procedures First Fit (FF) and the First Fit Decreasing(FFD) heuristic (see section 2.5). An example of crossover carried out by Falkenauer and Delchambre (1992) is given below. Consider the two object parts of the chromosomes shown below

ABCDEF (first parent)

abcd (second parent)

Two random sites are chosen as crossover positions in each chromosome, yielding for example

A|BCD|EF and

ab|cd|

The bins between the crossover sites in the second parent are injected into the first parent into the first crossing site, which yields

AcdBCDEF

This results in infeasible solution because some items appear twice in the solution and must be eliminated. Suppose the objects injected with bins c and d also appear in bins C, E and F. These bins are then eliminated leaving the solution

AcdBD

It could be that with the elimination of those three bins items that were not injected with the bins c and d have also been eliminated, Those items are thus missing from the solution. To fix this problem the FFD heuristic is applied to reinsert them yielding a solution like the one shown below for example:

AcdBDx

Where x is one or more bins formed by the reinserted items. To carry out the mutation a few random bins are selected and eliminated from the solution. The items that composed the eliminated bins are now missing from the solution and the FF heuristic is used to reinsert them back in random order. Hinterding and Khan (1995) used a GA to solve a multi stock one-dimensional cutting stock problem. They extended the work of Falkenauer and Delchambre (1992) where they devised a group based mapping for solutions an example of which is demonstrated in Figure 3.2. This representation is such that a valid group of items implies the stock length it should be cut from. This is the smallest stock length from which the group of items can successfully be cut from. This results in a chromosome of variable length. The crossover operation for this problem is a modification Faulkners' Grouping crossover [Falkenauer and Delchambre (1992)]. This crossover works as follows :

An insertion point is randomly chosen in parent1 and a segment is chosen from parent2. The offspring is constructed by first copying into it genes from parent1 up to the insertion point. Then genes are copied from the segment in parent2 into the offspring, lastly genes from parent1 after the insertion point are then copied into the offspring. It should be pointed out that only those items not yet included into the offspring chromosome are copied into the offspring. At the end of the above described process the list of items not yet included in the offspring chromosome is included using the first fit(FF) heuristic.

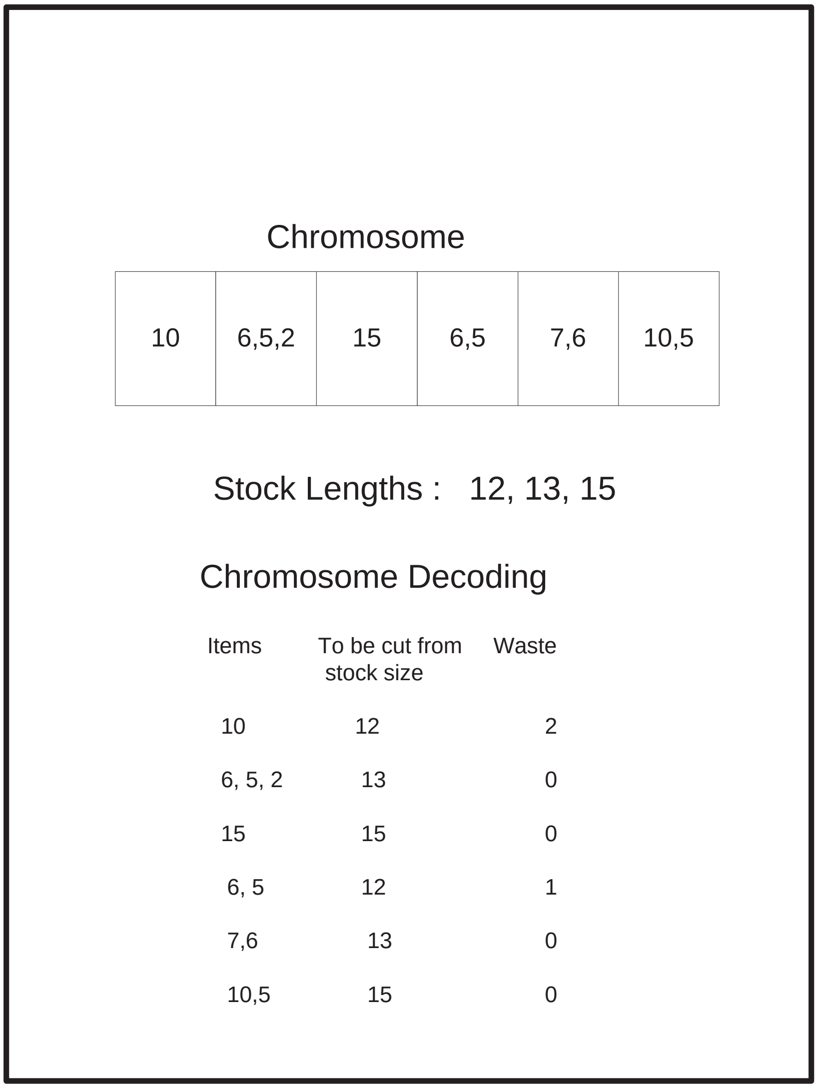


Figure 3.2: Hinterding and Khan (1995)'s representation

3.3.2 Related work on GAs applied to two-dimensional rectangular problems

One of the most popular approaches used by most researchers in using GAs when solving cutting and packing problems is a two-stage procedure, a hybrid genetic algorithm. In this the GA manipulates the encoded solutions, these solutions are then evaluated by the decoding algorithm, which transforms the encoded solutions into the corresponding physical layouts. The decoding procedure used can either be deterministic or heuristic. However the decoding procedure results in the loss of information from one generation to the next. This is because the domain knowledge is built into the decoding procedures.

3.3.2.1 GAs on Non-guillotine able Packing Problems

Jakobs (1996) proposed a hybrid genetic algorithm that allocates rectangular figures to a rectangular board of a fixed width and unlimited height with the aim of minimising the height of the occupied area. The GA is combined with the deterministic procedure that decodes the solutions to corresponding physical layouts. The decoding procedure used by Jakobs (1996) is the Bottom Left (BL) placement heuristic (see sub subsection 2.6.2.2 for explanation). Jakobs used a permutation π as a solution representation where the fitness function is determined by:

$$f : \pi \rightarrow \mathbb{R}_+$$

Since the height is not sufficient for the comparison of different packings, the fitness function also takes into account the largest resulting contiguous remainder see

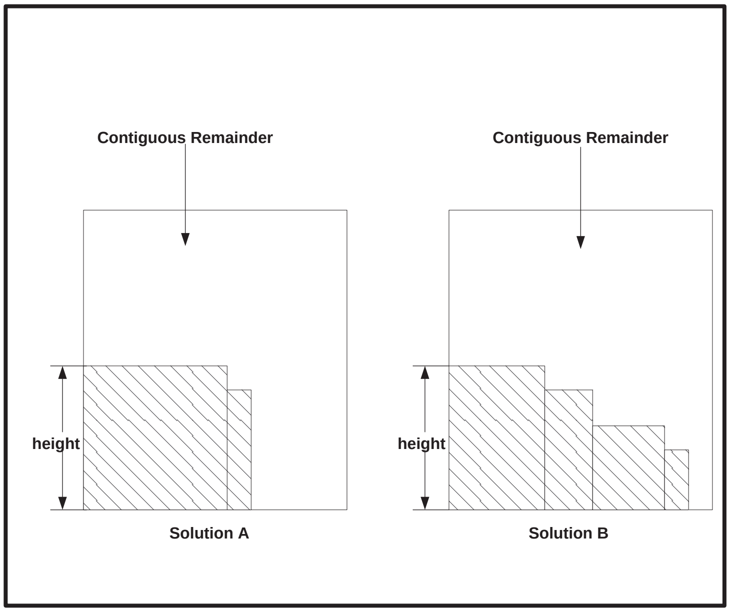


Figure 3.3: Contiguous Remainder of the packing Patterns

Figure 3.3 for illustration. In the illustration shown in Figure 3.3 let solution A be represented by π_1 and solution B be represented by π_2, In this situation it can be said that $f(\pi_1) > f(\pi_2)$ i.e. π_1 is a better packing than π_2 .

The contribution by Liu and Teng (1999) was aimed at improving the decoder used by Jakobs (1996), everything else remaining as proposed by Jakobs. Hopper and Turton (1999) designed a hybrid genetic algorithm using the permutation representation. The decoding procedure they used could access enclosed areas in the partial layout and placed the new items in the first bottom left position with sufficient area.

3.3.2.2 GAs on guillotine able Packing Problems

Kroger (1995) proposed a representation that ensures packing patterns are guillotine able. The relative arrangement of the rectangles is described as a slicing tree structure. In the tree leaf -nodes correspond to a rectangles to be packed, whereas all other nodes represent the hierarchy of guillotine cuts needed for the packing scheme. Apart from guaranteeing a guillotine able solution this representation contains the complete subtrees which can be manipulated separately. The fitness of the string is related to the height of the packing pattern.

A special crossover operator has been developed that preserve the knowledge that is stored in the subtrees. The mutation operator involved five different operators which are applied randomly, these involve swapping of adjacent subtrees, inversions of cut-line and rectangle orientation.

Hwang et al. (1994) used a permutation based representation to tackle the strip packing problem. They used a level-oriented heuristic as a decoder. See subsection 2.6.2.1 for typical level-oriented algorithms.

3.3.3 Related work on GAs applied to two-dimensional Irregular Packing problems

Two dimensional Irregular packing problems involve packing arbitrary shapes in well defined objects of fixed width and unbounded height. In most solution approaches the arbitrary items are approximated by polygons consisting of a list of vertices. Geometric algorithms are then made use of to determine feasible positions in the

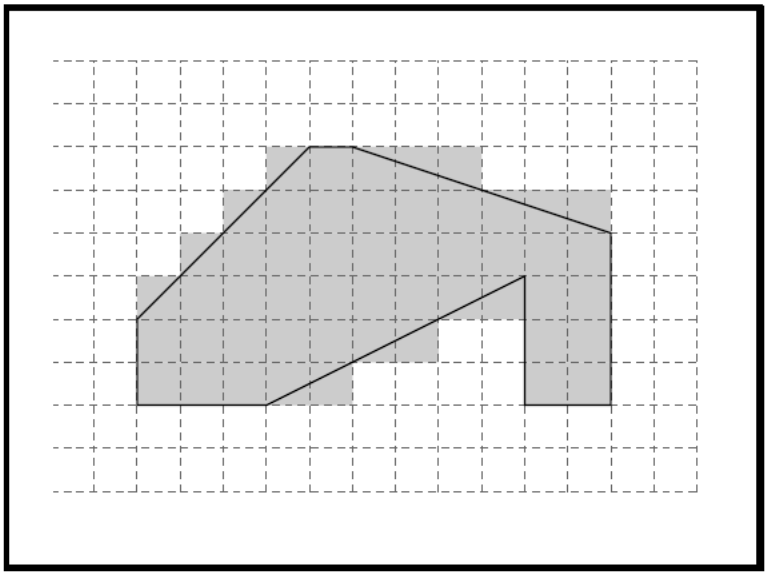


Figure 3.4: An example of a grid model

partial layout and eventually calculate the overlap. Another shape approximation technique is the grid approximation technique, where items are approximated by a list of equal sized squares using 2D matrices. An example of a grid approximation technique is shown in figure 3.4.

Work presented in this dissertation will only focus on shape approximation using polygons.

Jakobs (1996) extended his work on packing rectangular objects (see sub subsection 3.3.2.1) to packing polygons. He used a three step approach:

- Enclose Polygons into rectangles
- Apply a hybrid GA to the rectangles enclosing polygons as described in su bsubection 3.3.2.1
- *Shrinking-step*: Shift the polygons closer to each other

The final step is only arrived at when there is no longer any improvement brought about by the GA, i.e. the *Shrinking-step* deals with the irregular aspect of the problem. This algorithm moves polygons closer together using the idea of the BL-heuristic (see sub subsection 2.6.2.2), the polygons are shifted alternately as far as possible to the bottom and to the left whilst avoiding overlap and also tests reflections of the original polygons. Bounsaythip and Maouche (1997) applied a binary tree representation to a problem from the textile industry. The shapes are approximated with a special encoding technique that describes the contour of the polygon relative to the enclosing rectangle using a set of integer values which they called comb-coded shapes. For every side of the four sides of the rectangle such a contour is generated. The nodes on the tree contains information that indicates at which side of the stationary shape will the second shape be placed and its orientation. Petridis and S.Kazarlis (1994) developed a genetic algorithm with a dynamic fitness function, which does not make use of the decoding algorithm in the nesting process. The solution was encoded using binary strings for each reference vertex of the items om the layout. This encoding allowed for the traditional crossover operators to be used. This meant that overlapping could occur, a penalty function was used to discourage overlaps. The fitness function is dynamic, increasing the penalty term gradually as the search continues in order to move the population away from invalid solutions towards the end of the search. Petridis and S.Kazarlis (1994) tested their algorithm on jigsaw problems consisting of less than 15 pieces. Comparison showed that the optimal solution was more often found using the dynamic fitness function.

3.4 Summary

In this chapter a brief definition for optimisation was given. A summary of Genetic Algorithms was offered and how they work. A literature survey was offered on how Genetic Algorithms have been applied to solve various cutting and packing problems.

Chapter 4

The General Genetic Algorithm

Cutting and packing was introduced in chapter 2 and examples of relevant industries where cutting and packing problems need to be solved was provided. In chapter 3 Genetic algorithms were introduced and how they can be applied as optimisation procedures. Relevant work on how GAs were applied to cutting and packing problems was also presented. In this chapter a general GA is presented and a general solution encoding that is meant to represent all problems defined in section 2.4 is also presented. Algorithm 4 presents the idea behind the general genetic algorithm presented in this work and comments directing the reader to relevant sections where aspects of the algorithm are discussed in full detail.

Algorithm 4 General Genetic Algorithm

```
begin
    t← 0
    initialise P(t)   // See section  4.2 for discussion
      evaluate P(t) // See section 4.5 for discussion
  While (!(termination-condition)) do
      begin
        t←t+1
            select P(t) from P(t − 1)
            alter P(t)  // see section 4.4 for discussion
        evaluate P(t)
      end
end
```

4.1 Solution Representation

In section 3.3 a survey was presented on related work where different solution representations were presented and explained. In this section a generic solution representation is presented, which serves as a template solution for all the problems defined in section 2.4. The general solution representation consists of two parts.

Problem Code	**Problem Specific Encoding**

The problem code part is the 4-tuple code introduced in section 2.4. The code consists of the following fields (**P**roblem type, **D**imension, **O**rientation Constraint, **C**utting Technology Constraint), this code augments the problem specific encoding for every problem. The interpretation of this code was fully explained, the advantage of using this code is the ability to uniquely identify a problem with associated constraints.

A general representation for the problem specific part of the solution representation is given below

$$\{(x_1, i_1, \phi_1), (x_2, i_2, \phi_2), \ldots, (x_n, i_n, \phi_n)\}$$

The interpretation of every variable in the above given representation is problem specific. In other words every problem solution's representation is in this format. The full solution representation can be written as

$\overrightarrow{X} = [(\mathbf{P}, \mathbf{D}, \mathbf{O}, \mathbf{C}), \{(x_1, i_1, \phi_1), (x_2, i_2, \phi_2), \ldots, (x_n, i_n, \phi_n)\}]$

4.1.1 Interpretation of the solution for one-dimensional problems

In subsection 2.4.1 two one dimensional problems were defined, viz. 1D Bin packing problem, 1D Cutting stock problem. It was also stated that the two problems are closely related. The approach taken in this work is to look at these two problem types as one problem, i.e the one dimensional cutting stock problem is converted to one dimensional bin packing problem.

The meaning of the solution presented above for the one dimensional bin packing problem can be explained as follows:

P=BPP

D=1

O=*(blank)

C=*

x_k= the bin in which item i_k is assigned to

i_k= is the index of the item assigned to bin x_k.

ϕ_k=*

The general standard representation for the one dimensional bin packing problem is given by

$$\overrightarrow{X_1} = [(BPP, 1, *, *), \{(x_1, i_1, *), (x_2, i_2, *), \ldots, (x_n, i_n, *)\}]$$

The one dimensional problem is mainly a grouping problem, i.e we need to fit items from the item set to a smallest number of bins (groups). In such a situation the order of items should not matter. The approach taken in this work is the optimal grouping of items to a bin. An example to illustrate this solution encoding is provided below. Suppose we have bins of capacity 10 and a list of item sizes $L = [3, 6, 2, 1, 5, 7, 4, 9]$. One possible way to pack these items is shown in figure 4.1. Using the encoding introduced above the solution shown in figure 4.1 can be represented as

$$\overrightarrow{X} = [(BPP,\ 1,\ *,\ *), \{(1,1,*), (1,2,*), (2,\ 3,\ *), (2,\ 4,\ *), (2,\ 5,\ *), (3,\ 6,\ *), (4,\ 7,\ *), (5,\ 8,\ *)\}]$$

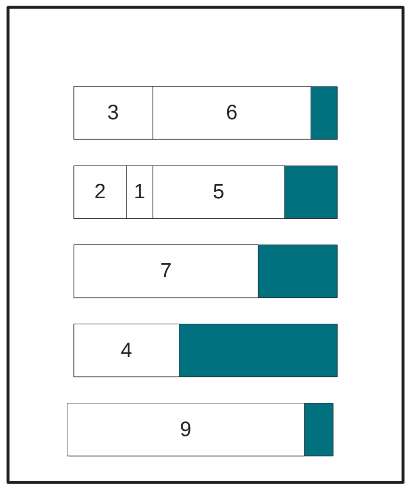


Figure 4.1: A possible solution for one dimensional bin packing problem, where the shaded areas represent waste.

4.1.2 Representation for 2D problems

The representation for 2D problems follows from the general representation introduced in section 4.1, which is

$$\overrightarrow{X} = [(\mathbf{P},\ \mathbf{D},\ \mathbf{O},\ \mathbf{C}),\ \{(x_1, i_1, \phi_1), (x_2, i_2, \phi_2), \ldots, (x_n, i_n, \phi_n)\}]$$

For two dimensional problems the solution representation is evaluated by means of a placement heuristic which is fully explained in section 4.5. What this implies is the problem specific part of the encoding can be considered ordered based representation, i.e. items are considered one item at a time for the placement. Let (x_i, y_i) be the bottom left corner of the rectangular item chosen as the reference for the rectangular item to be placed and be the reference vertex $v(x_i, y_i)$ if the item to be placed is the polygon. These items are to be placed in rectangular regions. Let the bottom left corner of the containment region (stock sheet) be the origin $(0, 0)$ with it's four sides parallel to the $X-$ and $Y-$ axes respectively. The meaning for the problem specific part variables is provided below:

x_k=	The x-coordinate value of the reference vertex for the kth item
i_k=	The index of the kth item
ϕ_k=	is the orientation of the kth item.

Now that the meaning of the variables for the encoding has been explained below the coding for each of the two dimensional problems defined in subsection 2.4.2 is presented.

Two dimensional Bin packing problems

For the problems described in sub subsection 2.4.2.1 the solution encoding is given below:

Items may be rotated by 90^0 and no guillotine cutting required:

$$\overrightarrow{X_2} = [(BPP,\ 2,\ 2,\ F),\ \{(x_1, i_1, \phi_1), (x_2, i_2, \phi_2), \ldots, (x_n, i_n, \phi_n)\}],$$

$\phi_i \in \{0^0,\, 90^0\}$

Items may be rotated by 90^0 and guillotine cutting is required:

$\overrightarrow{X_3} = [(BPP,\, 2,\, 2,\, G),\, \{(x_1, i_1, \phi_1), (x_2, i_2, \phi_2), \ldots, (x_n, i_n, \phi_n)\}],$

$\phi_i \in \{0^0,\, 90^0\}$

Items may not be rotated and no guillotine cutting required:

$\overrightarrow{X_4} = [(BPP,\, 2,\, 1,\, F),\, \{(x_1, i_1, \phi_1), (x_2, i_2, \phi_2), \ldots, (x_n, i_n, \phi_n)\}],$

$\phi_i = 0^0$

Items may not be rotated and guillotine cutting required:

$\overrightarrow{X_5} = [(BPP,\, 2,\, 1,\, G,)\, \{(x_1, i_1, \phi_1), (x_2, i_2, \phi_2), \ldots., (x_n, i_n, \phi_n)\}],$

$\phi_i = 0^0$

Two dimensional Strip packing problems

Problems in this category were defined in sub subsection 2.4.2.2, the solution encoding for each problem is illustrated below:

Items may be rotated by 90^0 and no guillotine cutting required:

$\overrightarrow{X_6} = [(SPP,\, 2,\, 2,\, F),\, \{(x_1, i_1, \phi_1), (x_2, i_2, \phi_2), \ldots, (x_n, i_n, \phi_n)\}],$

$\phi_i \in \{0^0,\, 90^0\}$

Items may be rotated by 90^0 and guillotine cutting is required:

$\overrightarrow{X_7} = [(SPP,\, 2,\, 2,\, G),\, \{(x_1, i_1, \phi_1), (x_2, i_2, \phi_2), \ldots, (x_n, i_n, \phi_n)\}],$

$\phi_i \in \{0^0,\, 90^0\}$

Items may not be rotated and no guillotine cutting required:

$\overrightarrow{X_8} = [(SPP,\, 2,\, 1,\, F),\, \{(x_1, i_1, \phi_1), (x_2, i_2, \phi_2), \ldots, (x_n, i_n, \phi_n)\}],$

$\phi_i = 0^0$

Items may not be rotated and guillotine cutting required:

$\overrightarrow{X_9} = [(SPP,\ 2,\ 1,\ G),\ \{(x_1, i_1, \phi_1), (x_2, i_2, \phi_2), \ldots, (x_n, i_n, \phi_n)\}],$

$\phi_i = 0^0$

Two dimensional Irregular Strip packing problems

As mentioned already in sub subsection 2.4.2.3 that, the major variant in this problem is the orientation constraint of the items. The following problems in this category can be coded as shown below:

Item orientation is fixed:

$\overrightarrow{X_{10}} = [(ISPP,\ 2,\ 1,\ *),\ \{(x_1, i_1, \phi_1), (x_2, i_2, \phi_2), \ldots, (x_n, i_n, \phi_n)\}],$

$\phi_i = 0^0$

Item orientation can be rotated by 180^0:

$\overrightarrow{X_{11}} = [(ISPP,\ 2,\ 2,\ *),\ \{(x_1, i_1, \phi_1), (x_2, i_2, \phi_2), \ldots, (x_n, i_n, \phi_n)\}],$

$\phi_i \in \{0^0,\ 180^0\}$

Item can be rotated at fixed 90^0 increments:

$\overrightarrow{X_{12}} = [(ISPP,\ 2,\ 4,\ *),\ \{(x_1, i_1, \phi_1), (x_2, i_2, \phi_2), \ldots., (x_n, i_n, \phi_n)\}],$

$\phi_i \in \{0^0,\ 90^0,\ 180^0,\ 270^0\}$

4.2 Initial Population Generation

In section 3.2 it was mentioned that at any time during the search a GA maintains a population of solutions. In the pseudo code presented in algorithm 4 the initial step is the initialisation of the population. In this section the population initialisation process for the problems described above is explained.

4.2.1 Initial Population generation for one dimensional problems

The usual way to generate the initial population is generating the population randomly. The problem with this procedure for the one dimensional problems dealt with here would be the generation of infeasible solutions. What is needed is the generation of solutions, which is both random and does not violate any of the constraints. For that purpose the First Fit (FF) heuristic is used as an initial population generator. Where items are randomly ordered and packed into bins using the FF heuristic. The version of the FF heuristic where items are arranged by non-increasing order was presented in section 2.5.

4.3 Initial Population generation for two dimensional problems

To generate the initial population it is ensured that every individual belonging to the initial population of solutions is feasible. In section 4.1 a general problem representation was introduced, which enables us to both uniquely identify problems and encode all the problems considered in this work in a standard format. For every item in the two dimensional problems, was represented by a 3-tuple (x_k, i_k, ϕ_k). Each variable in this 3-tuple has been described. x_k was defined as the x-coordinate of the reference vertex for each item (the bottom left corner for rectangular items). For rectangular items the only feasible values for x_k are in the interval $P_{x_i} = [0, W - w_k]$ where W is the width of the container and w_k is the width of the rectangular item r_k see figure 4.2 for illustration. The same goes with the third element of the 3-tuple, ϕ_k is always a member of a feasible set of orientation constraints, for example in problems where rectangular items can only be rotated by 90^0 , the set of feasible constraints O_c consists of only two elements, i.e. $O_c = \{0^o, 90^o\}$.

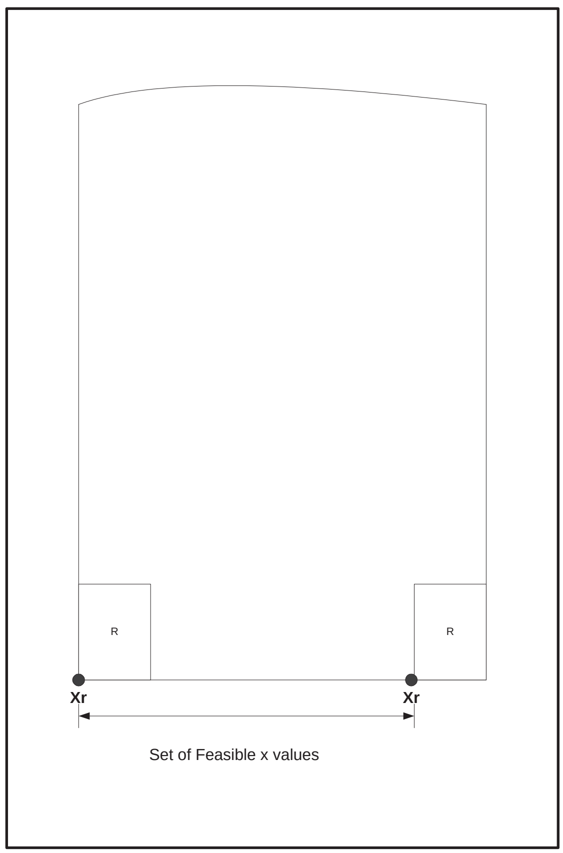


Figure 4.2: A set of Feasible x co-ordinates for rectangular items

The same argument holds for polygonal items, i.e. there is a set of feasible

x-coordinate positions for the reference vertex of each polygon item for the containment constraint and a set of feasible orientation constraints for each item. To generate initial population the following procedure is followed:

1. Randomly order items.
2. Randomly choose a feasible x-coordinate of the reference vertex for each item from the set of feasible x-coordinates .
3. Randomly choose an orientation from the set of feasible orientation constraints for each item.

4.4 Variation Operators in the general GA

A GA always has to have a means to pass on knowledge obtained so far about the search to future generations. The variation operators mainly introduce diversity in the genetic material that has to be passed on to future generations. The variation operators which are popularly known as crossover and mutation. In this section variation operators used in this algorithm are explained.

4.4.1 Variation Operators for 1D problems

For one dimensional problems a crossover operator has been designed which takes into account the grouping nature of these problems, i.e. items have to be packed into bins (groups). The operator is such that the offspring inherits as much information from both parents. Let $\overrightarrow{X_{1A}}$ and $\overrightarrow{X_{1B}}$ be two parents and let N_{BinA} and

N_{BinB} be the number of bins used in $\overrightarrow{X_{1A}}$ and $\overrightarrow{X_{1B}}$ respectively. For example suppose we have bins of capacity 10 and a list of item sizes

$L = [3, 6, 2, 1, 5, 7, 4, 9]$.

Let

$\overrightarrow{X_{1A}} =$

$[(BPP, 1, *, *), \{(1, 1, *), (1, 2, *), (2, 3, *), (2, 4, *), (2, 5, *), (3, 6, *), (4, 7, *), (5, 8, *)\}]$

and

$\overrightarrow{X_{1B}} =$

$[(BPP, 1, *, *), \{(1, 6, *), (1, 1, *), (2, 2, *), (2, 3, *), (3, 7, *), (3, 4, *), (4, 8, *), (5, 5, *)\}]$

be two parent solutions and let *child* be the resulting offspring from the crossing of the two parents. The offspring would be produced as follows:

1. A binary vector V_b is randomly created whose dimension is $max\{N_{BinA}, N_{BinB}\}$. Vector V_b will be used in the following step as a selection mechanism:
   For this example the dimension of $V_b = max\{5, 5\}$, $dim(V_b) = 5$, say the vector is randomly generated to be $V_b = [1\,1\,0\,1\,0]$.

2. V_b is used to select from $\overrightarrow{X_{1A}}$ those bins that correspond to vector positions in V_b whose entry is 1, and from $\overrightarrow{X_{1B}}$ those bins that correspond to vector positions in V_b whose entry is 0:
   According to this example this implies bins 1, 2 and 4 are selected from $\overrightarrow{X_{1A}}$ and bins 3 and 5 are selected from $\overrightarrow{X_{1B}}$ for transfer to the offspring solution.

3. Transfer those bins selected from both parents to the offspring one bin at a time, in transferring the bins we ensure that there is no conflict, i.e. no items

already present in the offspring solution are duplicated:
First and second bin are transfered from $\overrightarrow{X_{1A}}$ to *child*, so far no conflict has resulted. The third bin to be transfered is bin 3 from $\overrightarrow{X_{1B}}$ contains items [7 4]. If we transfer the bin as it is, a conflict will result because item 4 is already part of the *child* solution. This bin is transfered with out item 4. So far the offspring solution is
$child = [(BPP,\ 1,\ *,\ *), \{(1,1,*),(1,2,*),(2,3,*),(2,4,*),(2,5,*),(3,7,*)\}]$
After the transfer of all bins has been carried out the offspring will be
$child = [(BPP,\ 1,\ *,\ *), \{(1,1,*),(1,2,*),(2,3,*),(2,4,*),(2,5,*),(3,7,*)\}]$

4. After the above mentioned steps have been carried out it may be that some items are missing from the offspring solution, the FF (First Fit) heuristic is used to complete the partial offspring solution:
For an example the *child* solution above has items 6 and 8 missing, these items are allocated using the FF heuristic.

Then the result of the crossing of the two parent solutions will be

$$child =$$
$$[(BPP,\ 1,\ *,\ *), \{(1,1,*),(1,2,*),(2,3,*),(2,4,*),(2,5,*),(3,7,*),(4,6,*),(5,8,*)\}]$$

GAs at times tend to stagnate at a local optimum. This normally occurs in later generations when individuals have converged to a dominant individual. To discourage this tendency the mutation operator is used to diversify the population . The mutation for the one dimensional problems is as follows:

1. We randomly generate an integer number N_b in the interval $[1, N_{Bin}]$. Where N_{Bin} is the number of bins in the parent solution.
2. The bin numbered N_b is scattered .
3. The items that constituted N_b are randomly ordered and repacked using the FF heuristic to complete the offspring solution.

4.4.2 The variation operators for 2D Problems

In subsection 4.1.2 it is stated that the fitness evaluation (Which will be explained in section 4.5) of all solutions is done through a placement heuristic. This implies that the solution representation for 2D problems is of ordered nature, and the horizontal position (x-coordinate) of each item is also part of the encoding. The variation operators for the 2D problems take all of these into consideration.

4.4.2.1 Crossover Operator

As a crossover operator for 2D problems two crossover variants are used, viz. $cross\_var1$ and $cross\_var2$. The choice always has to be made as to which one to use, i.e. a coin has to be flipped to decide which of these two variants will be operational. A partially mapped crossover (PMX, see Michalewicz and Fogel (2000)) is slightly modified and applied for that purpose.

cross_var1

In subsection 4.1.2 the solution representation for 2D problems was represented with item characteristics that are part of the encoding, i.e. the x-coordinate of the

reference vertex and the orientation of the item. The *cross_var1* allows for the orientation, x-coordinate of the reference vertex components of the solution to be directly inherited from one parent. The ordering of the items is then achieved through breeding between both parents. Let $\overrightarrow{X_{p1}}$ and $\overrightarrow{X_{p2}}$ be two parent solutions representing n items and let O_1 be the offspring that results out of the breeding of the parents. For example consider the following situation where
$\overrightarrow{X_{p1}} =$
$[(SPP, 2, 2, F), \{(0, 6, 0^0), (9, 3, 90^0), (5, 2, 90^0), (2, 4, 0^0), (10, 5, 90^0), (3, 1, 90^0)\}]$
and
$\overrightarrow{X_{p2}} = [(SPP, 2, 2, F), \{(1, 1, 90^0), (5, 2, 90^0), (4, 3, 0^0), (6, 5, 90^0), (2, 4, 0^0), (8, 6, 0^0)\}]$
. We need to arrange efficiently a layout of 6 items.

The *cross_var1* works as follows:

1. Copy the x co-ordinate of the reference vertex and orientation of every item from solution $\overrightarrow{X_{p2}}$ to the offspring O_1 :

   In this example, at this stage the offspring becomes
   $O_1 = [(SPP, 2, 2, F), \{(1, X, 90^0), (5, X, 90^0), (4, X, 0^0), (6, X, 90^0), (2, X, 0^0), (8, X, 0^0)\}]$,
   (the symbol 'X' can be interpreted as "at present unknown").

2. Generate two random positions $p1$ and $p2$, such that $1 \leq p_1 < p_2 \leq n$:

   For example say p_1 is generated to be 4 and p_2 to be 5.

3. Create a one to one mapping between item indexes in positions p_1-p_2 from both parents:

The series of mappings for this example is:

$5 \leftrightarrow 4,\ 4 \leftrightarrow 5$

4. Copy every item index between positions p_1 and $p2$ from $\overrightarrow{X_{p2}}$ to O_1 to corresponding positions:

   After the copying the offspring becomes

   $O_1 = [(SPP, 2, 2, F), \{(1, X, 90^0), (5, X, 90^0), (4, X, 0^0), (6, 5, 90^0), (2, 4, 0^0), (8, X, 0^0)\}]$

5. For every item index from $\overrightarrow{X_{p1}}$ not in O_1 is copied with its position in the order to O_1 starting from the leftmost index to the right most excluding p_1-p_2 positions, conflict is avoided by making use of the mapping in stage 3:

   The final solution becomes:

   $O_1 = [(SPP, 2, 2, F), \{(1, 6, 90^0), (5, 3, 90^0), (4, 2, 0^0), (6, 5, 90^0), (2, 4, 0^0), (8, 1, 0^0)\}]$

From the description of this variant of crossover it should be obvious that, there is a possibility that infeasible solutions might be introduced into the population. To counteract this a penalty function is used to degrade the quality of infeasible solutions, more about this in section 4.5.

cross_var2

The major difference between these variants of crossover is that *cross_var1* allows a situation where breeding involves both item characteristics in the solution representation and the ordering of the items for the packing. *cross_var2* on the other hand is mainly concerned with the ordering of the items without separating the item characteristics and the ordering, i.e. when items change positions in the ordering , the item moves with the characteristics that define it. In other words the whole 3-tuple (x_k, i_k, ϕ_k) moves. The parents used to demonstrate *cross_var1* will again be used to demonstrate *cross_var2*. *cross_var2* breeds offspring as follows:

1. Generate two random positions $p1$ and $p2$, such that $1 \leq p_1 < p_2 \leq n$:

   For example, say the random process results in $p_1 = 3$ and $p_2 = 5$.

2. Create a one-to-one mapping of item indexes from both solutions in positions $p_1 - p_2$:

   For this example that would be:

   $2 \leftrightarrow 3$, $4 \leftrightarrow 5$ and $5 \leftrightarrow 4$.

3. Copy from parent $\overrightarrow{X_{p1}}$ items in position $p_1 - p_2$ with their related characteristics:

   This results in a partial offspring solution, which is

   $O_2 = [(SPP, 2, 2, F), \{(X, X, X), (X, X, X), (5, 2, 90^0), (2, 4, 0^0), (10, 5, 90^0), ((X, X, X)\}]$

   ,(the symbol 'X' can be interpreted as "at present unknown").

4. We complete the solution by copying items from parent $\overrightarrow{X_{p2}}$ starting from left to right excluding those items in positions $p_1 - p_2$ and try and avoid conflict by using the mapping in stage 2 :

   The resulting offspring finally is:

   $O_2 = [(SPP, 2, 2, F), \{(1, 1, 90^0), (9, 3, 90^0), (5, 2, 90^0), (2, 4, 0^0), (10, 5, 90^0), ((8, 6, 0^0)\}]$

4.4.2.2 2D Mutation Operator

The 2-swap mutation operator which is usually made use of in sequencing problems (see Michalewicz and Fogel (2000) for the TSP example) has been adapted and modified as the mutation operator for 2D problems. The operator works as follows:

1. Randomly choose two items $item1$ and $item2$.
2. Randomly generate a number $num \in \{0, 1\}$ to decide if the orientation of the chosen items will be randomly perturbated.
3. If $num = 1$ change the orientation of both items randomly (This applies if more than one orientation is allowed).
4. Exchange the position of $item1$ with that of $item2$.

4.5 Fitness Function

The fitness function is the mechanism used to judge the quality of the evolved solutions. The general fitness function can be summarised in equation 4.1.

$$
max\, f(X) = \begin{cases}
f_1(X) \quad if\, problemcode = \begin{cases} (BPP, 1, *, *) \\ (CSP, 1, *, *) \end{cases} \\
f_2(X) \quad if\, problemcode = \begin{cases} (SPP,\ 2,\ 2,\ F) \\ (SPP,\ 2,\ 1,\ F) \end{cases} \\
f_3(X) \quad if\, problemcode = \begin{cases} (SPP, 2, 2, G) \\ (SPP, 2, 1, G) \end{cases} \\
f_4(X) \quad if\, problemcode = \begin{cases} (BPP, 2, 2, F) \\ (BPP, 2, 1, F) \end{cases} \\
f_5(X) \quad if\, problemcode = \begin{cases} (BPP, 2, 2, G) \\ (BPP, 2, 1, G) \end{cases} \\
f_6(X) \quad if\, problemcode = \begin{cases} (ISPP, 2, 1, *) \\ (ISPP, 2, 2, *) \\ (ISPP, 2, 4, *) \end{cases}
\end{cases} \tag{4.1}
$$

4.5.1 Evaluation of one dimensional problems

Function f_1 is the fitness function for one dimensional problems, which is defined below.

Let eff_i be the measure of efficiency of bin i, let N be a total number of bins used and C be bin capacity and w_i be the weight of item i.

$$eff_k = \frac{\sum_{i \in Bin\,k} w_i}{C}$$

$$f_1 = \frac{\sum_{i=1}^{N} eff_i}{N} \tag{4.2}$$

What this means is the most efficient use of bins gets rewarded most, i.e. the algorithm seeks to maximise f_1.

4.5.2 Evaluation of nonguillotine-able 2D Strip packing problems

For the evaluation of the 2D strip packing problems a placement heuristic is made use of, which considers one item at a time. Items are placed on the strip in the order in which they appear in the solution string. The function f_2 is the evaluation of the 2D strip packing problem in which guillotine cutting is not a requirement. The placement heuristic for function f_2 works as follows:

For each item k the following steps are carried out in turn:

1. Item i_k is placed at the topmost position at horizontal position x_k, with the orientation of item k being that reflected by ϕ_k.

2. Item i_k is slid as far down as possible, until it collides with either the bottom edge of the strip or another item.

3. Item i_k is slid as far left as possible until it collides with another item or the left edge of the strip. This becomes the final position of the item i_k.

To demonstrate the above heuristic consider the solution

$\overrightarrow{X} =$

$[(SPP, 2, 2, F), \{(0, 6, 0^0), (9, 3, 90^0), (5, 2, 90^0), (2, 4, 0^0), (10, 5, 90^0), (15, 1, 90^0)\}]$

consisting of 6 rectangular items to be packed on strip whose width is 20 units. The item dimensions are given in table 4.1. Figure 4.3 shows how solution $\overrightarrow{X}$ would be decoded using the placement heuristic discussed above.

Item (i_k)	Height	Width
1	2	12
2	7	12
3	8	6
4	3	6
5	5	5
6	3	12

Table 4.1: Item dimensions example

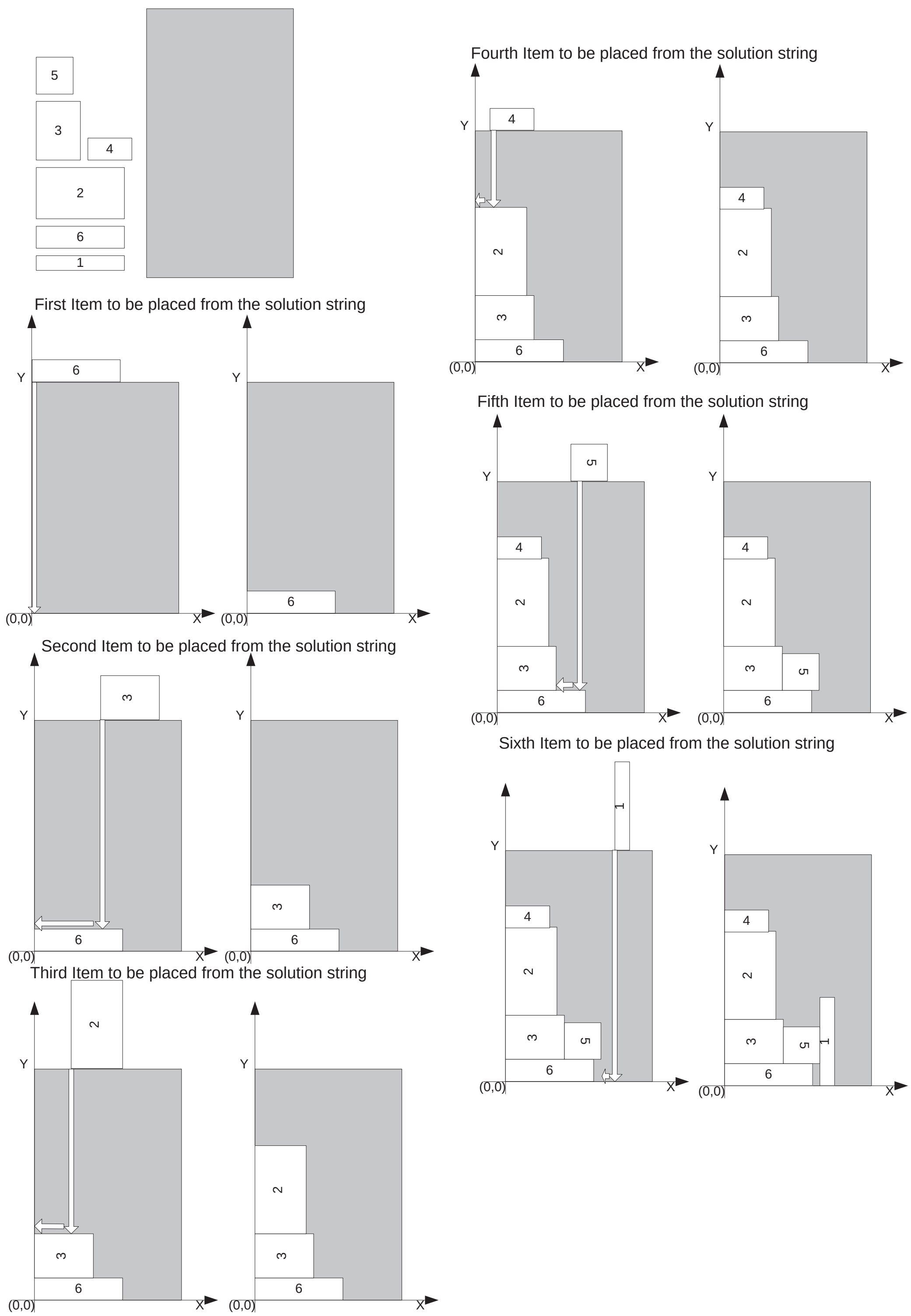


Figure 4.3: Placement-Heuristic Example for Strip packing problem without guillotine cutting.

In section 3.1 it is mentioned that the search space $\mathcal{S}$ consist of two subsets the feasible part $\mathcal{F} \subseteq \mathcal{S}$ and the infeasible part $\mathcal{U} \subseteq \mathcal{S}$. In the discussion on cross-over operator for 2D problems in sub subsection 4.4.2.1 it is mentioned that *cross_var1* can introduce infeasible solutions into the population. There are two possible violations of constraints that can occur in 2D problems, viz. overlap constraint, containment constraint. It is the violation of the latter constraint that *cross_var1* is guilty of, i.e placement of items outside the borders of the strip. In the design of the fitness function for 2D problems this has to be taken into account. Taking this into account the fitness function for 2D strip packing problems is given by:

$$f_2(X) = \begin{cases} P_2(X) & if\ X \in \mathcal{U} \\ Eff(X) & if\ X \in \mathcal{F} \end{cases}$$

$P_2(X)$ is a penalty function used as a constraint handling mechanism. Any solution in violation of the above mentioned constraint is "killed", i.e. the solution is made undesirable. $Eff(X)$ measures the efficiency of the packing. The total area of items to be packed is given by

$$A = \Sigma_{i=1}^{n} w_i h_i \tag{4.3}$$

Ideally the total area of the strip occupied by the items is supposed to be A , but in most instances this is not the case. A is a continuous lower bound for every instance I of this problem. Let A_p be the area that results after all items have been placed on the strip. A_p is given by

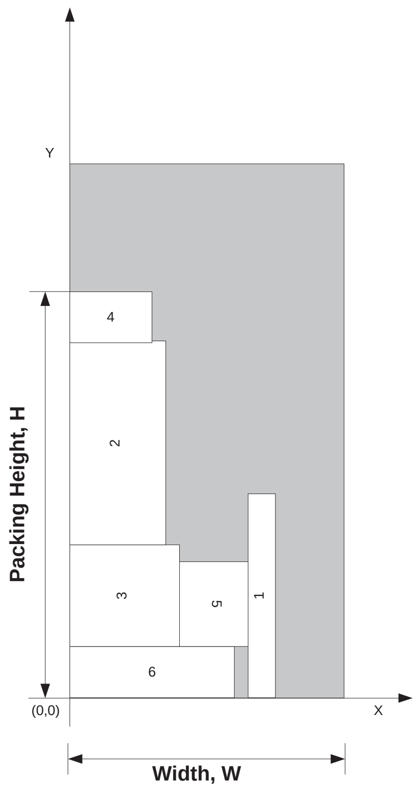


Figure 4.4: Example of packing Height

$$A_p = h_p W \tag{4.4}$$

where h_p is the packing height see figure 4.4 for an example of packing height.

$$Eff(X) = \frac{A}{A_p} \tag{4.5}$$

$Eff(X)$ reflects the efficient use of the strip, i.e. those individuals in the population that use the strip efficiently are rewarded the most. The lowest packing height possible is given by

$$h_L = \frac{A}{W} \tag{4.6}$$

It is desirable that an individuals packing height be close as possible to this height. To make infeasible solutions undesirable we move them as further from this bound by a factor K, such that we choose a penalty packing height $h_{penalty}$, where $h_{penalty} = Kh_L \gg h_L$.

$$P_2(X) = \frac{A}{Wh_{penalty}} \tag{4.7}$$

4.5.3 Evaluation of guillotine-able 2D Strip Packing Problems

The function f_3 is the fitness function for strip packing problems with guillotine constraint. This function is also evaluated by means of a placement heuristic, the only difference is the guillotine cutting constraint should be taken into account when placing items. Items are placed such that the guillotine constraint is never violated. An observation that is of great help when placing items on the strip with guillotine cutting required, is that guillotine cutting subdivides the strip into blocks whose top edge and bottom edge is parallel to the bottom edge of the strip, see figure 4.5 for illustration. Blocks consist of rectangular items and waisted space.

The placement heuristic to evaluate guillotine packing is similar to the placement heuristic for f_2 explained above. The only difference is taking the guillotine constraint into consideration. The placement heuristic works as follows:

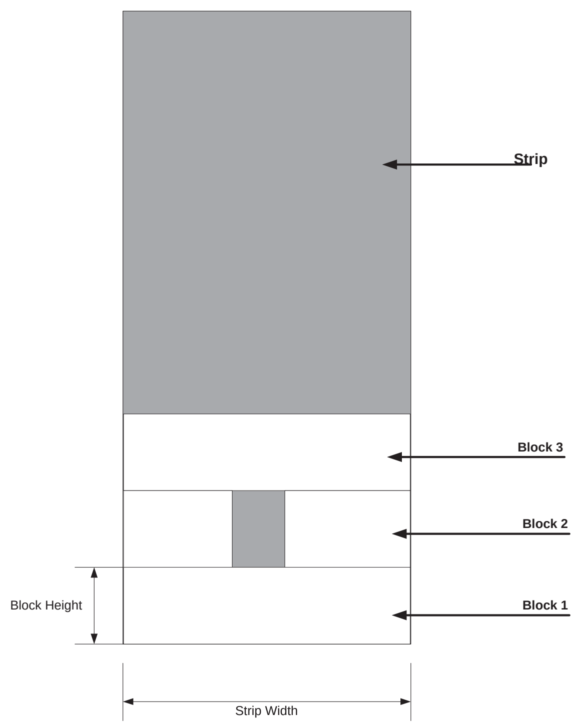


Figure 4.5: An example of Guillotine Block Packing

For every item i_k as sequenced by the solution string the following steps are carried out:

1. Item i_k is placed at the topmost position at horizontal position x_k, with the orientation of item k being that reflected by ϕ_k.

2. Item i_k is slid as far down as possible, until the item either collides with the bottom horizontal edge of the strip or collides with another item.

3. Item i_k is checked if it is in violation of the guillotine constraint, if it is Item i_k' s vertical position is corrected to satisfy the guillotine constraint.

4. Item i_k is shifted as far left as possible until it collides with another item or the vertical left edge of the strip.

The fitness function f_3 is also given by

$$f_3(X) = \begin{cases} P_3(X) & if \ X \in \mathcal{U} \\ F_3(X) & if \ X \in \mathcal{F} \end{cases}$$

$P_3(X)$ is a penalty function used to "kill" infeasible solutions and is worked out as $P_2(X)$. $F_3(X)$ is a function whose purpose is to value efficiently packed blocks and efficiently packed overall layout.

$$F_3(X) = Eff(X) + qB(X) \tag{4.8}$$

,

$0 < q \leq 1$

q is a weighting determined by the user to value the the efficiently packed block.

where $Eff(X)$ is as described in equation 4.5. Let Eff_{Bi} be an efficiency of block B_i, and H_{Bi} be the height of block B_i. Let A_i be the area of rectangle r_i. Let N_{Blocks} be the total number of blocks in a layout.

$$Eff_{Bk} = \frac{\Sigma_{r_k \in B_k} A_k}{WH_{Bk}} \tag{4.9}$$

Then $B(X)$ is given by

$$B(X) = \frac{\Sigma_{i=1}^{N_{Blocks}} Eff_{Bi}}{N_{Blocks}} \tag{4.10}$$

4.5.4 Fitness function for 2D Bin packing problems

For 2D bin packing problems both the guillotine-able and the nonguillotine-able versions, the placement heuristic presented in the section above is still used as a decoder. The strip packing problem above can be looked at as a problem where one needs to pack small rectangular items to a single open ended bin and 2D bin packing problem as the problem where small rectangular items have to be packed to multiple identical bins. The approach taken in this work is to partition the strip to an infinite number of identical bins. Below more details about this process are provided.

4.5.4.1 Evaluation of nonguillotine-able 2D Bin Packing Problems

For the evaluation of these problems we take the strip packing approach presented in section 4.5.2. The placement heuristic which gives the fitness function f_4 is as described below:

For each item i_k as sequenced by the solution string the following steps are carried out in turn:

1. Item i_k is placed at the topmost position at horizontal position x_k, with the orientation of item i_k being reflected by ϕ_k.

2. Item i_k is slid as far down as possible until it collides with the bottom edge of the strip or collides with another item. If item i_k is the first item then the first bin is opened and stays open until all items are placed.

3. If item i_k is not the first item, then item i_k has collided with an item in some bin k already opened, item i_k is checked if it can be wholly contained in bin k. If item i_k can not be wholly contained by bin k item i_k is placed in bin $k+1$ which is immediately on top of bin k if no such bin exists a new one, bin $k+1$ is opened.

4. After the final vertical position of item i_k is decided upon in some bin, item i_k is shifted as far to the left as possible until it collides with either the vertical left edge of the strip or some other item in the same bin. This becomes the final position for item i_k.

To illustrate this heuristic consider the following example. Suppose we have bins of dimensions 100×100. with items of the following sizes in table 4.2.Consider the following individual to be decoded by the above placement heuristic explained above.

$\overrightarrow{X} =$

$[(BPP, 2, 2, F), \{(0, 6, 0^0), (9, 3, 90^0), (5, 2, 90^0), (48, 4, 0^0), (10, 5, 90^0), (15, 1, 90^0)\}],$

Figures 4.6 to 4.10 illustrate how $\overrightarrow{X}$ is decoded by the placement heuristic for the 2D bin packing problem.

Item (i_k)	Height	Width
1	25	7
2	27	47
3	24	13
4	34	48
5	1	21
6	93	76

Table 4.2: Items dimensions for 2D bin packing problem

Figure 4.6: Placement of the first item

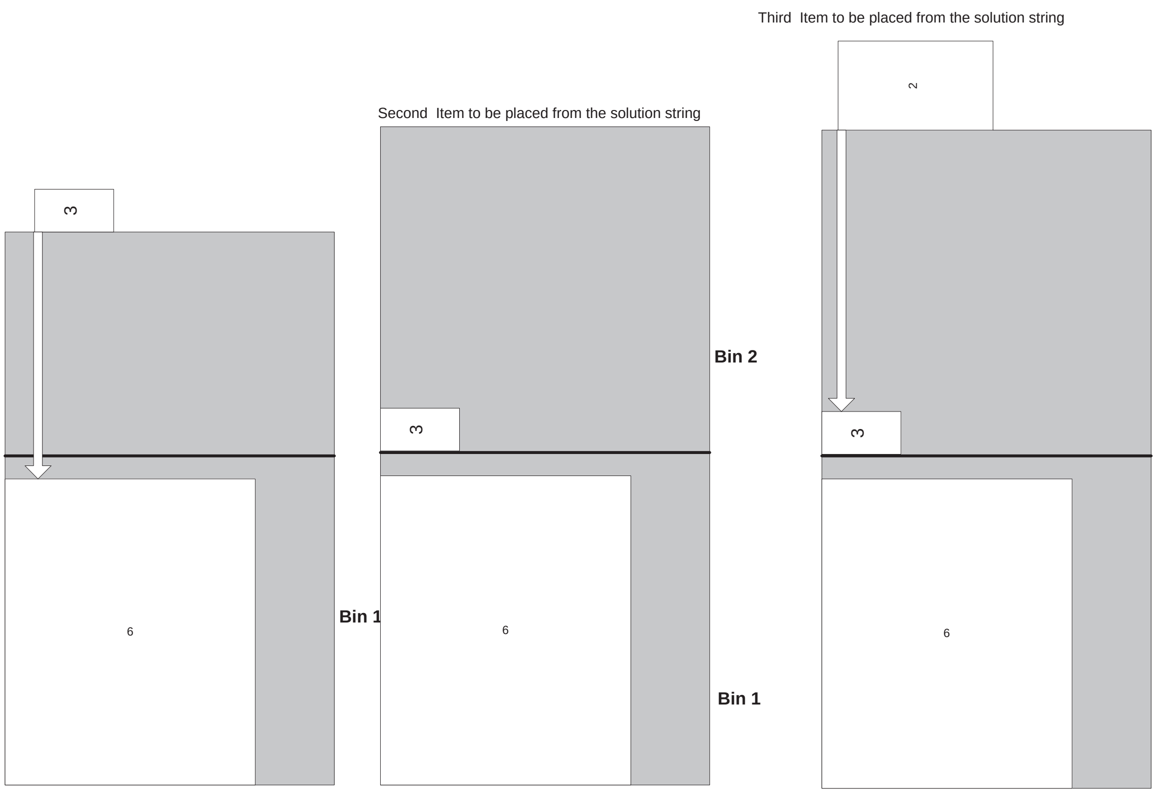


Figure 4.7: Placement of the second and third items

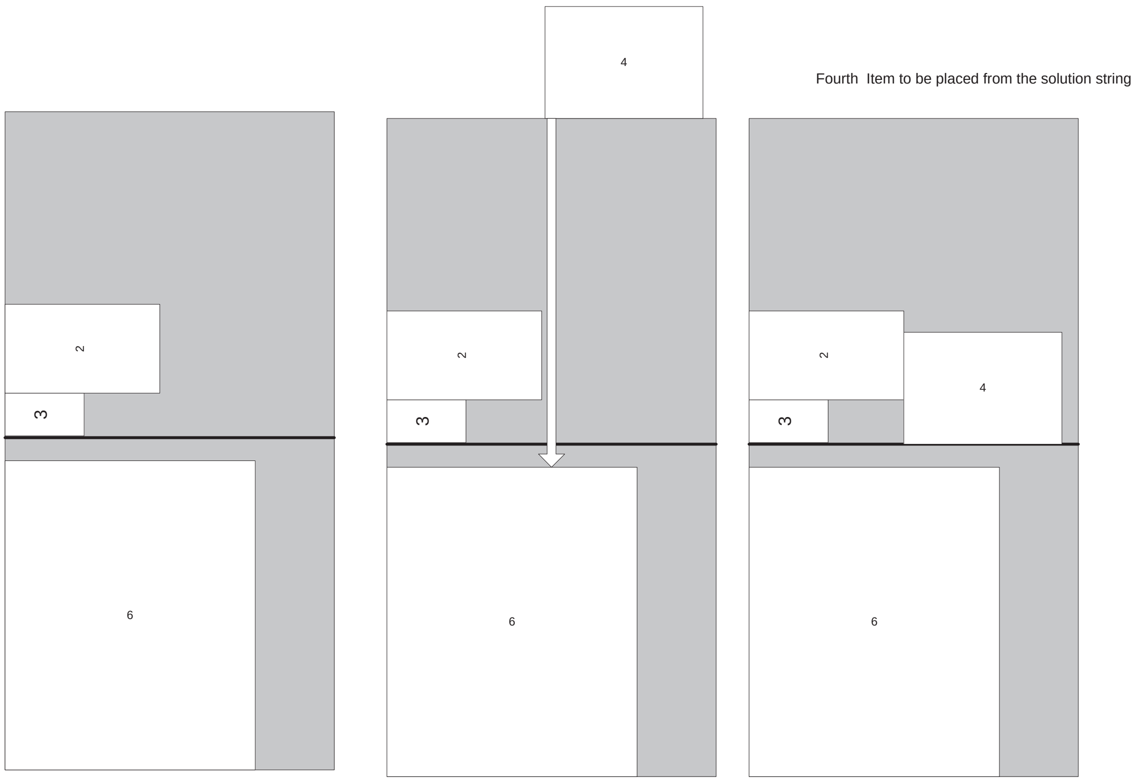


Figure 4.8: Placement of the fourth item

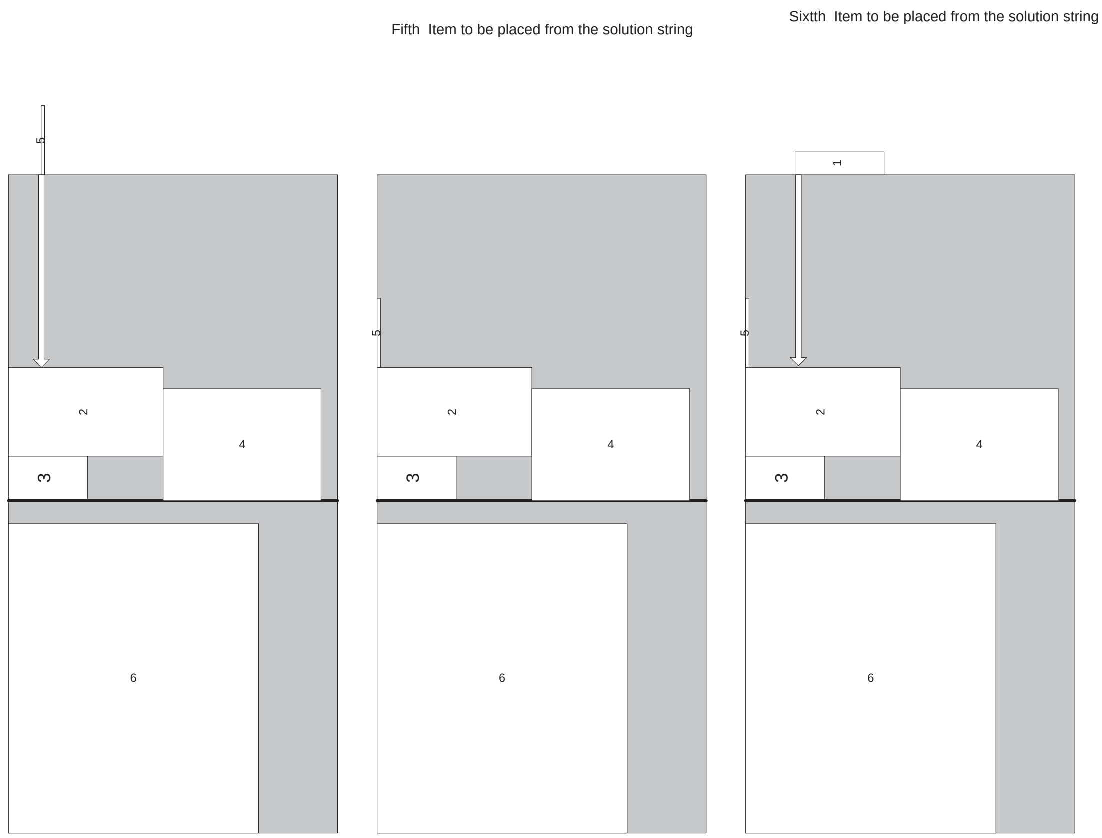


Figure 4.9: Fifth and sixth item to be placed on the solution string.

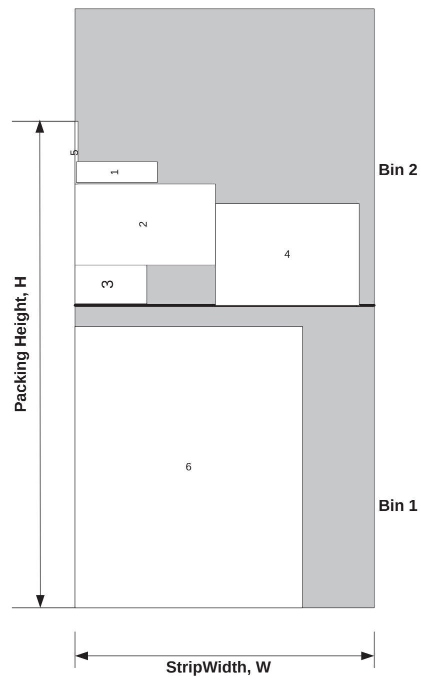


Figure 4.10: The complete layout represented by $\overrightarrow{X}$

$$f_4(X) = \begin{cases} P_4(X) & if\ X \in \mathcal{U} \\ F_4(X) & if\ X \in \mathcal{F} \end{cases} \tag{4.11}$$

where P_4 is the penalty function to penalise those individuals that place items outside the borders of the bins, and F_4 is the function that evaluates the packing efficiency of the bins and the packing efficiency of the strip that they partition. As indicated before the penalty function is a mechanism used to make infeasible individuals look unattractive. Let Eff_{Bink} be the efficiency of the kth bin, and let A_i be the area of rectangle r_i and $W$and H be the bin width and height respectively.

$$Eff_{Bink} = \frac{\Sigma_{A_i \in Bink} A_i}{HW}$$

The most inefficient assignment for problem instance I would result if every bin $Bink$ was assigned a single item from the list of items. The total number of bins used for the packing would be equal to the number of items n. The set of efficiencies E, would consist of n elements i.e

$E = \{Eff_{Bin1}, Eff_{Bin2}, \ldots, Eff_{Binn}\}$ which can be expressed as
$E = \{\frac{A_1}{HW}, \frac{A_2}{HW}, \ldots, \frac{A_n}{HW}\}$ The penalty function P_4 is given by

$$P_4(X) = min\{E\} \tag{4.12}$$

That is in an attempt to degrade infeasible solutions, they are all assigned the worse possible efficiency for problem instance I. Let N_{Bin} be the number of bins used in the layout. Let $BE(X)$ be the average bin efficiency for the entire layout, i.e.

$$BE(X) = \frac{\Sigma_{i=1}^{N_{Bin}} Eff_{Bini}}{N_{Bin}} \tag{4.13}$$

The function $F_4(X)$ to evaluate feasible individuals is given by:

$$F_4(X) = BE(X) + qEff(X) \tag{4.14}$$

where $Eff(X)$ is as described in equation 4.5, $0 < q < 1$.

4.5.4.2 Evaluation of guillotine-able 2D Bin Packing Problems

Evaluation of the two dimensional bin packing problem with a guillotine constraint is evaluated by means of the same heuristic that has been presented in this work. To be precise in section 4.5.2 the evaluation placement heuristic for 2D guillotine-able strip packing problems is presented. The same heuristic is used with one modification, i.e. a limit is put on the size of the strip, i.e the bin height becomes the height. The fitness function, f_5 for the guillotine-able bin packing problem is similar to the ones presented previously and is given by:

$$f_5 = \begin{cases} P_5(X) & if\ X \in \mathcal{U} \\ F_5(X) & if\ X \in \mathcal{F} \end{cases} \tag{4.15}$$

P_5 is a penalty function, whose purpose is similar to other penalty functions presented previously. P_5 is computed in the same way as P_4 in equation 4.12 above. Let Eff_k be the efficiency of the bin with items arranged with a guillotine pattern for bin k and let A_k be the area of item k. Let W and H be the width and height of the bin respectively. Let N_{bin} be the total number of bins used in the layout.

$$Eff_k = \frac{\Sigma_{i \in bin\, k} A_i}{HW}$$

$$F_5(X) = \frac{\Sigma_{i=1}^{N_{bin}} Eff_i}{N_{bin}} \tag{4.16}$$

4.5.5 Evaluation of 2D Irregular Strip packing Problems

In sub subsection 2.4.2.3, the 2D irregular strip packing problem was defined as a problem where one is given a set I consisting of n 2-D pieces that have arbitrary irregular shapes. These shapes have to be packed on a strip of constant width and height assumed to be infinite. This problem is prevalent in one form or another in industries such as the textile industries, shoe-leather cutting, furniture industry, aerospace industries and machine building. The approach taken in this work is to approximate the irregular 2-D pieces with polygons. This therefore means we have a set of pieces P of polygons, $P = \{P_1, P_2, \ldots, P_n\}$. The objective is to place the pieces in P on a strip such that the packing height is minimised, i.e. efficient utilisation of area. The fitness function f_6 for the 2D Irregular Strip packing Problems is also achieved by means of a placement heuristic which is very similar to the placement heuristic used in the problems above, with one minor variation, because of the arbitrariness of the geometry of the pieces the shift left stage is not part of this placement heuristic. The placement heuristic is as follows:

For each item i_k as sequenced by the solution string the following steps are carried out in turn:

1. Item i_k is placed at the topmost position at horizontal position x_k, with the orientation of item i_k being that reflected by ϕ_k.

2. Item i_k is slid as far down as possible, until it collides with either the bottom edge of the strip or another item. This becomes the final position of the piece i_k.

To illustrate this heuristic consider the solution

$$\overrightarrow{X} = [(ISPP, 2, 2, *), \{(4,1,0^0),(6,4,0^0),(2,6,0^0),(6,3,0^0),(0,2,0^0),(5,5,0^0)\}],$$

Figures 4.11 to 4.14 illustrate how the above solution can be decoded.

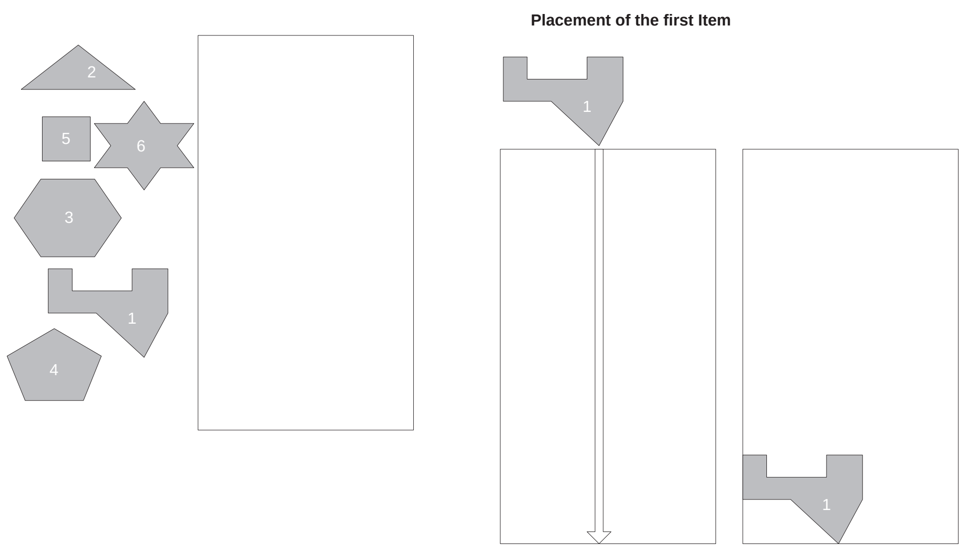


Figure 4.11: A List of Items to be placed and placement of Item1

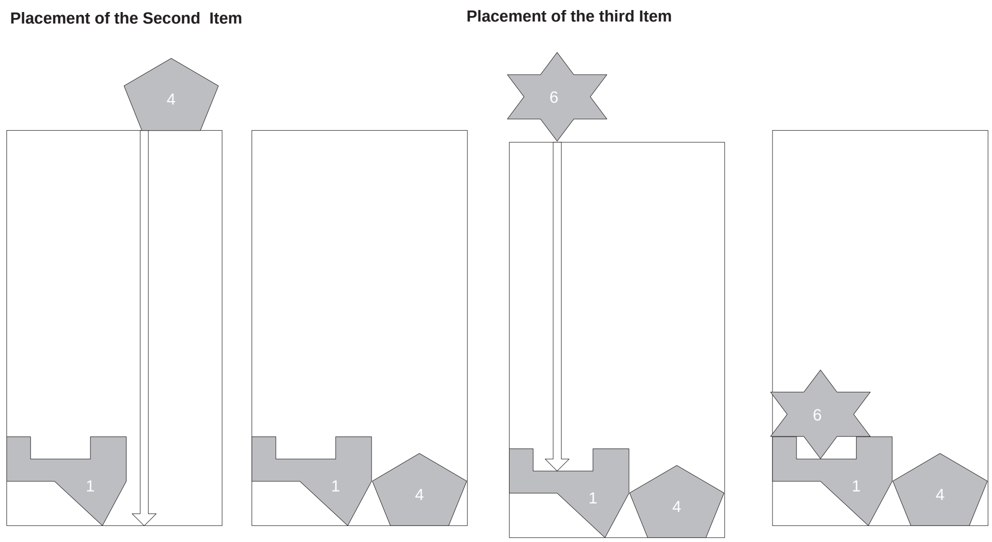


Figure 4.12: Placement of Item 4 and Item 6

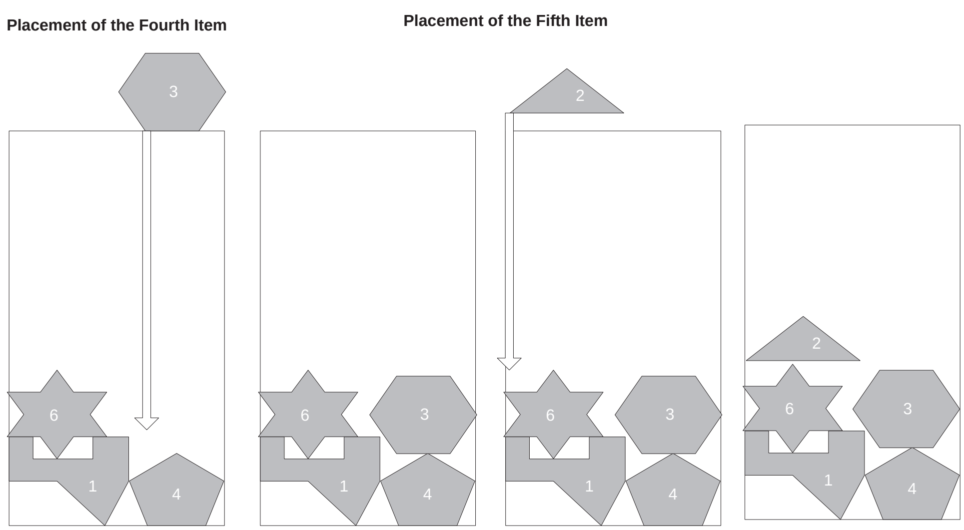


Figure 4.13: Placement of Item 3 and Item 2

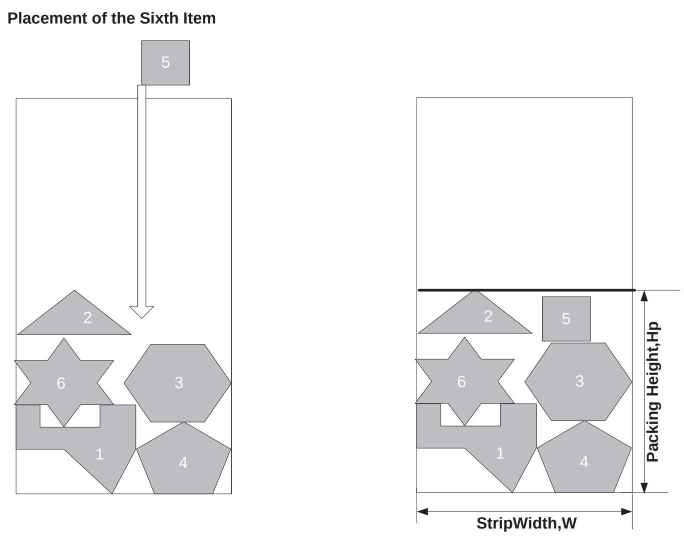


Figure 4.14: Placement of the last Item

The fitness function f_6 of the irregular strip packing problem is

$$f_6(X) = \begin{cases} P_6(X) & if\ X \in \mathcal{U} \\ F_6(X) & if\ X \in \mathcal{F} \end{cases} \tag{4.17}$$

Let A_k be the area of piece P_k. The total area A of the pieces is

$$A = \Sigma_{i=1}^{n} A_i$$

Let h_p be the packing height , The packing area A_p is

$$A_p = h_p W$$

where W is the width of the strip. Function F_6 is given by

$$F_6(X) = \frac{A}{A_p}$$

The penalty function P_6 for this problem is computed in the same way as penalty function P_2 in equation 4.7.

4.6 Summary

In this chapter the general genetic algorithm is explained and the general solution representation which functions as a template solution encoding for all problems looked at in this work. All the aspects of the general genetic algorithm are also presented.

Chapter 5

Implementation Issues

A general genetic algorithm was presented in chapter 4, where the general solution representation for all problems dealt with in this work was presented. The variation operators were also presented and the general fitness function, how various solutions get evaluated depending on the problem was also presented. In this chapter implementation issues are looked at. The algorithm presented in chapter 4 was implemented in MATLAB. MATLAB's genetic algorithm and direct search toolbox was used for the running of the genetic algorithm. A CD accompanying this document has all the necessary functions for the general genetic algorithm.

5.1 Computational Geometry

The two dimensional problems mainly consist of two dimensional small items to be assigned to two dimensional large objects. At this stage it would be proper to consider how to represent polygonal geometric objects (Recall rectangles are four sided convex polygons)

A survey on computational Geometry will not be offered in this work, as most of it is outside the scope of this work. To represent geometric structures it is necessary to be able to represent the most fundamental component of geometric object, i.e. the point. The textbooks represent the point in one way or another. For example Sedgewick (1992) represents a point as a C++ struct, O'Rourke (1998) represents a point as a typedef of an integer. Usually a cartesian co-ordinate representation of a point is used. One of the primitive operations needed for a point is to rotate it. Let (x, y) be the co-ordinates of a point P, Hearn and Baker (1997) have shown that if point P is rotated by angle θ about the origin. The new co-ordinates of the rotated point $(x^{'}, y^{'})$ are given by

$$\begin{bmatrix} x^{'} \\ y^{'} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{5.1}$$

To rotate a point (x, y) about any arbitrary point (x_r, y_r), by angle θ the co-ordinates $(x^{'}, y^{'})$ of the rotated point are given by

$$x^{'} = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$$

$$y^{'} = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta \tag{5.2}$$

One of the most common requirements is the rotation of a polygon about one of its vertices, p_v. Each vertex of the polygon has to be rotated except p_v. From what has been explained above this is reduced to rotating a set of points. An arbitrary polygon is generally represented by a list of points to represent each vertex [see O'Rourke (1998), Sedgewick (1992)]. The last vertex in the list is assumed to be connected to the first. The vertices are usually ordered in counterclockwise order. Now that the polygon representation has been defined there are fundamental operations that need to be performed on a polygon. These operations are usually referred to as primitive functions. One of the important primitive is to calculate the area of a triangle, given its vertices. An algorithm to do this is presented by O'Rourke (1998). Knowing how to work out the area of a triangle from the list of its vertices, from this an area of a polygon can be calculated. To do this an arbitrary vertex on the polygon is chosen and joined to all other vertices on the polygon forming triangles. The polygon area is now reduced to summing the area of the triangles. This procedure can be used to calculate the area of both convex and non-convex polygons. Another useful primitive is what is referred to as the left predicate, which is used to decide a relationship between three points, to illustrate this consider figure 5.1, if we want to determine the relationship between points $P1$, $P2$ and $P3$, i.e is $P3$ left of the line segment $\overrightarrow{P1P2}$ or on the right or collinear with points $P1$ and $P2$. The left predicate calculates the area of the triangle formed by the three points. If the area is positive it indicates that $P3$ is to the left of the line segment $\overrightarrow{P1P2}$. If the area is zero , the three points are collinear. If the area is negative it shows that $P3$ is to the right of $\overrightarrow{P1P2}$.

Another important primitive is given two line segments (where a line segment is represented by two vertices) $\overrightarrow{\mathbf{a}}$ and $\overrightarrow{b}$ on a plane, do the segments intersect?

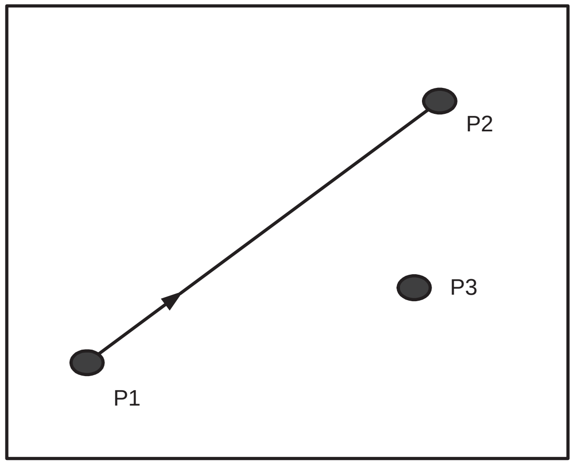


Figure 5.1: Left Predicate

Figure 5.2 shows an example of situations where the intersection between line segments has to be determined. O'Rourke (1998) describes an algorithm to do just this and states that in this situation it is necessary to leave the comfortable world of integer co-ordinates and return to the floating point values of representing the x and y-coordinates of the point of intersection.

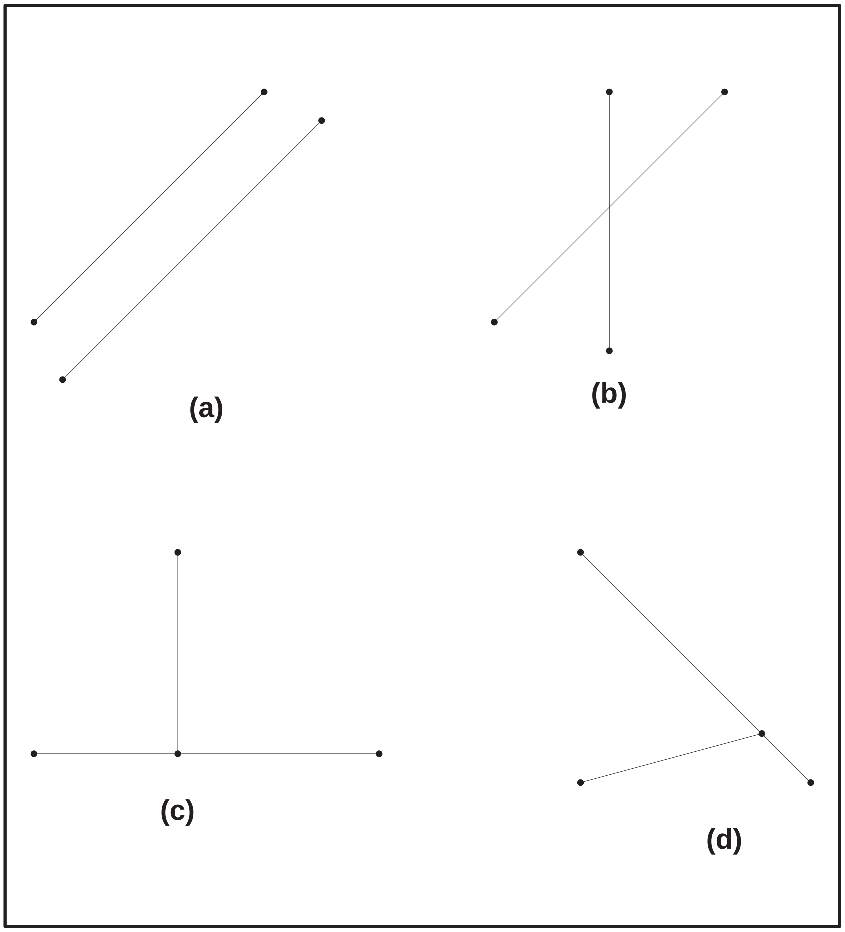


Figure 5.2: Line segment intersection

Now that the primitive operations on the polygon have been defined, higher level operations can now also be defined. Another important primitive in the area of cutting and packing is finding the convex hull of a polygon. A convex hull can be defined as follows:

Given a set P_p of points $P_p = \{p_1, p_2, \ldots p_n\}$ find a minimum convex polygon P_C that can enclose the points in the interior of the polygon. Figure 5.3 shows an

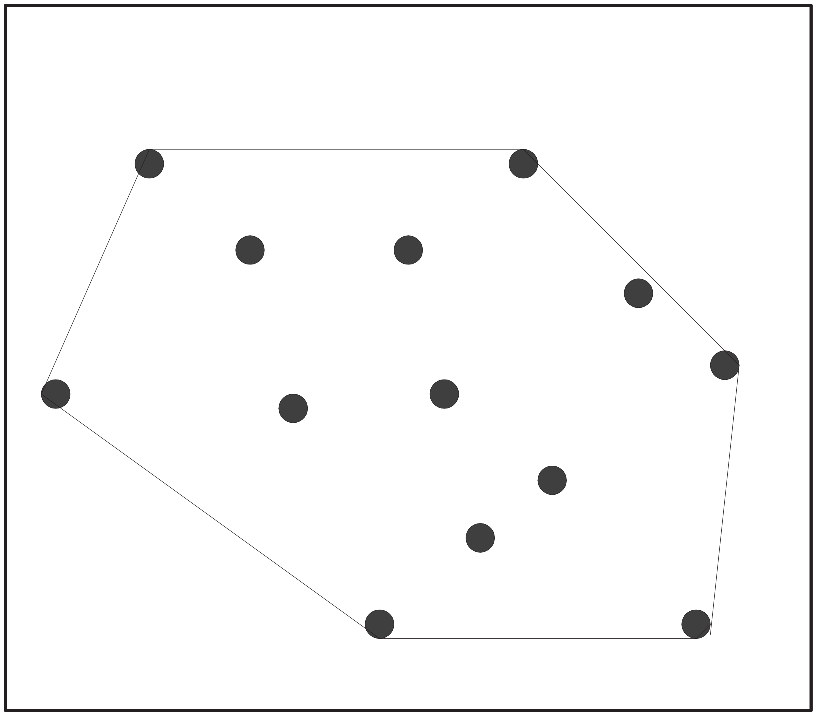


Figure 5.3: Convex hull of a set of points

example of a convex hull of a set of points.

The convex hull of a polygon P is a minimum convex polygon that can be wrapped around vertices of the polygon P. This can be visualised as stretching an elastic band around a shape . Figure 5.4 shows the convex hull of polygon $P1$, the convex hull, represented by the dashed line has been made slightly larger for illustration.

There are a number of algorithms available for the calculation of the convex hull. For more details on algorithms for the calculation of the convex hull see O'Rourke (1998); Preparata and Shamos (1985). For all the two dimensional problems in the

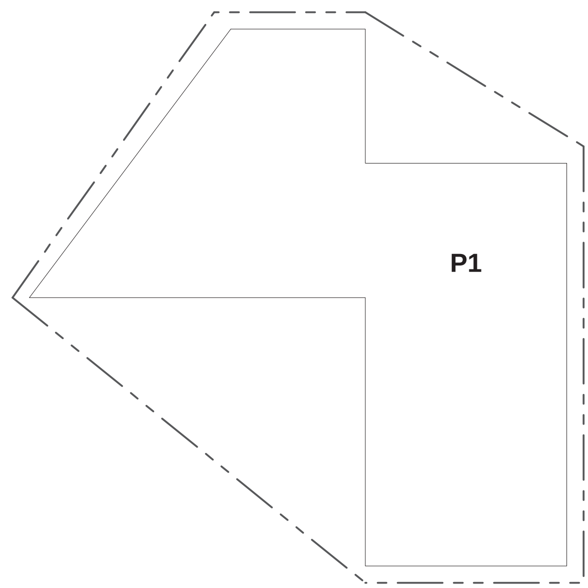


Figure 5.4: Convex Hull of P1, shown with a dashed edges

field of cutting and packing, a very important constraint is that the small items should be allocated to the big objects without overlap. A very crucial operation in cutting and packing is to decide if two arbitrary polygons intersect or overlap. The detection of overlap can be done in two ways depending on the requirements. The first method is just to return a boolean indicating if overlap has occurred or not. The second is more complex, but sometimes useful operation, is to return the polygon that represents the intersection area. In addition to this a consideration is to be made for convex and non-convex polygons. This operation is a lot more complex for non-convex polygons. The simplest way to go about this task, though not necessarily the best, is to use brute force, test each and every edge of a polygon against every edge of the other polygon. However one degenerate case has to be taken into account, i.e. if one polygon totally includes another. In O'Rourke (1998) an algorithm is presented to detect intersection between two convex polygons. The algorithm involves two lines "chasing" each other around the edges of the polygons and plotting points as the heads of the lines are advanced. At termination the algorithm returns the overlap area. This method is also described in Preparata and Shamos (1985). The implementation of algorithms for intersection detection is a very delicate process. For example if one polygon is touching another polygon should an intersection be reported? How to handle a situation when polygons intersect only at the vertices?

5.2 Representation of the Problems

The mathematical definition of problems dealt with in this work is given in section 2.4. In this section the representation of these problems in MATLAB is presented.

5.2.1 Representation of One-dimensional problems

The one dimensional bin packing problem was presented as a set of items J each item having a positive weight or size $w_i\,(i \in J)$. It is required to partition set J into a minimum number of subsets (bins), so that the sum of items in each bin does not exceed the given capacity C. A one dimensional bin packing problem is represented by the following Matlab structure. The **Problemtype** field is assigned a string "1BPP", this field is mainly used in distinguishing one problem from another when generating the initial population.

OneBinPacking Problem{

Width: //Bin Capacity

Items: // $1 \times n$ Vector for Item sizes

worse_eff: // worse possible assignment

Problemtype: // A string representing the problem type description

}

An example of a Matlab structure representing a one dimensional bin packing problem is shown in figure 5.5.

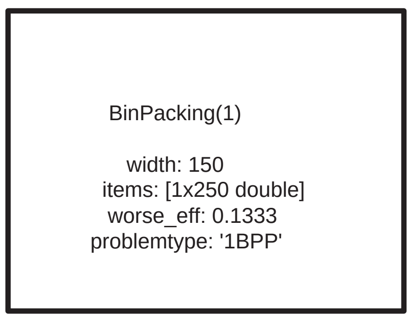


Figure 5.5: A Matlab structure showing the one-dimensional bin packing problem

5.2.2 Representation of Two-dimensional Problems

The common factor in all two dimensional problems is that a set consisting of two dimensional items (polygons or rectangles). These items have to be placed into a set containing two dimensional objects (regions). The object(s) set might consist of a single element e.g. strip packing problem or multiple elements e.g. two dimensional bin packing problem. Before the representation of the problem is presented, the

Representation of small items will be presented first.

Representation of a point

To represent a point $P(x, y)$ a 1×2 vector $[x\, y]$ is used. For example to represent a line segment we need two points $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$. A 2×2 square matrix $A = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix}$ would be used to represent a line segment.

Representing Polygons

A polygon is a region of a plane bounded by line segments forming a simple closed curve. An alternative definition would be as follows.

Let v_0, v_1, $v_2, \ldots, v_{n-1}$ be n points on a plane. The points are ordered cyclical, i.e. v_0 follows v_{n-1}. Let $e_0 = v_0v_1, \ldots, e_i = v_iv_{i+1}$, $e_{n-1} = v_{n-1}v_0$ be n segments connecting the points. Then these segments bound a polygon iff

1. The intersection of each pair of segments adjacent in the cyclic ordering is the single point shared between them: $e_i \cap e_{i+1} = v_{i+1}$
2. Non adjacent segments do not intersect: $e_i \cap e_j = \emptyset$ for all $j \neq i + 1$.

The points v_i are known as *vertices* and the segments e_i are called the *edges*.

This definition is that of a simple polygon, non simple polygons do not fulfill the above stated conditions. In tis work only simple polygons will be worked with as we have little use for nonsimple polygons. To implement the above definition of a polygon an $n \times 2$ matrix is used where n is the number of vertices of the polygon. For example a rectangle is a four sided convex polygon the following matrix would represent a rectangle.

$$polygon = \begin{bmatrix} x_0 & y_0 \\ x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \end{bmatrix}$$

Where the entries of the matrix are vertices of the four corners of the rectangle, figure 5.6 illustrates this further.

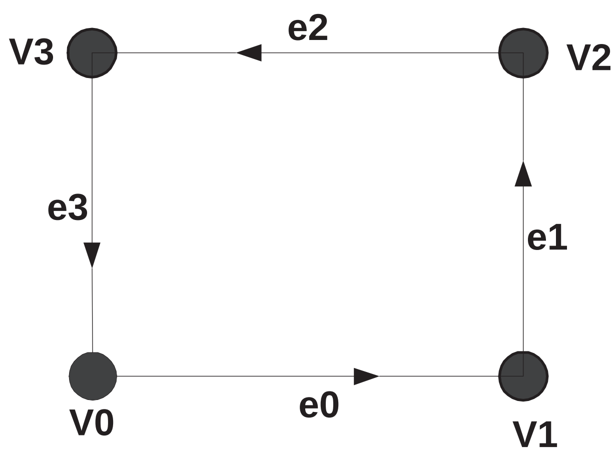


Figure 5.6: Representation of a rectangle

To fully represent a polygon the following Matlab structure is used:

Polygon{

polygon; // Matrix representing the polygon

Refpoint; // Reference point of *polygon*

Vertex_no; // Vertex number of **Refpoint**

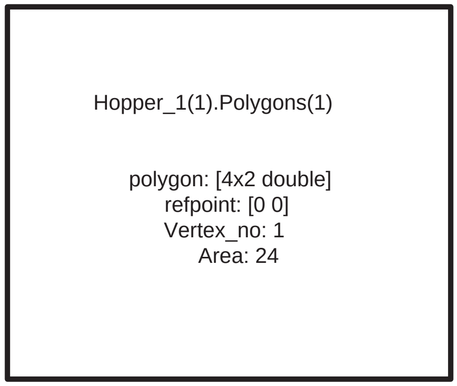


Figure 5.7: A Matlab structure for a polygon

Area; // Area of a polygon

}

An example of a polygon structure is shown in figure 5.7.

5.2.3 Representation of two dimensional bin packing problems

In sub subsection 2.4.2.1 four variants of the two dimensional bin packing problem are presented each variant represented by a unique problem code, in this section the Representation of these problems is presented.

Problem type (BPP,2,1,F)

To represent this problem type the following Matlab structure is used:

Problem BPP_2_1_F{

Width; // Bin width

Height; // Bin Height

StartHeight;//Initial Vertical position for each rectangle

Polygon[n]; // An array of Polygon structures

Orientations; // Number of feasible orientations=1

Worse_eff; // Worse possible efficiency (penalty function)

Problemtype; //A string representing the problem type

}

The **StartHeight** field is the initial vertical position for each rectangle. This position is worked out to be 10 times the continuous lower bound, i.e.

$$StartHeight = 10(\left\lceil \frac{\Sigma_{i=1}^{n} w_i h_i}{W} \right\rceil)$$

Problem type (BPP,2,2,F)

To implement this problem a structure similar to the one presented above is made use of. The only difference is the size of the **Polygon[n]** array and the number of feasible orientations, because rectangles in this problem can be rotated by 90^0 the **Orientations** field has the value 2. The size of the **Polygon[n]** array doubles, i.e it becomes **Polygon[2n]**. Where for every odd entry in the array n represents the 0^0 orientation of rectangle r_i and every even entry $2n$ represents the 90^0 rotation of the rectangle r_i.

The structure for this type of problem is:

Problem BPP_2_2_F{

Width; // Bin width

Height; // Bin Height

StartHeight; //Initial Vertical position for each rectangle

Polygon[2n]; // An array of Polygon structures

Orientations; // Number of feasible orientations=2

Worse_eff; // Worse possible efficiency (penalty function)

Problemtype; //A string representing the problem type

}

The guillotine-able versions of this problem are implemented analogously. The **Problemtype** field is assigned a string "2DBPP-G" to distinguish the problems.

5.2.4 Representation of two dimensional strip packing problems

To Represent the variants of the strip packing problem the following Matlab structure is used:

Strippacking Problem{

Width; // Strip width

Polygon[Orientations*n];// An array of polygon structures

LB_Area; // Total Area of rectangles

Orientations; // Number of feasible orientations

StartHeight; //Initial Vertical position for each rectangle

Problemtype; //A string representing the problem type

}

The above structure changes depending which variant of this problem is being solved. The size of the **Polygon[Orientations*n]** array depends on the number of feasible orientations for each rectangle. An example of the Representation of problem (SPP,2,2,F) is given in figure 5.8

```
Hopper_1(1)

        width: 20
     Polygons: [1x32 struct]
      LB_Area: 400
  startheight: 2000
 orientations: 2
  problemtype: '2DSPP-F'
```

Figure 5.8: A Matlab structure for the two dimensional strip packing problem

5.2.5 Representation of two dimensional Irregular strip packing problems

The Representation of the two dimensional irregular strip packing problem is similar to the strip packing in almost every respect, the difference is in the number of feasible orientations. In the irregular problem, because the shapes of the small items are irregular and arbitrary the number of possible feasible orientations can be very large. In this work the largest number of feasible orientations for this problem is 4. The two dimensional irregular strip packing problem is represented by the following structure:

Irregular Strippacking Problem{

Width; // Strip width

Polygon[Orientations*n];// An array of polygon structures

LB_Area; // Total Area of Polygons

Orientations; // Number of feasible orientations

StartHeight; //Initial Vertical position for each polygon

Problemtype; //A string representing the problem type

}

An example of a Matlab structure representing the two dimensional irregular problem is shown in figure 5.9.

```
marques

        width: 104
     Polygons: [1x96 struct]
      LB_Area: 7194
  startheight: 691.7308
 orientations: 4
  problemtype: '2DISPP'
```

Figure 5.9: A Matlab structure for a two dimensional irregular strip packing problem with 4 feasible orientations

5.3 Implementing the solution representation

In section 4.1 a general solution representation is introduced it is stated that the solution consists of two parts *problem code* and *problem specific encoding*. The general solution representation is

$\overrightarrow{X} = [(\mathbf{P}, \mathbf{D}, \mathbf{O}, \mathbf{C}), \{(x_1, i_1, \phi_1), (x_2, i_2, \phi_2), \ldots, (x_n, i_n, \phi_n)\}]$.

In this section the representation of the general solution is presented.

Representation of problem code

The problem code is represented by a 1×4 vector

$$PC = \begin{bmatrix} \mathbf{P} & \mathbf{D} & \mathbf{O} & \mathbf{C} \end{bmatrix}$$

Table 5.1 shows values that **P** can assume and table 5.2 shows values that **C** can assume.

P	Problem Type
1	Strip Packing problem
2	Bin Packing problem
3	Irregular strip packing problem

Table 5.1: P Values

C	Cutting Constraint
0	Not applicable (*)
1	Guillotine Cutting Constraint (G)
2	Free Cutting (F)

Table 5.2: C Values

Representation of problem specific encoding

A $3 \times n$ matrix is used to represent the problem specific encoding, where n is the number of items. an example of the matrix is shown below

$$PSE = \begin{bmatrix} x_1 & x_2 & . & . & . & x_n \\ i_1 & i_2 & . & . & . & i_n \\ \phi_1 & \phi_2 & . & . & . & \phi_n \end{bmatrix}$$

For one dimensional problems the orientation entries ϕ_k are blanks, 0 is used to represent blanks.

5.4 Initial Population Generation

The generation of the initial population of solutions for every problem has been implemented in the same Matlab function M-file called *CreatePopulationgeneration*. The code and commentry for the M-file function is shown in figure B.27.

5.5 Crossover Operator

The crossover operator for both 1D and 2D problems was implemented in a single Matlab M-file function, *generalxover*.Matlab code for the function is shown in figure B.28.

5.6 Slide and collision detection algorithm

For all two dimensional problems prior the evaluation of a solution, for every item the following operation has to be carried out. An item is slid as far down as possible until it collides with either the bottom edge of the container or collides with another item. In this section an algorithm to achieve this is presented. Let L be a list of n polygons items to be placed to some large object. Let P_{PL} be a set of items placed already and P_c be a candidate polygon to be placed. Let P_{PC} be a set of polygons placed already that are in the collision path of polygon P_c. The algorithm is shown in algorithm 5.

Algorithm 5 Slide &Collision Detection Algorithm

For $i = 1$ to n
- $P_c = L(i)$
- Place P_c at position $(x_i, P_c.startheight)$
- Select $P_{PC} \subseteq P_{PL}$
- Find the highest Vertex $V_{ymax}(x, y) \in P_{PC}$
- $Y_{max} = V_{max}(y)$
- Place P_c at position (x_i, Y_{max})
- Use binary search to place P_c as near as possible to the highest polygon in P_{PC}

End

To further explain how set P_{PC} is selected consider the situation in figure 5.10. The candidate polygon P_c is P_5, the set $P_{PL} = \{P_1, P_4, P_6, P_3, P_2\}$. The polygons in the collision path of P_5 are in set $P_{PC} = \{P_4, P_3\}$.

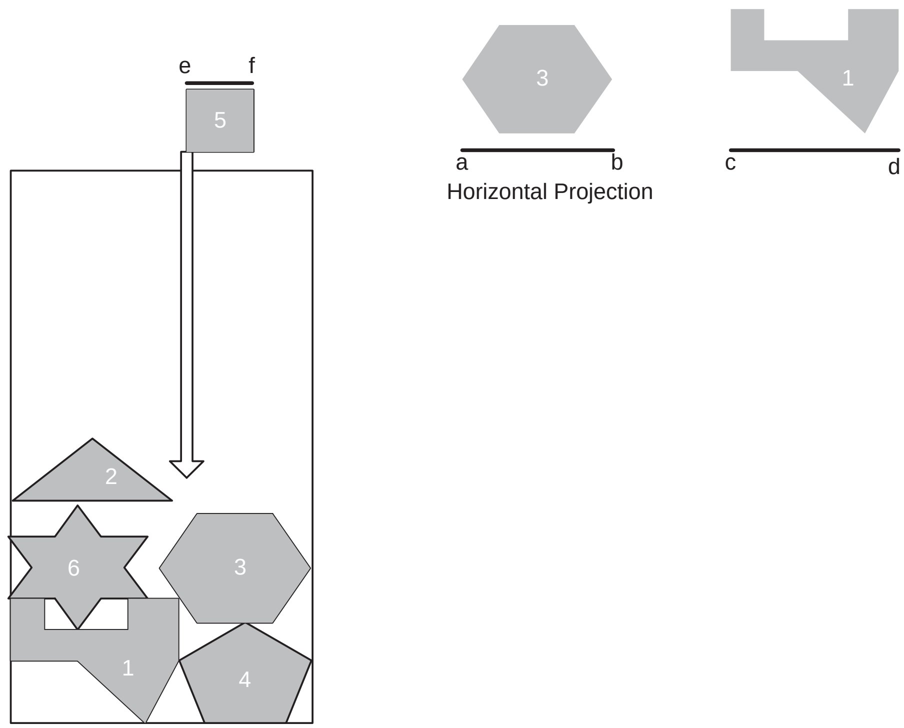
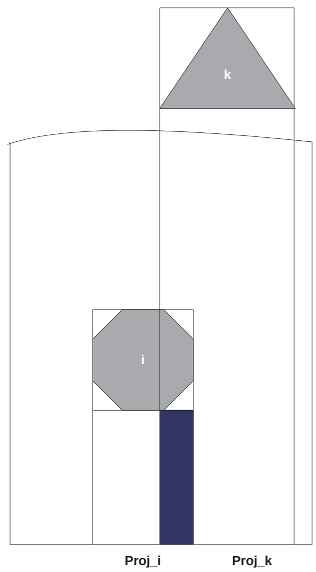


Figure 5.10: Placement of P_c

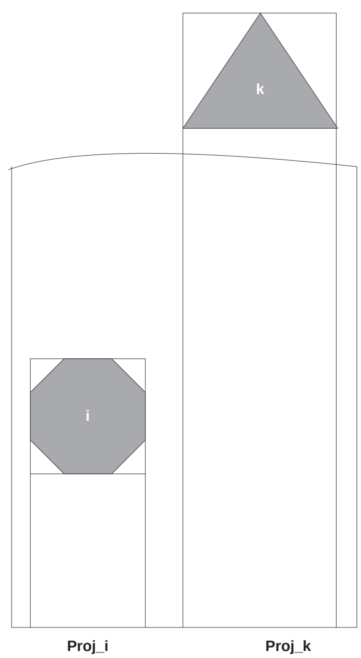
To select P_{PC} from P_{PL} one alternative would be to continually test for overlap on all polygons in P_{PL} as P_c is being slid downwards, but this would be inefficient. The approach taken in this work is an observation that if polygon i is in collision path of polygon k then there will be an overlap between the horizontal projection of polygon i, $proj_i$ and the horizontal projection of polygon k, $proj_k$, i.e

$$proj_i \cap proj_k \neq \emptyset$$

An example of this situation is shown in figure 5.11.



(a) Projection Overlap



(b) Projections non-Overlap

Figure 5.11: Overlap of Horizontal Projections

5.7 The Fitness Function

The general fitness function stated in equation 4.1 has been implemented in a Matlab m-file function *Eval_function.* The code for the function is shown in figure B.29.

5.8 Summary

In this chapter implementation details have been offered. A short review of geometry is offered. Matlab structure has been used to represent both the one and two dimensional problems that have to be solved. The representation of geometric properties for small items is discussed. The implementation of variation operators and sample m-file code is also presented. The slide and collision algorithm has also been presented.

Chapter 6

Computational Experiments

In order to evaluate the performance of the general GA presented in this work problem instances have been collected from literature. All experiments were conducted on a 3.4 GHz Pentium 4 processor. The algorithm was coded in Matlab and run using Matlab's genetic algorithm and direct search toolbox. For every problem the population size was set at 100 individuals, although this tended to slow down the speed of the algorithm. The GA was run for 2000 generations for every problem. After repeated runs for most problems it was decided that the crossover fraction should be between 0.3-0.45, the crossover fraction was kept at 0.3 for all problems. With 2 individual spots in the population reserved for elite children, i.e. two best individuals in every generation. A tournament of size 2 was used as a selection criteria. The stopping criteria was for the algorithm to run for 2000 generations if the best fitness does not improve after 1000 generations the algorithm stops or if the best fitness does not improve after 2000 seconds the algorithm stops. Almost all datasets used in this work can be downloaded from ESICUP (Euro special interest group on cutting and packing) home site (`http://www.apdio.pt/sicup/`).

6.1 Results for 1D problems

6.1.1 1D Bin Packing Problem

Problem datasets for the 1D bin packing problem are considered in E.Falkenauer (1996). The problems instances generated in Falkenauer's work consisted of two classes. The first class consist of integer item sizes uniformly distributed between 20 and 100, with bin capacity being 150. The second class consisted of items ranging from 25 to 50 in size with bin capacity of 100. The experiments were conducted on 19 problem instances of the class1 and 20 problem instances of the class2. The problem datasets are listed in appendix A in section A.1. The results of the experiments are listed in tables 6.1 and 6.2. For each problem tables A and A.1 gives.

- *Problem number*
- The theoretical minimum number of bins (Continuous Lower bound) $LB = \left\lceil \frac{\Sigma_{i=1}^{n} w_i}{C} \right\rceil$.
- *Time*=The entire period of time from start to when the algorithm stopped.
- z= The number of bins returned by the algorithm.

- *PR*= Performance ratio, a ratio of the solution returned by the general algorithm z over the lower bound LB given by $PR = \frac{z}{LB}$.

Problem Number	*LB*	*Time(s)*	*z*	*PR*
1	99	1147.5	104	1.05
2	100	1128.4	104	1.04
3	102	1123.5	107	1.05
4	100	1097.7	104	1.04
5	101	1153.4	105	1.04
6	101	1223.1	106	1.05
7	103	1200.7	107	1.04
8	105	1226.9	110	1.05
9	101	1202.5	105	1.04
10	105	1376.7	110	1.05
11	101	1646.8	106	1.05
12	105	1692.8	110	1.05
13	101	1299.6	106	1.05
14	99	1132.6	104	1.05
15	105	1188.5	110	1.05
16	97	1078	102	1.05
17	100	1087	104	1.04
18	100	1094.7	105	1.05
19	102	1106.7	107	1.05

Table 6.1: Class1 Results

Problem Number	*LB*	*Time(s)*	*z*	*PR*
1	21	256.7	22	1.05
2	21	245.11	21	1
3	21	244.9	21	1
4	20	246.07	22	1.1
5	21	249.68	21	1
6	21	241.08	21	1
7	21	250.98	22	1.05
8	20	280.58	22	1.1
9	20	245.54	21	1.05
10	21	281.4	22	1.05
11	20	249.56	22	1.1
12	20	254.27	22	1.1
13	20	258.42	21	1.05
14	21	253.25	22	1.05
15	21	284.04	22	1.05
16	20	251.03	22	1.1
17	21	250.68	22	1.05
18	21	425.05	21	1
19	20	257.25	22	1.1
20	20	398.47	22	1.1

Table 6.2: Class2 Results

6.1.2 1D Cutting Stock Problem

The test problems that were used for this problem type are considered in Hinterding and Khan (1995). The total items requested range from 20 to 126. For more details about problems see Appendix A, section A.2. The results for this problem type are listed in table 6.3. The table lists the *Problem Number*, *LB* the theoretical minimum number of bins that can be used which is given by $\left\lceil \frac{\Sigma_{i=1}^{n} d_i l_i}{L} \right\rceil$. The Stocks obtained by the algorithm z, the execution time and the performance ratio PR .

Problem Number	*LB*	*z*	*Time(s)*	*PR*
1	9	9	138.48	1
2	23	23	265.57	1
3	16	16	196.22	1
4	20	20	259.69	1
5	54	54	528.49	1

Table 6.3: 1D CSP results

6.2 Results for 2D strip packing problems

6.2.1 Results for NonGuillotine-able Problems

The results considered here are that for the variant (SPP,2,1,F), i.e the variant where the small rectangles can not be rotated. The test problems used in this work are considered in Martello et al. (2003). The problems consist of 38 problems collected from various sources. The number of items to be packed ranges from 10 to 200. These test problem can also be downloaded from ESICUP home site. The results for this problem type are shown in table 6.4. For each problem table 6.4 gives:

- Problem number and values of n (number of rectangles)
- LB the lower bound.
- z Best solution found by the general Genetic Algorithm.
- Total search time $Time$.
- PR the performance ratio.

An example of a layout generated by the general Genetic Algorithm is shown in figure 6.1, the layouts for the first 27 problems in table 6.4 are shown in appendix B, section B.1.

Problem no.	*n*	*LB*	*z*	*Time(s)*	*PR*
1	16	20	23	3581.9	1.15
2	17	20	23	2212.5	1.15
3	16	20	23	2026.4	1.15
4	25	15	18	2121.8	1.2
5	25	15	18	2042.3	1.2
6	25	15	17	2967.1	1.13
7	28	30	36	2150.5	1.2
8	29	30	37	3166.7	1.23
9	28	30	39	2333.9	1.3
10	16	23	25	2204.3	1.08
11	23	63	72	2653.6	1.14
12	62	636	730	8011.7	1.15
13	10	1016	1016	1105.1	1
14	20	1133	1215	4028.5	1.07
15	30	1803	1866	3818.7	1.03
16	50	2934	3340	5658.2	1.138
17	10	23	23	1165.5	1
18	17	30	30	2082	1
19	21	28	31	2494.6	1.11
20	7	20	20	869.82	1
21	14	36	36	2245.6	1
22	15	31	35	1699.9	1.13
23	8	20	20	971.27	1
24	13	33	34	3009.8	1.03
25	18	49	56	3192.1	1.14
26	13	80	80	1754.3	1
27	15	52	61	1959.3	1.17
28	22	87	87	2246.9	1
29	20	30	34	2383.6	1.13
30	40	57	65	2623.9	1.14
31	60	84	100	4363	1.19
32	80	107	130	4703.2	1.21
33	100	134	167	4086.9	1.25
34	40	36	44	3121.2	1.22
35	80	67	85	3001.1	1.27
36	120	101	133	4858.7	1.32
37	160	126	160	6903.4	1.27
38	200	156	209	4352.7	1.34

Table 6.4: Strip Packing Problem (SPP,2,1,F) results

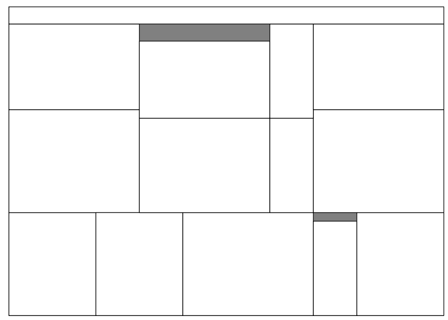


Figure 6.1: Layout example

The test problems for the (SPP,2,2,F) variant are taken from Hopper and Turton (2001). The test data consists of 21 problems presented in seven different sized categories (each category has three different problems of similar size and object dimension). These test problems are very difficult to solve as they are "perfect packings" obtained by cutting a given rectangle of fixed dimensions into smaller rectangular items. Table 6.5 shows results obtained by the general genetic algorithm. For problems in each category table 6.5 gives:

- Problem categories C1-C7
- The problem size n.
- The Optimum height of the packing for each test problem.
- The height obtained by the general algorithm z.
- The search time taken by the algorithm.

- *PR* the performance ratio.

Category	Problem	n	Optimum Heights	z	*Time(s)*	*PR*
C1	P1	16	20	22	2587.2	1.1
	P2	17	20	23	2112.5	1.15
	P3	16	20	23	2346.3	1.15
C2	P1	25	15	19	2207.4	1.27
	P2	25	15	19	2211.7	1.27
	P3	25	15	19	2525.4	1.27
C3	P1	28	30	36	3045.9	1.2
	P2	29	30	34	3173.8	1.13
	P3	28	30	36	2496.2	1.2
C4	P1	49	60	70	10122	1.17
	P2	49	60	72	3136.3	1.2
	P3	49	60	75	2661.9	1.25
C5	P1	73	90	117	2567.4	1.3
	P2	73	90	124	3764.8	1.38
	P3	73	90	109	8170.9	1.21
C6	P1	97	120	159	3796.5	1.33
	P2	97	120	160	3422.1	1.33
	P3	97	120	160	3387.5	1.33
C7	P1	196	240	330	6249.2	1.38
	P2	197	240	346	10911	1.44
	P3	196	240	352	5294.4	1.47

Table 6.5: Strip Packing Problem results where items can be rotated by 90^0

A layout example for one of these problems is shown in figure 6.2, the layouts for problems in categories C1-C5 are shown in appendix B, section

B.2.

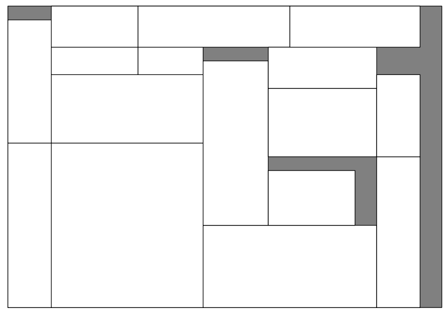


Figure 6.2: An example of layout for nonguillotine-able problems with 90^0 rotations

6.2.2 Results for Guillotine-able Strip packing problems

The test problems for guillotine-able strip packing are contributed by Hopper and Turton in 2002, The data have been generated from a 200×200 square which is a "perfect packing". The dataset consists of 35 problems in all but a subset of these has been solved in this work. The problem sizes range from 17 to 199 items, the results are displayed in table 6.6. The table gives the following results:

- The problem size n.
- The Optimum height of the packing for each test problem
- The height obtained by the general algorithm z.
- PR the performance ratio.

problem	*n*	*Optimum height*	*z*	*time(s)*	*PR*
1	17	200	259	5583.5	1.29
2	17	200	257	5017.5	1.28
3	17	200	239	13583	1.19
4	17	200	244	19428	1.22
5	17	200	242	23941	1.21
6	25	200	250	34453	1.25
7	29	200	299	3009.8	1.5
8	49	200	333	4202.8	1.665
9	73	200	375	4306.3	1.875
10	97	200	343	3192.3	1.715

Table 6.6: The results for guillotine-able strip packing problem where the rectangles can be rotated

The sample layouts for the problems in table 6.6 are shown in appendix B, section B.3.

6.3 Results for 2D Bin Packing Problem

The test problems used in this work have been adopted from Lodi et al. (1999). These test problems are featured in website `http://www.or.deis.unibo.it/research_pages/ORinstances/ORinstances.htm`. The test problems consist mainly of 10 classes the first six classes have been generated with he following properties:

Class1: w_j and h_j uniformly random in [1,10], W=H=10;

Class2: w_j and h_j uniformly random in [1,10], W=H=30;

Class3: w_j and h_j uniformly random in [1,35], W=H=40;

Class4: w_j and h_j uniformly random in [1,35], W=H=100;

Class5: w_j and h_j uniformly random in [1,100], W=H=100;

Class6: w_j and h_j uniformly random in [1,100], W=H=300;

The following four classes of problems were generated in the following manner: Items belong to one of four types:

Type 1: w_j uniformly random in $[\frac{2}{3}W, W]$,h_j uniformly random in $[1, \frac{1}{2}H]$;

Type 2: w_j uniformly random in $[1, \frac{1}{2}W]$,h_j uniformly random in $[\frac{2}{3}H, H]$;

Type 3: w_j uniformly random in $[\frac{1}{2}W, W]$,h_j uniformly random in $[\frac{1}{2}H, H]$;

Type 4: w_j uniformly random in $[1, \frac{1}{2}W]$,h_j uniformly random in $[1, \frac{1}{2}H]$;

For the 4 remaining classes W=H=100 , while items are as follows:

Class 7: type 1 with probability 70%, type 2, 3, 4 with probability 10% each.

Class 8: type 2 with probability 70%, type 1, 3, 4 with probability 10% each;

Class 9: type 3 with probability 70%, type 1, 2, 4 with probability 10% each;

Class10: type 4 with probability 70%, type 1, 2, 3 with probability 10% each;

Each class has five values of n: 20, 40, 60, 80, 100. For each class and value of n, ten instances were generated. In total there is a set of 500 test problems for each class.

Since this work was conducted under limited resources, it was decided to test our algorithm on class 1 for the (BPP,2,1,F) variant of the problem. The results are

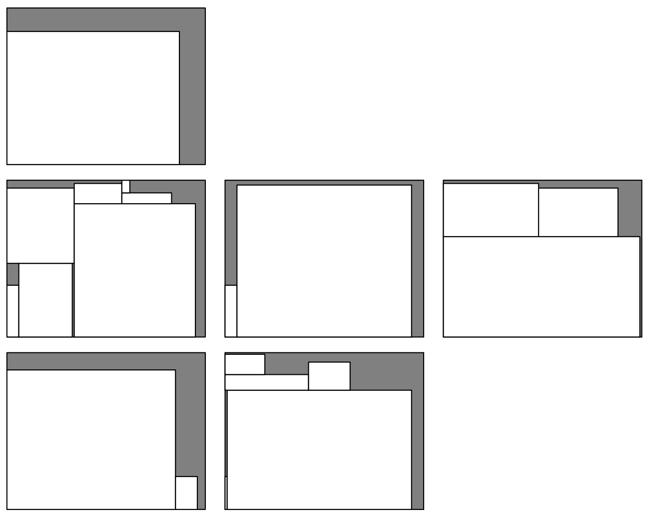


Figure 6.3: Layout for the two dimensional bin packing problem

shown in table 6.7. Table 6.7 gives averages of the performance ratio, PR of the solution by the general genetic algorithm z to the lower Bound $L_o = \frac{\Sigma_{i=1}^{n} w_i h_i}{WH}$, for problems of size n. The average total execution time for each problem size. Figure 6.3 shows one of the layouts genereted by this algorithm.

n	PR	Time(s)
20	1.2283	3278
40	1.3588	5543.7
60	1.3874	8974.1
80	1.3820	13543
100	1.368	19314

Table 6.7: 2D Bin Packing with free cutting and fixed orientation results

Sample layouts for this problem are shown in appendix B, section B.4. The computational results for the problem variant (BPP,2,2,F) are shown in table 6.8.

n	PR	Time(s)
20	1.208	2816.8
40	1.309	8285.9
60	1.3864	15153
80	1.3447	14629
100	1.4308	23996

Table 6.8: 2D Bin Packing with free cutting and where rectangles can be rotated

The computational results for the guilloteneable variant of this problem are shown in table 6.9.

n	PR	Time(s)
20	1.2083	716.9584
40	1.3095	1775.2
60	1.3864	2609.7
80	1.3447	3973.2
100	1.4308	2946

Table 6.9: Results for guillotine-able Bin Packing Problems

6.4 Results for 2D Irregular strip packing problem

To test the general genetic algorithm on this type of problem, test problems for this type of problem which are also featured in ESICUP website were used. Four test problems were used to test this algorithm all derived from the textile industry. These problem details are listed in table 6.10. The table shows the literature source that brought the problem to the attention of the academic community. The problem name, the number of shapes that have to be packed and the sheet width and the orientation constraints for each problem.

Problem Source	Problem Name	Shapes	Sheet Width	Rotational Constraints
Oliveira et al. (2000)	Shirts	99	40	0,180 Absolute
Oliveira et al. (2000)	Trousers	64	79	0,180 Absolute
Albano and Sapuppo (1980)	Albano	24	4900	90 Incremental
Marques et al. (1991)	Marques	24	104	90 Incremental

Table 6.10: Details about Irregular test problems in experiments

The summary of the results for this problem are listed in table 6.11. For the test problems that this algorithm was tested on, the packing efficiencies have been above 60%. An interesting study would be to compare these results to the best available results for these problems as the optimum for all of them is currently unknown.

Problem Name	Packing Efficiency	Time(s)
Shirts	61%	3409.8
Trousers	64%	4005
Albano	74%	2889
Marques	72%	3001

Table 6.11: Summary of results for Irregular Problems

A textile marker layout designed by the general Genetic Algorithm in this work is shown in figure 6.4.

The rest of the layouts generated by this algorithm are shown in appendix B, section B.6.

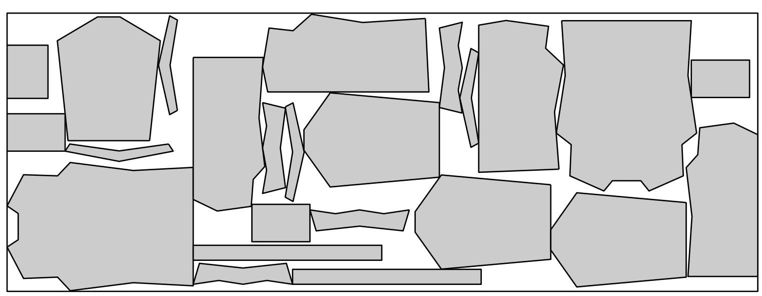


Figure 6.4: A textile marker layout generated by the general Genetic Algorithm

6.5 Discussion

The results presented above suggest a big room for improvement. For most problems a lower bound has been used as a measure to reflect the quality of the solution. It would be very interesting to compare the solutions found by the algorithm with actual optimum solutions. The total execution time has been disappointingly very long, however the execution times have been measured to give an overall picture and for the sake of completeness. Over all the solution quality provided by the general algorithm range from unacceptable deviation from the lower bound of above 40% to above average solutions of below 30%.

The results for the 1D problems are acceptable as the deviation from lower bound is less than 5%. Another feature of this algorithm is its sensitivity to input size of the problem. A perfect example of this is when trying to solve the 2D strip packing problem, where the deviation from the lower bound was almost directly proportional to the input size.

6.6 Summary

The computational results for the general genetic algorithm have been offered for all problem that were the target of this work. The performance of the algorithm was shown to vary from problem to problem. Another disappointment is the lamentably long execution times for most 2D problems.

Chapter 7

Conclusion

A study has been carried out on one-dimensional and two-dimensional cutting and packing problems. This included the definition of what cutting and packing problems are and examples of cutting and packing problems. In section 2.4 mathematical descriptions of all problems to be tackled in this work are defined. It was also pointed out that cutting and packing problems are NP-complete, therefore can not be solved in polynomial time.

A literature survey is offered in chapter 2 on some of the algorithms that have been used to solve these problems.

The objective of gaining an understanding of what genetic algorithms are, was well achieved. Chapter 3 dealt exclusively with genetic algorithms and how they have been applied on cutting and packing problems.

A general Genetic Algorithm was designed, details about how this algorithm works are dealt with in chapter 4. A novel general solution encoding has been introduced and a novel heuristic placement procedure has also been introduced in the design of this algorithm. A coding scheme that allows the algorithm to identify a problem with its constraints is also effectively made use of.

Computational tests were carried out for all problems dealt with in this work. The results have shown that the algorithm is a mixture of successes and failures. Successes in that the algorithm returned quality solutions for some problems and for some problems the solution quality was disappointing. The run time was also disappointing, but this should have been expected as the algorithm was implemented in MATLAB which is an interpreted language.

The following is recomended future work:

- An alternative implementation of this algorithm could fasten up the time taken for the running of the algorithm.
- The placement heuristic for two dimensional problems should give the downward movement a priority, i.e slide leftwards only if no downward movement is possible.
- To continue testing the algorithm on a variety of test problems both test problems from literature and real world problems.
- To do a comparative study between layouts generated by a human expert and those generated by the general genetic algorithm.

Bibliography

Adamowicz, M. and Albano, A. (1976). Nesting two-dimensional shapes in rectangular modules. *Computer Aided Design*, 8:27–33.

Albano, A. and Sapuppo, G. (1980). Optimal allocation of two-dimensional irregular shapes using heuristic search methods. *IEEE Trans. Syste., Man, Cybern., SMC-10*, 5:242–248.

Amaral, C., Bernardo, J., and Jorge, J. (1990). Marker-making using automatic placement of irregular shapes for the garment industry. *Computers and Graphics*, 14:41–46.

Baker, B. S., Coffman Jr., E. G., and Rivest, R. L. (1980). Orthogonal packings in two-dimensions. *SIAM Journal on Computing*, 9:846–855.

Beasley, J. E. (1985). An exact two-dimensional non-guillotine cutting tree search procedure. *Operations Research*, 33(1):49–64.

Bounsaythip, C. and Maouche, S. (1997). Irregular shape nesting and placing with evolutionary approach. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 4, pages 3425–3230.

Coffman, Jr., E. G., Garey, M. R., and Johnson, D. S. (1997). *Approximation algorithms (ed. D. Hochbaum)*, chapter Approximation algorithms for bin packing - a survey. PWS.

Coffman, Jr., E. G., Garey, M. R., Johnson, D. S., and Tarjan, R. E. (1980). Performance bounds for level oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9:808–826.

Davis, L. (1985). Applying adaptive search algorithms to epistatic domains. In *9th International Joint Conference on Artificial Intelligence*, pages 162–164, Los Angelos.

Dejong, K. A. (1993). Genetic algorithms are not function optimizers. In Whitley, L. D., editor, *Foundations of Genetic Algorithms 2*, pages 5–18. Morgan Kaufman, San Mateo, CA.

Dori, D. and Ben-Bassat, M. (1984). Efficient nesting of congruent convex figures. *Communications of the ACM*, 27:228–235.

Dyckhoff, H. and Finke, U. (1992). *Cutting and Packing in Production and Distribution: Typology and Bibliography*. (eds. Mueller, W.A. and Schuster, P.) Springer-Verlag, New York.

E.Falkenauer (1996). A hybrid grouping genetic algorithm. *Journal of Heuristics*, 6:5–30.

Falkenauer, E. and Delchambre, A. (1992). A genetic algorithm for bin packing and line balancing. In *Procedings of the IEEE 1992 International Conference on Robotics and Automation*.

Fowler, R. J., Paterson, M. S., and Tanimoto, S. L. (1981). Optimal packing and covering in the plane are NP-complete. *Inform. Process. Lett.*, 12(3):133–137.

Freeman, H. and Shapira, R. (1975). Determining the minimum area encasing rectangle for an arbitrary closed curve. *Communications of the ACM*, 81(7):409–413.

Garey, M. R. and Johnson, D. S. (1980). *Computers and Intractability: A guide to the theory of NP-completeness*. New York : W.H. Freeman.

Gilmore, P. and Gomory, R. (1961). A linear programming approach to the cutting stock problem. *Ops. Res.*, 9:849–859.

Gilmore, P. and Gomory, R. (1963). A linear programming approach to the cutting stock problem - part II. *Ops. Res.*, 11:863–888.

Gilmore, P. and Gomory, R. (1965). Multistage cutting stock problems of two and more dimensions. *Ops. Res.*, 13:94–120.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimisation and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Haessler, R. W. (1992). One-dimensional cutting stock problems and solution procedures. *Mathematical and Computer Modelling*, 16.

Hearn, D. and Baker, M. (1997). *Computer graphics: C version*. Upper Saddle River, N.J., Prentice-Hall.

Hinterding, R. and Khan, L. (1995). Genetic algorithms for cutting stock problems: with and without contiguity. *Lecture Notes in Computer Science*, 956:166–186.

Hopper, E. and Turton, B. (1998). Application of genetic algorithms to packing problems - a review. In *Procedings of the 2nd On-line World Conference on Soft Computing in Engineering Design and Manufacturing*, pages 279–288, New York. Springer Verlag.

Hopper, E. and Turton, B. (1999). A genetic algorithm for a 2d industrial packing problem. *Computers and Industrial Engineering*, 37:375–378.

Hopper, E. and Turton, B. (2001). An imperical investigation of meta-heuristic and heuristic algorithms for a 2d packing problem. *European Journal of Operational Research*, 128:34–57.

Hwang, S., Cheng, Y., and Horng, J. (1994). On solving rectangle bin packing problems using genetic algorithms. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 1583–1590.

Jakobs, S. (1996). On genetic algorithms for the packing of polygons. *European Journal Of Operational Research*, 88:165–181.

Kroger, B. (1995). Guillotineable bin packing: A genetic approach. *European Journal of Operational Reseasrch*, 84:645–661.

Liu, D. and Teng, H. (1999). An inproved bl-algorithm for the genetic algorithm of the orthogonal packing of rectangles. *European Journal Of Operational Research*, 112:413–420.

Lodi, A., Martello, S., and Vigo, D. (1999). Heuristic and metaheuristic approaches for a class of two dimensional bin packing problems. *Informs Journal on Computing*, 11:345–357.

Lodi, A., Martello, S., and Vigo, D. (2002). Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 123:379–396.

Lutfiyya, McMillin, Poshyanonda, and Dagli (1992). Composite stock cutting through simulated annealing. *Mathematical and Computer Modelling*, 16.

Marques, V., Bispo, C., and Sentieiro, J. (1991). A system of compaction of two-dimensional irregular shapes based on simulated annealing. In *IECON-91(IEEE)*.

Martello, S., Monaci, M., and Vigo, D. (2003). An exact approach to the strip-packing problem. *Journal of Computing*, 15:310–319.

Martello, S. and Toth, P. (1990). *Knapsack Problems*. Wiley and Sons.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures= Evolution Programs*. Springer-Verlag New York, Inc, New York,NY, USA.

Michalewicz, Z. and Fogel, D. B. (2000). *How to Solve It: Modern Heuristics*. Springer-Verlag Berlin Heidelberg.

Milenkovic, V., Daniels, K., and Li, Z. (1992). Placement and compaction of non-convex polygons for clothing. In *4th Canadian Conference on Computational Geometry*.

Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA ,USA.

Oliveira, J., Gomes, A., and Ferreira, J. (2000). A new constructive algorithm for nesting. *OR Spektrum*, 22:263–284.

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

Petridis, V. and S.Kazarlis (1994). Varrying quality function in genetic algorithms and the cutting problem. In *IEEE Conference on Evolutionary Computation*, pages 166–169.

Preparata, F. and Shamos, M. I. (1985). *Computational Geometry: An introduction*. Berlin: Springer.

Sedgewick, R. (1992). *Algorithms in C++*. Addison-Wesley, Reading, Los Angelos, California.

Smith, D. (1985). Bin packing with adaptive search. In *Genetic algorithms and their applications, Proc. 1st Int. Conf.*, pages 202–207, Pittsburgh PA.

Winston, W. L. (2004). *Operations Research: Applications and Algorithms fourth edition*. International Thomson Publishing.

Wäscher, G., Haußner, H., and Schumann, H. (2006). An improved typology of cutting and packing problems. *European Journal of Operational Research*, Article in Press.

Appendix A

Problem Datasets

A.1 1D Bin Packing test problems

A.1.1 Class 1 Problems

	Bin Capacity	150									
	Number of Items (n)	250									
1.	Items	42	69	67	57	93	90	38	36	45	42
		33	79	27	57	44	84	86	92	46	38
		85	33	82	73	49	70	59	23	57	72
		74	69	33	42	28	46	30	64	29	74
		41	49	55	98	80	32	25	38	82	30
		35	39	57	84	62	50	55	27	30	36
		20	78	47	26	45	41	58	98	91	96
		73	84	37	93	91	43	73	85	81	79
		71	80	76	83	41	78	70	23	42	87
		43	84	60	55	49	78	73	62	36	44
		94	69	32	96	70	84	58	78	25	80
		58	66	83	24	98	60	42	43	43	39
		97	57	81	62	75	81	23	43	50	38
		60	58	70	88	36	90	37	45	45	39
		44	53	70	24	82	81	47	97	35	65
		74	68	49	55	52	94	95	29	99	20
		22	25	49	46	98	59	98	60	23	72
		33	98	80	95	78	57	67	53	47	53
		36	38	92	30	80	32	97	39	80	72
		55	41	60	67	53	65	95	20	66	78
		98	47	100	85	53	53	67	27	22	61
		43	52	76	64	61	29	30	46	79	66
		27	79	98	90	22	75	57	67	36	70
		99	48	43	45	71	100	88	48	27	39
		38	100	60	42	20	69	24	23	92	32

Table A.1:

	Bin Capacity	150										
	Number of Items (n)	250										
2.	Items	84	36	65	84	34	68	64	33	69	27
		47	21	85	88	59	61	50	53	37	75
		64	84	74	57	83	28	31	97	61	36
		46	37	96	80	53	51	68	90	64	81
		66	67	80	37	92	67	64	31	94	45
		80	28	76	29	64	38	48	40	29	44
		81	35	51	48	67	24	46	38	76	22
		30	67	45	41	29	41	79	21	25	90
		62	34	73	50	79	66	59	42	90	79
		70	66	80	35	62	98	97	37	32	75
		91	91	48	26	23	32	100	46	29	26
		29	26	83	82	92	95	87	63	57	100
		63	65	81	46	42	95	90	80	53	27
		84	40	22	97	20	73	63	95	46	42
		47	40	26	88	49	24	92	87	68	95
		34	82	84	43	54	73	66	32	62	48
		99	90	86	28	25	25	89	67	96	35
		33	70	40	59	32	94	34	86	35	45
		25	76	80	42	91	44	91	97	60	29
		45	37	61	54	78	56	74	74	45	21
		96	37	75	100	58	84	85	56	54	71
		52	79	43	35	27	70	31	47	35	26
		30	97	90	80	58	60	73	46	71	39
		42	98	27	21	71	71	78	76	57	24
		91	84	35	25	77	96	97	89	30	86

Table A.2:

	Bin Capacity	150									
	Number of Items (n)	250									
3.	Items	81	39	75	66	85	36	60	56	50	75
		75	37	87	95	21	99	42	57	31	37
		42	40	69	91	45	97	84	90	52	43
		68	53	37	65	79	73	92	87	20	20
		73	42	52	20	24	76	71	72	21	21
		82	92	78	87	50	41	31	73	89	59
		88	40	71	69	45	57	49	68	84	32
		69	77	92	98	57	39	32	23	99	91
		48	21	70	43	73	69	65	57	67	28
		84	42	61	92	82	34	74	55	60	69
		26	25	67	77	67	79	47	84	50	21
		87	83	44	88	78	53	78	37	47	52
		32	88	85	82	55	41	60	66	78	72
		34	64	20	60	100	62	80	34	68	38
		32	32	37	82	98	90	58	97	56	34
		70	39	56	69	36	20	99	84	53	27
		88	53	42	45	42	31	54	60	55	27
		36	31	39	91	45	97	26	80	41	56
		70	97	48	87	23	32	75	100	97	51
		78	78	21	72	72	79	46	30	48	27
		95	48	67	58	46	92	21	82	91	40
		56	24	94	44	91	92	81	24	84	44
		83	37	98	85	88	95	29	35	100	55
		48	27	20	66	62	52	88	59	97	91
		81	81	86	48	43	60	72	88	90	48

Table A.3:

4.

Bin Capacity	150									
Number of Items (n)	250									
Items	38	60	53	55	90	48	55	57	59	25
	51	22	43	31	52	89	96	58	63	27
	46	43	30	44	71	66	64	28	83	88
	42	92	95	36	24	62	44	82	59	31
	96	44	61	78	72	62	76	65	22	41
	27	85	80	72	100	29	27	43	83	32
	33	53	95	99	20	23	72	50	50	27
	89	53	75	81	34	27	69	48	84	37
	69	54	51	49	49	54	100	55	45	83
	61	96	91	37	53	76	50	66	70	87
	92	35	53	95	47	56	55	86	32	99
	83	88	41	63	77	60	66	53	79	81
	96	34	99	47	74	87	44	77	52	99
	69	64	94	38	69	61	98	40	84	89
	49	64	53	41	34	85	35	55	61	68
	100	75	98	36	44	57	24	60	45	48
	60	94	71	70	64	62	93	20	69	37
	63	61	26	54	89	46	54	50	32	71
	62	40	26	59	62	27	60	50	74	34
	40	70	56	23	66	57	43	45	65	25
	82	82	37	66	47	44	94	23	24	51
	100	22	25	51	95	58	97	30	79	23
	53	80	20	65	64	21	26	100	81	98
	70	85	92	97	86	71	91	29	63	34
	67	23	33	89	94	47	100	37	40	58

Table A.4:

5.

Bin Capacity	150									
Number of Items (n)	250									
Items	73	39	49	79	54	57	98	69	67	49
	38	34	96	27	92	82	69	45	69	20
	75	97	51	70	29	91	98	77	48	45
	43	61	36	82	89	94	26	35	58	58
	57	46	44	91	49	52	65	42	33	60
	37	57	91	52	95	84	72	75	89	81
	67	74	87	60	32	76	85	59	62	39
	64	52	88	45	29	88	85	54	40	57
	91	55	60	37	86	21	21	43	77	75
	92	33	59	74	40	36	62	21	56	38
	22	45	94	68	83	86	75	21	40	44
	74	52	61	95	20	79	76	32	21	91
	83	39	31	81	41	90	74	100	38	33
	74	40	80	39	22	46	58	65	67	37
	82	64	26	80	74	20	62	82	40	28
	72	45	62	72	89	31	92	63	89	33
	25	54	66	100	20	90	87	48	28	46
	76	50	66	30	26	23	40	70	57	92
	52	54	27	58	66	65	93	83	37	62
	94	29	66	98	20	66	42	52	90	22
	30	34	65	81	90	44	88	51	97	79
	58	46	65	40	68	64	34	59	99	82
	86	88	52	76	76	50	51	92	59	22
	60	69	45	66	50	62	59	90	54	55
	92	23	97	73	39	88	34	92	74	90

Table A.5:

	Bin Capacity	150									
	Number of Items (n)	250									
6.	Items	55	28	45	71	56	45	63	26	20	34
		78	26	21	99	50	52	29	52	84	78
		84	89	93	83	97	35	29	80	99	86
		63	100	87	54	48	72	98	43	81	96
		77	92	32	66	82	52	30	52	97	56
		44	67	60	79	78	90	38	99	42	97
		63	39	69	67	91	38	37	51	98	30
		77	78	35	33	94	36	59	85	98	80
		79	68	61	27	95	83	91	90	38	93
		22	35	38	100	26	35	64	40	79	49
		88	41	28	62	78	65	90	35	50	62
		91	57	60	50	28	77	97	35	40	21
		73	30	75	50	27	58	59	94	60	55
		89	84	91	65	99	89	83	47	52	24
		66	98	51	21	23	78	41	99	52	36
		69	70	91	54	38	98	57	64	76	61
		31	27	23	22	61	65	35	37	75	54
		97	45	78	22	79	76	81	78	41	59
		28	58	90	78	57	63	24	27	79	67
		88	49	57	78	87	66	91	37	51	49
		84	32	62	36	52	72	59	77	54	46
		57	69	81	80	99	87	33	45	43	66
		28	30	54	23	79	69	56	24	82	58
		37	56	82	23	78	63	64	37	66	36
		41	71	48	42	26	45	26	86	64	54

Table A.6:

	Bin Capacity	150									
	Number of Items (n)	248									
		49	45	86	74	64	73	93	34	97	80
		24	87	100	75	89	78	46	31	68	63
		78	28	96	54	64	31	65	90	41	47
		71	51	63	44	93	46	83	68	57	89
		35	99	39	24	69	64	25	85	65	81
		61	40	64	88	43	99	53	98	70	38
		75	23	80	72	97	89	80	38	30	34
		22	61	48	22	28	99	55	89	67	24
		27	91	90	20	36	77	44	24	60	96
		83	53	76	27	91	58	78	23	31	99
		42	64	39	73	43	36	76	97	41	90
7.		24	82	55	93	63	61	39	73	54	77
	Items	100	46	69	74	41	32	56	68	98	61
		28	21	30	47	43	54	33	31	38	49
		40	44	93	20	81	71	36	71	36	42
		56	85	23	86	88	95	61	41	34	74
		37	82	30	98	86	37	93	100	69	25
		54	47	58	50	87	90	45	71	70	38
		49	42	33	78	48	94	99	100	84	91
		27	69	52	64	99	30	34	55	96	92
		48	88	76	38	73	90	99	45	84	94
		82	28	35	94	100	44	23	58	23	35
		84	75	30	58	61	100	63	99	85	60
		78	56	76	61	59	93	83	84	89	59
		75	32	21	62	27	64	44	83		

Table A.7:

8.	Bin Capacity	150									
	Number of Items (n)	250									
	Items	68	90	38	98	44	66	76	67	65	81
		95	62	34	33	56	75	40	72	49	95
		59	40	53	27	70	27	72	92	79	66
		92	47	87	32	51	94	22	79	75	70
		58	85	37	68	69	47	63	37	53	90
		85	88	68	100	86	93	26	44	77	72
		46	58	44	49	100	72	76	74	78	30
		79	30	88	29	70	69	26	53	86	48
		55	30	95	22	79	94	54	43	84	51
		80	90	61	43	71	72	82	83	91	56
		42	45	80	73	62	95	53	40	42	63
		80	79	86	59	22	62	72	51	60	55
		56	92	56	55	51	34	100	89	64	99
		87	74	38	28	50	86	92	98	30	30
		89	51	65	31	60	85	79	39	27	61
		84	41	53	77	77	94	86	91	49	47
		35	28	82	73	34	92	51	35	51	47
		64	89	72	89	22	52	75	85	73	83
		56	58	57	64	50	66	26	80	61	54
		40	89	46	45	59	51	79	73	95	42
		21	64	73	68	65	100	50	81	55	71
		44	63	76	36	73	74	98	36	97	23
		58	50	70	75	97	76	24	72	34	36
		67	45	55	94	63	100	95	54	40	62
		68	87	48	37	85	73	62	22	23	33

Table A.8:

9.

Bin Capacity	150									
Number of Items (n)	250									
Items	81	41	27	95	46	69	45	39	32	98
	41	46	100	86	84	39	67	34	92	59
	43	21	56	88	26	35	51	22	100	96
	49	95	38	62	63	97	42	62	100	43
	44	77	97	94	68	23	50	36	89	58
	97	27	64	65	54	58	24	35	33	63
	32	50	58	90	44	50	48	21	72	75
	21	74	28	95	77	69	96	24	57	85
	72	96	50	83	65	62	99	93	23	77
	94	31	50	33	79	73	23	55	44	78
	84	66	31	59	97	95	22	76	90	66
	29	100	90	92	50	49	47	43	37	40
	60	52	54	99	34	46	88	97	85	39
	32	51	95	54	99	86	48	90	28	25
	86	39	74	26	38	60	41	67	80	33
	37	62	71	87	31	72	84	84	53	85
	32	24	88	54	28	36	91	61	29	68
	69	35	30	88	85	87	70	70	59	26
	73	27	44	27	35	38	65	21	69	59
	35	70	40	84	42	92	24	46	78	60
	76	43	49	79	65	24	28	43	26	93
	62	91	21	21	32	34	86	27	79	34
	88	93	58	77	62	87	99	61	83	75
	99	93	39	85	31	69	48	67	50	24
	49	82	97	86	21	86	41	100	84	77

Table A.9:

10.

Bin Capacity	150
Number of Items (n)	250

Items	87	70	43	62	40	37	71	34	29	70
	41	78	74	51	71	47	21	32	37	80
	48	93	22	46	96	44	94	99	100	65
	61	34	25	35	60	52	90	81	93	74
	85	43	21	89	100	56	55	88	52	63
	40	41	62	35	77	72	75	93	55	95
	54	87	38	84	83	88	37	65	58	89
	89	48	85	25	100	28	20	96	52	100
	94	94	69	94	39	62	86	43	61	88
	78	72	71	31	45	72	87	60	91	100
	66	44	83	23	22	81	22	55	67	73
	68	42	83	40	86	63	33	24	54	48
	41	56	48	29	51	78	85	68	35	99
	74	42	26	49	65	92	51	43	97	91
	24	79	30	58	76	59	92	94	43	31
	87	59	56	74	91	88	85	70	59	80
	54	66	55	61	64	80	53	80	44	74
	22	91	91	83	51	57	20	83	46	54
	56	76	24	41	26	37	91	52	94	94
	49	61	69	79	38	78	25	57	70	81
	57	34	22	58	99	39	99	29	34	58
	94	46	41	56	86	92	81	82	38	42
	99	59	73	57	59	67	44	29	53	54
	40	83	55	66	72	80	25	92	78	97
	28	99	73	66	44	59	95	53	81	83

Table A.10:

	Bin Capacity	150									
	Number of Items (n)	250									
11.	Items	92	61	32	31	38	29	44	90	68	35
		78	56	25	26	61	90	20	43	37	65
		63	39	95	87	83	97	41	87	69	75
		82	45	80	78	89	98	32	24	55	63
		92	33	95	80	27	62	97	36	73	67
		35	82	37	61	82	45	26	56	91	53
		71	78	33	20	26	97	90	30	44	86
		82	25	56	34	54	97	91	42	74	83
		38	44	44	26	66	35	45	80	42	97
		26	61	59	92	92	81	33	86	87	100
		69	25	51	32	94	50	42	21	90	52
		32	66	77	22	64	51	41	81	54	70
		67	84	72	47	92	82	96	58	80	95
		36	60	42	41	51	29	99	57	21	48
		30	65	55	62	60	49	80	63	25	35
		54	27	68	64	35	52	87	40	52	41
		59	56	77	41	43	73	87	56	76	29
		46	39	92	40	72	54	20	56	68	27
		23	62	45	95	90	27	36	79	88	51
		95	96	66	57	96	25	33	84	67	75
		78	61	53	42	72	40	60	99	32	99
		70	39	90	73	71	23	61	49	100	35
		45	34	84	49	100	75	46	85	83	93
		90	68	20	100	73	25	66	70	40	83
		37	29	29	87	95	42	95	100	96	55

Table A.11:

12.	Bin Capacity	150									
	Number of Items (n)	250									
	Items	65	58	79	76	84	63	91	81	30	57
		71	67	33	27	99	36	48	66	68	66
		40	87	99	59	42	50	51	87	98	64
		32	41	56	85	87	95	46	75	37	54
		58	82	57	26	94	31	71	95	27	29
		38	37	55	94	70	90	29	98	27	95
		98	95	98	51	47	71	27	61	49	66
		93	89	34	60	33	97	74	95	44	96
		88	89	84	52	50	53	90	94	98	46
		62	68	45	77	49	82	51	95	33	94
		98	75	47	42	64	34	51	68	27	42
		87	65	44	62	84	75	70	44	84	54
		92	58	50	61	95	59	22	24	56	59
		45	54	43	70	97	97	29	42	55	67
		91	26	61	65	28	26	54	96	49	46
		100	68	58	43	36	78	40	22	41	82
		46	58	29	97	62	69	57	67	85	32
		93	43	47	99	20	81	70	91	23	80
		43	81	22	76	95	29	60	50	99	38
		79	20	67	63	89	85	97	100	33	100
		43	31	57	45	48	72	26	66	30	81
		43	62	86	64	89	22	100	73	38	63
		43	62	86	64	89	22	100	73	38	63
		80	98	71	82	28	67	88	57	44	78
		74	47	57	96	47	82	55	90	63	55

Table A.12:

13.	Bin Capacity	150									
	Number of Items (n)	248									
	Items	87	100	69	94	71	91	74	76	68	82
		96	85	96	85	79	71	56	86	46	55
		44	35	29	42	65	49	82	73	70	63
		94	63	71	86	27	93	80	42	45	93
		69	76	61	29	81	46	42	74	45	88
		96	40	31	47	82	60	43	20	80	69
		46	90	34	81	59	43	61	28	56	32
		90	60	66	70	77	43	92	85	45	74
		40	51	48	30	41	63	71	43	24	91
		48	65	41	34	47	88	73	57	50	68
		80	34	70	96	80	26	77	53	82	78
		74	87	69	97	87	64	31	77	25	60
		20	66	48	80	77	90	69	61	93	41
		35	28	68	59	27	34	24	56	42	29
		52	42	27	83	78	40	37	21	77	43
		45	76	53	36	61	52	53	41	76	83
		49	38	71	64	89	48	32	69	80	88
		41	46	37	60	63	20	47	40	93	46
		84	77	92	51	87	49	75	58	61	83
		53	22	79	80	92	96	49	53	22	50
		71	73	66	23	70	76	93	46	39	40
		93	41	36	60	35	25	99	79	52	22
		66	44	68	73	60	56	76	95	53	37
		68	87	20	38	95	86	47	68	66	37
		44	47	77	26	90	97	86	57		

Table A.13:

14.	Bin Capacity	150									
	Number of Items (n)	248									
	Items	72	83	38	84	82	88	47	43	59	92
		78	25	47	65	42	41	36	54	43	87
		51	65	98	82	34	21	94	100	80	95
		32	23	26	93	70	96	79	68	93	74
		76	99	75	44	94	93	38	44	45	49
		22	39	87	74	25	59	22	44	70	51
		68	33	25	77	55	75	87	42	79	50
		78	43	20	88	56	93	75	56	36	70
		47	94	24	35	47	26	48	40	48	77
		30	36	96	63	47	22	60	51	84	90
		46	98	59	94	59	54	38	79	77	73
		61	21	83	81	34	37	76	49	23	75
		79	98	100	29	88	83	80	100	56	61
		31	37	43	69	78	28	41	82	56	31
		25	22	46	68	63	75	64	76	65	98
		77	36	21	86	63	95	61	22	45	49
		35	63	43	71	23	53	100	41	50	51
		26	54	62	27	68	73	79	47	53	56
		85	93	36	97	29	65	20	32	49	83
		33	49	90	93	64	71	45	59	74	77
		58	91	88	60	67	44	42	89	79	40
		88	95	81	73	82	23	20	22	92	75
		23	74	25	79	62	48	21	74	28	78
		73	31	44	28	37	77	52	23	82	97
		52	90	94	28	95	37	51	21		

Table A.14:

15.	Bin Capacity	150									
	Number of Items (n)	249									
	Items	29	31	81	61	24	92	70	56	100	61
		85	83	53	44	70	65	25	39	71	26
		63	99	64	97	88	54	91	53	96	44
		49	94	63	65	90	37	30	28	53	83
		41	54	89	32	49	40	80	63	89	74
		89	20	25	75	31	56	92	85	40	97
		56	100	55	35	27	96	89	29	44	26
		49	73	72	50	52	77	35	97	79	45
		75	62	91	50	37	25	65	97	62	74
		81	72	100	57	49	83	23	92	63	55
		81	64	88	50	74	52	25	97	48	43
		49	33	86	35	71	21	90	95	88	80
		93	73	60	96	65	56	32	88	67	69
		63	26	51	59	85	41	91	70	92	44
		53	49	91	33	57	26	99	24	48	52
		92	43	46	47	96	36	88	55	76	51
		87	44	58	34	69	43	56	37	74	82
		64	75	99	36	54	76	72	21	33	61
		87	54	82	94	87	46	71	83	71	44
		87	20	31	67	93	100	94	97	64	63
		36	89	48	34	41	42	74	30	48	73
		37	100	49	58	50	86	79	91	98	63
		24	82	24	48	26	98	82	75	62	55
		82	87	74	87	32	73	28	95	84	29
		82	68	70	49	88	23	78	96	50	

16.

Bin Capacity	150									
Number of Items (n)	250									
Items	73	99	36	56	65	46	60	32	77	41
	32	94	77	63	35	78	24	95	96	81
	86	75	36	21	48	28	95	62	91	40
	26	88	43	45	22	54	28	48	88	80
	35	81	69	94	96	95	67	30	29	59
	40	65	31	74	39	57	95	46	32	82
	55	36	47	85	80	36	31	40	82	53
	59	57	31	82	72	38	69	53	74	79
	97	42	49	74	86	37	89	63	75	84
	38	42	59	80	23	20	95	46	98	97
	64	66	84	24	25	20	68	32	38	48
	27	74	86	54	81	73	77	40	48	81
	86	59	87	60	27	81	22	29	62	41
	76	57	31	79	30	83	29	65	97	49
	52	42	20	85	89	93	39	29	33	21
	26	73	28	28	38	33	96	50	73	53
	31	100	27	85	37	42	79	60	95	21
	87	34	46	88	57	41	66	38	79	27
	85	72	83	82	94	56	24	83	32	49
	78	30	33	50	37	49	25	44	86	22
	54	38	81	77	39	47	22	51	40	70
	83	86	69	73	31	80	84	70	55	68
	27	25	25	27	48	30	83	42	26	63
	72	74	83	55	36	44	95	81	73	53
	63	47	88	86	48	21	89	74	70	63

	Bin Capacity	150									
	Number of Items (n)	250									
17.	Items	56	68	67	56	44	64	75	96	80	58
		75	50	43	42	31	94	64	77	89	30
		45	74	53	57	56	47	31	55	58	28
		72	27	35	68	68	82	67	47	24	49
		40	67	96	80	88	39	93	32	47	81
		99	38	51	97	31	55	40	63	93	78
		30	39	55	67	24	72	71	43	31	79
		77	42	73	62	93	90	50	98	36	76
		72	35	48	53	33	64	51	32	82	68
		55	51	84	72	50	30	21	25	43	55
		56	65	73	24	100	21	47	97	90	83
		75	43	61	51	32	74	63	91	21	92
		71	74	42	100	21	63	72	42	54	57
		42	81	68	79	38	47	21	22	55	61
		40	35	76	83	100	31	62	36	75	82
		50	80	38	68	21	84	72	67	84	98
		39	68	86	63	98	67	75	37	35	41
		63	67	57	26	53	36	56	92	89	76
		49	23	23	49	24	56	74	34	64	100
		82	25	30	72	82	68	67	57	57	40
		33	40	27	52	89	52	97	31	48	50
		57	37	77	32	97	67	93	70	20	38
		71	49	78	40	94	21	66	96	86	85
		99	79	85	77	68	37	41	68	27	100
		96	74	46	79	43	59	50	39	42	80

18.

Bin Capacity	150									
Number of Items (n)	250									
Items	87	62	73	65	73	72	77	85	33	39
	58	100	87	24	35	34	28	70	49	36
	65	27	75	99	99	59	79	99	90	64
	42	82	58	56	89	80	97	82	44	92
	29	39	90	99	68	40	23	95	39	77
	59	74	94	67	72	90	60	49	21	20
	49	33	85	84	50	95	52	31	46	96
	73	66	33	90	77	79	27	91	54	62
	44	78	35	62	97	25	79	31	26	87
	30	24	31	24	53	90	66	21	58	28
	81	61	100	33	95	77	77	75	52	58
	95	47	27	29	74	84	49	25	57	90
	61	59	99	70	33	25	54	66	32	20
	32	47	28	71	33	55	81	56	21	83
	67	46	96	50	94	55	57	100	35	50
	21	97	30	34	57	74	99	63	40	96
	83	37	59	72	59	50	84	88	22	97
	81	22	55	31	66	23	88	89	28	77
	78	41	93	94	45	84	48	75	38	68
	34	37	40	78	60	94	58	71	70	30
	77	34	96	58	70	61	27	55	48	80
	26	59	31	55	80	75	73	48	22	35
	97	46	98	48	49	28	67	94	46	46
	37	45	48	42	31	67	23	98	58	55
	24	60	48	95	93	49	56	90	31	24

19.

Bin Capacity	150									
Number of Items (n)	250									
Items	71	39	69	32	82	75	60	39	80	61
	43	34	80	69	21	59	82	54	26	51
	96	76	76	45	41	73	91	23	98	90
	59	43	52	48	87	97	51	72	77	59
	83	65	40	79	30	31	99	40	42	66
	47	67	50	72	62	95	75	81	36	36
	70	89	95	62	56	23	37	50	46	30
	64	94	65	55	24	28	96	31	57	72
	96	63	40	79	89	97	50	37	93	52
	86	74	47	84	77	48	54	97	70	29
	40	74	71	46	46	63	48	74	25	77
	46	80	35	56	65	49	38	26	81	80
	73	38	27	97	47	88	42	62	45	33
	78	35	63	25	74	63	41	81	68	78
	52	30	22	100	42	53	60	58	92	74
	67	72	30	48	65	23	94	99	67	57
	73	44	63	53	87	54	62	100	30	20
	25	94	85	68	59	82	52	100	89	49
	74	44	23	39	21	65	80	93	36	97
	74	37	52	94	60	77	57	71	61	92
	98	86	55	89	24	88	53	85	39	89
	64	45	52	71	79	23	50	95	55	36
	95	41	36	94	52	36	76	72	52	42
	27	61	55	64	30	22	53	71	51	37
	96	74	63	54	81	77	55	29	89	41

A.1.2 Class 2 Problems

1.

Bin Capacity	100									
Number of Items (n)	60									
Items	36.6	26.8	36.6	43	26.3	30.7	41.4	28.7	29.9	49.5
	25.1	25.4	47.4	25.2	27.4	37	26.9	36.1	47.3	25.2
	27.5	47.2	25.9	26.9	44.4	25.8	29.8	43.9	27.3	28.8
	44.5	27.2	28.3	41.9	26.1	32	36.3	27.1	36.6	35.5
	27.3	37.2	46.6	26.2	27.2	35.7	29.2	35.1	39.5	25.5
	35	35	30.3	34.7	45	25.2	29.8	41	27.5	31.5

Table A.15:

2.

Bin Capacity	100									
Number of Items (n)	60									
Items	47.5	25.6	26.9	39.6	26.4	34	46.8	26.2	27	36.1
	30	33.9	44.4	25.1	30.5	36.6	25.2	38.2	40.9	27.7
	31.4	46.5	26	27.5	44.7	25.1	30.2	39.9	29.7	30.4
	42.3	25.8	31.9	47.3	26	26.7	42.6	26.1	31.3	40.3
	28.9	30.8	40.2	26.5	33.3	39.6	25.7	34.7	41.1	28.2
	30.7	46.2	25.8	28	41.2	25.4	33.4	37.6	25.5	36.9

Table A.16:

3.

Bin Capacity	100									
Number of Items (n)	60									
Items	49.4	25	25.6	42.9	26.6	30.5	37.8	26.8	35.4	48.2
	25.1	26.7	46.4	25.9	27.7	39.8	27.6	32.6	39	26
	35	48.2	25.1	26.7	43	26.2	30.8	40	26.1	33.9
	49.8	25	25.2	36.2	28.8	35	49.8	25	25.2	45.9
	26	28.1	40.1	27.1	32.8	36.7	28.8	34.5	35.2	27.9
	36.9	47.6	26.1	26.3	47.9	25.4	26.7	43.6	28	28.4

Table A.17:

4.

Bin Capacity	100									
Number of Items (n)	60									
Items	37.8	27.5	34.7	46.2	26.1	27.7	42.9	27.3	29.8	49.5
	25	25.5	37.1	26.2	36.7	39.1	29.3	31.6	49.3	25.2
	25.5	40.5	25	34.5	46.1	25.8	28.1	47.8	25.7	26.5
	35.4	27.8	36.8	45.1	25.6	29.3	48.5	25.4	26.1	47.7
	25.8	26.5	36.9	27	36.1	37.5	26.8	35.7	41.4	25.4
	33.2	45.9	26.3	27.8	45.6	26.3	28.1	42.6	27.7	29.7

Table A.18:

5.

Bin Capacity	100									
Number of Items (n)	60									
Items	37.9	29.3	32.8	47	25.1	27.9	41.1	25.3	33.6	41.4
	27.6	31	41.8	28.6	29.6	37.8	29.6	32.6	42.8	28.1
	29.1	45.5	26.4	28.1	49.4	25.2	25.4	47.8	25.8	26.4
	40.9	28.7	30.4	42.5	25.6	31.9	40.2	25.2	34.6	40.3
	28.3	31.4	40.1	28.4	31.5	43.4	28.2	28.4	49.6	25.1
	25.3	49.1	25.3	25.6	49.8	25	25.2	37.9	26.4	35.7

Table A.19:

6.

Bin Capacity	100									
Number of Items (n)	60									
Items	49.6	25	25.4	39.6	27	33.4	48.3	25.2	26.5	46.3
	26.1	27.6	38.9	28.3	32.8	38	26.8	35.2	41.6	28.2
	30.2	38.8	25.2	36	48.9	25.2	25.9	43.3	28.1	28.6
	38	28.5	33.5	37.2	30.1	32.7	37.2	28.1	34.7	35.5
	30.5	34	43.2	27.8	29	46.2	26.2	27.6	48.4	25.2
	26.4	42.2	28.2	29.6	46.9	26.2	26.9	35.8	28.1	36.1

Table A.20:

7.

Bin Capacity	100									
Number of Items (n)	60									
Items	40.3	28.7	31	42.7	28.1	29.2	45.1	26.5	28.4	45
	25	30	40	27.8	32.2	37.4	26.1	36.5	38	27.6
	34.4	46.4	25.2	28.4	39.4	26.9	33.7	37.5	29.7	32.8
	49.8	25	25.2	37.4	30.4	32.2	35.5	27.6	36.9	48.5
	25.3	26.2	35.7	27.5	36.8	42.4	25.9	31.7	47.1	25
	27.9	38.8	27.3	33.9	44.9	27.5	27.6	40.5	27.4	32.1

Table A.21:

8.

Bin Capacity	100									
Number of Items (n)	60									
Items	48	25.6	26.4	37.3	29.7	33	41.2	28.1	30.7	39.2
	28.6	32.2	43.2	26.5	30.3	47.8	25.5	26.7	40.6	26.8
	32.6	36.5	25.5	38	40.7	27.9	31.4	37.8	29.3	32.9
	36.2	27.3	36.5	48.7	25.2	26.1	42.2	26	31.8	41
	26.6	32.4	41	26.3	32.7	37	26	37	45.4	25
	29.6	36.6	28.1	35.3	47.6	26	26.4	46.5	25.6	27.9

Table A.22:

9.

Bin Capacity	100									
Number of Items (n)	60									
Items	48.5	25.6	25.9	49.1	25.1	25.8	35.3	31.2	33.5	49.8
	25	25.2	36.1	29.3	34.6	37.8	25.2	37	45.1	25.6
	29.3	43.9	26.9	29.2	45.3	26.3	28.4	39.8	25.9	34.3
	39.1	25.5	35.4	46.2	25.2	28.6	36.3	25.4	38.3	38.1
	27.8	34.1	45.4	25.1	29.5	35.6	28.3	36.1	45.3	27.2
	27.5	35.7	27.5	36.8	46.8	25	28.2	35.2	28	36.8

Table A.23:

10.

Bin Capacity	100									
Number of Items (n)	60									
Items	35.1	25.8	39.1	35.9	25.9	38.2	37.6	30.8	31.6	44.5
	27.5	28	42.6	27.6	29.8	46.8	25.9	27.3	45.3	25.8
	28.9	44.3	25.3	30.4	38	26.1	35.9	37.7	30.4	31.9
	48.3	25.4	26.3	45.1	26.1	28.8	37.3	26.3	36.4	41.7
	27.6	30.7	36.3	29.4	34.3	44.2	25.4	30.4	41.2	25.6
	33.2	36.9	26.2	36.9	42.9	25.2	31.9	39.7	26.6	33.7

Table A.24:

11.

Bin Capacity	100									
Number of Items (n)	60									
Items	40	27.7	32.3	41.9	28	30.1	38.8	25.7	35.5	39.8
	25.1	35.1	42.3	25.7	32	35.7	26	38.3	42.4	26.6
	31	44.8	26.8	28.4	41.7	25.1	33.2	36.6	29.9	33.5
	36	31.8	32.2	44.1	27.4	28.5	47.8	25.7	26.5	46.4
	26.5	27.1	42.8	28	29.2	47.2	25	27.8	49.1	25.3
	25.6	44	26.6	29.4	40.3	25	34.7	43.9	27	29.1

Table A.25:

12.

Bin Capacity	100									
Number of Items (n)	60									
Items	49.3	25.1	25.6	49.2	25.1	25.7	39.6	26.8	33.6	39.2
	25.6	35.2	49.2	25.3	25.5	44.7	27	28.3	47	25.2
	27.8	37.8	25.3	36.9	38.9	28	33.1	37.2	25.8	37
	48.1	25.3	26.6	39.5	27.8	32.7	40.9	26.8	32.3	45
	26.6	28.4	39.8	29.5	30.7	39.1	29.6	31.3	49.5	25.2
	25.3	39.9	28.8	31.3	35.2	26.7	38.1	38.5	28.4	33.1

Table A.26:

13.

Bin Capacity	100									
Number of Items (n)	60									
Items	47.2	25.4	27.4	42	27.6	30.4	49.5	25.1	25.4	39.5
	26.8	33.7	42.4	26.7	30.9	47	26	27	39.1	26.8
	34.1	46.2	25.2	28.6	44	25.6	30.4	43.3	28.1	28.6
	43.5	26.6	29.9	43.6	25	31.4	45	27.1	27.9	37.3
	25.5	37.2	43.8	26.1	30.1	44.2	25.6	30.2	39.3	25.5
	35.2	36.7	31.2	32.1	40.5	26.6	32.9	38.9	27.2	33.9

Table A.27:

14.

Bin Capacity	100									
Number of Items (n)	60									
Items	49.3	25.3	25.4	45.6	25.8	28.6	42.9	27.8	29.3	35.6
	31.7	32.7	49.2	25.1	25.7	48.8	25.1	26.1	44.8	25.6
	29.6	36.9	29.1	34	48	25.8	26.2	38.1	29.6	32.3
	48.5	25.6	25.9	41.9	26.4	31.7	45.9	25.4	28.7	44.4
	26	29.6	42.1	28	29.9	49.5	25	25.5	45.2	26.7
	28.1	35	29.7	35.3	36.1	25.3	38.6	43.4	27.8	28.8

Table A.28:

15.

Bin Capacity	100									
Number of Items (n)	60									
Items	47	26	27	46.4	25.8	27.8	36.7	26.6	36.7	49.2
	25.1	25.7	49.1	25.2	25.7	41.5	28.2	30.3	42.9	26
	31.1	45	27.3	27.7	48.4	25.4	26.2	39.9	27.4	32.7
	44.8	27	28.2	36.1	25	38.9	36	30	34	41.5
	25.1	33.4	35.3	31.1	33.6	46	25.9	28.1	47.4	25.4
	27.2	40	26.4	33.6	36.5	26.9	36.6	41.2	27.9	30.9

Table A.29:

16.

Bin Capacity	100									
Number of Items (n)	60									
Items	44.2	26.7	29.1	44.8	25.2	30	46.3	26.6	27.1	45.1
	26.1	28.8	39.2	27.9	32.9	45.4	25.8	28.8	43.1	27.3
	29.6	47.1	25.5	27.4	49.1	25.2	25.7	48.5	25.3	26.2
	40.9	29.2	29.9	48.7	25.2	26.1	38.3	28.1	33.6	42.6
	25.1	32.3	36	31.2	32.8	48.1	25.6	26.3	47.2	25.4
	27.4	45.1	25	29.9	38.9	26.4	34.7	41.3	29.1	29.6

Table A.30:

17.

Bin Capacity	100									
Number of Items (n)	60									
Items	40.2	26.4	33.4	49.8	25	25.2	43.9	27.9	28.2	48
	25.1	26.9	40.2	28.4	31.4	43.2	25.3	31.5	42.9	27.5
	29.6	38.2	26.4	35.4	49.2	25.3	25.5	45	27	28
	43.6	25.6	30.8	41.2	27.4	31.4	48.2	25.5	26.3	47.8
	25.8	26.4	45.5	26.6	27.9	48.1	25.2	26.7	49.7	25
	25.3	44.4	27.4	28.2	43.2	26.8	30	40.8	26.3	32.9

Table A.31:

18.

Bin Capacity	100									
Number of Items (n)	60									
Items	46.7	25.5	27.8	42.2	26.9	30.9	45.3	26.8	27.9	41.1
	25.6	33.3	43.6	27	29.4	45.5	25.1	29.4	48.9	25.3
	25.8	49.5	25	25.5	39.4	25.1	35.5	49.6	25	25.4
	40.6	29.2	30.2	43.7	27.8	28.5	45.9	26.2	27.9	46.9
	26.4	26.7	47.8	25.8	26.4	42.8	26.6	30.6	49.2	25.1
	25.7	40.3	25.5	34.2	42.5	28.5	29	45.9	26	28.1

Table A.32:

19.

Bin Capacity	100									
Number of Items (n)	60									
Items	49.5	25	25.5	37.3	29.6	33.1	36.6	25.4	38	47.1
	25.2	27.7	42	25.6	32.4	43.4	27.5	29.1	43.9	27.1
	29	37.7	25.3	37	44.3	27.6	28.1	42.4	26.1	31.5
	35.2	29	35.8	46.6	26.1	27.3	39.9	25.4	34.7	38.5
	27.8	33.7	37.7	30.4	31.9	49.3	25.1	25.6	49.2	25.3
	25.5	45.3	25.2	29.5	47.9	25.1	27	36.4	27.5	36.1

Table A.33:

20.

Bin Capacity	100									
Number of Items (n)	60									
Items	36.1	25.7	38.2	39.1	25.9	35	39.5	26.3	34.2	42.7
	25.7	31.6	45.9	25.5	28.6	36.7	27	36.3	46	26.7
	27.3	45.9	25.9	28.2	49.3	25.3	25.4	46	25.3	28.7
	36.6	29.2	34.2	47	25.4	27.6	40.5	25.2	34.3	49.9
	25	25.1	48.8	25.1	26.1	38.4	25.8	35.8	40.7	28.8
	30.5	41.5	26.1	32.4	42.3	27.9	29.8	36.8	30.3	32.9

Table A.34:

A.2 1D Cutting Stock test problems

1.

Stock Length	14							
Number of Items	20							
Item Length	3	4	5	6	7	8	9	10
Demand	5	2	1	2	4	2	1	3

Table A.35:

2.

Stock Length	15							
Number of Items	50							
Item Length	3	4	5	6	7	8	9	10
Demand	4	8	5	7	8	5	5	8

Table A.36:

3.

Stock Length	25							
Number of Items	60							
Item Length	3	4	5	6	7	8	9	10
Demand	6	12	6	5	15	6	4	6

Table A.37:

4.

Stock Length	25							
Number of Items	60							
Item Length	5	6	7	8	9	10	11	12
Demand	7	12	15	7	4	6	8	1

Table A.38:

5.

Stock Length	4300								
Number of Items	126								
Item Length	2350	2250	2220	2100	2050	2000	1950	1900	1850
	2	4	4	15	6	11	6	15	13
Demand	1700	1650	1350	1300	1250	1200	1150	1100	1050
	5	2	9	3	6	10	4	8	3

Table A.39:

Appendix B

Layouts for 2D Problems

B.1 Layouts for Strip Packing Problems with fixed orientation and free cutting

Figure B.1: Problems 1-3

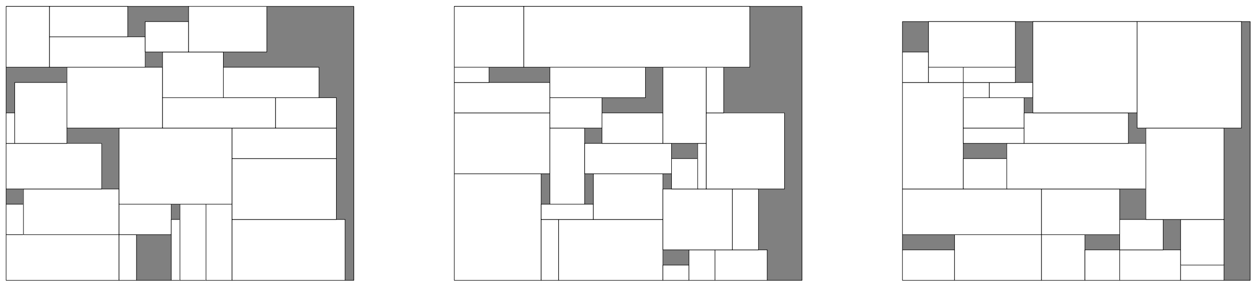


Figure B.2: Problems 4-6

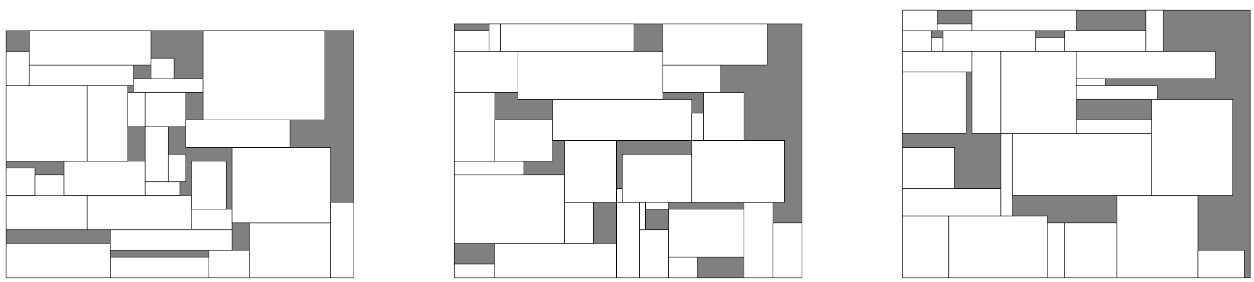


Figure B.3: Problems 7-9

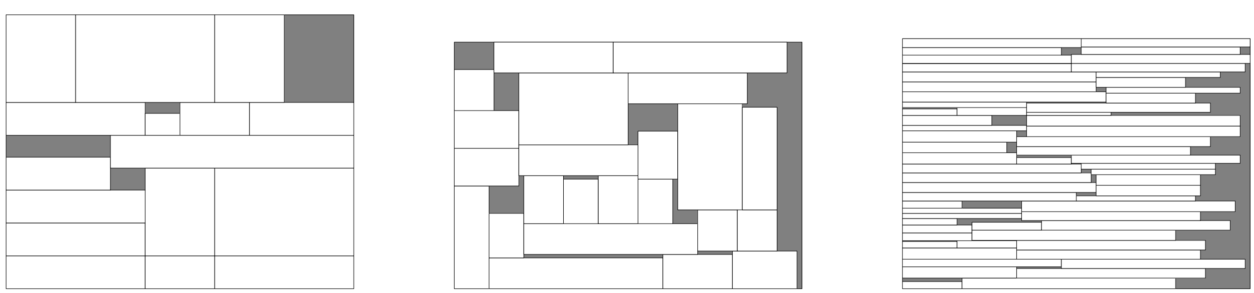


Figure B.4: Problems 10-12

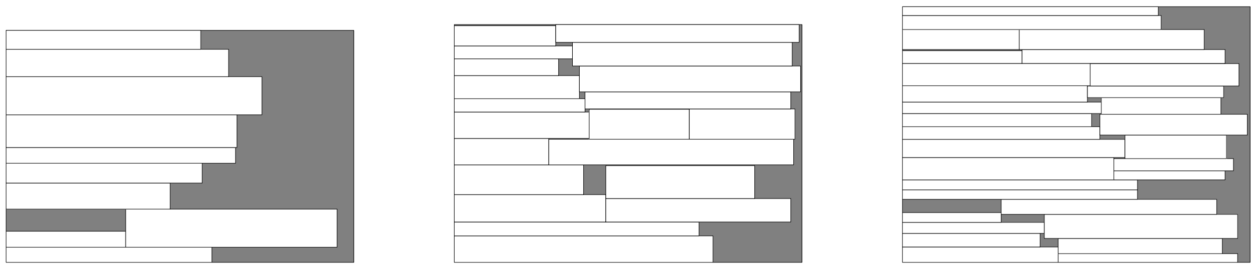


Figure B.5: Problems 13-15

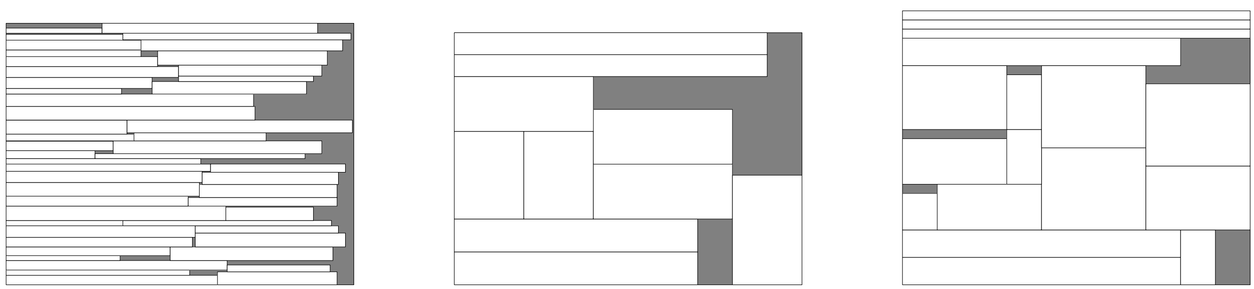


Figure B.6: Problems 16-18

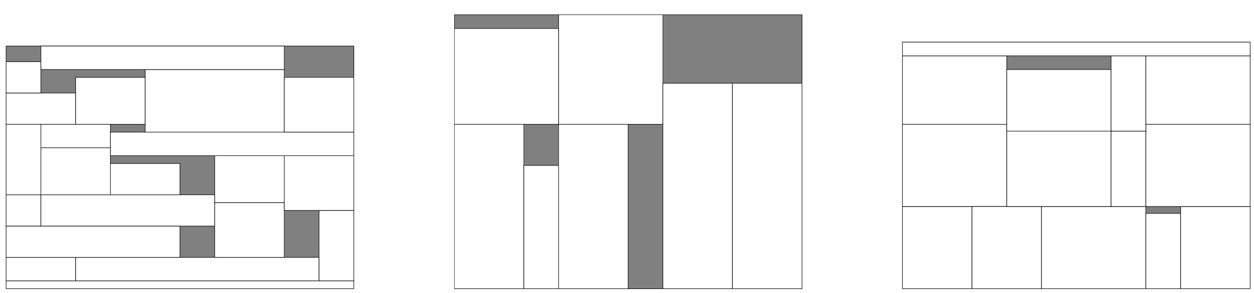


Figure B.7: Problems 19-21

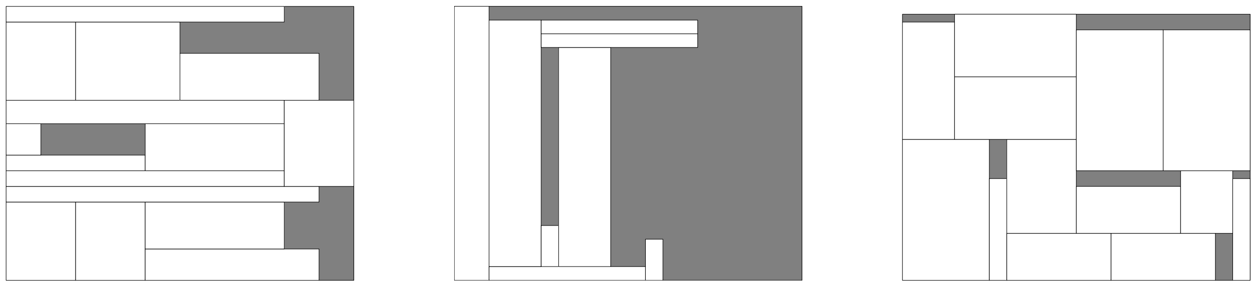


Figure B.8: Problems 22-24

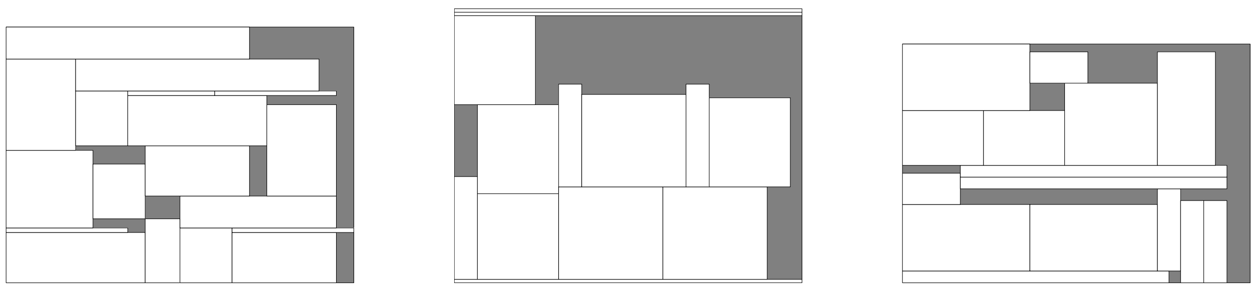


Figure B.9: Problems 25-27

B.2 Layouts for Strip Packing Problems with rotatable orientation with free cutting

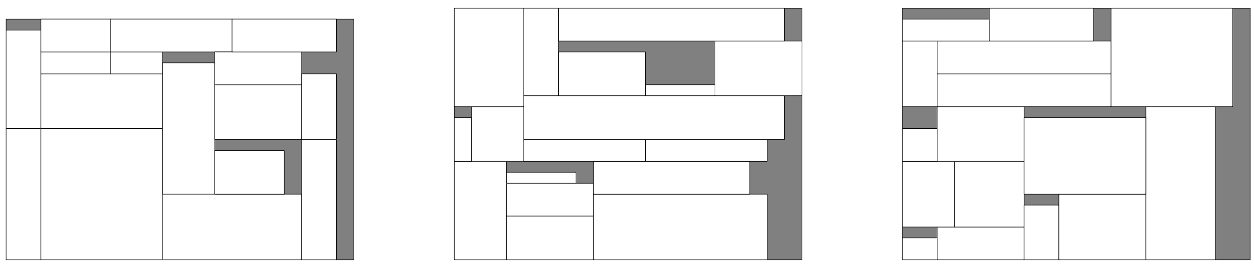


Figure B.10: Layouts for C1

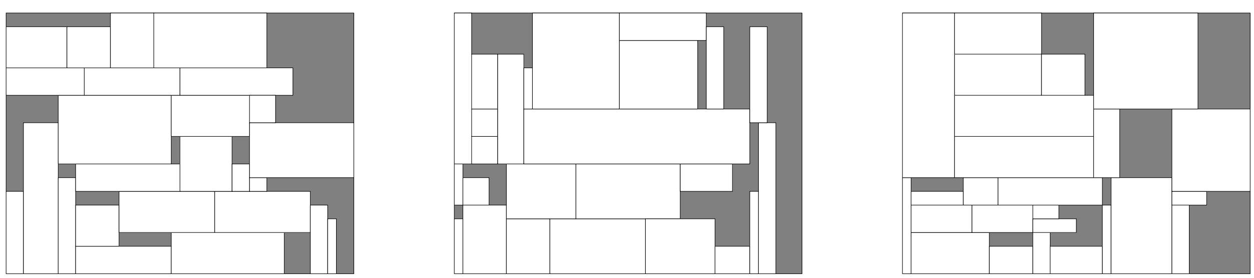


Figure B.11: Layouts for C2

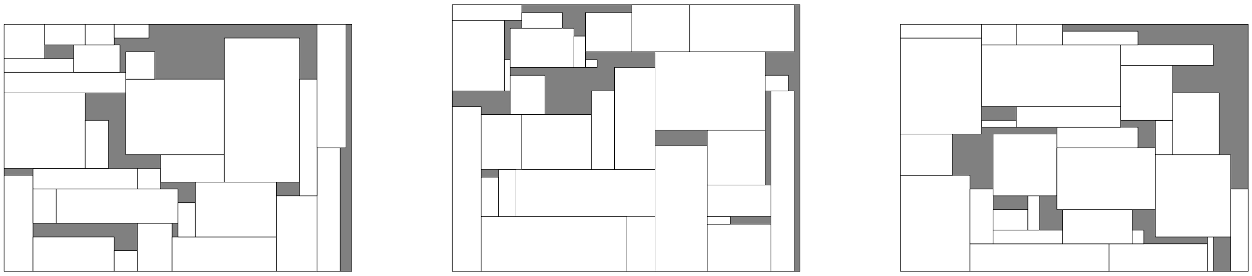


Figure B.12: Layouts for C3

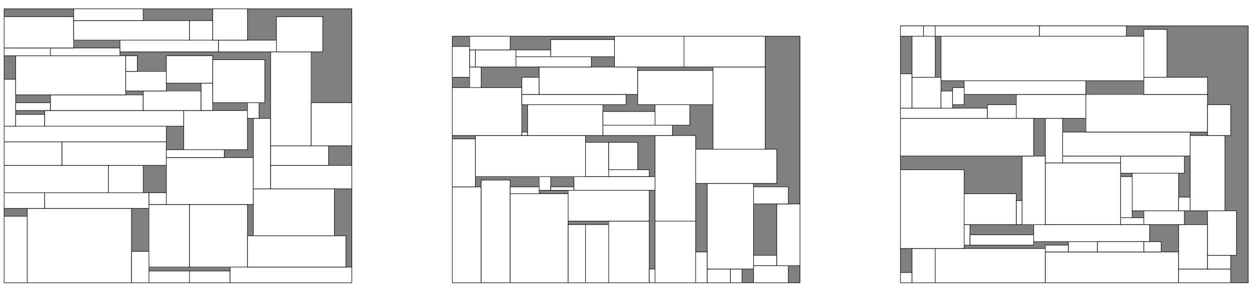


Figure B.13: Layouts for C4

Figure B.14: Layouts for C5

B.3 Layouts for guillotine-able Strip Packing Problems with rotatable orientation

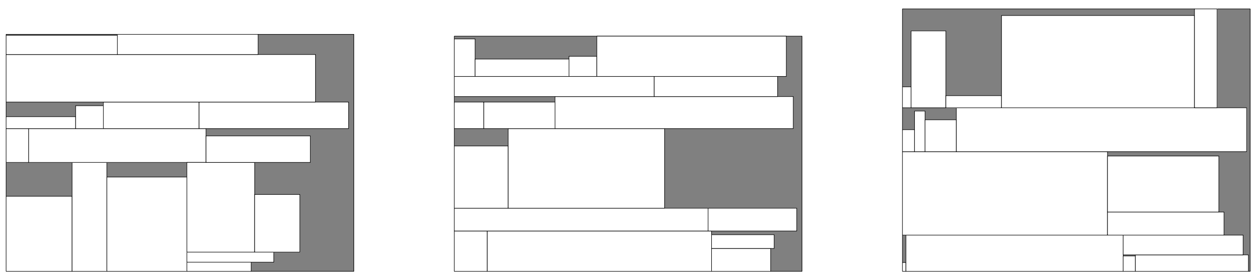


Figure B.15: Layouts for Guillotine-able Strip packing problems 1-3

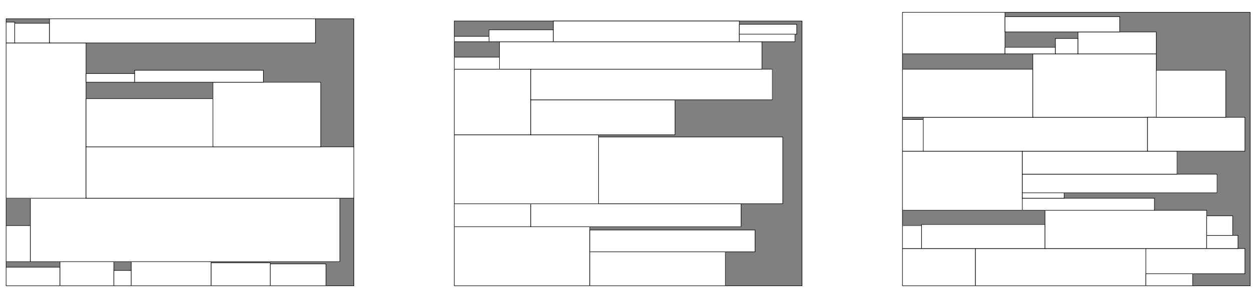


Figure B.16: Layouts for Guillotine-able Strip packing problems 4-6

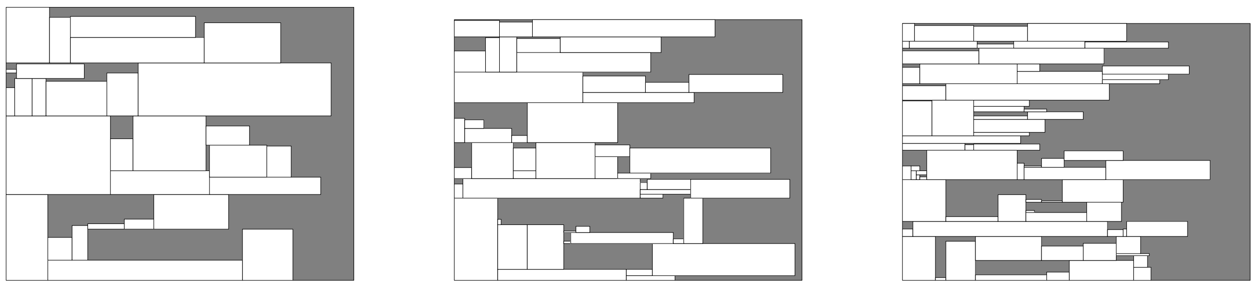


Figure B.17: Layouts for Guillotine-able Strip packing problems 7-9

B.4 Layouts for Bin Packing Problems with fixed orientation and free cutting

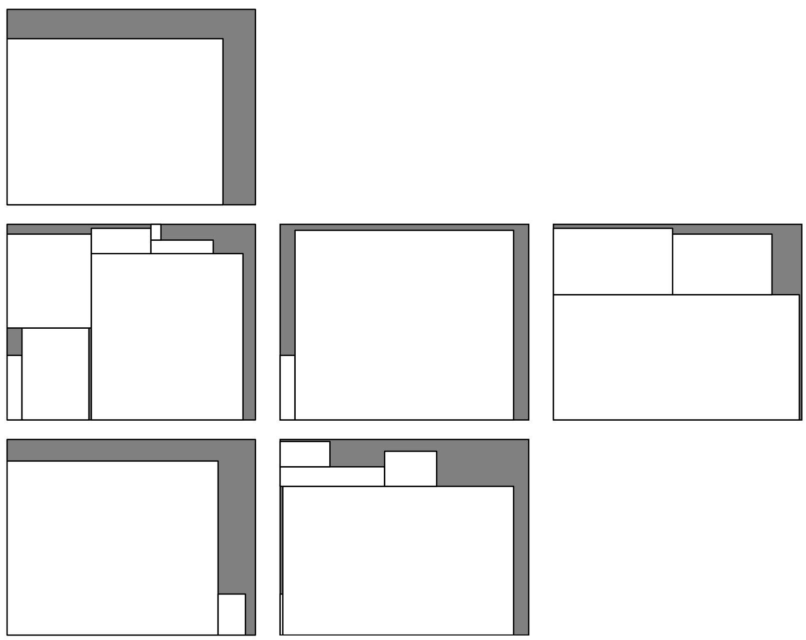


Figure B.18: Layout1 with 20 Items

Figure B.19: Layout2 with 40 Items

Figure B.20: Layout3 with 60 Items

B.5 Example Layouts for guillotine-able bin packing problems

Figure B.21: Example Layout for guillotine-able Bin Packing problem

Figure B.22: Example2 Layout for guillotine-able Bin Packing problem

B.6 Layouts for Irregular strip packing problem

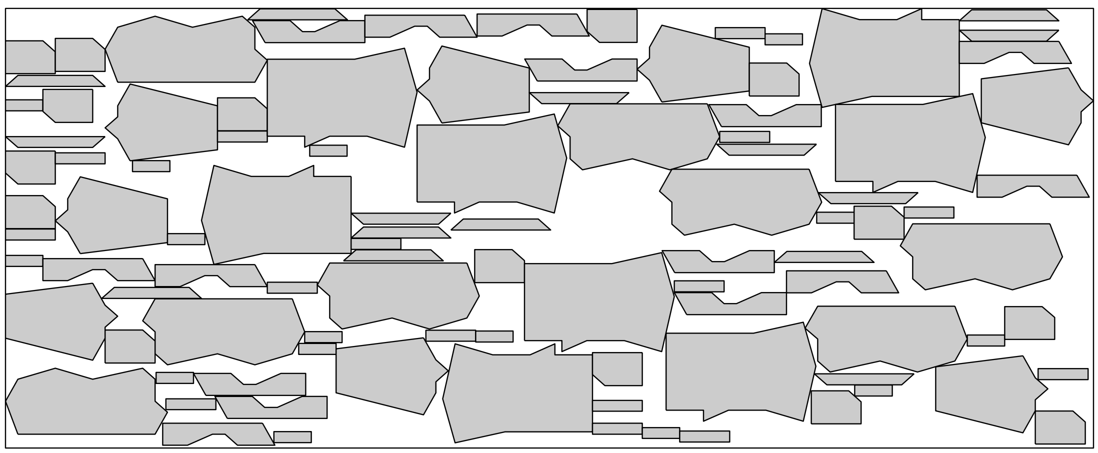


Figure B.23: Layout for Shirts

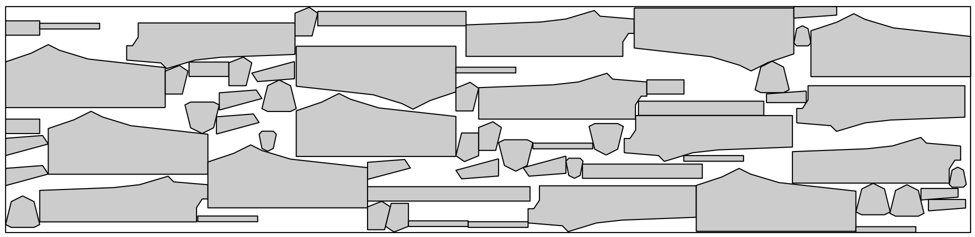


Figure B.24: Layout for trousers

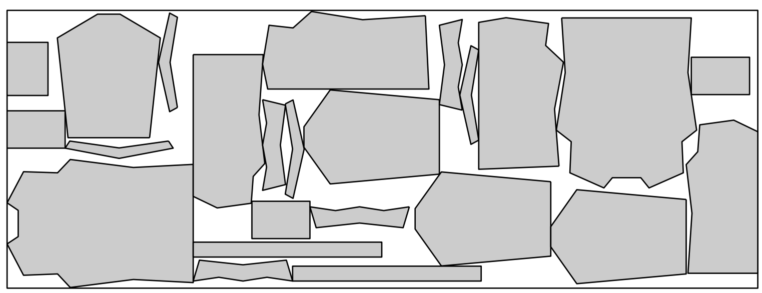


Figure B.25: Layout for Albano

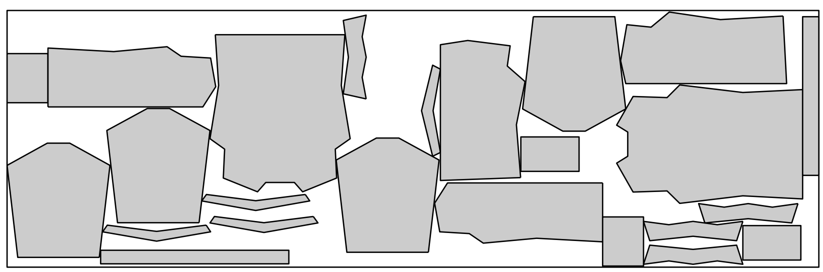


Figure B.26: Layout for Marques

```
 function pop=CreatePopulation(NVARS,FitnessFcn,options)

% This function creates a population of solutions for One Dimensional
%and Two Dimensional C&P Problems
%POP = CREATEPOPULATION(NVARS,FITNESSFCN,OPTIONS) creates a population
%  of solutions POP each with a length of NVARS.
%
%   The arguments to the function are
%     NVARS: Number of variables
%     FITNESSFCN: Fitness function
%     OPTIONS: Options structure used by the GA
% by V.Mancapa

% A Problem global variable assigend to the problem
% that has to be solved.
global Problem

Pop_size=sum(options.PopulationSize);
for i= 1:Pop_size
    %Create the ith individual
    pop(i,:)=CreateIndividual(Problem);
end
```

Figure B.27: Function for generaton of population of solutions

```
function xoverKids  =generalxover(parents,options,NVARS,FitnessFcn,unused,thisPopulat

%   generalxover Custom crossover function for 1D and 2D C&P problems.
%   XOVERKIDS = GENERALXOVER(PARENTS,OPTIONS,NVARS, ...
%   FITNESSFCN,UNUSED,THISPOPULATION) crossovers PARENTS to produce
%   the children XOVERKIDS.
%
%   The arguments to the function are
%     PARENTS: Parents chosen by the selection function
%     OPTIONS: Options structure used by the GA
%     NVARS: Number of variables
%     FITNESSFCN: Fitness function
%     THISPOPULATION: Matrix of individuals in the current population

%by V.Mancapa
nkids=length(parents)/2;
j=1;
for i=1:nkids
    %Select Parent1 From this Population
    parent1=thisPopulation(parents(j),:);
    j=j+1;
    %Select Parent2 From this Population
    parent2=thisPopulation(parents(j),:);
    j=j+1;
    %Cross Parent1 and Parent2
    xoverKids(i,:)=generalcrossover(parent1,parent2);
end
```

Figure B.28: Xover operator M-file function

```
function IndividualScore=Eval_function(soln)
%   A fitness function.
%  IndividualScore = EVAL_FUNCTION(SOLN) Calculate the fitness
%   of an individual for 1D and 2D C&P problems.
%By V.Mancapa




switch soln(1)
    case 1
        %Evaluation for one-dimensional bin packing problem
        true_solution=[soln(5:end)];
        IndividualScore=One_BPPEval(true_solution);

    case 2
        switch soln(2)
            case 1
                 switch soln(3)
                     case 2
       %Evaluation for 2D Strippacking problem without the guillotine constraint
                        true_solution=[soln(5:end)];
                        IndividualScore=StripPacking(true_solution);
                     case 1
        %Evaluation for 2D Strippacking problem with the guillotine constraint
                         true_solution=[soln(5:end)];
                         IndividualScore=Guillotine_StripPacking( true_solution);
                    end
            case 3
          %Evaluation for Irregular Strippacking problem
                true_solution=[soln(5:end)];
                IndividualScore=IrregularStripPacking(true_solution);

            case 2
                switch soln(3)
                    case 2
         %Evaluation for 2D Binpacking problem with free cutting
                            true_solution=[soln(5:end)];
                            IndividualScore=Two_Binpacking(true_solution);
                    case 1
          %Evaluation for 2D Binpacking problem with guillotine cutting constraint
                        true_solution=[soln(5:end)];
                         IndividualScore=Two_D_Binpacking_Guillotine(true_solution);
                    end

        end
end
```