
**IMPLEMENTING ARTIFICIAL
INTELLIGENCE SEARCH METHODS TO
SOLVE CONSTRAINED TWO-DIMENSIONAL
GUILLotine-CUT CUTTING STOCK
PROBLEMS**

by

**Jan Adriaan Oberholzer
M.Com., Hons. B.Com.**

Thesis submitted in fulfilment of the requirements for the degree

**DOCTOR PHILOSOPHIAE IN
COMPUTER SCIENCE**

at the

**POTCHEFSTROOMSE UNIVERSITEIT VIR CHRISTELIKE
HOËR ONDERWYS**

**PROMOTER: Prof T. Steyn
CO-PROMOTER: Prof. J.M. Hattingh**

Potchefstroom, November 2003




Potchefstroomse Universiteit
vir Christelike Hoër Onderwys

DECLARATION

I, Jan Adriaan Oberholzer, hereby declare on this 4th day of November 2003 that:

- i the work within this thesis is my own original work;
- ii all sources used or referred to have been documented and recognised; and
- iii this thesis has not been previously submitted in full or partial fulfilment of the requirements for an equivalent or higher qualification at any other recognised educational institution.


J. A. Oberholzer

I would like to express my thanks to:

My promoters, **Professors Tjaart Steyn and Giel Hattingh**, for their interest and guidance even at times when their own schedules were hectic;

My **parents** for their support, patience, understanding, love and enthusiasm. As always, you remain the only constant in my life. I love you!

Abstract

The main focus of this thesis will be on the *constrained two-dimensional guillotine-cut cutting stock (C2DGC) problem*. Stock cutting involves the process of cutting certain small demand items from a larger object. During this process, waste material is generated, which is called trim loss. The cutting stock problem presents itself in many industrial processes where the cutting of material is concerned, for instance the cutting of wood in the furniture industry, the cutting of glass and plastic sheets in the glass industry, the cutting of paper in the cardboard industry and the cutting of steel bars in metallurgy, to name but a few. The cutting stock problem aims to find one or more solutions to a cutting problem so that the optimal amount of the stock sheet is utilized. This, in turn, implies that the trim loss (waste) will be kept to a minimum.

Artificial intelligence search methods as well as existing exact C2DGC problem solution methods are investigated and evaluated critically. Different artificial intelligence search methods are then combined with the existing C2DGC problem solution methods, forming feasible algorithms to solve C2DGC problems. Existing C2DGC problem solution methods are also enhanced using innovative ideas. Numerical tests are then conducted to test the effectiveness and efficiency of each original and enhanced algorithm.

Die sentrale fokus van hierdie proefskrif is die sogenaamde *begrensde tweedimensionele guillotine-snit materiaalsny (C2DGC) probleem*. Materiaalsnyprobleme behels die sny van gegewe kleiner bestel-items vanuit 'n groter voorraadplaat. Materiaalsnyprobleme kom te voorskyn in verskeie industriële prosesse waar materiale gesny moet word. Voorbeelde hiervan is die sny van hout in die skrynwerkbedryf, die sny van glas en plastiek in die glasbedryf, die sny van papier in die kartonbedryf en die sny van staaltawe in die staalbedryf. In die oplossing van materiaalsnyprobleme word gepoog om een of meer oplossings vir 'n gegewe probleem te vind sodat die voorraadplaat optimaal benut word. Hierdie proses impliseer dat onbenutte dele op die voorraadplaat tot 'n minimum beperk sal word.

Kunsmatige intelligensie soekmetodes asook bestaande eksakte C2DGC probleem-oplosmetodes word ondersoek en krities geëvalueer. Verskillende kunsmatige intelligensie soekmetodes word dan met die bestaande C2DGC probleem-oplosmetodes gekombineer om uitvoerbare algoritmes te vorm waarmee C2DGC probleme opgelos kan word. Bestaande C2DGC probleem-oplosmetodes word ook verbeter deur middel van innoverende idees. Numeriese toetse word dan gedoen om die effektiwiteit en kwaliteit van elke bestaande en verbeterde algoritme te toets.

TABLE OF CONTENTS

Table of figures.....	viii
Table of tables.....	xi
Table of charts.....	xiv

CHAPTER 1: Introduction

1.1 Introduction.....	1
1.2 Problem statement.....	2
1.2.1 Exact methods.....	2
1.2.1.1 Problem areas concerning exact methods.....	3
1.2.2 Non-exact methods.....	4
1.2.2.1 Problem areas concerning non-exact methods.....	4
1.3 Scope of the work.....	5
1.4 Objectives of the study.....	6
1.4.1 Gaining an understanding of what artificial intelligence search methods are and how they function.....	6
1.4.2 Gaining an understanding of what the C2DGC problem models and algorithms entail.....	6
1.4.3 Developing algorithms that solve C2DGC problems.....	7
1.4.4 Investigate the effectiveness and efficiency of these algorithms.....	7
1.4.5 To develop an integrated software package implementing these algorithms.....	7
1.5 Methodology.....	7

1.6 Organization of the thesis.....	8
1.6.1 Chapter 1: Introduction.....	8
1.6.2 Chapter 2: Defining key concepts and terms.....	9
1.6.3 Chapter 3: Uninformed search methods.....	9
1.6.4 Chapter 4: Informed search methods.....	9
1.6.5 Chapter 5: C2DGC problem solution methods.....	9
1.6.6 Chapter 6: Algorithmic enhancements.....	10
1.6.7 Chapter 7: Numerical tests and results.....	10
1.6.8 Chapter 8: Conclusion.....	10

CHAPTER 2: Defining key concepts and terms

2.1 Introduction.....	11
2.2 Defining key concepts.....	11
2.2.1 Artificial intelligence.....	11
2.2.1.1 Thought processes and reasoning.....	12
2.2.1.2 Behaviour.....	12
2.2.2 Search methods.....	13
2.2.2.1 Uninformed search methods.....	13
2.2.2.2 Informed search methods.....	13
2.2.3 The C2DGC problem.....	14
2.2.3.1 Mathematical formulation.....	15
2.2.3.2 Problems related to the C2DGC problem.....	16
2.3 Previous research done in the field of 2DGC problems.....	17
2.4 Summary.....	18

CHAPTER 3: Uninformed search methods

3.1 Introduction	19
3.2 Evaluating different search methods	19
3.2.1 <i>Asymptotic notation</i>	20
3.3 Types of uninformed search methods	21
3.3.1 <i>Breadth-first search</i>	21
3.3.2 <i>Uniform-cost search</i>	25
3.3.3 <i>Depth-first search</i>	27
3.3.4 <i>Depth-limited search</i>	29
3.3.5 <i>Iterative-deepening depth-first search</i>	30
3.4 Final thoughts	30
3.5 Summary	31

CHAPTER 4: Informed search methods

4.1 Introduction	32
4.2 Types of informed search methods	32
4.2.1 <i>Generate-and-test</i>	33
4.2.2 <i>Hill climbing</i>	34
4.2.2.1 <i>The foothill problem</i>	36
4.2.2.2 <i>The ridge problem</i>	36
4.2.2.3 <i>The plateau problem</i>	36
4.2.3 <i>Dynamic hill climbing</i>	37
4.2.3.1 <i>Dynamic coordinate frame</i>	38
4.2.3.2 <i>Exploitation of local optima</i>	38
4.2.4 <i>Beam search</i>	39

4.2.5 Best-first search.....	41
4.2.6 Branch-and-bound search.....	43
4.2.7 Branch-and-bound search with underestimates.....	45
4.2.8 Branch-and-bound search using the dynamic programming principle.....	48
4.2.9 The A* search method.....	50
4.2.9.1 Admissibility of the A* method.....	50
4.2.9.2 Monotonicity of the A* method.....	50
4.2.9.3 Informedness and the A* method.....	51
4.2.9.4 Underestimation of h	51
4.2.9.5 Overestimation of h	52
4.2.9.6 Graceful decay of admissibility.....	53
4.3 Summary.....	54

CHAPTER 5: C2DGC problem solution methods

5.1 Introduction.....	56
5.2 Exact methods to solve the C2DGC problem.....	57
5.2.1 The Wang method.....	58
5.2.1.1 Rectangle building with the bottom-up approach.....	59
5.2.1.2 Trim loss.....	60
5.2.1.2.1 Internal trim loss.....	60
5.2.1.2.2 External trim loss.....	61
5.2.1.2.3 Total trim loss.....	62
5.2.1.3 Acceptable waste percentages (β).....	62
5.2.1.4 Wang's two original algorithms.....	62
5.2.1.5 Specifying values for β and optimality conditions.....	65
5.2.1.6 Solving sample C2DGC problem instances with Wang's method.....	66
5.2.1.6.1 Implementing Wang's method using breadth-first search.....	67
5.2.2 A computational improvement to Wang's algorithm one.....	75

5.2.2.1 Horizontal completeness.....	75
5.2.2.2 Vertical completeness.....	75
5.2.2.3 Dynamically diminishing the β value.....	76
5.2.3 <i>The modified Wang method (WAM)</i>	77
5.2.3.1 Trim loss.....	78
5.2.3.1.1 Estimated external trim loss.....	78
5.2.3.1.1.1 Unbounded two-dimensional knapsacks.....	83
5.2.3.1.2 Estimated total trim loss.....	87
5.2.3.2 Solving sample C2DGC problems with the modified Wang method (WAM) using the A* search method.....	87
5.3 Non-exact methods to solve the C2DGC problem.....	90
5.4 Exact methods' algorithmic properties and considerations.....	91
5.4.1 <i>Problems with the Wang method</i>	91
5.4.2 <i>Problems with the WAM method</i>	92
5.5 Summary.....	93

CHAPTER 6: Algorithmic enhancements

6.1 Introduction.....	94
6.2 Optimization techniques.....	95
6.2.1 <i>Detection of duplicate patterns: symmetric strategies</i>	95
6.2.1.1 Pattern coding.....	96
6.2.1.2 Pattern domination.....	98
6.2.1.3 Symmetric (duplicate) patterns on opposite directions.....	99
6.2.1.4 Symmetric (duplicate) patterns on the same direction.....	100
6.2.2 <i>Cutting order</i>	101
6.2.3 <i>Demand rectangle rotation</i>	102

6.3 Improving the lower bounds of the WAM method.....	103
6.3.1 <i>Partial stock sheet propagation (PSSP) method.....</i>	<i>111</i>
6.4 Upper bounds and the waste gap.....	123
6.4.1 <i>Upper bound propagation and the waste gap.....</i>	<i>124</i>
6.4.2 <i>Beam search, upper bounds and the waste gap.....</i>	<i>129</i>
6.5 Strategies for handling the value of beta (β).....	131
6.5.1 <i>Lower bound using the WAM lookup table.....</i>	<i>132</i>
6.5.2 <i>Increasing the value of beta (β).....</i>	<i>133</i>
6.6 Summary.....	136

CHAPTER 7: Numerical tests and results

7.1 Introduction.....	138
7.2 Numerical results.....	138
7.2.1 <i>AWA and AWAM algorithms versus DWA and DWAM algorithms.....</i>	<i>139</i>
7.2.2 <i>AWA and AWAM algorithms versus the MAWAM algorithm.....</i>	<i>145</i>
7.2.3 <i>AWA and AWAM algorithms versus A*WA and A*WAM algorithms.....</i>	<i>150</i>
7.2.4 <i>Adding symmetrical duplicate pattern removal to the A*WA and A*WAM algorithms.....</i>	<i>156</i>
7.2.5 <i>Partial stock sheet propagation (PSSP) algorithm.....</i>	<i>159</i>
7.2.6 <i>Normalized results.....</i>	<i>163</i>
7.2.7 <i>Increasing the beta (β) value.....</i>	<i>168</i>
7.2.8 <i>Industry-sized problem instances.....</i>	<i>171</i>
7.2.8.1 <i>PSSP algorithm with initial underestimations of zero.....</i>	<i>175</i>

7.3 Summary.....	178
-------------------------	------------

CHAPTER 8: Conclusion

8.1 Introduction.....	180
------------------------------	------------

8.2 Objectives of the study.....	182
-----------------------------------------	------------

8.2.1 Gaining an understanding of what artificial intelligence search methods are and how they function.....	182
-----------------------------------------------------------------------------------------------------------------	-----

8.2.2 Gaining an understanding of what the C2DGC problem models and algorithms entail.....	183
-----------------------------------------------------------------------------------------------	-----

8.2.3 Developing algorithms that solve C2DGC problems.....	183
------------------------------------------------------------	-----

8.2.4 Investigate the effectiveness and efficiency of these algorithms.....	184
-----------------------------------------------------------------------------	-----

8.2.5 To develop an integrated software package implementing these algorithms.....	184
---------------------------------------------------------------------------------------	-----

8.3 New research.....	184
------------------------------	------------

8.4 Further research.....	185
----------------------------------	------------

BIBLIOGRAPHY.....	187
--------------------------	------------

TABLE OF FIGURES

CHAPTER 2: Defining key concepts and terms

Figure 2.1: Two cutting patterns, where (a) is a guillotine pattern and (b) is not.....	15
------------------------------------------------------------------------------------------------	----

CHAPTER 3: Uninformed search methods

Figure 3.1: Showing that $f(n) = 8n + 128 = O(n^2)$	20
Figure 3.2: The order of node generation for breadth-first search.....	22
Figure 3.3: A breadth-first search algorithm.....	24
Figure 3.4: Uniform-cost search example: (a) The state space. (b) Progression of the search, each node labeled with a value $g(n)$	25
Figure 3.5: A uniform-search algorithm.....	27
Figure 3.6: The order of node generation for depth-first search.....	28
Figure 3.7: A depth-first search algorithm.....	29

CHAPTER 4: Informed search methods

Figure 4.1: Hill climbing: (a) The foothill problem. (b) The ridge problem. (c) The plateau problem.....	35
Figure 4.2: A hill climbing search algorithm.....	37
Figure 4.3: The order of node generation for beam search.....	39
Figure 4.4: A beam search algorithm.....	40
Figure 4.5: The order of node generation for best-first search.....	42
Figure 4.6: A best-first search algorithm.....	43

Figure 4.7: Node generation considerations for branch-and-bound search.....	44
Figure 4.8: A branch-and-bound search algorithm.....	45
Figure 4.9: Node generation for branch-and-bound search with underestimates.....	46
Figure 4.10: A branch-and-bound search algorithm using underestimates.....	47
Figure 4.11: The principal of dynamic programming.....	48
Figure 4.12: A branch-and-bound search algorithm using the dynamic programming principle.....	49
Figure 4.13: Underestimation of h.....	52
Figure 4.14: Overestimation of h.....	53
Figure 4.15: An A* search algorithm.....	54

CHAPTER 5: C2DGC problem solution methods

Figure 5.1: An illustration of the bottom-up rectangle building approach.....	59
Figure 5.2: Internal trim loss, as generated by horizontal and vertical builds.....	60
Figure 5.3: External trim loss.....	61
Figure 5.4: Wang's first algorithm (using β_1).....	63
Figure 5.5: Wang's second algorithm (using β_2).....	64
Figure 5.6: Partial representation of all stored nodes for example problem EP1 using breadth-first search combined with Wang's algorithm ($\beta = 0.24$).....	69
Figure 5.7: All stored nodes, in the sequence they are generated, for problem instance EP1 using breadth-first search and Wang's algorithm.....	71
Figure 5.8: Wang's method does not generate non-guillotine cuts.....	72

Figure 5.9: Partial representation of generated nodes for problem instance P5 using breadth-first search and Wang's algorithm ($\beta = 0.00$).....	74
Figure 5.10: Estimated external trim loss, generated by placing rectangles over the L section.....	79
Figure 5.11: Dividing the L section into different views.....	80
Figure 5.12: Partial representation of generated nodes for problem instance P5 using A* search and the modified Wang algorithm ($\beta = 0.00$).....	89
Figure 5.13: The modified Wang algorithm.....	90

CHAPTER 6: Algorithmic enhancements

Figure 6.1: Some constructed patterns, as entered into CList.....	97
Figure 6.2: The effect of cut ordering.....	101
Figure 6.3: Demand rectangle rotation.....	102
Figure 6.4: Example problem instance.....	103
Figure 6.5: An example generated rectangle.....	104
Figure 6.6: Propagated builds.....	105
Figure 6.7: An optimal solution for EP1.....	107
Figure 6.8: The PSSP algorithm.....	123

TABLE OF TABLES

CHAPTER 3: Uninformed search methods

Table 3.1: Time and space complexity examples for a problem where $b = 10$	24
--------------------------------------------------------------------------------------------	----

CHAPTER 5: C2DGC problem solution methods

Table 5.1: Set of nine C2DGC problems (P1-P8 as presented by Daza)...	56
Extract from table 5.1: Two problem instances (P5 and EP1) from table 5.1.....	67

CHAPTER 6: Algorithmic enhancements

Table 6.1: Original underestimates as generated by the unbounded knapsack function.....	108
Table 6.2: Maximum trim loss values for each dimension.....	108
Table 6.3: Maximum trim loss allowed by beta.....	109
Table 6.4: Internal trim loss of builds generated by the original Wang method when solving the simplified problem.....	109
Table 6.5: Altered values.....	110
Table 6.6: Altered underestimates.....	110
Table 6.7: Final updated underestimates.....	111
Table 6.8: Original underestimates as stored in array A.....	114
Table 6.9: Array B, initialization values.....	116

Table 6.10: Updated values of array B after running the original Wang method.....	117
Table 6.11: Further updated values in array B after the propagation algorithm was run.....	120
Table 6.12: Final values in array A after comparison with array B.....	122
Table 6.13: Initial upper bounds before propagation for P8 and beta = 0.00.....	125
Table 6.14: Propagated upper bounds for P8 and beta = 0.00.....	126
Table 6.15: Propagated upper bounds for P8 and beta = 0.01.....	128
Table 6.16: Least total trim loss of solution patterns using beam search.....	130
Table 6.17: Initial lower bound on beta (β).....	133
Table 6.18: Calculation of beta (β) increase fraction.....	136

CHAPTER 7: Numerical tests and results

Reference table 7.1: DWA, DWAM, AWA and AWAM algorithms.....	140
Table 7.1: Results for AWA and AWAM versus DWA and DWAM.....	141
Reference table 7.2: MAWAM algorithm.....	146
Table 7.2: Admissible heuristic function.....	147
Reference table 7.3: A*WA and A*WAM algorithms.....	152
Table 7.3: A* search algorithms.....	153
Reference table 7.4: SA*WA and SA*WAM algorithms.....	157
Table 7.4: A* search algorithms with symmetrical duplicate pattern removal.....	158
Table 7.5: PSSP algorithm results.....	160
Table 7.6: Results for other partial areas to solve the sub-problem.....	163
Table 7.7: Summary of results before normalization.....	165
Table 7.8: Normalized results using the AWA algorithm as the norm.....	167
Table 7.9: Constant versus fractional beta (β) value increases.....	170
Table 7.10: Set of four C2DGC problem instances from PG Glass Pty. Ltd.....	171

Table 7.11: Numerical results for larger problem instances.....	173
Table 7.12: PSSP algorithm with initial underestimates of zero.....	177

TABLE OF CHARTS

CHAPTER 6: Algorithmic enhancements

Chart 6.1: Change in total trim loss as the value of beta increases.....	134
--------------------------------------------------------------------------	-----

CHAPTER 7: Numerical tests and results

DWA, DWAM, AWA and AWAM algorithms

Chart 7.1: N Values for all problem instances.....	142
Chart 7.2: L Values for all problem instances.....	144

MAWAM algorithm

Chart 7.3: N Values for all problem instances.....	148
Chart 7.4: L Values for all problem instances.....	149
Chart 7.5: Execution times for all problem instances.....	150

A* search algorithms

Chart 7.6: N Values for all problem instances.....	154
Chart 7.7: L Values for all problem instances.....	155
Chart 7.8: Execution times for all problem instances.....	156

Chart 7.9: Scalability of the MAWAM method.....	174
-------------------------------------------------	-----

Chart 7.10: Scalability of the PSSP algorithm with initial underestimates of zero.....	178
-------------------------------------------------------------------------------------------	-----

CHAPTER 1: Introduction

"Enthusiasm without knowledge is like running in the dark."

- Fred Hatfield.

1.1 Introduction

According to Fred Hatfield, enthusiasm is not the sole driving force behind all success stories. In order to achieve a set goal, it is important to remember that being aware of a few facts concerning a specific problem are better than being aware of none at all. Therefore, always attempt to define your problem and the possible solutions as best you can before attempting to solve the problem itself. Hatfield states another simple truth in this famous quote and although it can be applied to many situations, it is particularly well suited to the field of Computer Science, and especially Artificial Intelligence. It clarifies the fact that no matter how eager a researcher is, without the proper scientific knowledge of a subject or field of study and a thorough comprehension of its principles, it is impossible to predict and realize its possibilities and potential.

Stock cutting involves the process of cutting certain small demand items from a larger object. During this process, waste material is generated, which is called trim loss. The cutting stock problem presents itself in many industrial processes where the cutting of material is concerned, for instance the cutting of wood in the furniture industry, the cutting of glass and plastic sheets in the glass industry, the cutting of paper in the cardboard industry and the cutting of steel bars in metallurgy, to name but a few. The cutting stock problem aims to find one or more solutions to a cutting problem so that the optimal amount of the stock sheet is utilized. This, in turn, implies that the trim loss will be kept to a minimum (Morabito & Garcia, 1998:469).

As the heading implies, this chapter guides the reader into the work by explaining the problem statement, the scope of the work, objectives of the

study and the methodology that was followed. After having read this chapter it should be clear what the work is all about and what to expect from it.

1.2 Problem statement

It is imperative to describe why it is necessary to further research the specific subject, namely the practical implementation of artificial intelligence search methods to solve constrained two-dimensional guillotine-cut cutting stock problems. From this point onward, the constrained two-dimensional guillotine-cut cutting stock problem will be referred to as the C2DGC problem, as it is used in the literature concerning cutting problems.

Morabito and Garcia (Morabito & Garcia, 1998:469-470) state that a large Brazilian hardboard industry generates waste material in their cutting process at an estimated amount of 20 tons per day. This translates to a financial loss of \$1 million per year because of good quality hardboard scraps that has to be discarded. These pieces are seen as useless for practical purposes because of their small size.

In the area of the nesting problem, which involves the packing of irregular shapes and is often used in the ship building industry, a European company (Esprit Automation Ltd.) has recently granted £50,000 to research concerning the development of more efficient algorithms that will provide solutions to nesting problems, resulting in less trim loss (Kendall, 2000:21).

It should be noted that methods do exist that can be implemented to solve C2DGC problems, and among these exact and non-exact methods can be identified.

1.2.1 Exact methods

A method is exact if it finds the highest-quality (optimal) solution when a problem has several different solutions. Christofides and Whitlock (1977),

Nilsson (1980), Wang (1983), Bagchi and Mahanti (1983), Pearl (1984), Sen and Bagchi (1989), Vasko (1989) and Oliveira and Ferreira (1990), amongst others, have done work concerning exact methods to solve C2DGC problems. This thesis and the research done for it deals with exact methods to solve constrained stock cutting problems, focusing on the method proposed by Wang (1983). Vasko (1989) and Oliveira and Ferreira (1990) made enhancements to the original method as proposed by Wang and these enhancements are also considered. Even though much work and research have been done in the C2DGC field, some problems still remain with respect to exact methods, which will be described shortly.

1.2.1.1 Problem areas concerning exact methods

Stock cutting problems, including C2DGC problems, are inherently difficult combinatorial optimization problems. An exponential explosion of possible search paths quickly materializes when solving all but the most trivial of textbook problems. This leads to a situation where the practicality of these methods are questioned where industry problems are concerned. For this reason, Wang proposed a method (which later became known as the Wang method) that utilizes a proportion parameter called beta (β) that is used to inhibit the exponential explosion of explored alternatives. It accomplishes this by not generating patterns containing more trim loss (waste) than is allowed by beta (β). This parameter prunes away significant portions of the problem search space when its value (possible range is $0 \leq \beta \leq 1$) is low, but even with this enhancement, larger problems still suffer from an exponential explosion of possible search paths. For this reason, Vasko introduced computational improvements to the Wang method, but the real revolution came when Oliveira and Ferreira introduced the modified Wang method (WAM method). The WAM method is still an exact one, but uses a heuristic function to lead the search more efficiently, thereby generating optimal solutions quicker. It was believed that this algorithm would be the answer to solve larger problem instances, but as will be shown, the

calculation of the values required by the WAM heuristic function becomes ineffective for larger, industry sized problems.

A second problem where the Wang method is concerned, is determining an initial value for beta. If the value is underestimated, it has to be increased and the search must then be undertaken again, resulting in unnecessary work being done. If the value is overestimated, the algorithm searches through unnecessary portions of the search space. The last problem is determining by what amount the beta value is to be incremented if the initial value was an underestimation.

1.2.2 Non-exact methods

Non-exact methods to solve stock cutting problems, including the C2DGC problem, exist in the form of heuristic search methods such as greedy searches, beam searches, depth-limited searches and hill-climbing searches (refer to chapter 3 and 4 for further details concerning these search methods).

1.2.2.1 Problem areas concerning non-exact methods

A great deal of uncertainty exists regarding the efficiency and effectiveness of algorithms based on these methods. Furthermore, it is difficult to evaluate the results given by the algorithms derived from the non-exact methods, as the solutions are not always optimal (the method does not guarantee that it finds the highest-quality solution if there are several different solutions). Non-exact methods are therefore not very well suited for academic research or scientific experimentation. For this reason, non-exact methods are only used here for algorithmic enhancements. For example, beam search will be used to calculate upper bounds for the Wang and WAM methods because it finds a solution fast, even if it is more often than not a non-optimal solution.

These facts demonstrate that the C2DGC problem is indeed a worthwhile and active topic with many opportunities for further research.

1.3 Scope of the work

Rigorous research efforts in the field of stock cutting have led to the emergence of a myriad of sub-problem areas. These research areas are all complex and multifaceted in their own right, and for this reason, definitive and concrete boundaries are placed on the scope of this study. The main focus of this work will be the *constrained two-dimensional guillotine-cut cutting stock problem*. The following describes the above-mentioned phrase:

- **Constrained:** An upper bound is placed on the number of each required demand rectangle size that can be cut from the stock sheet. Therefore, with a given set of demand rectangles of type r_i ($i = 1, 2, 3, \dots n$), each type will have a demand constraint of b_i (Viswanathan & Bagchi, 1993:768). This implies that the given problem will indicate the maximum number of demand b_i for rectangles of type r_i that may be cut from the stock sheet;
- **Two-dimensional:** This aspect of the problem implies that each demand rectangle type r_i , will have given dimensions (ℓ_i, w_i) for each i , where ℓ_i is the length and w_i the width of type r_i . Furthermore, these demand rectangles will be cut from a stock sheet of length L and width W (Fayard & Zissimopoulos, 1995:620). On the other hand, one-dimensional cutting problems consist of a stock sheet of length L and width W , and demand rectangles $r_1, r_2, r_3 \dots r_n$, where r_i represents the i th demand rectangle with length ℓ_i and width w_i (Gau & Wäscher, 1995:573). In three dimensional situations, an extra parameter is added to the two-dimensional cutting problem in the form of a third dimension;
- **Guillotine-cut:** According to Wang (Wang, 1983:573), guillotine cuts are obtained by only considering successive edge-to-edge cuts made

on the stock sheet and successively produced sheets. Cutting from one edge of the stock sheet to another is always required when cutting glass and almost always when cutting wood or thin metal (Christofides & Hadjiconstantinou, 1995:21); and

- *Cutting stock problem*: This is normally used as a generic term for the entire class of cutting and packing problems.

It is assumed that the constraints mentioned above are placed on the research criteria. These limits will therefore direct the research.

1.4 Objectives of the study

The next step is an attempt to define objectives for the research.

1.4.1 Gaining an understanding of what artificial intelligence search methods are and how they function

The first objective is essential, as in our context, this provides the theoretical background that is needed to solve cutting stock problems. Firstly, it shows that search methods are indeed a suitable means for representing and solving cutting problems. Secondly, it presents options available from which to choose the best-suited search methods for the problem.

1.4.2 Gaining an understanding of what the C2DGC problem models and algorithms entail

In section 1.3, a short description of the C2DGC problem was given, showing its complexity. For a clear and unambiguous understanding of the problem, however, an in-depth discussion is necessary. Firstly, this discussion defines the C2DGC problem thoroughly. Secondly, it highlights previous research efforts that provided useful results that are applicable to this study. Therefore, only by reaching this objective is it possible to continue with the

task of developing effective problem solving algorithms, and improve on previous results obtained by other researchers.

1.4.3 Developing algorithms that solve C2DGC problems

A very important part of this research includes developing working, efficient and preferably exact algorithms, which investigate the practicalities of the theoretical results. The best-suited search methods will be identified to use as the basis for these algorithms. From the execution of the algorithms, results are attained and recorded.

1.4.4 Investigate the effectiveness and efficiency of these algorithms

The results obtained from solving sample problems serve as a measure of how efficiently and effectively the algorithms performed, by comparing the results of the various approaches.

1.4.5 To develop an integrated software package implementing these algorithms

All the algorithms that were developed will be implemented in an integrated software package. This will demonstrate the effectiveness of different problem-solving algorithms.

1.5 Methodology

When planning a thesis of this nature, a comprehensive study of existing literature needs to be done, identifying all possible sources from which facts, statistics, data, diagrams and any other form of useful information pertaining to the subject and field of study can be obtained. These sources could be obtained in the university library by doing a comprehensive search on the computer database using the title of the thesis or key concepts in the title. The

sources used range from magazine articles, scientific journal articles, textbooks, interviews, Internet sites and electronic full-text databases.

The full-text Internet databases used for this study include:

- ScienceDirect at <http://sciencedirect.com/>; and
- Citeseer at <http://citeseer.nj.nec.com/>.

Resources used for the planning and development of the integrated software package includes software engineering coding standards as set forth by the Ellemtel Telecommunication Systems Laboratories¹. Borland C++ Builder is used as a coding platform. With the help of the integrated software package, empirical studies will be done, using textbook-sized problems as well as larger, industry-sized problems. This will help to determine how well certain problem solving methods scale when given larger problems to solve.

1.6 Organization of the thesis

In this section a description is given to explain the purpose of each chapter and its structure.

1.6.1 Chapter 1: Introduction

The first chapter discusses the problem statement, objectives of the study, methodology and the organization of the thesis.

¹ Copyright © 1990-1992 by Ellemtel Telecommunication System Laboratories
Box 1505
125 25 Älvsjö
Sweden
Tel: international extension + 46 8 727 30 00

1.6.2 Chapter 2: Defining key concepts and terms

The key terms necessary for fluently reading and understanding the thesis are defined in this chapter. Definitions for concepts such as artificial intelligence, search methods and the C2DGC problem will be given. These are standard definitions as it is used in the field of stock cutting and artificial intelligence research areas and communities.

1.6.3 Chapter 3: Uninformed search methods

The main goal of this chapter is to introduce the reader to search methods. These methods are very basic and form the building blocks from which informed search methods are constructed. Problem solving search methods are essential for writing algorithms that have to make decisions by finding sequences of actions that lead to desired states.

1.6.4 Chapter 4: Informed search methods

The concepts introduced in chapter 3 are expanded upon in chapter 4, where informed search methods are discussed. As the name implies, *informed* search methods utilize domain specific knowledge about a problem to guide a search in the correct direction.

1.6.5 Chapter 5: C2DGC problem solution methods

Existing exact and non-exact C2DGC problem solving methods are discussed in this chapter. These methods are analyzed and problems are identified with algorithmic implementations of these methods. A list of possible problems to investigate is given, and these will be looked at in further chapters.

1.6.6 Chapter 6: Algorithmic enhancements

Chapter 6 introduces modifications to the existing algorithmic implementations of the Wang and modified Wang methods. These modifications aim at enhancing the performance of these algorithms and also at eliminating the problems identified in chapter 5. These enhancements include optimization techniques, beta handling strategies for the Wang and modified Wang methods, calculating sharper lower bounds and explicitly managing upper bounds.

1.6.7 Chapter 7: Numerical tests and results

By solving standard problem instances with the algorithms discussed in chapters 5 and 6, and implementing the enhancements made in chapter 6, numerical results will be obtained and these results will be discussed in chapter 7.

1.6.8 Chapter 8: Conclusion

The last chapter summarizes the goals set forth for the study and how these goals were achieved. Furthermore, new problems that arose, which falls outside of the scope of this study, and opportunities that presented itself during the study are discussed to outline possible ideas that can be used as the basis for further study in this field. Lastly, the new research and newly developed algorithms are highlighted, which shows the contributions of the study.

CHAPTER 2: Defining key concepts and terms

2.1 Introduction

In this chapter the key terms and concepts used throughout this thesis are defined. In section 2.2, the terms artificial intelligence, search methods and the C2DGC problem are defined. Descriptions of related fields of study within the cutting and packing (CP) problem environment are also given, with supplied definitions for each field. Section 2.3 discusses previous research done in the field of C2DGC problems, and section 2.4 summarizes the contents of the chapter.

2.2 Defining key concepts

If the title of this thesis is carefully studied, three terms are identified that need to be defined. The first, and most obvious, is the term *artificial intelligence*. Secondly, the term *search methods* are described and lastly the *C2DGC problem* is defined.

2.2.1 Artificial intelligence

Winston defines the concept of artificial intelligence very broadly, and states that it is “the study of ideas which enable computers to do things that make people seem intelligent” and continues in stating “the central goals of artificial intelligence are to make computers more useful and to understand the principles which make intelligence possible” (Winston, 1977:1).

Although it is a general definition, the idea of using artificial intelligence as a problem-solving tool becomes apparent when Winston’s description of the concept is considered. This attribute of artificial intelligence provides a means by which cutting problems may be solved.

An in-depth definition can be found in the book *a Modern approach to artificial intelligence* by Russel and Norvig (Russel & Norvig, 1995:4), in which they state that there are two major paths taken when defining artificial intelligence; the first being *thought processes and reasoning*, whereas the second deals with *behaviour*.

In short, Russel and Norvig (Russel & Norvig, 1995:5) divide the definitions into the two above-mentioned categories. These two categories will then be subdivided, each into two more categories, which will then form four basic definitions:

2.2.1.1 Thought processes and reasoning

They state that where thought processes and reasoning are concerned, systems can be developed that *think like humans* and systems that *think rationally*.

In this category a few definitions have been introduced by different authors, some of the best-known being: "*The study of mental faculties through the use of computational models.*" (Charniak and McDermott, 1985) and "*The exciting new effort to make computers think... machines with minds, in the full and literal sense.*" (Haugeland, 1985).

2.2.1.2 Behaviour

Furthermore, Russel and Norvig (Russel & Norvig, 1995:5) distinguish between systems that *act like humans* and systems that *act rationally*.

Once again reference has been made to different authors' definitions, which reads: "*The art of creating machines that perform functions that require intelligence when performed by people.*" (Kurzweil, 1990) and "*A field of study which seeks to explain and emulate intelligent behaviour in terms of computational processes.*" (Schalkoff, 1990).

2.2.2 Search methods

Search methods are universal problem solving mechanisms, used in the field of artificial intelligence. The steps required to solve problems that are addressed by search methods, which include cutting stock problems, are not known in advance, and have to be determined by a systematic trial-and-error exploration of alternatives (Korf, 1996:1).

The alternatives generated by the trial-and-error process, can be viewed as part of the state space, which consists of an initial state, the alternative states, and a goal state. The problem is therefore reduced to reaching the goal state from the initial state. For this reason, searches undertaken by search methods are often referred to as state space searches (Bundy, 1997:115).

Two forms of search methods that are useful in the study of cutting problems are uninformed and informed search methods.

2.2.2.1 Uninformed search methods

Uninformed search methods, also called brute-force or blind search methods, require no domain specific knowledge to function (Korf, 1996:1). Although uninformed search methods are an impractical tool for solving nontrivial problems (problems where the state space is too large to consider every possibility), it serves as an invaluable tool in describing the basic ideas behind informed search methods (discussed in chapter 4).

2.2.2.2 Informed search methods

Informed search methods, also referred to as heuristic search methods, implement rules that expand nodes in the state space that are most likely to lead to an acceptable problem solution. These rules, used for decision-making, are based on domain specific knowledge of the state space.

Unfortunately, like all rules of discovery and invention, heuristics are fallible. A heuristic¹ is only an informed guess of the next step to be taken in solving a problem (Luger & Stubblefield, 1993:116-117).

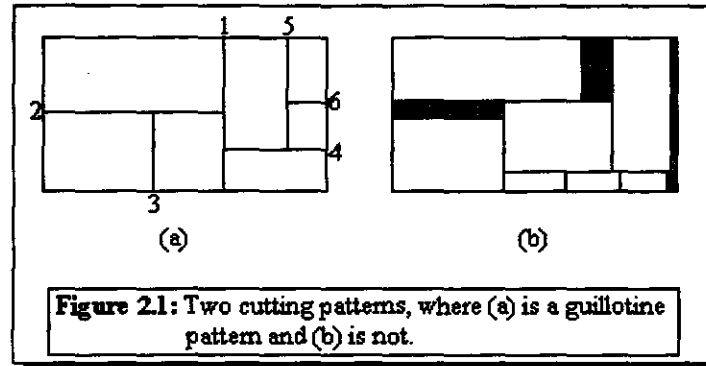
2.2.3 The C2DGC problem

The C2DGC problem belongs to a well-known family of problems called CP. These problems are natural combinatorial optimization problems, as found in the fields of Computer Science, Industrial Engineering, Logistics, Manufacturing, etc. (Cung et al., 2000:186).

The C2DGC problem forms that part of the CP problem domain where rectangles are cut from a rectangular stock sheet, with the aim of minimizing the total trim loss (waste material) generated by the cutting process. The size of the problem domain is fortunately reduced by two constraints that are placed on feasible cuts that have to be explored as possible solutions. The first constraint is an upper bound assigned to every demand rectangle, which is the number of times a certain demand rectangle type can be cut from the stock sheet. The second constraint is that all cuts have to be guillotine cuts, which are made from the one edge of the stock sheet to the other and are parallel to the edges of the stock sheet (Daza et al., 1995:633).

Figure 2.1, adapted from Christofides and Hadjiconstantinou (Christofides & Hadjiconstantinou, 1995:22), illustrates the difference between a guillotine and a non-guillotine cutting pattern. In the figure, pattern (a) is a guillotine cutting pattern because all cuts are edge-to-edge cuts and parallel to the edges of the stock sheet or successively produced sheets. Pattern (b), on the other hand, does not conform to the definition of a guillotine cutting pattern.

¹ Refer to chapter 4, section 4.1, page 32, for information pertaining to heuristics



The cuts in figure 2.1 (a) are numbered in the order in which they could be made, although other sequences are obviously also possible.

Due to the combinatorial characteristics of the C2DGC problem, it is possible to represent the search space as a formal tree structure, which integrates seamlessly with artificial intelligence search methods. Artificial intelligence search methods offer the capability of constructing these tree-like search spaces facilitating searching through it for possible optimal solution patterns.

2.2.3.1 Mathematical formulation

Let S be a stock sheet of length L and width W , and let R be a set of demand rectangles of type r_i ($i = 1, 2, 3, \dots, n$), where each type will have a demand constraint of b_i , a length of l_i and a width of w_i . From this, the guillotine cutting pattern with a minimum trim loss must be determined that uses no more than b_i replicates of demand rectangle r_i ($i = 1, 2, 3, \dots, n$) (Wang, 1983:574). The problem statement, as adapted from Wang (Wang, 1983:574), can also be stated in the form

$$\begin{array}{ll} \text{Maximize}_G & \sum_{i=1}^n x_i l_i w_i \\ \text{Subject to} & 0 \leq x_i \leq b_i \\ & x_i \text{ integer} \quad (i = 1, 2, \dots, n) \end{array}$$

where x_i is an integer indicating the number of times the demand rectangle r_i appears in the guillotine cutting pattern G under consideration.

2.2.3.2 Problems related to the C2DGC problem

As noted in chapter 1, rigorous research efforts in the field of stock cutting have led to the emergence of a myriad of modeling areas. These will shortly be described, as knowledge obtained from research in some of these fields will be used and referred to in this thesis.

- *Assortment problem:* Assortment problem solutions intend to minimize the number or area of stock sheets used by optimally placing given demand rectangles. Solutions to the problem are applied in the industry, for example solving cutting stock problems of rectangular steel bars (Li & Tsai, 2001:1245-1246). According to Baker (Baker, 1999:84), it is not possible to place all required demand rectangles on a single stock sheet, and therefore solution algorithms have to decide which rectangles to hold and which to use to reach optimal cutting patterns spanning two or more stock sheets.
- *Bin-packing:* Bin-packing problem solutions attempt to partition or pack a certain number of objects into a minimum number of bins (Chao et al., 1995:133). Problems where all the objects must be loaded into the bin are differentiated from those where some objects might be left out of the solution. The first type of problem is referred to as the three-dimensional bin-packing problem and the second type is known as the three-dimensional knapsack problem (Bortfeldt & Gehring, 2001:143).
- *Knapsack problem:* The knapsack problem often presents itself as a relaxation methodology in one-, two-, and three-dimensional cutting and packing problems. When given demand rectangles need to be cut from a stock sheet, or objects need to be optimally placed in a container or bin, the knapsack approach may be used to help solve

the problems (Fayard & Zissimopoulos, 1995:618). This problem can be viewed as a bin-packing problem when some additional constraints are placed on the value assigned to each object (Kendall, 2000:12).

- *Nesting problem*: The two-dimensional nesting problem concerns itself with the optimal placement of irregularly shaped *stencils* onto an irregularly shaped *surface*. The problem usually presents itself in the fabric and clothing industry. Some constraints are placed on the *stencil* placement, for instance on the allowable degrees of stencil rotation and on the area of placement if the fabric contains any patterns (Heckmann & Lengauer, 1998:473).

2.3 Previous research done in the field of 2DGC problems

Numerous approaches have been proposed to solve both constrained and unconstrained two-dimensional guillotine-cut cutting stock problems (2DGC), and the techniques used range from dynamic programming and linear programming to recursive procedures, incremental development algorithms and artificial intelligence search methods.

According to Cung (Cung et al., 2000:186), the study of cutting problems started nearly sixty years ago, but during the last ten years a renewed interest in the field has sparked the imaginations of numerous researchers and it has led to the development of new problem solving models.

The unconstrained two-dimensional guillotine-cut cutting stock (U2DGC) problem was extensively studied by Gilmore and Gomory (1965, 1967) and Beasley (1985). They implemented dynamic programming and linear programming methods to solve the U2DGC problem, where the number of times a specific demand rectangle is allowed to be used in the cutting pattern is unlimited. Herz (1972) solved the same problem by using recursive search procedures. Hinxman (1976) and later Morabito, Arenales and Arcaro (1992)

used the problem reduction methodology to solve unconstrained cutting problems.

Christofides and Whitlock (1977) used the results obtained by Gilmore and Gomory (1967) and a classical transportation problem to devise an exact tree search algorithm to solve the C2DGC problem. This solution implemented dynamic programming procedures to solve the constrained cutting problem. Wang (1983) proposed and implemented two incremental development algorithms to solve the C2DGC problem, and Vasko (1988) and Oliveira and Ferreira (1990) improved upon some aspects of these algorithms.

Viswanathan and Bagchi (Viswanathan & Bagchi, 1993:768) state that artificial intelligence search methods and their application to constrained cutting problems have been studied extensively by Nilsson (1980), Bagchi and Mahanti (1983), Pearl (1984) and Sen and Bagchi (1989). Viswanathan and Bagchi proposed the use of the best-first search method to solve the C2DGC problem. Furthermore, Held and Karp (1971) proposed an implementation of the travelling salesperson problem to solve the C2DGC problem.

2.4 Summary

This chapter introduced the reader to the C2DGC problem, and also discussed the basic principles behind artificial intelligence search methods. Chapter 3 continues with an in-depth study of uninformed search methods, which lays the foundation for the discussion of informed search methods in chapter 4. As was stated in section 2.2.3, artificial intelligence search methods are ideally suited for searching through C2DGC search spaces (represented as tree structures).

CHAPTER 3: Uninformed search methods

3.1 Introduction

This chapter deals with uninformed artificial intelligence search methods. These methods are discussed by explaining how to implement them. Furthermore, the methods are evaluated with certain criteria as listed in section 3.2. Section 3.3 introduces and discusses uninformed search methods, and sections 3.4 and 3.5 conclude the chapter with final thoughts and a summary of the chapter content.

3.2 Evaluating different search methods

A simple yet effective criterion to measure the effectiveness and efficiency of search methods has been proposed by Russel and Norvig (Russel & Norvig, 1995:73). The majority of work in search methods has gone into finding the most effective search method for a given problem. To aid researchers in the evaluation process of the different search methods, the following four criteria can be used:

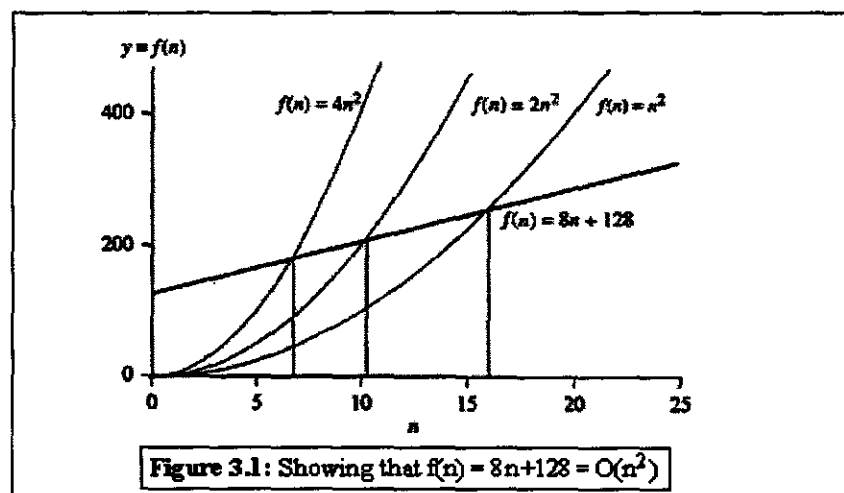
- *Completeness*: is the method guaranteed to find a solution when one exists for the problem?
- *Time complexity*: how long does it take to find a solution?
- *Space complexity*: how much memory does it need to perform the search?
- *Exactness*: does the strategy find the optimal solution when there are several different solutions?

The results obtained for these four measures when the method is tested, are all important factors to consider when deciding on a search method for a specific problem. If, for instance, a method's time complexity is acceptable (the method finds a solution within an acceptable period of time) but its space complexity is infeasible (the method cannot find a solution because not

enough physical memory is available) for many problem instances, the method is not an acceptable tool for solving the problem. Furthermore, it can be logically deduced that if a method is exact it is also complete, but if it is complete it is not necessarily exact (Russel & Norvig, 1995:73).

3.2.1 Asymptotic notation

According to Preiss (Preiss, 1999:36-37), P. Bachman devised a notation in 1982, which describes the asymptotic behaviour of functions. This notation later became known as the *big oh* (O) notation. Big oh notation is used to describe the asymptotic upper bound of functions, with omega notation describing a lower bound. The big oh notation will be used for the evaluation of worst-case time-complexities of search methods.



The following defines big oh notation mathematically:

Consider a function $f(n)$ that is non-negative for all integers $n \geq 0$. It is said that " $f(n)$ is big oh $g(n)$," which is written as $f(n) = O(g(n))$, if there exists an integer n_0 and a constant $c > 0$ such that for all integers $n \geq n_0$, $f(n) \leq cg(n)$.

Preiss (Preiss, 1999:36-37) gives an illustrative example that demonstrates the functioning of big oh. When the function $f(n) = 8n + 128$ is considered, it is shown in figure 3.1 that $f(n)$ is non-negative for all integers $n \geq 0$. To show that the asymptotic complexity of the function $f(n)$ is $O(n^2)$, it is necessary, according to the definition of big oh, to find an integer n_0 and a constant $c > 0$ such that for all integers $n \geq n_0$, $f(n) \leq cn^2$. It does in fact not matter what these constants are, as long as they exist.

Working with the function $f(n)$ mentioned above while the value of 1 is chosen for the constant c , we have:

$$\begin{aligned} f(n) \leq cn^2 &\rightarrow 8n + 128 \leq n^2 \\ &\rightarrow 0 \leq n^2 - 8n - 128 \\ &\rightarrow 0 \leq (n - 16)(n + 8) \end{aligned}$$

Since $(n+8) > 0$ for all values of $n \geq 0$, it can be deduced that $f(n) \leq cn^2$ if $(n - 16) \geq 0$. Therefore n_0 can be chosen as 16. It is thus clear that for $c = 1$ and $n_0 = 16$, $f(n) \leq cn^2$ for all integers $n \geq n_0$. From this follows that $f(n)$ is $O(n^2)$. It is indeed possible to further prove a stronger result for this function $f(n)$ so that $f(n) = O(n)$.

3.3 Types of uninformed search methods

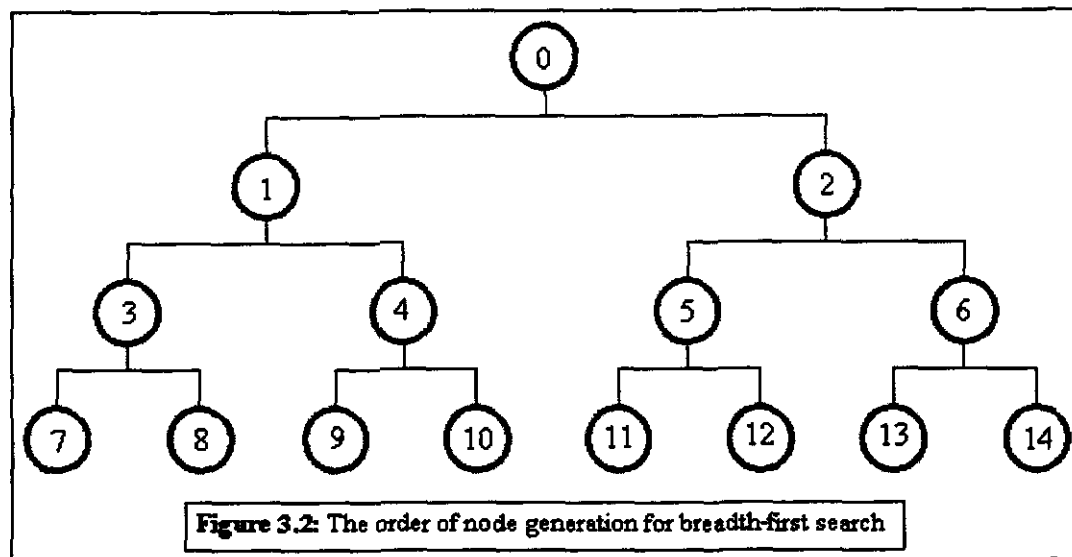
The following is a discussion of different types of uninformed search methods. The methods are evaluated according to the four criteria described in section 3.2.

3.3.1 Breadth-first search

Barr (Barr, 1981:47) defines breadth-first search as a "method that expands nodes in the order of their proximity to the start node, measured by the number of arcs between them". In other words, the start node is expanded first, and then all nodes expanded from the start node are expanded next,

and so on. All nodes on the n 'th level of the search tree are expanded before the algorithm moves on to the level $n+1$.

Breadth-first search is therefore a strategy that systematically searches through all possible nodes on level one of the search tree, and only then it moves on to the second level. Figure 3.2 shows how breadth-first search expands nodes in a simple binary tree. When a solution for a problem exists, breadth-first search will always find the shallowest goal node first. In terms of the four criteria (according to section 3.2, page 19), breadth-first search is complete, and it is exact *provided the path cost is a non-decreasing function of the depth of the node* (Russel & Norvig, 1995:74).



According to Bundy (Bundy, 1997:27), the breadth-first search method is admissible¹, because of the fact that the strategy is guaranteed to terminate with the minimal cost path to the goal node.

At this point it might seem as if breadth-first search is indeed a practical and acceptable method for problems that require state-space searches. When

¹ An admissible method guarantees to find a solution path of minimal cost for any problem instance if any solution path exists. Chapter 4, page 50, introduces a formal definition of the concept of admissibility.

the time and space complexity of this method is evaluated, however, their impeding influences are revealed. The main reason for excessive time and memory usage is the branching factors encountered by almost all non-trivial problems. This can be illustrated with a state space where each state can be expanded to yield b new states, where b is referred to as the branching factor. Suppose now that the problem's solution has a solution depth of d , then the maximum number of nodes that will be expanded before the solution is reached, is:

$$G = 1 + b + b^2 + b^3 + \dots + b^d \quad (3.1)$$

The result obtained from equation 3.1 is the *maximum* number of nodes that could be expanded, but the possibility to find the goal node before reaching the last node of the final layer is high. This implies that the actual number of expanded nodes will most likely be less than G . The asymptotic time and space complexity of the breadth-first search algorithm can therefore be expressed as $O(b^d)$.

According to Ciesielski (Ciesielski, 2001), under some quite reasonable assumptions the space and time complexity at different depths of the search tree for a problem with a branching factor of 10 can be summarized as in table 3.1. The branching factor value of 10 represents that of a reasonable C2DGC problem's branching factor. The depth of a search tree for a C2DGC problem can easily reach a depth of 10 or more, and table 3.1 shows that at least 1 terabyte of memory and 128 days of processing time is required to solve it at depth 10. The values for table 3.1 were calculated assuming that 100 bytes of memory are used for storing a single node and that 1000 nodes can be expanded per second.

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	1 second	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

Table 3.1: Time and space complexity examples for a problem where $b=10$

```

/* OPEN and CLOSED are lists */

OPEN = Start node, CLOSED = empty.

While OPEN is not empty do
{
  Remove leftmost node from OPEN, call it X.

  If X is a goal
    return success.
  else
  {
    Generate children of X.
    Put X on CLOSED.
    Eliminate the children of X already on
    OPEN or CLOSED.
    Put remaining children of X on the right
    end of OPEN.
  }
}

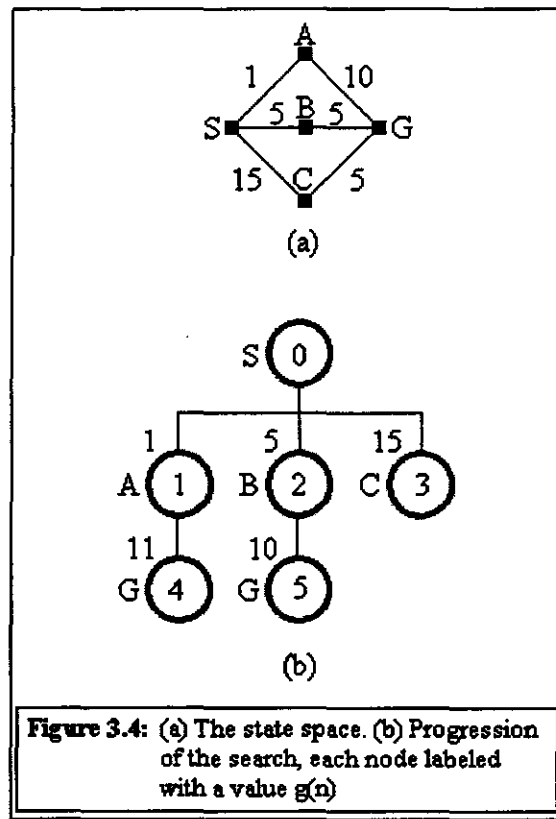
```

Figure 3.3: A breadth-first search algorithm

To conclude the discussion on breadth-first search, a general breadth-first search algorithm is given in figure 3.3.

3.3.2 Uniform-cost search

A slightly modified version of the breadth-first search algorithm results in the uniform-cost search method. According to Barr (Barr, 1981:47), uniform-cost search will always find the “cheapest path from the start state to the goal state.” The cheapest path implies that the solution will not necessarily find the shortest solution path, but the least-cost solution path. An important factor for uniform-cost algorithms is that a nonnegative cost must be associated with every path (arc) joining two nodes in a search tree. The algorithm produces a pure breadth-first search when the costs associated to all nodes are equal. Russel and Norvig (Russel & Norvig, 1995:75) depict an instance of the systematic functioning of the uniform-cost strategy graphically as in figure 3.4.



The figure shows the state space in (a), with the cost of each path (arc) associated with it. The problem is to traverse from the start state S to the goal state G and to minimize the path cost. As shown in (b) in figure 3.4, the first iteration of the algorithm expands the initial state, yielding the nodes A , B and C . The costs of these nodes are evaluated and the least-cost node, which is A in this instance, is expanded next. Once A is expanded, the path SAG is generated with a cost of 11. Since SAG represents a path from the start state to the goal state, it is a solution path, but is not yet recognized as the best solution because nodes B and C have not yet been considered for expansion. The next step is to expand node B , which generates SBG with a cost of 10. The only incomplete path left is SC with a path cost of 15 before expansion. Therefore, the path SBG is the cheapest possible solution path and is therefore the optimal solution.

The uniform-cost search method is also known as the Dijkstra single-source shortest-path algorithm. The method is complete and it also guarantees that whenever a node is expanded, a lowest-cost path to that node has been found (exactness) *provided the path cost is a non-decreasing function of the depth of the node* (Russel & Norvig, 1995:76). The worst-case asymptotic time complexity of uniform-cost search is $O(b^{c/m})$, where c is the cost of an optimal solution and m is the minimum cost arc within the state space. The algorithm unfortunately requires the same order of memory as breadth-first search algorithms to solve problems, in other words $O(b^d)$. (Korf, 1996:7).

Figure 3.5 gives a uniform-cost algorithm, which closely resembles the breadth-first search algorithm in figure 3.3. One difference is that whenever a new node is selected that will be expanded, the algorithm does not select the node sequentially. It directs the selection process by evaluating the costs of the currently expanded child nodes, and selecting the minimum value.

In the uniform-cost algorithm given below, the cost of the path from node X to node X_s is denoted by $c(X, X_s)$. The cost of a path from the start node to any

node X (or in other words the cost of node X) is denoted by $g(X)$. The cost of the start state will be set to zero.

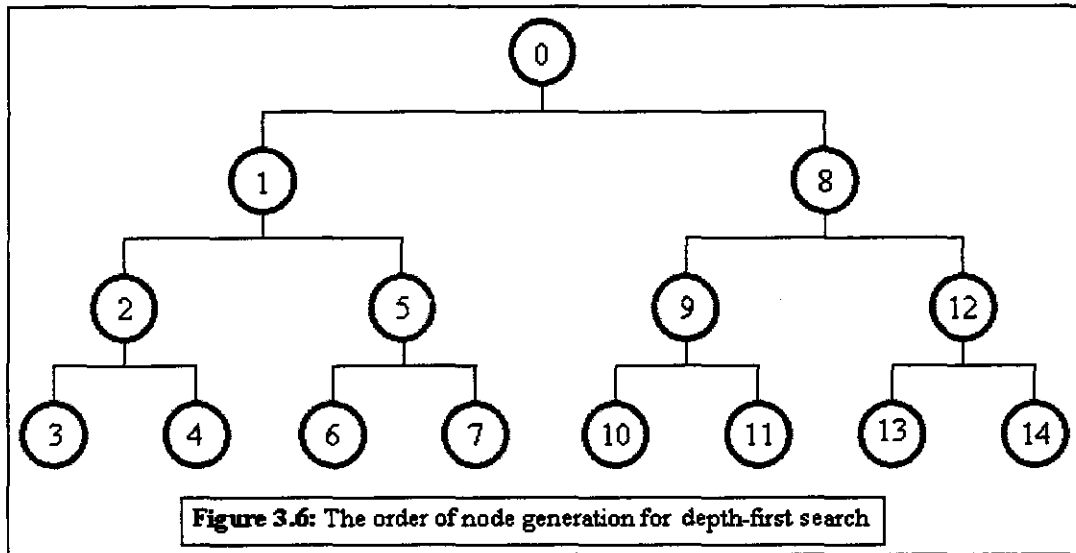
```
/* OPEN and CLOSED are lists */  
  
OPEN = Start node, CLOSED = empty.  
While OPEN is not empty do  
{  
    Select the node with the minimum cost from  
    OPEN, call it X.  
  
    If X is a goal node  
        return success.  
    else  
    {  
        Generate children of X.  
        Put X on CLOSED.  
        For every successor node  $X_s$  of X, compute  
        the cost of  $X_s$  as  $g(X_s) = g(X) + c(X, X_s)$ .  
        Place all successor nodes on the OPEN  
        list.  
    }  
}
```

Figure 3.5: A uniform-cost search algorithm

3.3.3 Depth-first search

Luger and Stubblefield (Luger & Stubblefield, 1993:89-96) state "depth-first search goes deeper into the search space whenever this is possible. Only when no further descendants of a state can be found are its siblings considered." Depth-first search methods implement a strategy known as backtracking. This strategy starts its search at the start state (usually the root node) and pursues that path until it reaches a goal node or a dead end in the state space. If the algorithm determines that a goal node has been reached, it terminates and returns the solution path. Otherwise it backtracks to the

most recent node in the path that has unexamined siblings and continues down one of those branches.



The advantage of depth-first search is that it requires much less memory than breadth-first search does. The amount of memory that is required increases linear to the search depth, as opposed to exponentially for breadth first search. This is easily explained, as only those nodes that are in the path from the root node to the current node need to be stored. The asymptotic space-complexity is therefore favorable and can be expressed as $O(b.d)$ (where b is the branching factor for the problem and d is the depth of the maximum-depth node). Another way to express the space-complexity is that the method only requires storage space for $b.d$ nodes, where b is equal to the branching factor for the problem and d is equal to the depth of the maximum-depth node. Time-complexity, on the other hand, still remains a problem as the same number of nodes is expanded for depth-first search as was expanded for breadth first search. The only difference is the order in which these nodes are expanded, and the asymptotic time-complexity will therefore also be $O(b^d)$. One of the main disadvantages of depth-first search is that it may not terminate on an infinite tree and simply go down the

leftmost path forever (Korf, 1996:8). Depth-first search is therefore neither exact nor complete (Russel & Norvig, 1995:78).

To conclude the discussion on depth-first search, a general depth-first algorithm is given in figure 3.7.

```
/* OPEN and CLOSED are lists */  
  
OPEN = Start node, CLOSED = empty.  
  
While OPEN is not empty do  
{  
  Remove leftmost node from OPEN, call it X.  
  
  If X is a goal  
    return success.  
  else  
  {  
    Generate all successors of X.  
    Put X on CLOSED.  
    Eliminate any successors that are already on  
    OPEN or CLOSED.  
    Put remaining successors on LEFT end  
    of OPEN.  
  }  
}
```

Figure 3.7: A depth-first search algorithm

3.3.4 Depth-limited search

To solve the depth-first search's problem of getting stuck on an infinite path in a search tree, depth-limited search was proposed. This algorithm places a bound on the maximum depth the algorithm is allowed to search to, therefore preventing all nodes on lower levels than the bound in the graph to be expanded. This leads to a search method that neither guarantees to find a shortest path to a solution if one exists, nor to find a solution even if one

exists (Bundy, 1997:33). Therefore the method is neither complete nor exact. The time complexity for the method is $O(b^d)$ where b is the branching factor and d is the depth limit. The space complexity is $O(b.d)$.

3.3.5 Iterative-deepening depth-first search

According to Ciesielski (Ciesielski, 2001), iterative-deepening search performs a depth-first search of the state space with a depth-bound of 1. If this search fails to find a solution for the problem, it continues with a depth-first search of the search space with depth-bound 2. This continues with the depth-bound increased for every iteration.

It might seem, at first glance, as if the iterative-deepening strategy is inefficient, because when the depth-bound is increased from level d to level $d+1$, the search is repeated for all levels up to level d . However, since typical search spaces grow exponentially with the increase of the search depth d , the search on level $d+1$ dominates the total search time. In fact, iterative-deepening performs very well where asymptotic time ($O(b^d)$) and space ($O(b.d)$) complexities are concerned (Bundy, 1997:61). Furthermore, the method is complete and exact (Russel & Norvig, 1995:79-80).

3.4 Final thoughts

Luger and Stubblefield (Luger & Stubblefield, 1993:99) state that all the uninformed search strategies, as discussed in this chapter, namely breadth-first, uniform-cost, depth-first, depth-limited and iterative-deepening search methods, can be shown to have worst-case exponential time complexities. This fact is true for all uninformed search methods, and the only searching approaches that reduce this complexity, employ heuristics to guide the search. Therefore, chapter 4 introduces informed search methods that implement heuristics to guide the search more effectively.

3.5 Summary

Uninformed search methods have now been discussed and evaluated with the set criteria. The next chapter deals with informed search methods that expand upon uninformed search methods.

CHAPTER 4: Informed search methods

4.1 Introduction

George Polya (Polya, 1945) defines *heuristic* as “the study of the methods and rules of discovery and invention”. Heuristics form the basis of informed search methods and according to Bundy (Bundy, 1997:52), informed search methods is an advanced technique to do state space searches, with the state space normally represented in the form of a graph or tree.

The word heuristic comes, in fact, from the Greek word *heuriskein*, meaning “to discover”. It is also the origin for the word *eureka*, derived from Archimedes’ reputed exclamation *heurika* (“I have found”), uttered when he had discovered a method for determining the purity of gold.

As mentioned in chapter 2, heuristics, like all rules of discovery, are fallible. Heuristics are often based on previous experience or intuition, and therefore it leads to an informed guess of the next step that should be taken to solve a specific problem. Even though heuristics cannot predict the *exact* branching of the state space tree, it can drastically improve the performance of the search methods (Luger & Stubblefield, 1993:117).

This chapter deals with different heuristic search methods in section 4.2 and section 4.3 summarizes the contents of the chapter.

4.2 Types of informed search methods

The following is a discussion of different types of informed search methods.

4.2.1 Generate-and-test

In the literature concerning search methods, various authors voice different opinions pertaining to the *generate-and-test* search method. Barr (Barr, 1981:30) states that the generate-and-test method merely generates new states in the state space and then tests them against a specific goal state. This translates to an uninformed search method that utilizes no information concerning the state space. Rich and Knight (Rich & Knight, 1991:64), on the other hand, explain that only in its most basic form can the generate-and-test method be seen as an uninformed search method. The method can be implemented by using the following algorithm:

- **Step 1:** Generate a possible solution: Some methods require that a path be generated from the start state, and for other problems this means generating a particular state in the state space;
- **Step 2:** Test to see whether the generated state is indeed equal to the goal state; and
- **Step 3:** If, at this stage, a solution has been found, terminate the algorithm. Otherwise return to step 1.

In its most basic form, the generate-and-test algorithm is an exhaustive depth-first search algorithm, which generates all possible states of the search tree and tests them. The method can also generate random searches, which does not guarantee that a solution will be found. Random generate-and-test algorithms are often referred to as British Museum algorithms, which is a reference to a method for finding an object in the British Museum by wandering around in random directions (Rich & Knight, 1991:64).

Up to this point, the generate-and-test algorithm still presents itself as an uninformed search method, but by implementing a general purpose heuristic such as the *nearest neighbor heuristic*, some states that seem unlikely to lead to a solution, are not expanded further. Researchers studying the

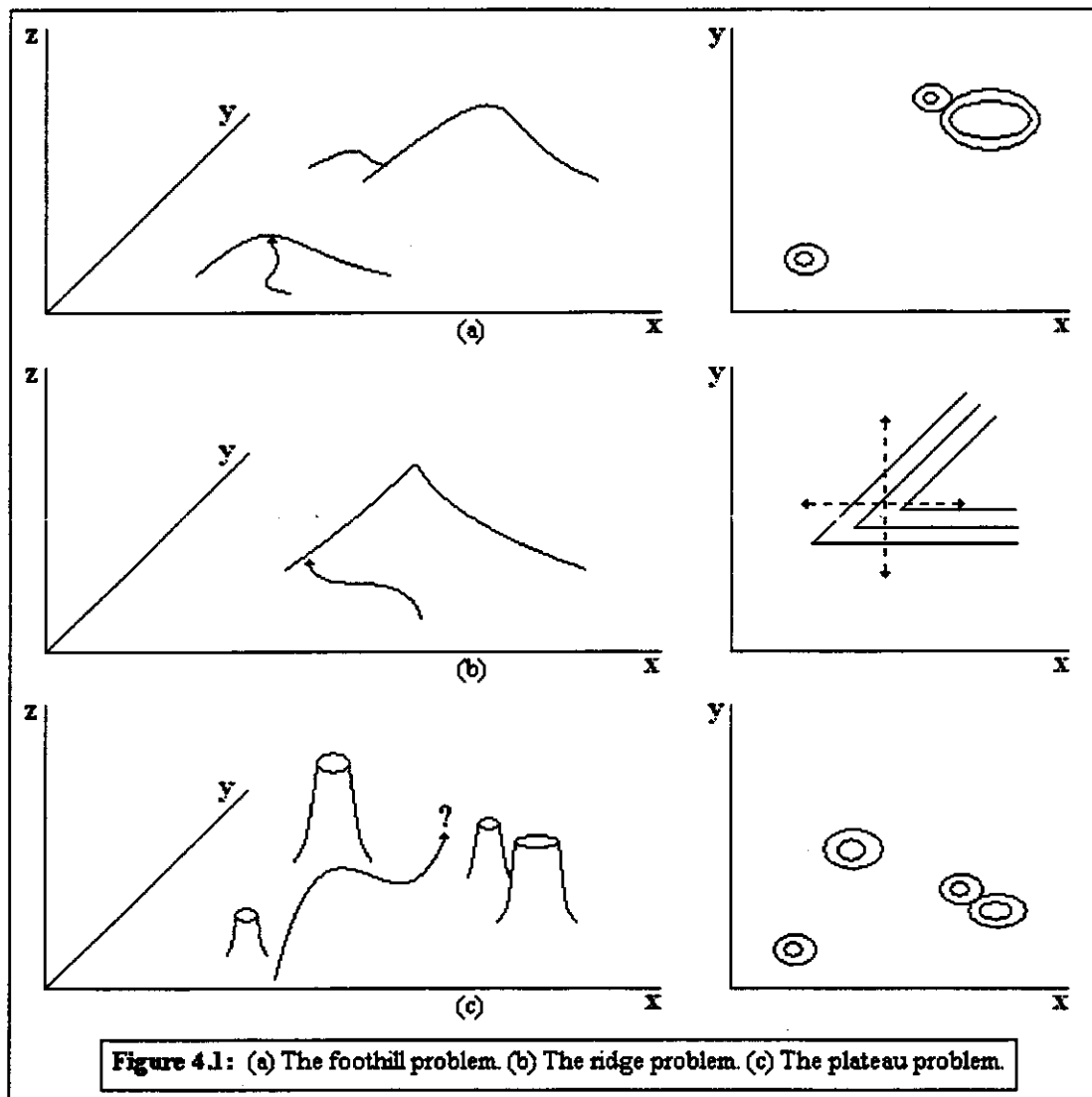
traveling salesperson problem, where the algorithm selects a city that has not been visited and is closest to the current city, devised the nearest neighbor heuristic. The nearest neighbor heuristic is similar to the heuristic implemented by best-first search algorithms. This conforms to the idea of informed search, and therefore such a generate-and-test search method is regarded as an informed search method (Rich & Knight, 1991:41).

4.2.2 Hill climbing

Winston (Winston, 1992:70) describes the classical hill climbing search method intuitively by stating that search efficiency may improve spectacularly if there exists a way to order the branches under each node so that the most promising ones are explored. In some situations, measurements can be made to determine a reasonable ordering. Some examples are:

- The temperature in a room is uncomfortably hot. The thermostat in the room can be used to change the temperature, but the markings on it have been removed, leaving it up to the user to choose which way to move the thermostat switch; and
- The television's picture has deteriorated over a period of time. The brightness, color, tint and tuning controls have to be adjusted to obtain a better picture.

Both of these problems conform to an abstraction in which there are some adjustable parameters and a way of measuring the performance associated with any particular set of values for the parameters. Hill climbing is therefore a depth-first search method which implements a heuristic that orders the alternatives at each decision point. Movement proceeds through the alternative that offers the best improvement to the situation in one step. The required measurements may be absolute or relative, precise or appropriate (Winston, 1992:70-73).



Bundy (Bundy, 1997:54) writes that hill climbing is a search method that is used to determine the maximum or minimum value of an evaluation function. The method considers the local neighborhood of a node, calculates the maximum or minimum values for all the neighbors and chooses those nodes with the largest or smallest values. Hill climbing differs from other methods that use evaluation functions in that it does not implement backtracking. It rather follows one path down the search tree and does not retain previous unexpanded nodes that were promising. This property is what makes the method computationally very efficient, but also explains why it is not guaranteed to find a solution for all problem instances.

Winston (Winston, 1992:72-74) states that the foothill, ridge and plateau problems are the three main pitfalls that hinder hill climbing and illustrates it as in figure 4.1.

4.2.2.1 The foothill problem

The source of the *foothill problem* is secondary peaks in the state space, which causes hill climbing to find only locally exact solutions. Unfortunately, secondary peaks usually divert the search in wrong directions, preventing the discovery of global exact solutions. This situation is depicted in figure 4.1 (a).

4.2.2.2 The ridge problem

A more subtle and frustrating problem is the *ridge problem*, shown in figure 4.1 (b). The contour map shows that each decision made by the hill climbing method moves the current position across contour lines, even though no local or global maximum is near the current position. Increasing the number of search directions might help solve the problem.

4.2.2.3 The plateau problem

The *plateau problem* occurs when mostly flat area in the state space separates the peaks. The local improvement operation fails to yield meaningful paths and all standard-step probes leave the performance unchanged. This will ultimately lead to a situation where no solution is offered by the method, as depicted in figure 4.1 (c).

Figure 4.2 shows a general implementation of the hill climbing search method. The main difference between the depth-first and hill climbing search methods are indicated by the text written in *italic* in figure 4.2.


```

/* OPEN and CLOSED are lists */

OPEN = Start node, CLOSED = empty.

While OPEN is not empty do
{
    Remove leftmost node from OPEN, call it X

    If X is a goal
        return success.
    else
    {
        Generate all successors of X.
        Put X on CLOSED.
        Eliminate any successors that are already on
        OPEN or CLOSED.
        Sort the new successors by the estimated
        distances between these nodes and the
        goal.
        Put remaining successors on LEFT end
        of OPEN.
    }
}

```

Figure 4.2: A hill climbing search algorithm

4.2.3 Dynamic hill climbing

Yuret and De la Maza (Yuret & De la Maza, 1993:2) recognize the same three problems encountered by basic hill climbing, but offer solutions to the first two, namely the foothill and ridge problems. The plateau problem, though, is referred to as a hopeless one by Yuret and De la Maza, because the state space offers no information about its structure. In this case, any random search method will perform as well as any other heuristically informed method. A dynamic hill climbing method is proposed, which implements measures that attempt to resolve the effects of the foothill and the ridge problems.

4.2.3.1 Dynamic coordinate frame

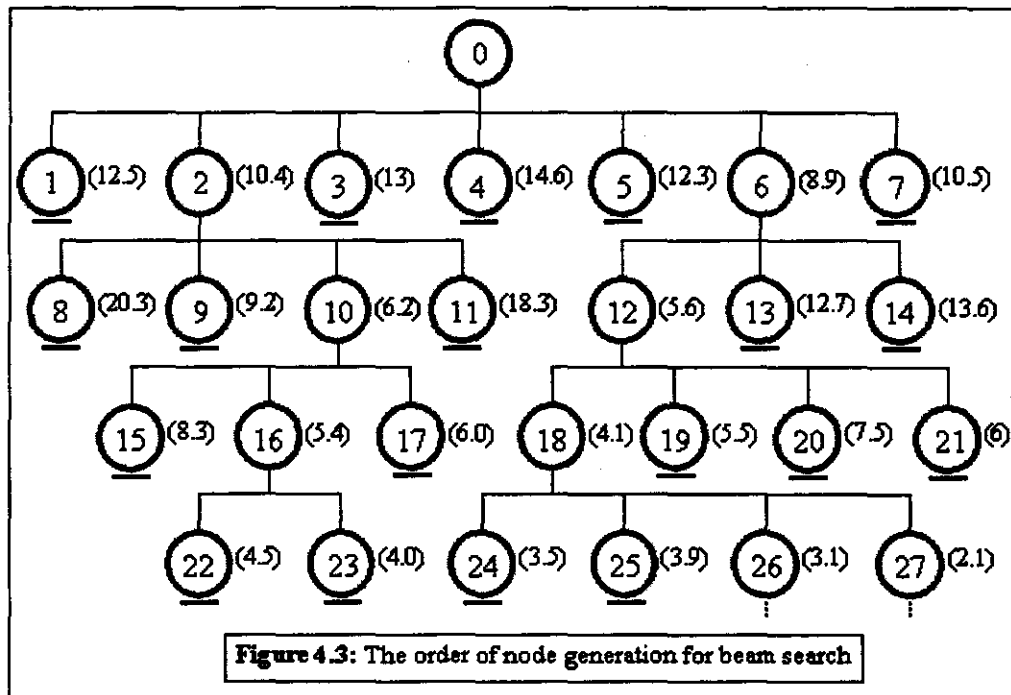
Basic hill climbing techniques usually implement static coordinate frames, with which movement is restricted to, for instance, the four basic compass directions: north, west, south and east. As shown in figure 4.1 (b), these directions may sometimes all lead to worse states. One solution to the ridge problem might be, as mentioned in section 4.2.2.2 (page 36), to increase the number of search directions to also include combinations of search directions for instance northwest, northeast, southwest and southeast. This, unfortunately, presents the problem of combinatorial explosions of search states that is difficult or impossible to resolve. Dynamic hill climbing, on the other hand, adopts a dynamic coordinate frame approach. This implies that whenever the basic hill climber gets stuck, a more appropriate coordinate frame is automatically calculated. The process leads to a situation where the number of search directions remains constant, but these directions are only changed when necessary (Yuret & De la Maza, 1993:2-3).

4.2.3.2 Exploitation of local optima

A possible measure against the foothill problem is to consider the advantage of individual optimal solutions as an advantage in the selection procedure. This implies that as in many genetic algorithms, whenever a local optimum is reached its chance of survival surpasses that of other possible solutions. The goals of diversity-based strategies are to protect the search from early convergence and to explore the state space as homogeneously as possible. In practice, these strategies usually take into account the fitness and diversity of locally optimal solutions (Yuret & De la Maza, 1993:3).

4.2.4 Beam search

Beam search employs a simple heuristic to alter the breadth-first search method. The search still progresses in a level-by-level fashion through the search space, but only the best m nodes are considered for further exploration and the rest of the nodes on the level are ignored. Each node on a certain level is evaluated with a chosen heuristic evaluation function and the m nodes with the lowest cost are then chosen and the rest is ignored. Beam search ensures that the number of nodes explored remains manageable, even if the branching factor is large and the search needs to probe deeply into the search tree. This search method does not, however, ensure that a solution for the problem instance will be found even if one exists, or that if a solution is found that it will be optimal. As with basic hill climbing, beam search does not implement backtracking, but rather than following only the one most promising path down the search tree like hill climbing does, beam search follows the m most promising paths. Figure 4.3 illustrates the first few steps of how beam search expands a search tree where the beam width m is 2. The values next to the nodes indicate the cost associated with each node (Luger & Stubblefield, 1993:147).



```

/* OPEN and CLOSED are lists */

OPEN = Start node, CLOSED = empty.

Remove leftmost node from OPEN, call it X.
If X is a goal
    return success.
else
{
    Generate children of X.
    Put X on CLOSED.
    Eliminate children of X already on OPEN or CLOSED.
    Put remaining nodes on OPEN.
}

Evaluate the costs of all nodes on OPEN, keep only the
m best nodes on OPEN.
Delete the rest of the nodes on OPEN.

While OPEN is not empty do
{
    For Count = 1 to m
    {
        Remove leftmost node from OPEN, call it X.
        If X is a goal
            return success.
        else
        {
            Generate children of X.
            Put X on CLOSED.
            Eliminate children of X already on OPEN or CLOSED.
            Put remaining nodes on right end of OPEN.
        }
    }

    Evaluate the costs of all nodes on OPEN, choose only the
m best nodes and put these on the left end of OPEN.
    Delete the rest of the nodes on OPEN.
}

```

Figure 4.4: A beam search algorithm

To conclude the discussion on beam search, an algorithmic implementation of the method is displayed in figure 4.4. The algorithm closely resembles a breadth-first search algorithm, except that it only chooses the m best nodes

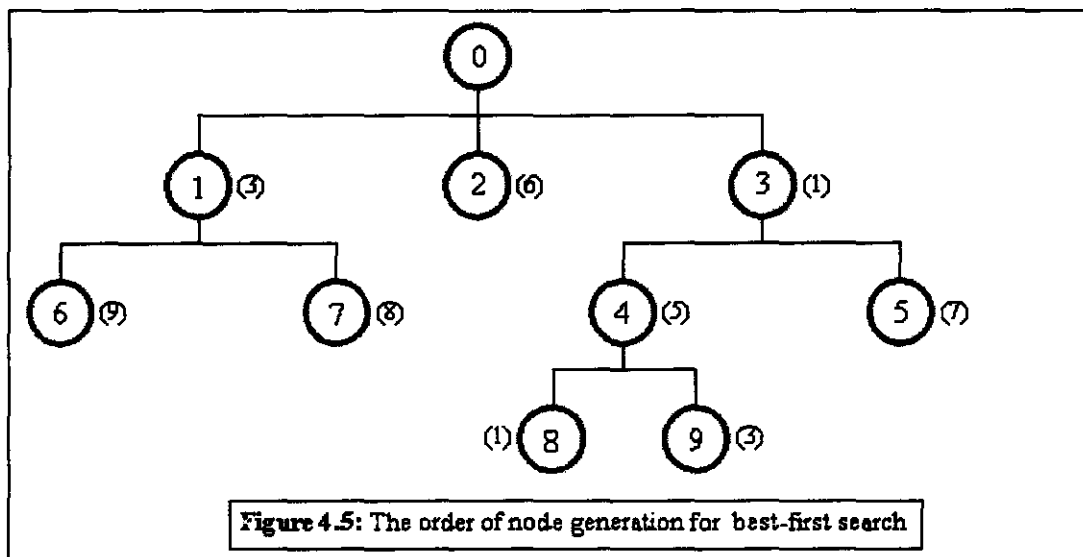
on every level of the search tree to explore further, whereas breadth-first search explores all possible paths on any level of the search tree.

4.2.5 Best-first search

Best-first search differs only slightly from basic hill climbing. In best-first search methods, the focus shifts after each cycle to the best node found globally. Basic hill climbing, on the other hand, moves its attention to the best node found locally, meaning that forward motion is always generated through the seemingly best descendant. Best-first search generates forward motion from the best node so far, no matter where it is located in the partial developed search tree (Winston, 1992:75).

The best-first search method uses an estimate value to determine the next node that should be expanded. It is calculated using a heuristic evaluation function h that estimates the minimum path cost from the current node to the goal. The cost of node n is denoted as $h(n)$.

Figure 4.5, adapted from Rich and Knight (Rich & Knight, 1991:74), shows the first few steps for the best-first search procedure. Initially, only the root node (node 0) is part of the state space, therefore it will be expanded first. This expansion yields nodes 1, 2 and 3, with respective costs of 3, 6 and 1. These values represent the estimated costs of reaching a goal node from the current position ($h(n)$). This number must be minimized, and if nodes 1, 2 and 3 are analyzed, node 3 is found to have the minimum cost. Therefore node 3 is expanded next and the same procedure is followed until a goal node is found.



Rich and Knight (Rich & Knight, 1991:74) concludes their discussion on best-first search by stating that a bit of depth-first search is done at the most promising node in the state space. Eventually, if a goal node is not found, the current branch will start to look less promising than one of the top-level branches that had previously been ignored. The now more promising, previously ignored branch will be explored. The old branch is not forgotten however, and its last node is placed in the set of generated but unexplored nodes. The search can return to it whenever all the others get bad enough that it is again the most promising path.

A general algorithm for the best-first search method is given in figure 4.6. The algorithm displays similarities to that of the basic hill climbing algorithm, with two differences. The first is that for every iteration the lowest cost node in the list OPEN is placed in X and not the leftmost node in the list. Secondly, the newly generated successors of the current node is not sorted, but merely placed in the OPEN list.

```

/* OPEN and CLOSED are lists */

OPEN = Start node, CLOSED = empty.

While OPEN is not empty do
{
    Remove leftmost node from OPEN, call it X.

    If X is a goal
        return success.
    else
    {
        Generate all successors of X.
        Put X on CLOSED.
        For every successor node  $X_s$  of X,
        compute the cost of  $X_s$  as  $h(X_s) =$ 
        remaining distance to the goal.
        Eliminate any successors that are already
        on OPEN or CLOSED.
        Put remaining successors on OPEN.
        Sort the entire OPEN list with least-cost
        nodes (lowest  $h$ ) in the front of the list.
    }
}

```

Figure 4.6: A best-first search algorithm

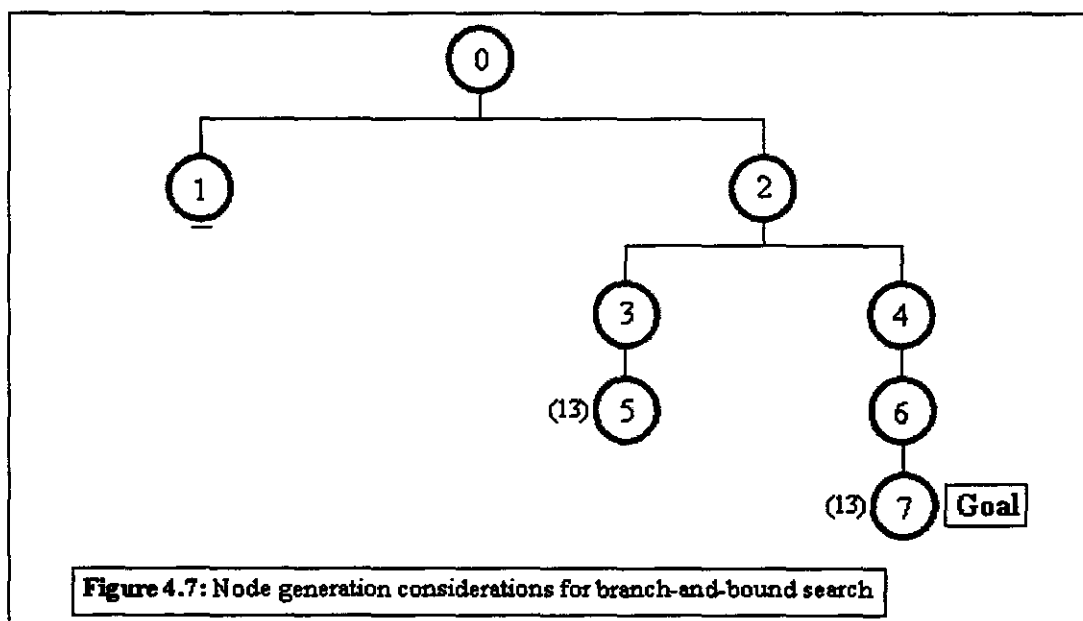
4.2.6 Branch-and-bound search

A method that can be implemented to generate optimal solutions for problem instances is called branch-and-bound search. The branch-and-bound method always keeps track of all nodes that could be expanded next. It therefore globally selects the lowest cost node at any given time during the search process and expands it (generates its children). At this stage it might seem to be exactly the same as best-first search, but the heuristic evaluation function used to determine the best node (node to be explored next) differs slightly from that of best-first search.

The branch-and-bound method uses the exact cost of every node (the cost of reaching the current node from the start node) and denotes it as $g(n)$. The lowest-cost, globally unexplored node is then chosen as the node to be expanded next. This process repeats until the goal is reached, and because the lowest cost node is always the one chosen for expansion, the first goal to be found is *likely* to be the optimal solution.

To turn *likely* into *certain*, all nodes in the OPEN list have to be expanded until their costs are equal to or more than that of the solution. The reason for this is that the last step taken to reach the goal may be costly enough to make another open node able to generate a less costly solution.

Figure 4.7, adapted from Winston (Winston, 1992:83), shows how branch-and-bound search functions when searching for an optimal solution. The cost to reach node 7 (a goal node) is 13. Similarly, the cost to reach node 5 is also 13 and any additional movement along that branch will make it more expensive than 13. Therefore, node 5 does not need to be expanded further, because any resulting node including node 5 in the solution, will have a more expensive cost than node 7.




```

/* OPEN and CLOSED are lists */

OPEN = Start node, CLOSED = empty.

While OPEN is not empty do
{
    Remove leftmost node from OPEN, call it X.

    If X is a goal
        return success.
    else
    {
        Generate all successors of X.
        Put X on CLOSED.
        For every successor node  $X_s$  of X, compute
        the cost of  $X_s$  as  $g(X_s)$  = cost of reaching
        node  $X_s$  from the start node.
        Eliminate any successors that are already
        on OPEN or CLOSED.
        Put remaining successors on OPEN.
        Sort the entire OPEN list with least-cost
        nodes (lowest  $g$ ) in the front of the list.
    }
}

```

Figure 4.8: A branch-and-bound search algorithm

A general algorithm for the branch-and-bound search method is given in figure 4.8.

4.2.7 Branch-and-bound search with underestimates

In some cases, you can improve branch-and-bound search greatly by guessing the remaining cost of reaching the goal from the current node, as well as using facts about costs already accumulated to get to the current node.

The branch-and-bound method with underestimates consists of two components. The first is the exact cost of reaching the current node n from

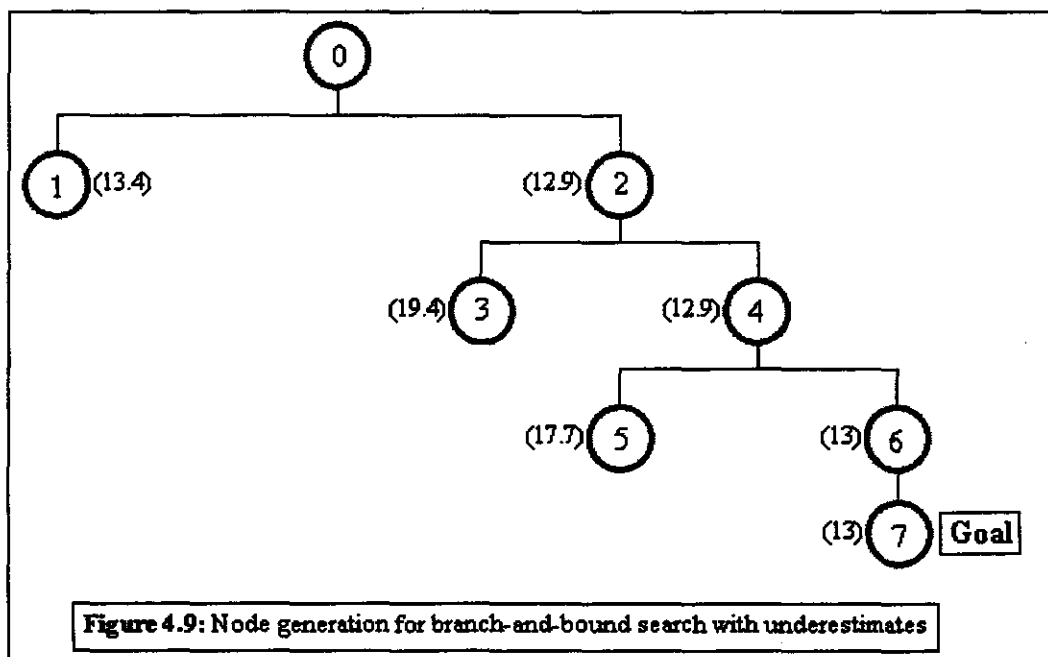
the start node, which is denoted as $g(n)$. The second component is the underestimated cost from the current node n to a goal node and is denoted as $h(n)$. Therefore, the following function is used to guide the branch-and-bound method with underestimates:

$$f(n) = \text{estimated cost of the cheapest solution through } n$$

which can be written as

$$f(n) = g(n) + h(n)$$

according to Russel and Norvig (Russel & Norvig, 1995:77). Note that by using *underestimates* ($h(n)$) the optimal path will never be overlooked, because an underestimate of the remaining cost of reaching the goal node from the current node n added to the exact cost of reaching the current node n from the start node will always yield an underestimate of the total cost of reaching the goal. Therefore, if a goal node is reached by expanding the lowest-cost (underestimated values) nodes repeatedly, no extra work need to be done after all nodes in the OPEN list have been expanded with a cost of less than or equal to that of the goal node.



Branch-and-bound search augmented by underestimates determines that a solution through the nodes 0-2-4-6-7 is optimal. In figure 4.9 the numbers beside the nodes are accumulated distance ($g(n)$) plus underestimates of distance remaining ($h(n)$). Underestimates quickly push up the lengths associated with bad solutions. In the example in figure 4.9, fewer nodes are expanded than would be expanded with branch-and-bound search operating without underestimates.

A general algorithm for the branch-and-bound search method using underestimates is given in figure 4.10. The algorithm closely resembles branch-and-bound search, and the only addition is the inclusion of the $h(n)$ value when computing the costs of each successor node.

```

/* OPEN and CLOSED are lists */

OPEN = Start node, CLOSED = empty.

While OPEN is not empty do
{
    Remove leftmost node from OPEN, call it X.

    If X is a goal
        return success.
    else
    {
        Generate all successors of X.
        Put X on CLOSED.
        Calculate node costs for each successor  $X_s$  of X
        by adding the values of  $g(X_s)$  and  $h(X_s)$ .
        Eliminate any successors that are already
        on OPEN or CLOSED.
        Put remaining successors on OPEN.
        Sort the entire OPEN list with least-cost
        nodes (lowest  $g+h=f$ ) in the front of the list.
    }
}

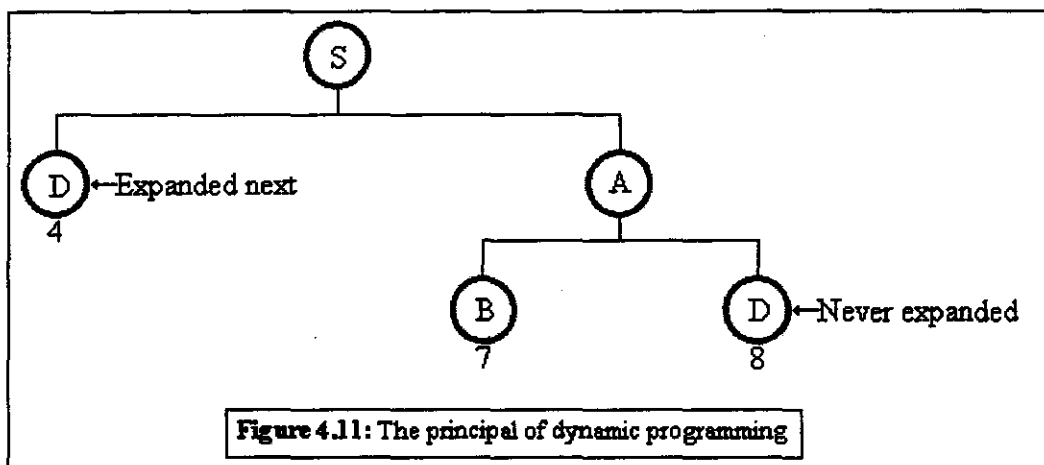
```

Figure 4.10: A branch-and-bound search algorithm using underestimates

4.2.8 Branch-and-bound search using the dynamic programming principle

A second method to improve the basic branch-and-bound method is by eliminating portions of the search tree structure. This is accomplished by ignoring (not expanding) more expensive duplicate nodes that also exist in other portions of the search space.

Figure 4.11 illustrates this concept. The numbers beside the nodes are accumulated distances ($g(n)$). There is no point in expanding the instance of node D at the end of S-A-D, because getting to the goal via the instance of D at the end of S-D is obviously more efficient.



This example illustrates a general principle. Assume that the path from a starting point S to an intermediate point I does not influence the choice of paths for traveling from I to a goal point G. Then the minimum cost from S to G through I is the sum of the minimum cost from S to I and the minimum cost from I to G. Consequently the *dynamic-programming* principle holds that, when a least-cost path from S to G is required, all paths from S to any intermediate node I can be ignored except the minimal-length path from S to I.

A general definition of the dynamic-programming principle:

The best way through a particular, intermediate place is the best way to it from the starting place, followed by the best way from it to the goal. There is no need to look at any other paths to it or from the intermediate place.

A general algorithm for the branch-and-bound search method using the dynamic-programming principle is given in figure 4.12. Note that this algorithm does not include underestimates.

```
/* OPEN and CLOSED are lists */  
  
OPEN = Start node, CLOSED = empty.  
  
While OPEN is not empty do  
{  
  Remove leftmost node from OPEN, call it X.  
  
  If X is a goal  
    return success.  
  else  
  {  
    Generate all successors of X.  
    Put X on CLOSED.  
    Calculate node costs for each successor  $X_s$  of X  
    as  $g(X_s)$  = cost of reaching  $X_s$  from the start node.  
    Eliminate any successors that are already  
    on OPEN or CLOSED.  
    If two or more nodes are identical, delete all  
    but the least-cost node from OPEN.  
    Put remaining successors on OPEN.  
    Sort the entire OPEN list with least-cost  
    nodes (lowest  $g$ ) in the front of the list.  
  }  
}
```

Figure 4.12: A branch-and-bound search algorithm
using the dynamic programming principle

4.2.9 The A* search method

The A* search method is a branch-and-bound search, with an estimate of remaining distance ($h(n)$), combined with the dynamic programming principle. If the estimate of remaining distance is a lower bound on the actual distance, then the A* method is exact and it produces optimal solutions.

4.2.9.1 Admissibility of the A* method

If the A* method is used with an evaluation function in which $h(n)$ (estimated cost of reaching the goal from node n) is less than or equal to the exact cost of reaching the goal from node n , for all n , the resulting algorithm is admissible. Therefore, in the interests of admissibility, the value of $h(n)$ for all n must never be overestimated (Luger & Stubblefield, 1993:132-133).

Admissible heuristics are by nature optimistic, because they think the cost of solving the problem is less than it actually is. This optimism transfers to the f function as well, and therefore if h is admissible, $f(n)$ never overestimates the actual cost of the best solution through n (Russel & Norvig, 1995:97).

If an algorithm is admissible, it implies that the algorithm is guaranteed to terminate with the least-cost solution, if such a solution exists (Bundy, 1997:12).

4.2.9.2 Monotonicity of the A* method

According to Luger and Stubblefield (Luger & Stubblefield, 1993:133), a heuristic function h is monotone if:

1. For all states n_i and n_j , where n_j is a descendant of n_i ,

$h(n_i) - h(n_j) \leq \text{cost}(n_i, n_j)$, where $\text{cost}(n_i, n_j)$ is the actual cost of going from state n_i to n_j .

2. The heuristic evaluation of the goal state is zero, or $h(\text{Goal}) = 0$.

In other words, if a search space is examined that was created with a monotone heuristic function, the node costs (value of function f) never decreases as one moves down any given path in the search space (Russell & Norvig, 1995:97).

The advantage of a monotone heuristic function is that it guarantees that if a state is discovered using that heuristic, the same state will not be found later in the search at a cheaper cost.

4.2.9.3 Informedness and the A* method

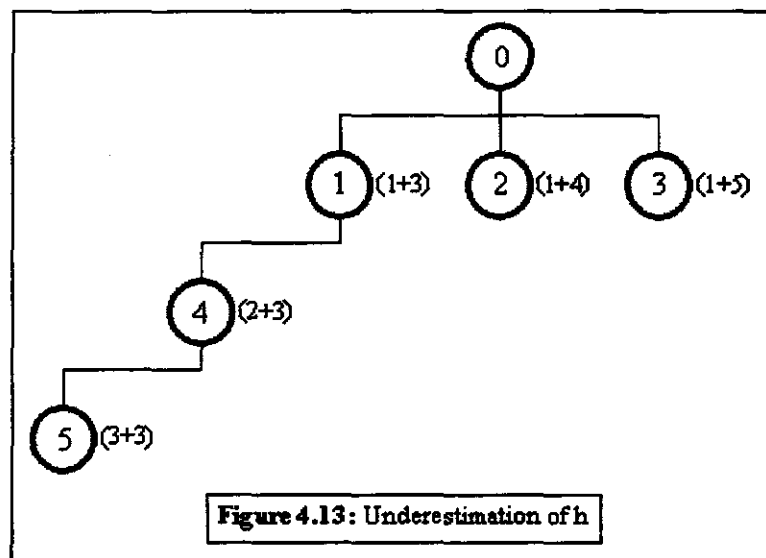
For two A* heuristics h_1 and h_2 , if $h_1 \leq h_2$, for all states n in the search space, heuristic h_2 is said to be more informed than h_1 . Therefore, if a heuristic h_2 is more informed than h_1 , then the set of states examined by h_2 is a subset of those expanded by h_1 (Luger & Stubblefield, 1993:135).

4.2.9.4 Underestimation of h

Consider the situation shown in figure 4.13. According to Rich and Knight (Rich & Knight, 1991:78) all arc costs are fixed and equal to one. For each node, f is indicated as the sum of g and h , as is specified by the definition of the A* method.

The figure depicts a situation where five iterations of an algorithm implementation of the A* method have been executed and nodes 1, 2, 3, 4 and 5 have been expanded. Initially, though, only node 0 is expanded.

When the node costs are evaluated, node 1 is identified as the lowest-cost node with $f(1) = 4$, therefore, it is expanded first. Suppose node 1 has only one successor namely 4, which also appears to be three steps away from a goal state. If the cost of node 4 is evaluated, it is shown to be $f(4) = 5$. This is, in fact, the same as the cost for node 2, but this situation is resolved by favoring the path that is currently being followed. This implies that node 4 is expanded next and for this problem instance, node 4 also has only one successor namely node 5. Node 5 also appears to be 3 steps away from reaching a goal node and it becomes clear that unnecessary steps are being used up and no progress is made. The node cost of node 5 is, however, $f(5) = 6$. This is greater than that of node 2, therefore the current path is abandoned and node 2 is expanded next. It is deduced that by underestimating the value of h , some effort will possibly be wasted in the search.

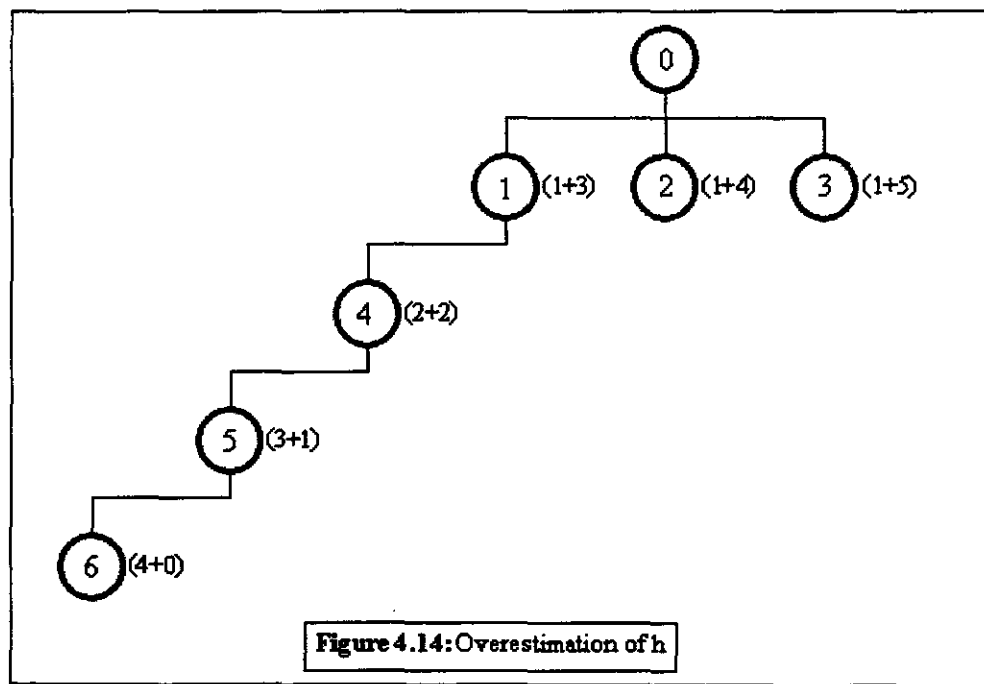


4.2.9.5 Overestimation of h

Now the instance in figure 4.14 is considered. As was the situation in figure 4.13, the initial nodes' costs are once again evaluated and node 4 is expanded. The expansion sequence remains the same as that for the

instance illustrated in figure 4.13 up until node 5 is expanded. Then, the node costs of nodes 5, 2 and 3 are evaluated. Node 5 is still the most promising node and is expanded to yield node 6, which represents a goal state. The length of the solution path is therefore 4.

Suppose, however, that a direct path exists between node 3 and a goal state. This path will never be found because $h(3) = 5$ is an overestimate, even though it has a path length of only 2.



4.2.9.6 Graceful decay of admissibility

For most real world problems, the only way to guarantee that h is never overestimated, is to set it to zero. Then, unfortunately, the method degenerates to basic branch-and-bound search. A corollary to this theorem does exist, however, which is called *graceful decay of admissibility*. According to Rich and Knight (Rich & Knight, 1991:79), the graceful decay of admissibility is defined as follows:

If h is rarely overestimated by more than δ , then an A^ algorithm will rarely find a solution whose cost is more than δ greater than the cost of the optimal solution.*

To conclude the discussion of the A^* method, a general algorithmic implementation of the method is given in figure 4.15.

```
/* OPEN and CLOSED are lists */  
  
OPEN = Start node, CLOSED = empty.  
  
While OPEN is not empty do  
{  
    Remove leftmost node from OPEN, call it X.  
  
    If X is a goal  
        return success.  
    else  
    {  
        Generate all successors of X.  
        Put X on CLOSED.  
        Calculate node costs for each successor  $X_s$  of X  
        by adding the values of  $g(X_s)$  and  $h(X_s)$ .  
        Eliminate any successors that are already  
        on OPEN or CLOSED.  
        If two or more nodes are identical, delete all  
        but the least-cost node from OPEN.  
        Put remaining successors on OPEN.  
        Sort the entire OPEN list with least-cost  
        nodes (lowest  $g+h=f$ ) in the front of the list.  
    }  
}
```

Figure 4.15: An A^* search algorithm

4.3 Summary

Chapter 4 concludes the discussion on the basic and more advanced ideas behind search methods. Chapter 5 leads the reader into a detailed discussion

on different existing exact and non-exact problem solving methods. These methods are analyzed and problems are identified with algorithmic implementations of these methods.

CHAPTER 5: C2DGC problem solution methods

5.1 Introduction

As was mentioned in chapter 1, the C2DGC problem has a wide range of commercial and industrial application areas. The need for optimal solutions for the problem arises in the steel, glass, wood and metallurgy industries to name but a few. Therefore, well-defined, structured, effective, efficient and sometimes exact algorithms are required by industries and this chapter acts as a guide to understanding the methods that exist (exact and non-exact) from which these algorithms can be deduced.

Sample problem instances are also solved throughout this chapter illustrating the functioning of the discussed solution methods.

Problem	Stock plate length (L) and width (W)	Demand rectangles' length (L), width (W) and upper bound (b)
P1	(15,10)	(2,1,1); (3,2,2); (3,3,2); (3,4,5); (8,2,3); (3,7,1); (8,4,2).
P2	(70,40)	(17,9,1); (11,19,4); (12,21,3); (14,23,4); (24,15,1); (24,15,2); (25,16,4); (27,17,2); (18,29,3); (21,31,3); (32,22,2); (23,33,3); (34,24,2); (35,25,2); (36,26,1); (37,27,1); (38,28,1); (39,29,1); (41,30,1); (43,31,1).
P3	(70,40)	(29,5,1); (9,39,4); (55,9,1); (39,15,1); (11,16,2); (23,21,3); (29,14,4); (16,19,3); (9,36,2); (22,4,2).
P4	(70,40)	(22,18,2); (40,10,1); (13,27,3); (23,18,2); (29,8,4); (16,4,1); (47,9,1); (19,19,4); (13,16,2); (36,16,4).
P5	(8,4)	(2,1,1); (3,3,3); (1,4,2); (2,2,3).
P6	(30,12)	(3,6,3); (7,4,2); (7,5,1); (8,2,3); (8,5,3); (9,6,2); (9,7,1); (10,4,4); (15,5,3); (19,6,2); (4,8,2); (4,9,1).
P7	(47,32)	(13,14,1); (15,11,2); (14,7,3); (12,20,4); (17,11,3); (16,14,3); (24,8,2); (8,17,3); (13,16,1); (19,16,2).
P8	(55,50)	(11,13,3); (17,28,3); (23,23,2); (16,18,3); (18,15,1); (18,23,1); (15,24,4); (16,24,2); (27,29,2); (31,10,4).
EP1	(5,5)	(1,1,1); (3,2,5).

Table 5.1: Set of nine C2DGC problems (P1-P8 as presented by Deza)

Daza et al (Daza et al, 1995:642) present a set of eight C2DGC problem instances in their article, some of which were also studied by Christofides and Whitlock (1977). These eight problem instances were solved by algorithms developed by Daza, and the problem specifications are summarized in table 5.1.

Section 5.2 discusses existing exact C2DGC problem solving methods, whereas section 5.3 describes existing non-exact methods. Section 5.4 concentrates on the properties of algorithms based on these methods, and also identifies certain problem-areas within these algorithms that require further investigation. Section 5.5 summarizes the content of the chapter.

5.2 Exact methods to solve the C2DGC problem

As discussed in chapter 2, section 2.3 (pages 17-18), the C2DGC problem has been researched by Christofides and Whitlock (1977). They used dynamic programming procedures and results obtained by Gilmore and Gomory (1965, 1967) who studied the U2DGC problem, to solve the C2DGC problem. Furthermore, Held and Karp (1971) proposed an implementation of the travelling salesman problem to solve the C2DGC problem. Lastly, Wang (1983), Vasko (1988) and Oliveira and Ferreira (1990) used a rectangle-building method, as proposed by Wang, to solve the C2DGC problem. This approach (referred to as the *Wang method* or *Wang's method* in the literature) will be used as the foundation from which modified algorithms will be derived in this thesis.

The Wang method is used as a basis because:

- It integrates effectively with artificial intelligence search methods and the tree structures they use to represent state spaces; and
- It can be shown that when an effective pruning criterion is used, the Wang method (and algorithms derived from it) is exact.

5.2.1 The Wang method

The Wang method (WA), as was presented by Wang (Wang, 1983:573-577) in 1983, proposed an ingenious way of building larger rectangles by joining smaller ones. Wang proposed two algorithms based on this method to solve C2DGC problems. By using her method, rectangles are gradually generated considering the original demand rectangles, rotated versions of the original demand rectangles and new rectangles generated in each state of the algorithm. Every newly generated rectangle may or may not contain trim loss. This trim loss is called internal trim loss. From the discussion of the C2DGC problem in chapter 2, section 2.2.3 (pages 14-17), the problem is defined as:

Let S be a stock sheet of length L and width W , and let r be a set of demand rectangles of type r_i ($i = 1, 2, 3, \dots n$), where each type will have a demand constraint of b_i , a length of l_i and a width of w_i . From this, the guillotine cutting pattern with a minimum trim loss must be determined that uses no more than b_i replicates of demand rectangle r_i ($i = 1, 2, 3, \dots n$).¹

Daza et al (Daza et al, 1995:635) state that for each newly generated rectangle, three feasibility criteria are considered:

- Rectangle dimensions must be less than or equal to the stock sheet dimensions². These rectangles are referred to as feasible rectangles;
- Internal trim loss must be less than or equal to a certain predetermined percentage of the stock sheet; and
- The number of demand rectangles of a specific type cut from the stock sheet must be less than or equal to its upper bound.

¹ Refer to chapter 2, section 2.2.3.1, pages 15-16, for the mathematical formulation of the problem

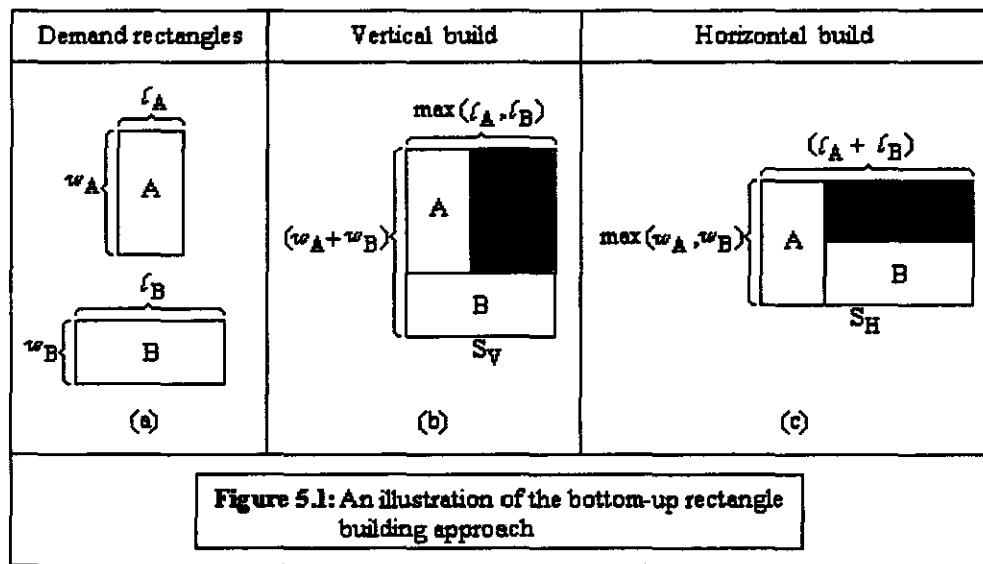
² Note that for each newly generated rectangle only the comparisons $l \leq L$ and $w \leq W$ need to be done

5.2.1.1 Rectangle building with the bottom-up approach

The bottom-up rectangle building approach is based on the observation that by using horizontal and vertical builds on demand rectangles and their rotated equivalents, all the possible *guillotine cuts*³ can be obtained on the initial stock sheet. Wang (Wang, 1983:573-577) first proposed these construction methods for guillotine-cutting problems, but it can be linked to particular cases of methods used by Albano and Sapuppo for solving the irregular-shape cutting stock problem (Cung *et al*, 2000:188).

A *vertical build* of two rectangles $A = l_A \times w_A$ and $B = l_B \times w_B$ is a rectangle S_V having dimensions $\max(l_A, l_B) \times (w_A + w_B)$ and containing A and B . A *horizontal build* of A and B is a rectangle S_H of dimensions $(l_A + l_B) \times \max(w_A, w_B)$ that contains A and B .

Figure 5.1 illustrates how vertical (b) and horizontal (c) builds are constructed by using given demand rectangles (a).



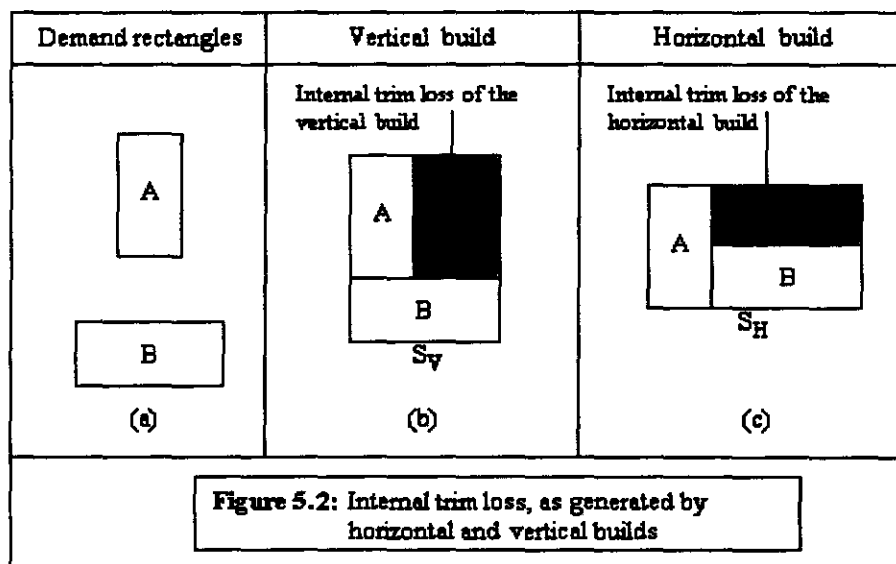
³ Refer to chapter 2, section 2.2.3, pages 14-15, for more information pertaining guillotine cuts

5.2.1.2 Trim loss

Trim loss, or waste as it is sometimes referred to, arise if there is a mismatch in the relevant dimensions of the rectangles being combined, and/or if a complete⁴ rectangle (a rectangle which cannot be used successfully as a component in a further vertical or horizontal build because then the resulting build would exceed either the length or width of the stock sheet, or both) does not have the same dimensions as the stock sheet. Three types of trim loss may occur, namely *internal trim loss*, *external trim loss* and *total trim loss*.

5.2.1.2.1 Internal trim loss

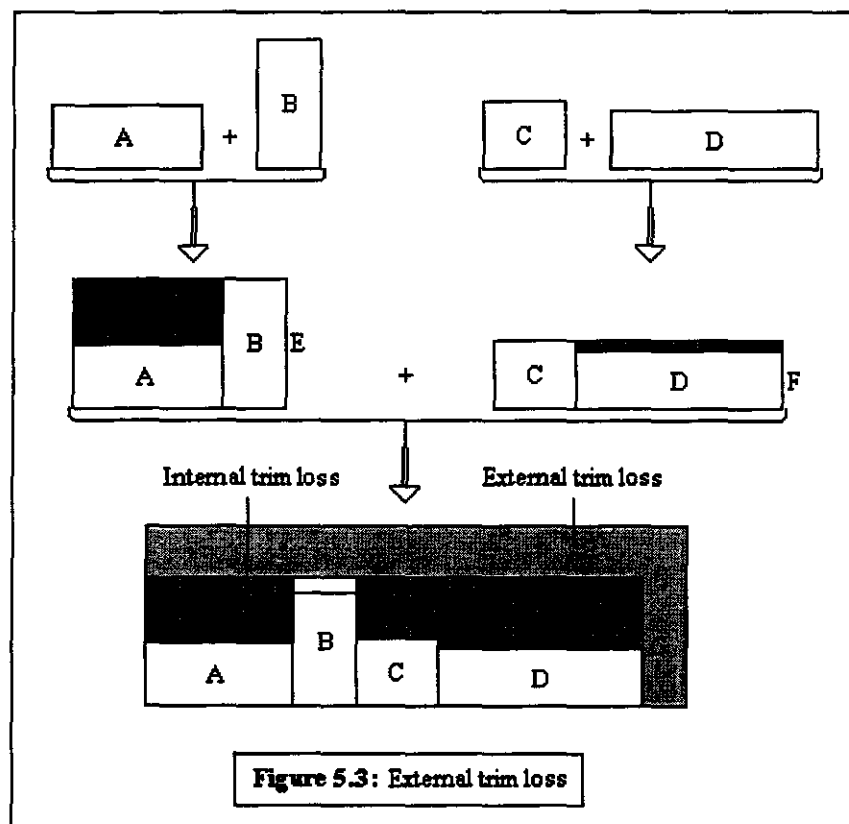
Internal trim loss is the area *within* a generated rectangle that is wasted. This idea is illustrated in figure 5.2, where (a) represents the demand rectangles and (b) and (c) illustrate how internal trim loss is generated for vertical and horizontal builds respectively.



⁴ Section 5.2.2, pages 75-77, uses the concept of completeness to enhance the original Wang algorithm

5.2.1.2.2 External trim loss

External trim loss is the area *outside* a complete demand rectangle or complete generated rectangle that is wasted when it is placed on (cut from) the stock sheet. To grasp the concept of external trim loss, consider figure 5.3. In this instance, rectangles *A* and *B* and rectangles *C* and *D* are combined to form *E* and *F* respectively. Subsequently, joining *E* and *F* generates rectangle *G*. From the previous definition of internal trim loss, it is evident that the shaded sections in rectangles *E* and *F* represent the internal trim loss of these rectangles respectively. Rectangle *G*, however, has its own internal trim loss, which is the sum of the internal trim loss of rectangles *E* and *F*, plus the trim loss generated when they were joined.



When the newly generated rectangle *G* is placed over (cut from) the stock sheet, an *L* section is formed as a residual on the stock sheet. If rectangle

G is a complete rectangle (a rectangle which cannot be used successfully as a component in a further vertical or horizontal build), then the external trim loss is the unused area remaining on the stock sheet, as indicated in figure 5.3.

5.2.1.2.3 Total trim loss

Total trim loss for a complete demand rectangle or complete generated rectangle is defined as the internal trim loss plus the external trim loss of that rectangle.

5.2.1.3 Acceptable waste percentages (β)

As was stated earlier (section 5.2.1, page 58), Wang proposed two algorithms that are based on her method of rectangle generation. Both algorithms utilize internal trim loss to evaluate whether a cut is feasible by comparing it to a parameter β , which represents the maximum acceptable waste percentage of any generated rectangle produced by the algorithms. Wang's first algorithm uses a parameter β_1 that is measured with respect to the area of the stock sheet $L \times W$. The second algorithm utilizes the parameter β_2 that is measured with respect to the area of a rectangle that was generated with either a horizontal build S_H or a vertical build S_V .

5.2.1.4 Wang's two original algorithms

The method as proposed by Wang will now be described and two algorithms to implement this method will follow the discussion. Firstly, at most b_i demand rectangles of type r_i must be specified with dimensions (ℓ_i, w_i) for each i , where $i = 1, 2, 3, \dots, n$ and ℓ_i is the length and w_i the width of type r_i . Furthermore, these demand rectangles will be cut from a stock sheet of length L and width W . Wang's method will now start an iterative building process where each rectangle i is combined horizontally and vertically with every other rectangle j . This combinatory process will then

form new rectangles that may contain internal trim loss, which must in turn also be combined with all the original demand rectangles and the newly generated rectangles. Only those generated rectangles with an internal trim loss of less than $\beta_1 L W$, will be stored for further consideration, the rest is discarded. It should be noted that rotated demand rectangles are also considered in this process, therefore the original demand rectangles are rotated and these rotated versions are also seen as part of the original demand rectangles in the building process. The combinatory process continues until no further rectangles can be generated with horizontal and vertical builds. The best solution (cutting pattern) for the predetermined β_1 value is then chosen as the generated rectangle with the least total trim loss (internal + external trim loss).

The two algorithms, as proposed by Wang (Wang, 1983:576-577), are illustrated in figure 5.4 and figure 5.5.

Step 1. (a) Choose a value for β_1 , $0 \leq \beta_1 \leq 1$;
 (b) Define $L^{(0)} = F^{(0)} = \{r_1, r_2, \dots, r_n\}$, and set $k = 1$;

Step 2. (a) Compute $F^{(k)}$ which is the set of all rectangles T satisfying:
 (i) T is formed by a feasible horizontal or vertical build of two rectangles from $L^{(k-1)}$;
 (ii) the amount of trim loss in T does not exceed $\beta_1 L W$; and
 (iii) those rectangles r_i appearing in T do not violate the bound constraints b_1, b_2, \dots, b_n ;
 (b) Set $L^{(k)} = L^{(k-1)} \cup F^{(k)}$. Remove any identical cutting pattern from $L^{(k)}$;

Step 3. If $F^{(k)}$ is nonempty, set $k \leftarrow k + 1$ and go to Step 2. Otherwise,

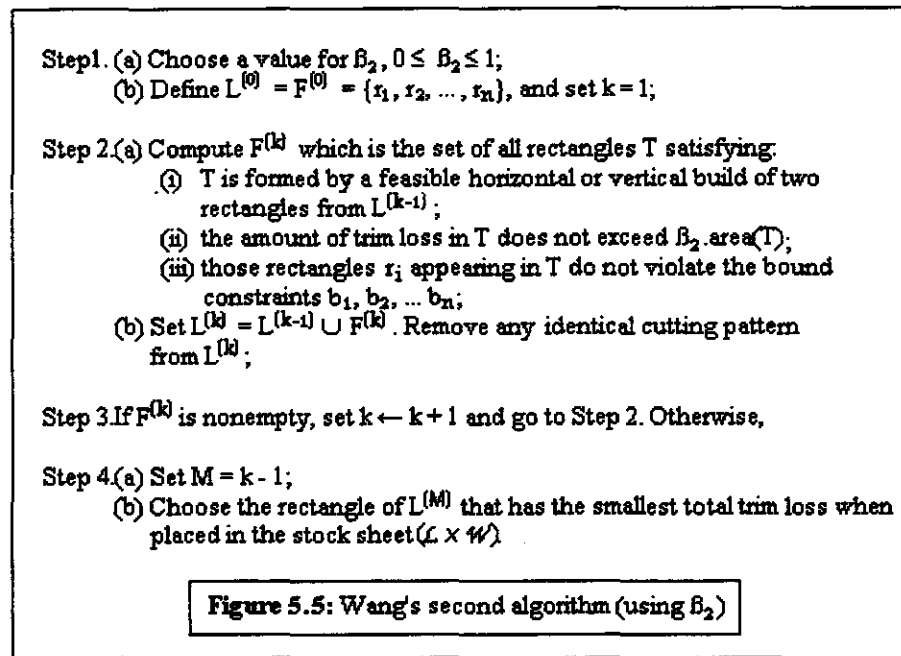
Step 4. (a) Set $M = k - 1$;
 (b) Choose the rectangle from $L^{(M)}$ that has the smallest total trim loss when placed in the stock sheet ($L \times W$).

Figure 5.4: Wang's first algorithm (using β_1)

As is lucidly apparent when comparing the algorithms presented in figures 5.4 and 5.5, the only difference between the two algorithms is that for

algorithm two, β_1 is replaced with β_2 in step 1.(a) and step 2.(a)(ii) of algorithm one is replaced in algorithm two by:

(a)(ii) the amount of trim loss in T does not exceed $\beta_2 \cdot \text{area}(T)$.



The algorithms in figures 5.4 and 5.5 show the iterative process of rectangle building used by Wang's method. It should be noted, however, that it only uses a single, predetermined beta value and executes the iterative process only once with this value. It is up to the user that implements the algorithm to increment the value of beta (by some chosen value) when an optimal (section 5.2.1.5, pages 65-66, deals with the optimality condition for the Wang algorithm) solution is not found with the initial beta, as Wang did not specify how this value is to be handled. Later in the thesis this problem is discussed further.

Furthermore, the Wang algorithms (WA) based on Wang's method of rectangle building are categorized as uninformed search strategies (breadth-first search) because no information concerning the problem

domain is considered when solving the cutting problem. The method merely sequentially generates all possible combinations with horizontal and vertical builds, and rejects those nodes (rectangles) that do not satisfy both the feasibility criteria of the problem and the pruning criterion. The pruning criteria for the two algorithms are based on the values assigned to β_1 and β_2 . Furthermore, the best solution is found only if the cost of each pruned away node is higher than the optimum value.

The method used by WA is therefore classified as an uninformed method although a numerical function is associated with each rectangle. The reason for this is that the function only prunes away some branches of the search tree, but it does not direct the search process.

5.2.1.5 Specifying values for β and optimality conditions

The two algorithms as given by Wang are nearly identical except for the two small differences as noted in section 5.2.1.4 (pages 62-65). For the purposes of this study, Wang's algorithm 1 will be used as a basis for further research (as used by Vasko and Daza). Therefore when referring to Wang's algorithm and β it is assumed that reference is being made to Wang's algorithm 1 and β_1 .

In her article *Two algorithms for constrained two-dimensional cutting stock problems*, Wang (Wang, 1983:573-577) does not explicitly specify which method to use to obtain an initial value for β . Neither does she state how this value should be handled if an optimal solution is not found using the initial chosen β value. From the literature, it would seem that many researchers have adapted a policy of starting with an initial β value of zero and systematically incrementing the value until an optimal solution is reached (refer to chapter 6, section 6.5, pages 131-136) for more information on initial β values and incrementing β values).

Wang (Wang, 1983:578) proves that the optimality condition for algorithm one can be written as shown in the following theorem:

Theorem T1: *If the total trim loss X of the pattern T obtained from algorithm one with a fixed value of β satisfies $X \leq \beta \cdot L \cdot W$ then T is an optimal pattern.*

where L and W represents the stock sheet length and stock sheet width respectively and the pattern T is the pattern with the least total trim loss found by algorithm one. It should be noted that Wang did not include this optimality condition in the two algorithms in figures 5.4 or 5.5. It is up to the user that implements these algorithms to determine whether a solution found by the algorithm using an initial beta value is optimal (with the optimality condition stated above), and if it is not, the beta value should be increased and the process repeated until an optimal value is found.

Sample C2DGC problems will now be solved with the Wang method to graphically illustrate how the rectangle building process functions.

5.2.1.6 Solving sample C2DGC problem instances with Wang's method

The process of solving a C2DGC problem, with the use of artificial intelligence search methods and an algorithm based on Wang's method, requires the integration of these two concepts. Firstly, artificial intelligence search methods offer a means by which the rectangle building process can be represented in a formal tree search structure. Secondly, the method conceived by Wang then provides the mechanism required to generate the nodes that form the search tree.

To better grasp the combinatorial process involved in the Wang method, illustrations are given of a breadth-first search representation of the Wang method utilizing horizontal and vertical builds to generate new rectangles, and then Wang's method is implemented using the A* method.

5.2.1.6.1 Implementing Wang's method using breadth-first search

By implementing the Wang method using a breadth-first search⁵ method, an algorithm is created that also expands nodes on a level-by-level fashion, but where some nodes are not considered for further expansion due to its excessive internal trim loss (in other words they are pruned). The value of beta (β) as defined by the Wang method, is used to determine whether a node is to be expanded further by determining if the internal trim loss of the node is less than or equal to stock sheet length (L).stock sheet width (W). β .

Problem	Stock plate length (L) and width (W)	Demand rectangles' length (L), width (W) and upper bound (b)
P5	(8,4)	(2,1,1); (3,3,3); (1,4,2); (2,2,3).
EP1	(5,5)	(1,1,1); (3,2,5).

Table extract: Two problem instances (P5 and EP1) from table 5.1

Two problem instances will be solved for the purposes of demonstrating how breadth-first search combined with the Wang algorithm functions. The first is a very small instance named **EP1** introduced in table 5.1. Because of its small size, all possible feasible builds that can be obtained with the Wang method using breadth-first search will be shown (figure 5.7). The second problem, **P5** as shown in table 5.1, is a slightly larger problem instance where all possible builds cannot be shown due to paper size constraints (these two problem instances are also shown in the table extract above).

Figure 5.6 is a partial representation of how feasible builds for problem **EP1** is obtained by the Wang rectangle building process with a beta value

⁵ Refer to chapter 3, section 3.3.1, pages 21-24

of 0.24. The least total trim loss pattern for this problem contains a total trim loss of 6, and only when the value of beta reaches 0.24 can it be proven that this is the optimal solution ($6 \leq 5.5 \cdot (0.24)$) (according to theorem T1).

As figure 5.6 demonstrates:

- the initial step taken by the algorithm is to generate patterns using only the basic demand rectangles as well as their rotated equivalents (builds 1-3, in part [a], figure 5.6). These three initial builds are located on the first level of the breath-first search tree;
- Now, the first step is to start combining rectangle number 1, as indicated in part [a], figure 5.6, with itself (by means of horizontal and vertical builds) if possible. A horizontal build with itself is not possible, as the new build would exceed the stock sheet length, therefore the fourth build is a vertical combination of build number 1 with itself (build 4, part [a], figure 5.6);
- Next, build number 1 must be combined horizontally and vertically, if possible, with build 2. Horizontal and vertical builds between builds 1 and 2 are possible, therefore builds 5 (part [b], figure 5.6) and 6 (part [c], figure 5.6) are generated;
- Next, build number 1 must be combined horizontally and vertically, if possible, with build 3. Horizontal and vertical builds between builds 1 and 3 are possible, therefore builds 7 (part [d], figure 5.6) and 8 (part [e], figure 5.6) are generated;



- Next, build number 1 must be combined horizontally and vertically, if possible, with build 4. Neither a horizontal nor vertical build can be made between builds 1 and 4. In both instances the stock sheet dimensions would be exceeded;
- Next, build number 1 must be combined horizontally and vertically, if possible, with build 5. A horizontal build is not possible, because the stock sheet length would be exceeded. A vertical build is possible, but if it is made, the resulting build would contain an internal trim loss value of 7. This is not allowed by the beta value, therefore the build is not stored;
- Next, build number 1 must be combined horizontally and vertically, if possible, with build 6. Neither a horizontal nor vertical build can be made between builds 1 and 6. In both instances the stock sheet dimensions would be exceeded;
- Next, build number 1 must be combined horizontally and vertically, if possible, with build 7. A horizontal build is not possible, because the stock sheet length would be exceeded. A vertical build is possible, and build number 9 (part [f], figure 5.6) is generated and stored;
- Next, build number 1 must be combined horizontally and vertically, if possible, with build 8. A horizontal build is not possible, because the stock sheet length would be exceeded. A vertical build is possible, and build number 10 (part [g], figure 5.6) is generated and stored;
- Next, build number 1 must be combined horizontally and vertically, if possible, with build 9. Neither a horizontal nor vertical build can be made between builds 1 and 9. In both instances the stock sheet dimensions would be exceeded;
- Next, build number 1 must be combined horizontally and vertically, if possible, with build 10. Neither a horizontal nor vertical build can be made between builds 1 and 10. In both instances the stock sheet dimensions would be exceeded;
- At this time, the end of the list of stored nodes has been reached, now to continue the process, all stored nodes will be combined with build number 2. Therefore, build number 2 must be combined

horizontally and vertically, if possible, with build 1. Horizontal and vertical builds between builds 2 and 1 are possible, therefore builds 11 (part [h], figure 5.6) and 12 (part [i], figure 5.6) are generated; and

- This process continues until all stored nodes have been combined, if possible, with themselves as well as each other.

Figure 5.6 shows that the sequence of node generation is accomplished by always combining the node that has to be expanded next with itself as well as all other currently stored nodes. This process will continue until the final stored node must be combined with itself as well as all the other stored nodes, but no feasible build is possible.

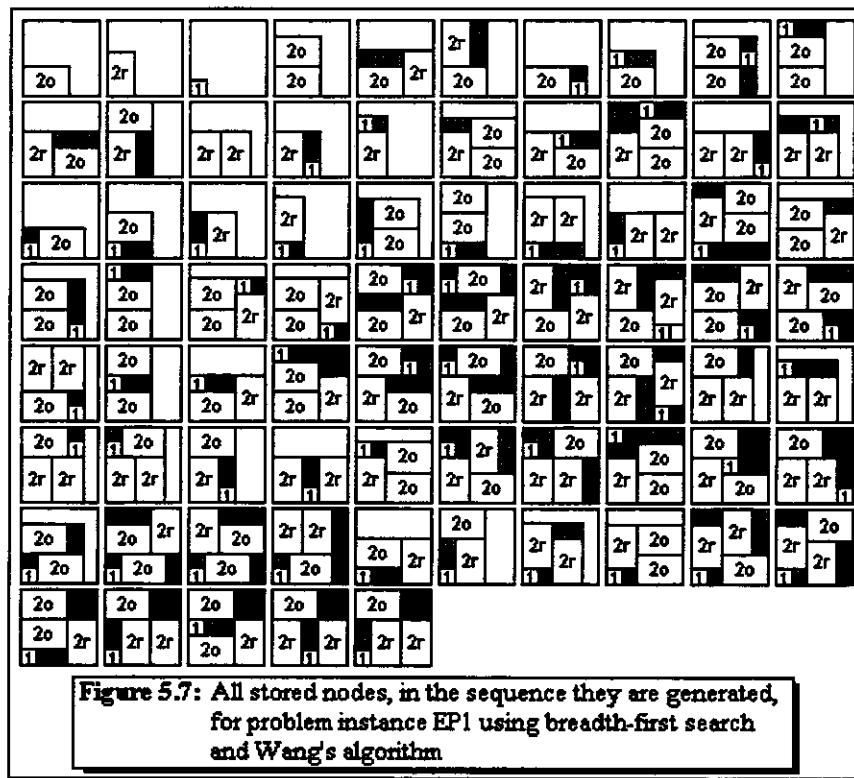


Figure 5.7 illustrates all stored builds that are generated by the Wang method combined with a breadth-first search. They are also shown in the order in which they are generated.

By examining the total trim loss of each complete build (a build which cannot be used successfully as a component in a further vertical or horizontal build because then the resulting build would exceed either the length or width of the stock sheet, or both) it is evident that several optimal solutions are found by the Wang method for this problem instance. Reasons for this are either unique placements of the original demand rectangles on the stock sheet, or symmetrical duplicate patterns. An example of an optimal solution for problem **EP1** is:



with a total trim loss of 6, which is less than or equal to $L.W.B$ $((5).(5).(0.24) = 6)$, and according to theorem T1 (section 5.2.1.5, page 66) the build shown above is then an optimal solution.

An important aspect that is illustrated by figure 5.7, which might not be obvious at first glance, is that the Wang method refrains from generating non-guillotine cutting patterns. For instance, the pattern shown in figure 5.8 could be generated for problem **EP1** and is one that contains no trim loss (in other words it represents the best possible layout for a non-guillotine cutting pattern), but it is not a guillotine pattern. This pattern is therefore not generated by the Wang method.

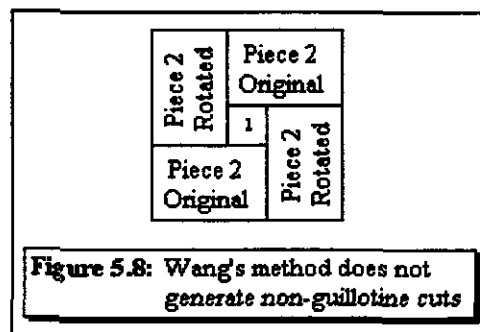
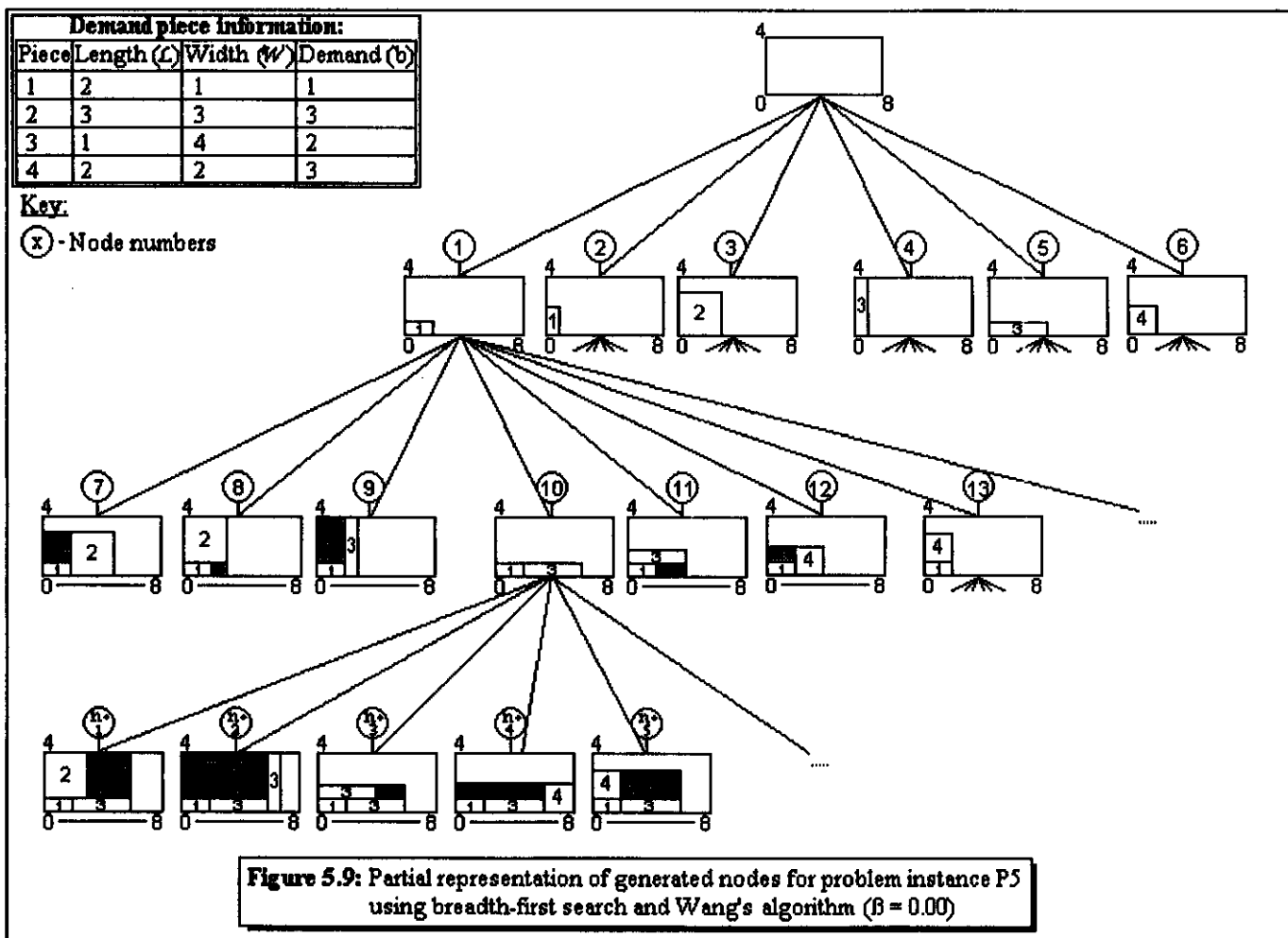


Figure 5.9 illustrates the initial steps when solving problem instance **P5**. As figure 5.9 demonstrates, the first step taken by the algorithm is to generate initial patterns using only the basic demand rectangles as well as their rotated equivalents. Figure 5.9 is a partially generated breadth-first search tree, with nodes that are pruned if the internal trim loss of the node is less than or equal to stock sheet length (L).stock sheet width (W). β . It should be noted though that because of paper-size constraints the entire second and third level of the search tree could not be displayed in figure 5.9. This is why the children of node 10 are numbered $n+1$, $n+2$, ..., $n+5$, where n is equal to the number of the final build of level 2. Furthermore, shaded areas within builds indicate the internal trim loss of those builds, and builds with lines drawn underneath them, are not expanded further because their internal trim losses exceed that allowed by β .

These first patterns (patterns 1 to 6) also include the rotated equivalents of the original demand rectangles. It should be noted though that if an initial demand rectangle is square it has the exact same length and width (demand rectangles 2 and 4), then only one pattern is created, because if the pattern is rotated its dimensions stay exactly the same. This simple addition to the algorithm prevents the generation of costly and unnecessary duplicate sub-trees in the search space. Equally important is to note that only the initial demand rectangles are rotated in the rectangle generation process. Rectangles containing more than one demand rectangle is not explicitly rotated because rotated versions of these rectangles will be created when the rotated original demand rectangles are combined. As an example, observe in figure 5.9 that rectangle number 9 originates from the horizontal combination of rectangles 1 and 4. A rotated equivalent to rectangle 9 will be created once rectangles 2 and 5 are vertically combined, therefore eliminating the need to explicitly rotated rectangle 9. In other words, the process of rectangle combination considers both horizontal and vertical builds, as well as the rotation of original demand rectangles (implying the rotation of all other, more complex builds).



By completing the iterative process as described in figure 5.4 (Wang's algorithm 1), the optimal solution for any given C2DGC problem will always be found, given a large enough value for beta, enough time and computer memory (Wang, 1983). It is, however, possible to enhance the performance of the Wang method by using an informed search method (such as the A* search method) that will guide the search more effectively and efficiently through the search space.

5.2.2 A computational improvement to Wang's algorithm one

Vasko (Vasko, 1989:109-115) studied the two algorithms as proposed by Wang, and made some computational improvements to the first of the two algorithms. He based this improvement on the ideas of horizontal and vertical completeness of cutting patterns, as well as dynamically diminishing the value of β . These three concepts will now be defined.

5.2.2.1 Horizontal completeness

A rectangle, resulting from successive vertical and/or horizontal builds, which cannot be used successfully as a component in a horizontal build because combining it with any r_i (or rotated r_i) would result in a rectangle with length exceeding L (stock sheet length) is said to be *horizontally complete*. Once a rectangle has been identified as horizontally complete, then it is no longer considered as a candidate component for horizontal builds. Also, once a rectangle is known to be horizontally complete, then its length is increased to L and its internal trim loss is recalculated.

5.2.2.2 Vertical completeness

A rectangle, resulting from successive vertical and/or horizontal builds, which cannot be used successfully as a component in a vertical build because combining it with any r_i (or rotated r_i) would result in a rectangle with width exceeding W (stock sheet width) is said to be *vertically complete*.

Once a rectangle has been identified as vertically complete, then it is no longer considered as a candidate component for vertical builds. Also, once a rectangle is known to be vertically complete, then its width is increased to w and its internal trim loss is recalculated.

5.2.2.3 Dynamically diminishing the β value

Before a rectangle enters $F^{(k)}$ (refer to figure 5.4, section 5.2.1.4, pages 62-65) it is tested for both horizontal and vertical completeness. If a rectangle is complete (both horizontally and vertically) its length is increased to L , its width is increased to w , and its internal trim loss (which is equal to its total trim loss because the rectangle is the same size as the stock sheet) is recalculated. If this internal trim loss is less than or equal to $\beta \cdot w \cdot L$ (Wang's optimality condition, theorem T1) the build is stored in $F^{(k)}$, otherwise it is not. Its internal trim loss is then compared to the best total trim loss known. If it is less than the best known total trim loss, the value of β is recalculated according to this new best total trim loss. It should be noted that when using an algorithm based on the branch-and-bound search method that utilizes the dynamic programming principle (chapter 4, section 4.2.8, pages 48-49) or algorithms based on methods derived from it (for instance the A* search method as discussed in chapter 4, section 4.2.9, pages 50-54), this dynamic diminishing of the beta value is accomplished implicitly. The reason for this is that an algorithm based on the branch-and-bound method utilizing the dynamic programming principle will never expand a rectangle with an internal trim loss that is greater than the total trim loss of the best known total trim loss rectangle.

The following is a description of the enhancements that Vasko made to Wang's algorithm one (figure 5.4, section 5.2.1.4, page 63):

- When defining $L^{(0)}$ each r_i and rotated r_i is checked for both vertical and horizontal completeness. If a rectangle is either horizontally or vertically complete (or both), then its length or width dimension (or

both dimensions) as well as its internal trim loss are appropriately adjusted. If a horizontally complete rectangle, vertically complete rectangle or a rectangle that is both horizontally and vertically has an internal trim loss greater than $\beta.W.L$ (Wang's optimality condition, theorem T1), then it is not included in $L^{(0)}$; otherwise it is included.

- In Step 2a⁶, if a rectangle T is horizontally (vertically) complete, then it is not considered during the horizontal (vertical) building process.
- In Step 2a⁵, once a horizontal (vertical) rectangle has been built, as part of the process of determining if a rectangle T should be included in $F^{(k)}$, a check for horizontal (vertical) completeness is made. If the rectangle just built is horizontally (vertically) complete, then its length (width) dimension and internal trim loss are adjusted appropriately. If the newly calculated internal trim loss of rectangle T is less than or equal to $\beta.W.L$ the rectangle is included in $F^{(k)}$, otherwise it is not.
- If a rectangle T should enter $F^{(k)}$ and it is complete (both horizontally and vertically), then its length and width dimension as well as its internal trim loss are adjusted appropriately. If the newly calculated internal trim loss of rectangle T is less than or equal to $\beta.W.L$ the rectangle is included in $F^{(k)}$, otherwise it is not. If it is entered into $F^{(k)}$ the internal trim loss of T is compared to the best total trim loss known; β is updated if the internal trim loss is less than the best total trim loss.

5.2.3 The modified Wang method (WAM)

According to Daza et al (Daza et al, 1995:635), Oliveira and Ferreira (1990) studied Wang's method and resulting algorithms extensively and they developed an improvement on her method. It is denoted as the WAM method, which is an abbreviation for the phrase Wang's modified method.

⁶ Refer to Wang's algorithm one, Step 2a, section 5.2.1.4, figure 5.4, page 63

The WAM method, according to Daza et al (Daza et al, 1995:635), requires that for each newly generated rectangle three feasibility criteria be considered:

- Rectangle dimensions must be less than or equal to the stock sheet dimensions;
- The estimated total trim loss (internal trim loss + estimated external trim loss) must be less than a certain percentage of the stock sheet's area. The concepts of estimated external trim loss and estimated total trim loss are described in sections 5.2.3.1.1 (pages 78-83) and 5.2.3.1.2 (page 87) respectively; and
- The number of demand rectangles of a specific type cut from the sheet must be less than or equal to its upper bound.

These three criteria closely resemble those listed for Wang's original method, but the second criterion differs slightly. The WAM method now requires internal as well as estimated external trim loss (instead of only internal trim loss as for WA) to be taken into account when evaluating the criteria.

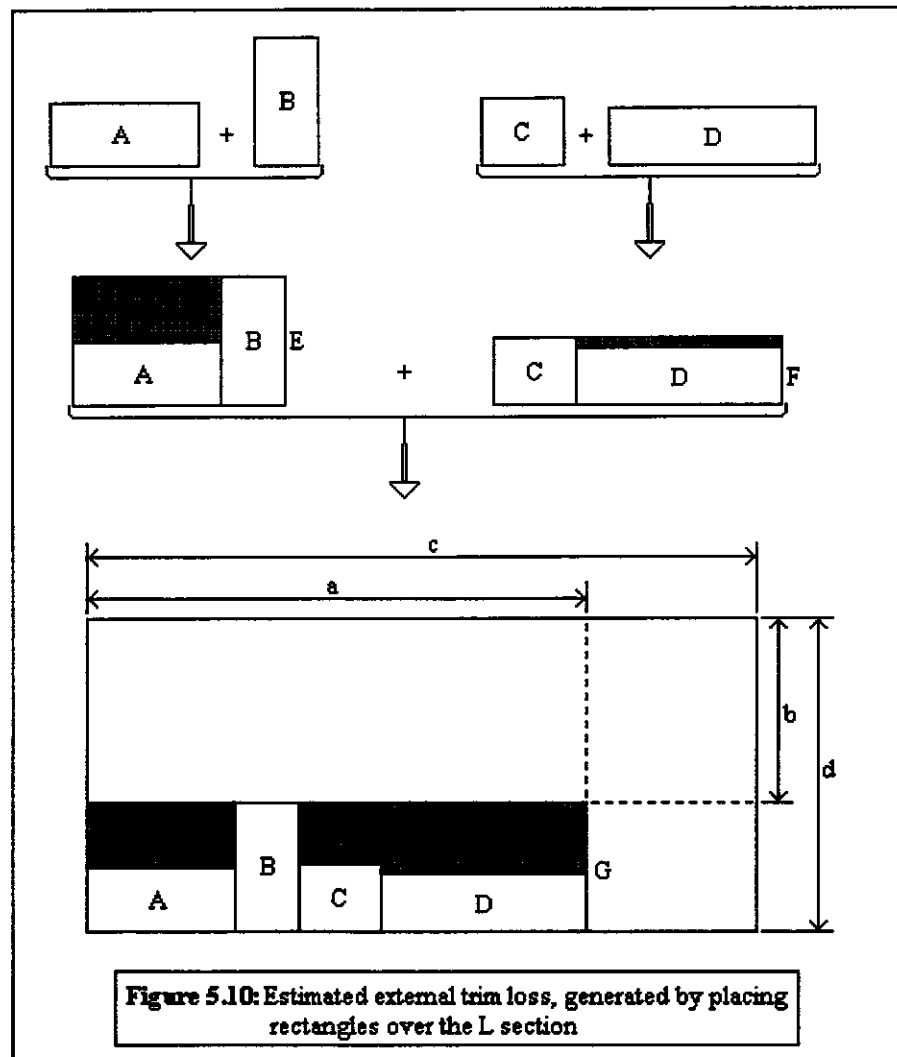
5.2.3.1 Trim loss

Internal, external and total trim loss were discussed in sections 5.2.1.2.1 – 5.2.1.2.3 (pages 60-62) as it is used extensively by WA as well as by the WAM method. Estimated external trim loss and estimated total trim loss, on the other hand, are new concepts introduced and used by the WAM method.

5.2.3.1.1 Estimated external trim loss

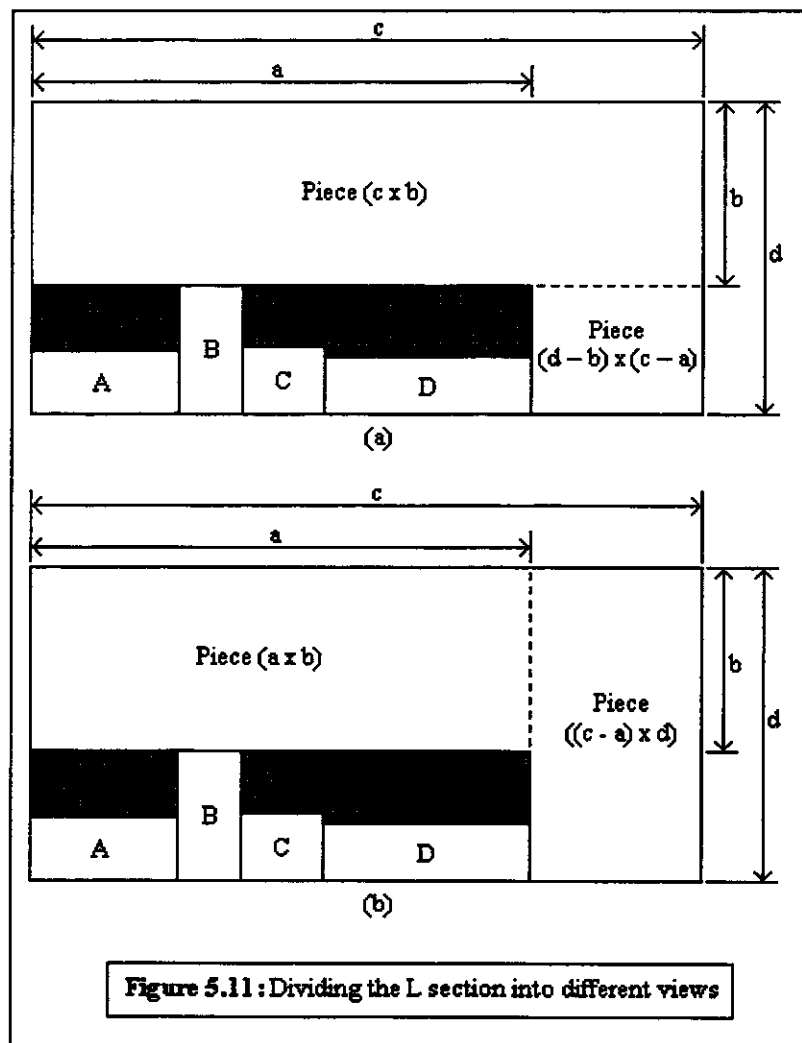
To grasp the concept of estimated external trim loss, consider figure 5.10. In this instance, rectangles *A* and *B* and rectangles *C* and *D* are combined to form *E* and *F* respectively. Subsequently, joining *E* and *F* generates rectangle *G*. From the previous definition of internal trim loss, it is evident

that the shaded sections in rectangles E and F represent the internal trim loss of these rectangles respectively. Rectangle G , however, has its own internal trim loss, which is the sum of the internal trim loss of rectangles E and F , plus the trim loss generated when they were joined.



When the newly generated rectangle G is placed over the stock sheet, an L section is formed as a residual on the stock sheet. At this point in the rectangle building process, it is evident that remaining available demand rectangles must be used to fill the area shown as the L section in figure 5.10. If this area could be filled exactly with the remaining demand rectangles leaving no extra trim loss, the situation would have been ideal,

but unfortunately this does not happen often. Therefore, a method is needed that can estimate the amount of trim loss that will be generated when remaining demand rectangles are placed over the L section. Since we are working with guillotine cuts this L section can be viewed in different ways. According to figure 5.10 one way is to consider it to be a combination of two rectangles with dimensions $(c \times b)$ and $(d - b) \times (c - a)$, and another is by viewing it as a combination of two rectangles with dimensions $(a \times b)$ and $((c - a) \times d)$. These two views are illustrated in figure 5.11.



Each rectangle $(c \times b)$, $(d - b) \times (c - a)$, $(a \times b)$ and $((c - a) \times d)$ may be cut according to a certain pattern and this process generates some internal and external trim loss. An estimate of the total trim loss for such rectangles is therefore required, and consequently a method to accomplish this is discussed.

Oliveira and Ferreira (1990) proposed a method of estimating this trim loss. Firstly, the L section is divided (as shown in figure 5.11) and by using the exact method as prescribed by Gilmore and Gomory (1966), rectangles are theoretically assigned over the divided L section. This exact process involves solving an unbounded two-dimensional knapsack problem (section 5.2.3.1.1.1, pages 83-87, explains this problem and process) once, before the rectangle building process starts, for the entire stock sheet dimension (L, W) . This unbounded knapsack is solved using a dynamic programming procedure, therefore every possible dimension included within the larger stock sheet dimension will have an internal trim loss value associated to it when the procedure finishes. These values are then stored in a lookup table and used as estimated external trim loss for any given dimension within the stock sheet. These values are underestimates of the exact external trim loss that could be incurred, because of the relaxation (the upper bound b_i on the maximum number of demand rectangles for each type r_i is ignored) of the model that was used.

If T_{ni} denotes the estimated internal trim loss on node n in the search space when rectangles are assigned over piece i , and additionally it is denoted that, according to figure 5.11, (which is a typical representation of a node n) i has been defined to be the following sequence of pieces⁷:

- $i = 1$, the piece of dimensions $c \times b$;
- $i = 2$, the piece of dimensions $(d - b) \times (c - a)$;
- $i = 3$, the piece of dimensions $a \times b$; and
- $i = 4$, the piece of dimensions $(c - a) \times d$.

⁷ Note that the L section in figure 5.11 is covered in two different ways where i is equal to 1 and 2 and where i is equal to 3 and 4

Then the value of $h(n)$ in function 5.1 represents the estimated trim loss of node n in the search space:

$$h(n) = \text{Min}\{T_{n1} + T_{n2}, T_{n3} + T_{n4}\} \quad (5.1)$$

According to Daza et al (Daza et al, 1995:639), however, by using small examples it can be shown that function 5.1 is neither monotone⁸ nor admissible⁶. This situation may be improved upon by taking future trim loss into account, with the purpose of diminishing the value of $h(n)$. The reason for this is that, because function 5.1 is not admissible, it yields values for $h(n)$ that are greater than the exact ones (overestimates⁹). To satisfy the admissibility condition, the following function is defined by :

$$h_2(n) = \text{Max}\{T_{n1} + T_{n4} - T_{n5}, 0\} \quad (5.2)$$

where T_{n5} is the unconstrained trim loss of the rectangle in the upper right hand corner of the stock sheet of node n . In reference to figure 5.10 (page 79), this is the rectangle of dimensions $b \times (c - a)$. Note that as a condition $h_2(n) \leq h(n)$ must be valid, and that if $T_{n5} \geq T_{n1} + T_{n4}$ then the condition $h_2(n) = 0$ is satisfied. On the other hand, if $T_{n5} < T_{n1} + T_{n4}$, two possibilities exist. The first possibility is that $h(n) = T_{n1} + T_{n2}$ and $h_2(n) = T_{n1} + T_{n4} - T_{n5}$ also satisfies the condition because $T_{n4} \leq T_{n2} + T_{n5}$. Secondly, if $h(n) = T_{n3} + T_{n4}$ and $h_2(n) = T_{n1} + T_{n4} - T_{n5}$, then similarly we have $T_{n1} \leq T_{n3} + T_{n5}$.

Function 5.2 is more effective than function 5.1, but Daza et al (Daza et al, 1995:639-641) proves that in order to obtain monotonicity, the function should be rewritten as:

$$h_3(n) = \begin{cases} \text{Max}\{T_{n1} + T_{n4} - T_{n5}, 0\} & \text{if } h_3(n) \text{ is monotone,} \\ g(n) + h(n_i) - g(n_j) & \text{in other cases} \end{cases} \quad (5.3)$$

⁸ Refer to chapter 4, sections 4.2.9.1 and 4.2.9.2, pages 50-51, for further information pertaining to admissibility and monotonicity

⁹ Refer to chapter 4, sections 4.2.9.4 and 4.2.9.5, pages 51-53, for further information pertaining to underestimating and overestimating the value of $h(n)$

Where node n is obtained from a horizontal or vertical build between nodes n_i and n_j . Note that the admissibility of this function is a direct consequence of its monotonicity. A generalization of the monotonicity requirement for C2DGC problems is represented by the following inequality:

$$h(n_i) - h(n) \leq g(n_j) + c(n_i, n_j) \quad (5.4)$$

where $c(n_i, n_j)$ is the additional trim loss generated by the horizontal or vertical build resulting from the combination of nodes n_i and n_j so that we can write that $g(n) = g(n_i) + g(n_j) + c(n_i, n_j)$. Refer to section 4.2.9.2, pages 51-52, for further information pertaining the cost function $c(n_i, n_j)$.

5.2.3.1.1 Unbounded two-dimensional knapsacks

As stated above, the unbounded two-dimensional knapsack problem is solved by the exact method as described by Gilmore and Gomory (1966). Gilmore and Gomory (Gilmore & Gomory, 1966:1045) give a broad definition of problems falling in the knapsack category:

"Knapsack problems can arise directly in two ways. Firstly, a portion of space is being packed with objects, each having a value, and the knapsack problem is then to find the most valuable packing. Alternatively and equivalently, if a portion of space is being cut into pieces of different values, the knapsack problem is to find the most valuable way of cutting."

An example of the first set of problems would be the packing of containers in one, two or three dimensions. An example of the second set of knapsack problems would be the cutting of glass or any other material from a larger stock sheet or multiple stock sheets, also in one, two or three dimensions. The difference between the bounded and the unbounded problems are that with bounded problem instances only a certain number of each demand rectangle is required, and with

unbounded problem instances an unlimited number of each demand rectangle is required. Gilmore and Gomory (1966:1046) define a so-called two-dimensional knapsack function G as follows:

“One is given rectangles of positive dimensions (ℓ_i, w_i) , $i = 1, \dots, m$ that have nonnegative values Π_1, \dots, Π_m associated with them; then $G(x, y)$ is the maximum of $\Pi_1 Z_1 + \dots + \Pi_m Z_m$, where Z_1, \dots, Z_m are nonnegative integers such that there exists a way of dividing a rectangle (x, y) into Z_i rectangles (ℓ_i, w_i) , for $i = 1, \dots, m$.”

The calculation of $G(x, y)$ for given x and y is not an easy task. Therefore, for problem instances utilizing the guillotine cutting constraint, Gilmore and Gomory (Gilmore & Gomory, 1966:1046) defines another knapsack function F similar to G .

“ F is defined like G except that in dividing a rectangle (x, y) into Z_i rectangles (ℓ_i, w_i) for $i = 1, \dots, m$, the following restriction is imposed: The division must take place by a series of straight lines that extend from one edge of a stock sheet to the opposite edge, parallel to the other two edges; we will call them ‘guillotine cuts’.”

Furthermore, they prove a functional theorem for such knapsack functions F . From the discussion above it is clear that when the rectangles (ℓ_i, w_i) of worths Π_i , $i = 1, \dots, m$, are given, the knapsack function $F(x, y)$ defined from them satisfies the following three sets of inequalities:

$$F(x, y) \geq 0, \tag{5.5}$$

$$F(x_1 + x_2, y) \geq F(x_1, y) + F(x_2, y), \tag{5.6}$$

$$F(x, y_1 + y_2) \geq F(x, y_1) + F(x, y_2), \tag{5.7}$$

$$F(\ell_i, w_i) \geq \Pi_i \quad (i = 1, \dots, m) \tag{5.8}$$

Inequalities (5.5) and (5.7) speak for themselves. The inequalities (5.6) are a consequence of the permitted method of cutting a large rectangle

(x,y) into the smaller rectangles (ℓ_i, w_i) . F is not the only function to satisfy these inequalities, although it is the minimal function in the sense of the following theorem.

Theorem T2: F is a knapsack function defined from the rectangles (ℓ_i, w_i) with values Π_i , $i = 1, \dots, m$, if and only if F satisfies (5.5), (5.6), (5.7) and (5.8), and (5.8) is: For any G satisfying (5.5) to (5.7), $F(x,y) \leq G(x,y)$ for all x and y .

An efficient method for computing the two-dimensional knapsack function F , according to Gilmore and Gomory (Gilmore & Gomory, 1966:1051) is by a modified dynamic programming technique that is based upon the functional equation:

$$F(x,y) = \max\{F_0(x,y), F(x_1,y) + F(x_2,y), F(x,y_1) + F(x,y_2); \quad (5.9)$$

$$x \geq x_1 + x_2, 0 < x_1 \leq x_2, y \geq y_1 + y_2, \text{ and } 0 < y_1 \leq y_2\}$$

where $F_0(x,y) = \max\{0, \Pi_i; \ell_i \leq x \text{ and } w_i \leq y\}.$

They then prove the following theorem to justify the above-mentioned method.

Theorem T3: The functional equation (5.9) is satisfied only by the knapsack function.

Using these fundamental theorems and refining them computationally, Gilmore and Gomory (Gilmore & Gomory, 1966:1067-1068) eventually propose an algorithm they call the '*Basic Two-Dimensional Step-Off Algorithm*'. It can be divided into four major parts, and in part I only general initializations take place. In parts II and III it is assumed that a general step-off point has been defined. In part II step-offs from (x_2, y_2) take place along the line $y = y_2$ while in part III step-offs take place along the line $x = x_2$, where the former step-off are taken first. Finally in

part IV a new step-off point is determined in the order discussed above. Recall that F_0 is defined in equation 5.9:

- I. Let $F(x,y) = F_0(x,y)$ for $0 \leq x \leq L$ and $0 \leq y \leq W$. Let $I^*(l_i, w_i) = l_i$ and $w^*(l_i, w_i) = w_i$ for $i=1, \dots, m$, and let $I^*(0,0) = w^*(0,0) = 0$. Let $x_2 = y_2 = 1$ and go to step II.1;
- II.
 1. Let $x_1 = 1$;
 2. If $x_1 + x_2 \leq L$ then let $V = F^*(x_1, y_2) + F^*(x_2 + y_2)$ and go to step II.3. Otherwise go to step III.1;
 3. If $V > F^*(x_1 + x_2, y_2)$ then let $F^*(x_1 + x_2, y_2) = V$, let $I^*(x_1 + x_2, y_2) = x_1$, and let $w^*(x_1 + x_2, y_2) = y_2$ and go to step II.4. If $V = F^*(x_1 + x_2, y_2)$ then let $I^*(x_1 + x_2, y_2) = x_1$ and go to II.2. Otherwise go to step III.1.
 4. If $x_1 < x_2$ then let $x_1 = x_1 + 1$ and go to step II.2. Otherwise go to step III.1;
- III.
 1. Let $y_1 = 1$;
 2. If $y_1 + y_2 \leq W$ then let $V = F^*(x_2, y_1) + F^*(x_2 + y_2)$ and go to step III.3. Otherwise go to step IV.1;
 3. If $V > F^*(x_2, y_1 + y_2)$ then let $F^*(x_2, y_1 + y_2) = V$, let $w^*(x_2, y_1 + y_2) = y_1$, and let $I^*(x_2, y_1 + y_2) = x_2$ and go to step III.4. If $V = F^*(x_2, y_1 + y_2)$ then let $w^*(x_2, y_1 + y_2) = y_1$ and go to III.4. Otherwise go to step III.4;
 4. If $y_1 < y_2$ then let $y_1 = y_1 + 1$ and go to step III.2. Otherwise go to step IV.1;
- IV.
 1. If $x_2 < L$ then let $x_2 = x_2 + 1$ and go to step II.1. Otherwise go to step III.2;
 2. If $y_2 < W$ then let $y_2 = y_2 + 1$ and $x_2 = 1$ and go to step II.1. Otherwise stop.

Finally, the following implication of the Gilmore and Gomory method for computing knapsack functions is worthy of notification. Since the knapsack function considered by Gilmore and Gomory differ from the C2DGC problem considered in this thesis and more specifically from the model treated by Wang, it is interesting that since they treat the

unconstrained problem, they find knapsack function values $F(x,y)$ for the relevant (x,y) points on the grid by solving a relaxation of the C2DGC problem. As such, the solutions give underestimates of the (internal) trim loss. These values are then used to compute estimated external trim loss.

5.2.3.1.2 Estimated total trim loss

Estimated total trim loss is defined as the sum of the internal and estimated external trim loss for any given original demand rectangle or generated rectangle.

5.2.3.2 Solving sample C2DGC problems with the modified Wang method (WAM) using the A* search method

Oliveira and Ferreira (1990) studied the Wang method (WA) extensively and developed an improved version of the method. The new method was called the modified Wang method (WAM) and introduced the idea of using future information concerning the state space to predict which partial cuts would potentially yield better results than other partial cuts when evaluated further. The result of their study was the introduction of estimated external trim loss, calculated by solving an unconstrained two-dimensional knapsack problem as described in section 5.2.3.1.1.1 (pages 83-87). The heuristic evaluation function 5.1 (section 5.2.3.1.1, page 81) as proposed by Oliveira and Ferreira (Oliveira and Ferreira, 1990:257-259), can be used to calculate the estimated external trim loss ($h(n)$). Unfortunately, this evaluation function is neither admissible nor monotone¹⁰. Therefore, it is suggested that the evaluation function 5.3 (section 5.2.3.1.1, page 82) as proposed by Daza et al (Daza et al, 1995:639) be used for the calculation of the estimated external trim loss.

As described in chapter 4, the A* search method uses a heuristic function to evaluate the cost of each node as it is constructed during the search

¹⁰ Refer to section 5.2.3.1.1, pages 78-83, for a detailed discussion of the evaluation function

process, and if the cost of the node is less than a certain set cost, the node will be expanded and stored, otherwise it will not be expanded or stored. The A* method's heuristic function (refer to chapter 4, section 4.2.9, pages 50-54) requires two parameters to determine whether a node should be evaluated or not, and can be written as:

$$f(n) = g(n) + h(n)$$

where $g(n)$ is the cost of the node up to the current state, and $h(n)$ is an estimated cost that may be incurred if the node is part of a complete cutting pattern for the sheet. By adding these two values, a node cost, $f(n)$, is calculated.

To use the A* method employing the modified Wang method's pruning criteria, the value of $g(n)$ will be set to the internal trim loss of the current node, and the value of $h(n)$ will be set to the estimated external trim loss as shown in evaluation function 5.3 (section 5.2.3.1.1, page 82). This leads to a situation where some domain specific knowledge as well as future information is used to evaluate the current node. Furthermore, throughout the process of rectangle combination, the three feasibility criteria, as described in section 5.2.3 (page 78), are constantly kept under consideration.

When the pruning criterion as presented by Wang is employed with a β value of 0.00 for problem P5, then the pruning value is:

$$\beta \cdot \text{stock length} \cdot \text{stock width} = (0.00) \cdot (8) \cdot (4) = 0$$

With a pruning value of 0, all nodes in the search space with an estimated total trim loss (internal trim loss plus estimated external trim loss) of more than 0 are not considered for further evaluation.

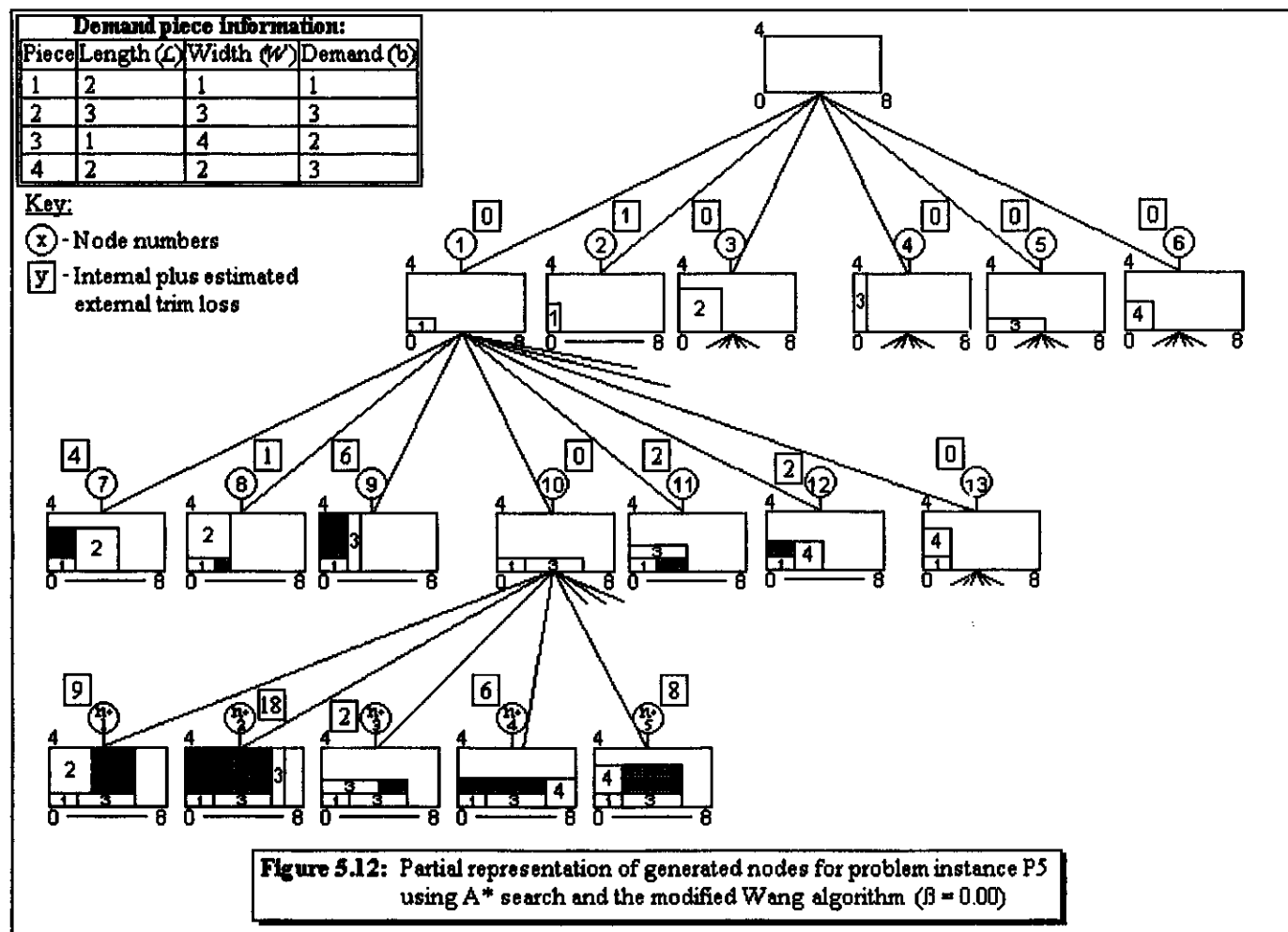
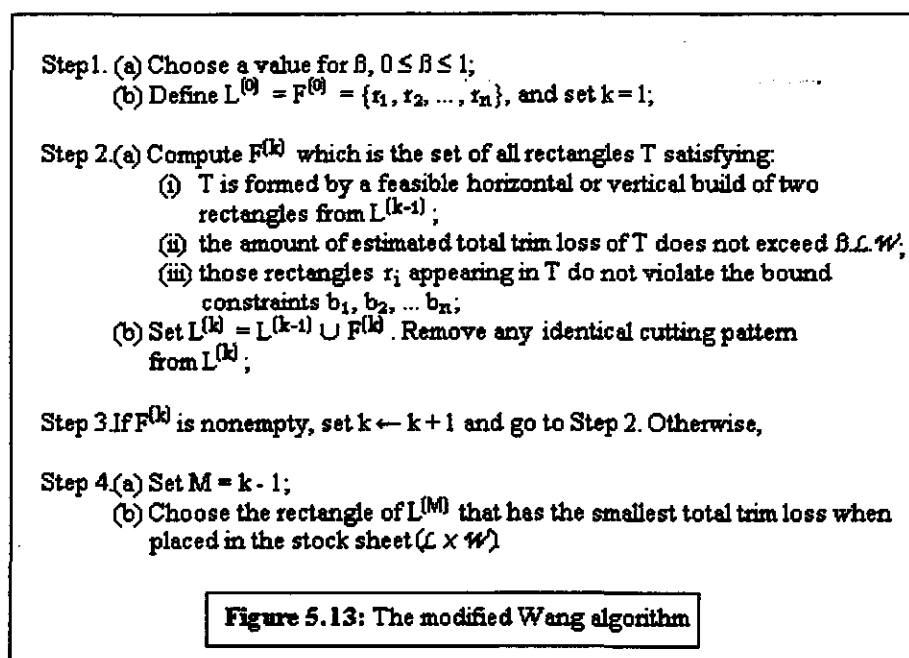


Figure 5.12 shows the partially developed search tree with estimated total trim loss associated with each node shown in the boxes next to the node numbers. These values are calculated using evaluation function $f(n) = g(n) + h(n)$, and the value of β is chosen as 0.00. The value of $h(n)$ is calculated using the evaluation function 5.3 (section 5.2.3.1.1, page 82). As was discussed in section 5.2.1 (page 58), the method employed by the Wang algorithms (WA) is an uninformed method that utilized a pruning criterion. The WAM method, on the other hand, can be classified as an informed method because information about the search space is used to guide the search. This domain-specific knowledge is represented in the form of estimated external trim loss and can be calculated as illustrated in section 5.2.3.1.1, pages 78-83. Figure 5.13 represents the WAM algorithm:



5.3 Non-exact methods to solve the C2DGC problem

As was stated in chapter 1, section 1.2.2 (pages 4-5), non-exact methods to solve stock cutting problems, including the C2DGC problem, exist in the form of heuristic search methods such as greedy searches, beam searches, depth-

limited searches and hill-climbing searches. These search methods can be combined with the Wang and modified Wang methods, but it yields algorithms that are not guaranteed to find the optimal solution for any given problem instance. This does not mean that non-exact algorithms are of no use. On the contrary, some of these algorithms will be implemented and used to calculate upper bounds for beta (β). Refer to chapter 6, section 6.4.2 (pages 129-131) for numerical results obtained from using beam search to generate upper bounds for β and defining a waste gap.

5.4 Exact methods' algorithmic properties and considerations

One of the advantages of using non-exact methods to derive problem solving algorithms, is that in most cases it executes and finds solutions faster than exact methods. Therefore some industries, where raw materials are cheap and time is a more important resource, will prefer to use non-exact algorithms that do not necessarily find optimal solutions but do find acceptable solutions fast.

Exact algorithms, on the other hand, will usually spend more time than non-exact ones when solving a given problem instance, but the quality of the solution will be better. For this reason, and in the interest of scientific experimentation, exact methods and algorithms are preferred for this study.

As was stated in section 5.2 (page 57), the Wang and modified Wang methods will be used as the basic methods from which enhanced algorithms will be derived in this thesis. After completing a theoretical study of the Wang and modified Wang methods, a few problems presented itself, as indicated below.

5.4.1 Problems with the Wang method

As was stated in section 5.2.1.4 (pages 62-65), algorithms derived from the Wang method are uninformed, because it only prunes certain areas of the

search space, with no heuristic function that leads the search in a certain direction. This is the first major drawback of the Wang method. Secondly, Wang did not define a strategy for the handling of the beta value. Neither in terms of calculating an initial value for beta nor for the increase of the beta value when an optimal solution was not found by the algorithm with the initial beta value. Lastly, algorithms based on the Wang method generate great amounts of data in the rectangle building process, and the inherent information in this data is mostly ignored and not utilized.

5.4.2 Problems with the WAM method

To remedy the fact that the Wang method is an uninformed one, Oliveira and Ferreira (1990) devised the modified Wang method (WAM). The WAM method utilizes a heuristic function (see section 5.2.3, pages 77-87) that guides the search process and this may yield computational improvements over the Wang method as were demonstrated by using example instances by Oliveira and Ferreira (Oliveira & Ferreira, 1989:260-265). The heuristic function uses lower bounds that represent underestimates of external trim loss. These are calculated once at the beginning of the solving process. These bounds are calculated using Gilmore and Gomory's unbounded two-dimensional knapsack function (refer to section 5.2.3.1.1.1, pages 83-87). The problem is that these bounds are not tight enough, causing the algorithm to underestimate the value of $h(n)$ significantly, therefore generating more nodes in the process than a more informed heuristic would. Secondly, the lack of a good strategy to handle the value of beta still remains a problem as for the Wang method. Lastly, for small textbook problems, the calculation of lookup tables for the lower bounds using the unbounded two-dimensional knapsack function is feasible. Unfortunately though, for larger industry sized problems, the calculation of these lower bounds becomes tedious and a time-consuming process (refer to chapter 7, section 7.2.8, pages 171-178).

The problems mentioned in sections 5.4.1 and 5.4.2 lead the author to believe that there is still much opportunity to enhance these methods and the

algorithms derived from it. Therefore, an effort will be made to investigate the following aspects:

- lower bounds as it is used by the modified Wang method. Sharper lower bounds are needed and this should lead to faster execution times for algorithms derived from the WAM method;
- the use of non-exact methods to generate upper bounds;
- the use of information contained in the data generated by the building process involved in the Wang and modified Wang method's algorithms. It was observed that the algorithms produce a great deal of data, but most of it is discarded and not used by the algorithms. Firstly, the author proposes the use of this data to generate upper bounds for the problem instance. Secondly these upper bounds can be used to manage the value of beta by defining a waste gap;
- strategies to handle the value of beta, as this is a problem for both the Wang and modified Wang methods; and
- lower bound calculation and lookup table generation for larger, industry-sized problems. For larger problems the lower bound calculation process becomes tedious and time-consuming.

5.5 Summary

Chapter 5 concluded the discussion on existing exact and non-exact problem solving methods for the C2DGC problem. Furthermore, it postulated that inefficiencies still exist within these existing methods (section 5.4, pages 90-93). Chapter 6 will delve deeper into these issues by presenting algorithmic modifications and additions, most of which is proposed by the author, to enhance the performance of the existing algorithms.

CHAPTER 6: Algorithmic enhancements

6.1 Introduction

The Wang (chapter 5, section 5.2.1, pages 58-75) and modified Wang (chapter 5, section 5.2.3, pages 77-87) methods are basically enumeration methods that generate cutting patterns by means of successive horizontal and vertical builds. These methods also use a pruning criterion, which allows only builds containing less than a certain amount of trim loss to be stored, and the rest is discarded. This pruning criterion utilizes a value called beta (β), as a proportion parameter, from which an upper bound on the amount of trim loss that is allowed for builds is calculated. Therefore, upper bounds play an important role in the Wang and modified Wang methods. Usually, algorithms based on these methods will start with a beta value equal to 0.00 and if the optimal solution is not found using this initial value, beta is increased by an arbitrary value (usually 0.01). For many problems 0.00 is not a suitable initial value for beta, and for problems requiring a large value of beta to reach an optimal solution, the arbitrary increase of 0.01 for the value of beta is arguably not the best strategy. Therefore, it is obvious that an initial lower bound on the value of beta would also be useful, as well as a strategy to manage the value of beta.

Furthermore, the modified Wang method relies heavily on lower bounds as produced by Gilmore and Gomory's unbounded two-dimensional knapsack function. These lower bounds are mostly not sharp enough, and for this reason an improvement to the lower bounds may expedite the solution process.

This chapter deals with certain algorithmic enhancements that can be made to the existing algorithms derived from the Wang and modified Wang methods. Section 6.2 will describe optimization techniques that can be applied to the existing algorithms to increase performance. Section 6.3 describes a method devised by the author to sharpen the underestimates as given by the

unbounded two-dimensional knapsack, which is used by the modified Wang method. Section 6.4 describes a method devised by the author to generate upper bounds using data generated by the Wang and modified Wang methods, and indicates how these upper bounds can be used to manage the value of beta. Furthermore, section 6.4 describes a beam search method to also calculate upper bounds. Section 6.5 further describes strategies for determining an initial lower bound on the value of beta and also effective means of incrementing the beta value.

6.2 Optimization techniques

The basic knowledge to find solutions for the C2DGC problem using the Wang and modified Wang methods has now been discussed. It should, however, be noted that some other strategies such as the exploitation of symmetry, the order of cutting and demand rectangle rotation may lead to better results when implemented effectively. Therefore, these concepts will now be discussed.

6.2.1 Detection of duplicate patterns: symmetric strategies

When algorithms for solving the C2DGC problem are derived from, for example, the WA and WAM methods, the probability of generating symmetrically duplicate patterns is high. To prevent these algorithms from generating duplicate patterns, all previously generated and accepted patterns are stored in a list, denoted as CList. Any new qualifying pattern is compared to the patterns in CList to establish whether the specific pattern already exists or is symmetrically equivalent to a pattern in CList.

According to Cung et al (Cung et al, 2000:196), authors such as Tschöke and Holthöfer (1995) have proposed that a procedure be applied to every newly generated pattern in order to detect an equivalent pattern throughout CList. Although their strategy is very effective and detects many duplicate patterns, it requires a great deal of computational effort because of the great

number of comparisons that have to be made. This procedure's time complexity is therefore unacceptable.

6.2.1.1 Pattern coding

To improve upon the time complexity, Cung et al (Cung et al, 2000:196) has proposed a method whereby patterns that are stored in CList are coded, and tests are then defined that use these codes to avoid duplicate patterns from being stored. The pattern coding system can be defined as follows:

“Let A be a pattern obtained by combining vertically (respectively horizontally) a set of sub-patterns $A^v_1, A^v_2, \dots, A^v_r$ (respectively $A^h_1, A^h_2, \dots, A^h_s$), where A^v_i (respectively A^h_j) denotes an NV-pattern (respectively NH-pattern). Let l_A (respectively w_A) be the length (respectively width) of pattern A .

On one hand, it is assumed that each pattern A that is an NV pattern (respectively an NH pattern) has two identifiers denoted by I'_A and I^h_A , where I'_A (respectively I^h_A) represents the ordered set that indicates how A is obtained in CList from the NV (respectively NH) pattern, where CList is the list of stored best sub-problems transferred from the Open list to CList on account of the fact that it has the best upper bound for the evaluation function. It is assumed that the identifiers of each pattern of the Open list are undefined, where the Open list refers to the list of patterns that still have to be considered for further builds. Refer to figure 4.15, section 4.2. On the other hand, if A is not an NV-pattern (respectively NH-pattern), then I'_A (respectively I^h_A) is represented by the set of identifiers of the NV-pattern (respectively NH-pattern) which contribute to construct A , in other words the set of identifiers represented by $I'_{A1}, I'_{A2}, \dots, I'_{Ar}$ (respectively $I^h_{A1}, I^h_{A2}, \dots, I^h_{As}$). The number of distinct NV-patterns, which contribute to produce A (which have different identifiers), is denoted by D^v_A and the number of the same NV-patterns (with the same identifier), which contribute to construct A , is denoted by N^v_A . Let R be the rectangle obtained by combining horizontally two patterns A and B . Then, the code

of R is obtained as follows (because of an error in the article by Cung et al (Cung et al, 2000:196), these definitions have been altered slightly):

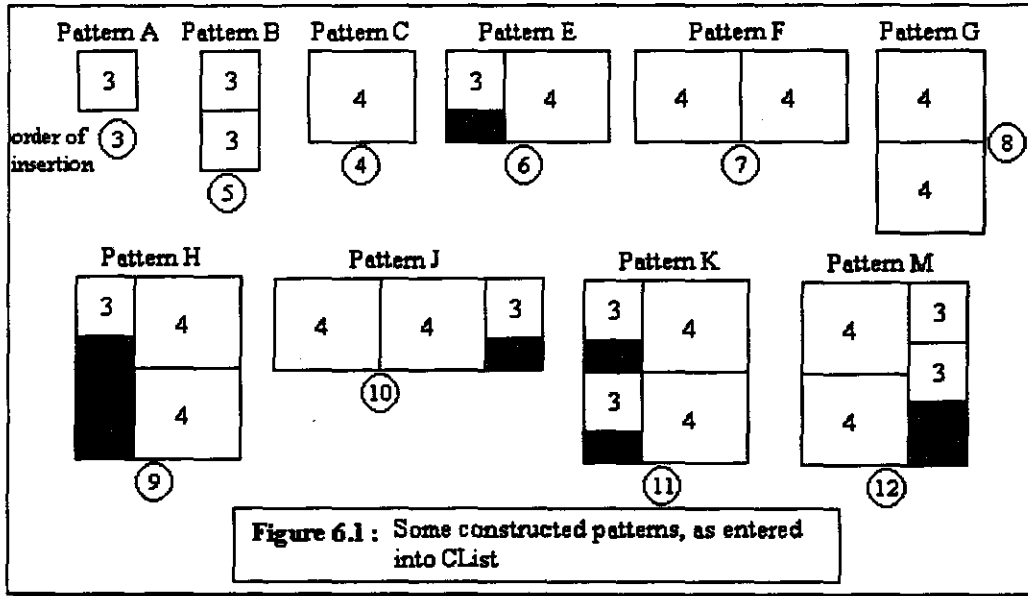
Horizontal: $I_R^h = \text{order of insertion}$; $D_R^h = 1$; $N_R^h = 1$.

Vertical: $I_R^v = I_A^v \cup I_B^v$; $D_R^v = |I_R^v|$ (the cardinality of D_R^v) and if $D_R^v = 1$ then $N_R^h = N_A^h + N_B^h$, otherwise, $N_R^h = \text{undefined}$.

Now, let S be the rectangle obtained by combining vertically two patterns A and B . Then, the code of S is obtained as follows:

Horizontal: $I_S^h = I_A^h \cup I_B^h$; $D_S^h = |I_S^h|$ (the cardinality of D_S^h) and if $D_S^h = 1$ then $N_S^h = N_A^h + N_B^h$, otherwise, $N_S^h = \text{undefined}$.

Vertical: $I_S^v = \text{order of insertion}$; $D_S^v = 1$; $N_S^v = 1$.



The code for each produced pattern is computed as follows:

- Piece A: Initial piece (NH-pattern and NV-pattern):
 - $D_A^v = D_A^h = 1$, $N_A^v = N_A^h = 1$, $I_A^v = I_A^h = \{3\}$;
- Piece B: (NV-pattern but not a NH-pattern), vertical combination of two instances of the pattern A:
 - $D_B^v = 1$, $N_B^v = 1$, $I_B^v = \{5\}$, $D_B^h = 1$, $N_B^h = 2$, $I_B^h = \{3\}$;
- Piece C: Initial piece (NH-pattern and NV-pattern):
 - $D_C^v = D_C^h = 1$, $N_C^v = N_C^h = 1$, $I_C^v = I_C^h = \{4\}$;

- Piece E: (NH-pattern but not a NV-pattern), horizontal combination of A and C:
 - $D^v_E = 2, N^v_E = \text{undef.}, I^v_E = \{3, 4\}, D^h_E = 1, N^h_E = 1, I^h_E = \{6\};$
- Piece F: (NH-pattern but not a NV-pattern), horizontal combination of two instances of C:
 - $D^v_F = 1, N^v_F = 2, I^v_F = \{4\}, D^h_F = 1, N^h_F = 1, I^h_F = \{7\};$
- Piece G: (NV-pattern but not a NH-pattern), vertical combination of two instances of C:
 - $D^v_G = 1, N^v_G = 1, I^v_G = \{8\}, D^h_G = 1, N^h_G = 2, I^h_G = \{4\};$
- Piece H: (NH-pattern but not a NV-pattern), horizontal combination of the patterns A and G:
 - $D^v_H = 2, N^v_H = \text{undef.}, I^v_H = \{3, 8\}, D^h_H = 1, N^h_H = 1, I^h_H = \{9\};$
- Piece J: (NH-pattern but not a NV pattern), horizontal combination of the patterns A and F:
 - $D^v_J = 2, N^v_J = \text{undef.}, I^v_J = \{3, 8\}, D^h_J = 1, N^h_J = 1, I^h_J = \{10\};$
- Piece K: (NV-pattern but not a NH-pattern), vertical combination of two instances of the pattern E:
 - $D^v_K = 1, N^v_K = 1, I^v_K = \{11\}, D^h_K = 1, N^h_K = 2, I^h_K = \{6\}.$
- Piece M: (NH-pattern but not a NV-pattern), horizontal combination of the patterns G and B:
 - $D^v_M = 2, N^v_M = \text{undef.}, I^v_M = \{5, 8\}, D^h_M = 1, N^h_M = 1, I^h_M = \{12\}.$

Now that a method for coding patterns has been established, it is possible to show how these codes may be implemented to reduce computational effort at every new node that is generated by a C2DGC algorithm. Three basic methods are described by Cung et al (Cung et al, 2000:197-201), namely pattern domination, symmetrical duplicate patterns in the same direction and symmetrical duplicate patterns in opposite directions.

6.2.1.2 Pattern domination

The fact that patterns are constructed by consecutive horizontal and/or vertical builds creates a situation where two or more patterns with the same

dimensions may be created. By doing a few translations on these patterns, it might become obvious that one of these patterns contains fewer pieces than another with the same dimensions. This is demonstrated in figure 6.1, where the pattern H is dominated by the patterns K and M .

The following proposition shows that some patterns can be neglected, because of pattern domination, if some conditions are satisfied.

Proposition 1: *Let A and B be two patterns. Suppose that R is a feasible pattern obtained using a horizontal build between A and B . Let b'_k , $k \in S = \{1, \dots, n\}$, be the number of times the k -th piece is used in R . Then R is a dominated pattern if and only if*

$$\exists k \in S: b_k - b'_k > 0 \text{ and } (l_R, w_R) \geq (l_k, w_k)$$

where $w_R = w_A - w_B$ and $l_R = l_B$ if $w_A > w_B$, $w_R = w_B - w_A$ and $l_R = l_A$ otherwise.

Cung et al (Cung et al, 2000:198) provides a proof for this proposition in their article, and states that proposition 1 only treats horizontal builds between patterns A and B , but that the same principle can be applied to vertical builds to eliminate dominated patterns.

6.2.1.3 Symmetric (duplicate) patterns on opposite directions

The second symmetric strategy builds on the first one. It states that patterns are symmetrical duplicates if they are the same size and are composed of exactly the same demand rectangles. Once again referring to figure 6.1, patterns K and M can be identified as duplicate patterns.

Proposition 2 describes how these duplicate patterns can be identified when they are generated and therefore more than one instance of a pattern, or its symmetrical duplicate, will not be stored in CList.

Let A denote a pattern obtained by combining horizontally the NH-patterns A_1, A_2, \dots, A_r and B is a pattern obtained by combining horizontally the NH-patterns B_1, B_2, \dots, B_s , with $d = l_A - l_B$ where $d \geq 0$.

Proposition 2: *A vertical build between two patterns A and B has a symmetric pattern if there exist two vectors $x = (x_i)$ where $i = 1, 2, \dots, r$ and $y = (y_j)$ where $j = 1, 2, \dots, s$ satisfying the following inequalities:*

$$\begin{cases} \sum_{i=1}^r l_{A_i} \cdot x_i - \sum_{j=1}^s l_{B_j} \cdot y_j \leq d \\ \sum_{i=1}^r x_i < r \text{ and } \sum_{j=1}^s y_j \leq s \\ x_i \in \{0,1\} \text{ and } y_j \in \{0,1\} \end{cases}$$

To avoid exponential growth of the lists that need to be maintained for this proposition, Cung et al (Cung et al, 2000:200) states that a simpler approach can be taken to eliminate only some symmetrical constructions. Therefore, let A and B be two patterns and suppose the following inequality exists, $0 \leq l_{A_i} - l_{B_j} \leq d$, where l_{A_i} and l_{B_j} represent respectively the subpatterns of A and B , stored in CList according to their lengths. In this case, a vertical build between the previous A and B patterns are forbidden.

6.2.1.4 Symmetric (duplicate) patterns on the same direction

The third symmetric strategy is implemented to cover symmetric patterns that might be generated but are not covered by propositions 1 and 2. It rejects the storage of patterns that were constructed by combining two patterns vertically (respectively horizontally), for instance A and B . It is assumed that both patterns A and B are composed of different sub patterns combined vertically (respectively horizontally). For example, the pattern J in figure 6.1 could be constructed by combining the patterns F and A horizontally, but also by horizontally combining the patterns E and C . The

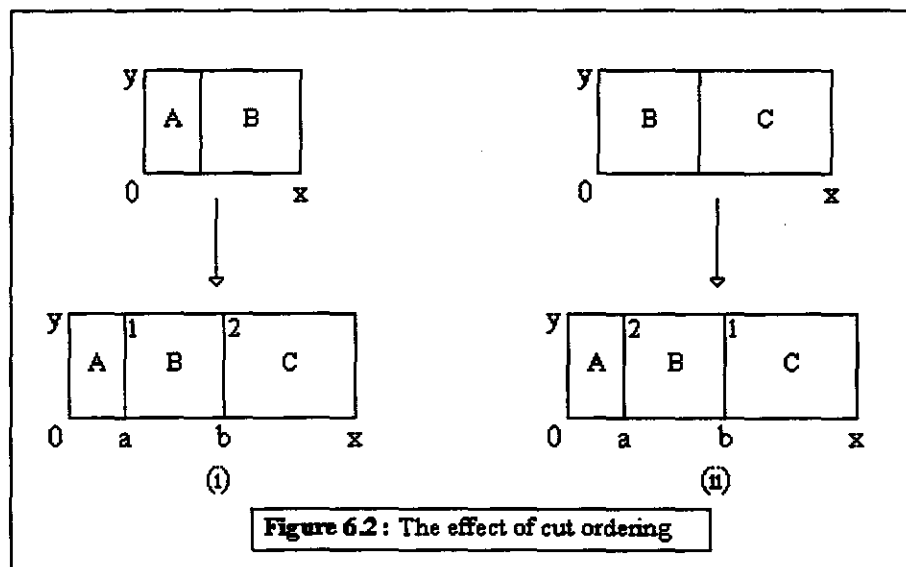
following proposition handles the horizontal instance, but the vertical instance could be treated in the same way.

Proposition 3: *The horizontal build between two patterns A (taken from the list Open) and a pattern B (taken from CList) is discarded if one of the three following cases are verified:*

- 1) $D_A^h = 1$ and $D_B^h = 1$ and $I_A^h = I_B^h$ and $(N_A^h - N_B^h < -1$ or $N_A^h - N_B^h > 1)$;
- 2) $D_A^h = 1$ and $D_B^h \neq 1$ and $I_A^h \subset I_B^h$; or
- 3) $D_A^v \neq 1$.

Cung et al (Cung et al, 2000:201) finally states that proposition 3, if implemented correctly, eliminates many unnecessary branches that would have been developed in the search tree. More complex build combinations could, however, be rejected if other competitive strategies were to be formulated using the coding standard.

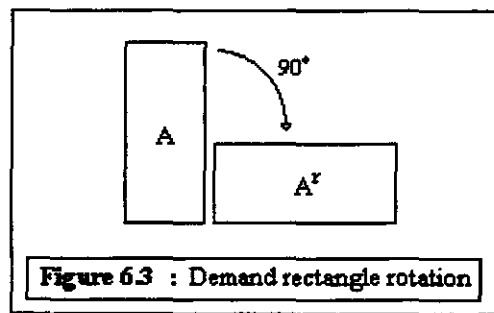
6.2.2 Cutting order



Another form of pattern duplication that may occur, results from the order in which pieces are cut from the stock sheet. Christofides and Hadjiconstantinou (Christofides & Hadjiconstantinou, 1995:20) explain the situation as follows: Let the chosen rectangle (i) be cut into smaller rectangles (a, y) and ($x-a, y$) by an x -cut at $r = a$. A second x -cut is performed on ($x-a, y$) at $r = b$ such that $a < b \leq \lfloor \frac{1}{2}(x-a) \rfloor$ at some successor node results in producing three rectangles A, B and C as shown in pattern (i) in figure 6.2. The same set of rectangles can be generated by pattern (ii) in figure 6.2, where the numbers next to the x -cuts indicate the order in which the cuts are made.

The consequence of implementing cut ordering is to eliminate from explicit consideration different sequences of cuts when these lead to the same final cutting pattern. This can be done, without missing any unique cutting pattern, by introducing an arbitrary cut ordering so that if a rectangle (x, y) is cut at, for instance, $r = a$, then all subsequent x -cuts on the two resultant rectangles must be greater than or equal to a (Christofides & Hadjiconstantinou, 1995:21).

6.2.3 Demand rectangle rotation



Demand rectangle rotation is an optional feature that can be included in the algorithms derived from C2DGC problem solving methods. An example of a rotated demand rectangle is demonstrated in figure 6.3. Christofides and Hadjiconstantinou (Christofides & Hadjiconstantinou, 1995:23) and

Viswanathan (Viswanathan, 1993:769) both state that their algorithms assume the following:

- *The orientation of the demand rectangles is considered to be fixed, in other words a demand rectangle of length l and width w is not the same as a demand rectangle of length w and width l .*

When rotated demand rectangles are considered, as by Wang (Wang, 1983:573-586) and Cung et al (Cung et al, 2000:188), it can be proven that the method is complete, therefore generating all possible cutting patterns. The demand constraint b_i for the i 'th demand rectangle still holds though, implying that only b_i instances of the demand rectangle as well as its rotated equivalent are allowed to be cut from the stock sheet. For this reason this situation is preferred to the one where rotated demand rectangles are not considered.

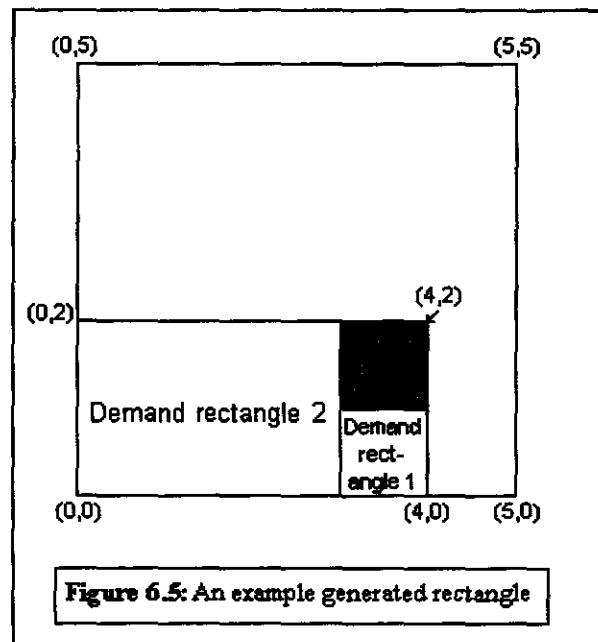
6.3 Improving the lower bounds of the WAM method

It can be observed that the Wang and modified Wang methods produce information in the process of searching for solutions using a certain value of beta. If, for instance, all generated builds (if any is found) are considered that has a certain dimension (l, w) using a specific beta value, the one that produces the least internal trim loss will in general be an improved lower bound over the one found by the unbounded two-dimensional knapsack function for the same dimension (l, w) . This improved lower bound for the dimension (l, w) is only a valid underestimate for the specific beta value that was used to calculate it.

Problem	Stock plate length (L) and width (W)	Demand rectangles' length (l), width (w) and upper bound (b)
EP1	(5,5)	(1,1,1); (3,2,5).

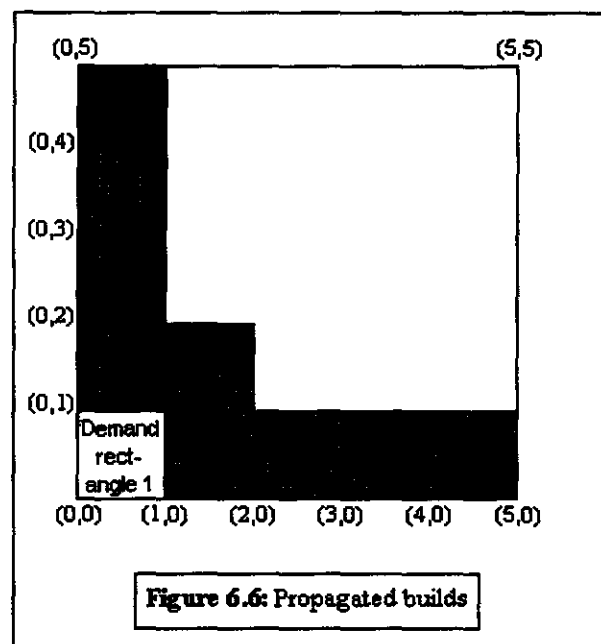
Figure 6.4: Example problem instance

On the other hand, the underestimates calculated by the unbounded two-dimensional knapsack function are valid for all values of beta. The concept of improved underestimates can be illustrated using a simple example. Consider the small example problem instance **EP1** (table 6.4), first introduced in chapter 5, section 5.1, table 5.1, page 56, where the stock sheet has dimensions (5,5) and two demand rectangle types exist that may be cut from the stock sheet. Firstly, because a demand rectangle with the dimensions (1,1) exists, the unbounded two-dimensional knapsack will use the fact that it has an unlimited number of these at its disposal, and fill the demand rectangle with it. This happens even though only one instance is allowed to be cut from the stock sheet, and therefore produces underestimates of 0 at each dimension within the larger stock sheet. It is already obvious that the unbounded knapsack fails to generate good lower bounds for this problem instance. When the iterative process of rectangle generation (based on Wang's method) is executed, different rectangular builds are produced with different dimensions. Figure 6.5 shows one of these possible generated rectangles (see also chapter 5, figure 5.6, build number 7 for a demonstration of how the build in figure 6.5 is constructed).



The build that is illustrated in figure 6.5 is a horizontal build constructed by combining demand rectangles 1 and 2 and will only be stored by the Wang method if a beta value of 0.04 or larger is used. It forms a new rectangle with dimensions (4,2) and an internal trim loss of 1. This build is also the best possible build for the dimension (4,2) (refer to chapter 5, figure 5.7, page 71) for a complete illustration of all stored builds using a beta value of 0.24. In figure 5.7 it can be seen that no build with less than 1 unit of internal trim loss exists for the dimension (4,2) when using a beta value of 0.24) and 1 is therefore an acceptable lower bound for that dimension if a beta value of 0.04 or larger is used to solve the problem. The underestimates produced by the unbounded two-dimensional knapsack function that is equal to 0 for the dimension (4,2), can therefore be replaced with the value of 1, which is a more accurate underestimate. This concept is used in the stock sheet propagation algorithm proposed by the author in section 6.3.1, pages 111-123.

A further observation is that the Wang and modified Wang methods may not generate builds with internal trim loss values allowed by beta for each dimension within the stock sheet (L, W). Figure 6.6 illustrates this concept, using problem instance **EP1**.

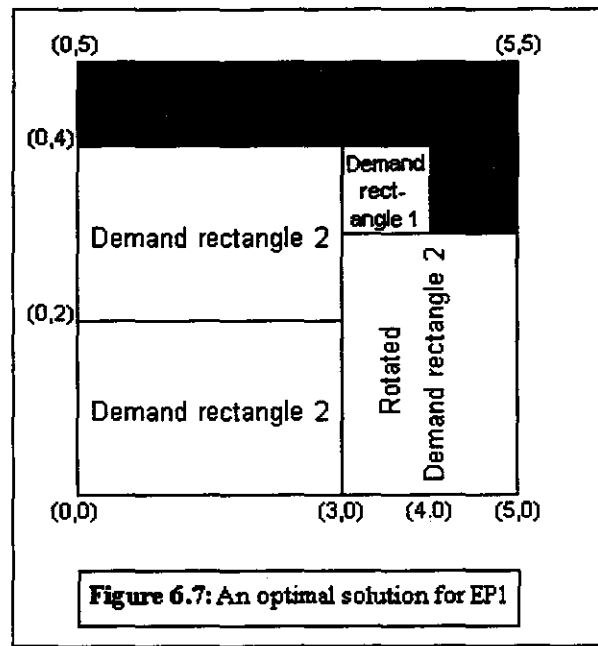


Problem instance **EP1** contains one demand rectangle of dimensions (1,1), and if it is placed at position (0,0) on the stock sheet, no more demand rectangles exist with a length or width dimension of 1 and no demand rectangle of dimensions (2,2) exists either. This implies that through inspection (for this small problem instance) it is impossible to generate builds with the following dimensions (as shown in the shaded sections in figure 6.6 (this can also be seen in chapter 5, figure 5.7, which shows all possible builds for **EP1** with a beta value of 0.24)):

(2,1), (3,1), (4,1), (5,1), (2,2), (1,2), (1,3), (1,4) and (1,5)

It may however be possible to artificially construct builds for the dimensions stated above. This can be achieved by adding trim loss to slightly smaller builds (the build (1,1) in this instance) that were constructed, which will fill the dimensions to the required size. This process of constructing missing builds is called propagation, and section 6.3.1, pages 111-123, introduces a method created by the author (PSSP method), which uses propagation and data obtained by the building process to improve the WAM lower bounds.

Before continuing with the discussion of the PSSP method, a small example instance (**EP1**, figure 6.4) will be used to further illustrate the concepts of underestimate updating and build propagations, as they form very important parts of the PSSP method. Firstly, the optimal guillotine-cut cutting pattern obtainable for **EP1** (using a beta value of 0.04 or higher) is a pattern with a total trim loss of 6 (internal trim loss of 1), and is illustrated in figure 6.7 (see also chapter 5, section 5.2.1.6.1, figure 5.7, page 71, for an illustration of this pattern in the complete list of stored nodes for **EP1** using a beta value of 0.24).



Even though a pattern with a total trim loss of 6 (internal trim loss of 1) for this problem instance could already be generated with a beta value of 0.04, it is necessary to increase the beta value to 0.24 ($(0.24) \cdot 5.5 \leq 6$) to prove that 6 is indeed the best obtainable trim loss value for the problem (according to Wang, chapter 5, section 5.2.1.5, theorem T1, page 66), and that figure 6.7 indeed illustrates an optimal solution.

As noted earlier, if the problem is solved with the modified Wang method (WAM) and underestimates are calculated using an unbounded two-dimensional knapsack function (chapter 5, section 5.2.3.1.1.1, pages 83-87, Gilmore and Gomory (1966)), all underestimate values should be 0 for this problem instance (EP1). This is due to the unbounded nature of the knapsack function and the fact that one demand rectangle of dimensions (1,1) exists. These values can be improved upon though, by solving a simplified problem instance using a smaller stock sheet but the same demand rectangles and demand constraints. The solving of this simplified problem instance is fast and can lead to better underestimates. This principle will now be demonstrated using an example problem instance (EP1). The calculated underestimates

using an unbounded two-dimensional knapsack function for the WAM method for problem (EP1) is given in table 6.1:

	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

Table 6.1: Original underestimates as generated by the unbounded knapsack function

These underestimates are not accurate enough, but it is possible to increase the values by following a few steps. Firstly, initialize values in an array with values calculated as follows:

$$\text{Cell}(x,y) = x.y$$

where x and y indicate index values for cells (stored elements) within the two-dimensional array and at the index (x,y) the value of x.y should be stored. Note that where cells are referenced, the x indicates the row and the y indicates the column, therefore the reference cell(x,y) is equal to cell(row,column). These values indicate the maximum trim loss that could exist for every dimension in the two-dimensional array, and is represented in table 6.2:

	1	2	3	4	5
1	1	2	3	4	5
2	2	4	6	8	10
3	3	6	9	12	15
4	4	8	12	16	20
5	5	10	15	20	25

Table 6.2: Maximum trim loss values for each dimension

It should be noted that a lot of these values exceed the maximum internal trim loss as allowed by the currently used beta value (which is 0.24 in this case). Therefore, replace the values in the array in table 6.2 that are larger than $\lfloor 5.5.(0.24) \rfloor + 1 = 7$ with the integer value 7. It should be noted that it is not

absolutely necessary to replace the values that is larger than 7 with 7, but it establishes uniformity in the table and shows with clarity which values exceed the maximum internal trim loss as allowed by the currently used beta value.

	1	2	3	4	5
1	1	2	3	4	5
2	2	4	6	7	7
3	3	6	7	7	7
4	4	7	7	7	7
5	5	7	7	7	7

Table 6.3: Maximum trim loss allowed by beta

The array has been initialized, and to take the process further, the problem instance will be solved with the original Wang method. Note that only half of the stock sheet length (3) will be used and the full stock sheet width, resulting in a simplified problem to be solved. The dimensions and upper bounds of the demand rectangles will remain the same as for the original problem. Refer to chapter 5, section 5.2.1.6.1, figure 5.7, page 71, for a representation of all possible feasible builds ($\beta=0.24$) that are stored by the Wang method for **EP1**.

	1	2	3
1	0	-1	-1
2	-1	-1	0
3	-1	0	2
4	-1	1	0
5	-1	-1	2

Table 6.4: Internal trim loss of builds generated by the original Wang method when solving the simplified problem

When the simplified problem is solved, builds with internal trim loss as indicated by table 6.4 are attained, where -1 indicates the algorithm did not find a build for that specific dimension. These values are now compared with those in table 6.3 (maximum internal trim loss allowed by beta). If a build was found with the Wang method and it's internal trim loss (represented in table

6.4) is less than that of the maximum allowed by beta (represented in table 6.3), the value in table 6.3 is replaced by the value in table 6.4. When this is complete, the following table is produced:

	1	2	3
1	0	2	3
2	2	4	0
3	3	0	2
4	4	1	0
5	5	7	2

Table 6.5: Altered values

Unfortunately, some values in table 6.5 are still overestimates of the maximum internal trim loss that could be generated by certain builds. For instance, cell (2,1) has a value of 2, but a build with an internal trim loss of 1 could be constructed by taking the build with dimensions (1,1) and adding 1 unit of trim loss at the bottom of the build (build propagation). This will lead to a build with an internal trim loss of 1. Therefore, we need to run a propagation algorithm to verify that no overestimated internal trim losses remain in table 6.5. After this algorithm (the propagation algorithm is discussed in more detail in section 6.3.1, pages 111-123) was run, the values in table 6.6 are generated.

	1	2	3
1	0	1	2
2	1	3	0
3	2	0	2
4	3	1	0
5	4	3	2

Table 6.6: Altered underestimates

Compare the values stored in table 6.6 with the propagated builds in figure 6.6. The values in table 6.6 represent the internal trim loss as illustrated in the propagated builds of figure 6.6.

All that remains to be done now is to compare the values in table 6.1 with the values in table 6.6. Keep in mind that the values in table 6.6 are

underestimates of internal trim loss that could occur in any build, therefore it is acceptable underestimates for the modified Wang method. If the value for a given dimension in table 6.6 is larger than its corresponding value in table 6.1, the value in table 6.1 must be replaced by the value in table 6.6. In this instance, all the values in table 6.1 are 0, and table 6.6 can be used as the first part of an updated lookup table for the modified Wang method. It should be noted that this is an exceptional case, and that usually the values in the original lookup table will not be all 0. In those instances, the two tables must be carefully compared. Table 6.7 represents the new table of underestimates to be used with the WAM method:

	1	2	3	4	5
1	0	1	2	0	0
2	1	3	0	0	0
3	2	0	2	0	0
4	3	1	0	0	0
5	4	3	2	0	0

Table 6.7: Final updated underestimates

It is obvious that table 6.7 represents much sharper underestimates than table 6.1, and this guarantees that the modified Wang method using the updated underestimates will perform better (or in certain cases the same) than the modified Wang method that used the original underestimates. This statement is explained further in the next section.

6.3.1 Partial stock sheet propagation (PSSP) method

The partial stock sheet propagation (PSSP) method is a new method devised by the author. It uses the modified Wang method and its characteristics as a basis and then refines the estimates used by the method. Characteristics of the modified Wang method (WAM) are:

- An unbounded two-dimensional knapsack function is solved once at the beginning of the problem solving process providing underestimates of internal trim loss for each smaller dimension within the larger stock sheet. This unbounded knapsack function is based on a dynamic programming recursion function (Gilmore & Gomory, section 5.2.3.1.1.1, pages 83-87);
- The solutions obtained from this function is stored in a two-dimensional array (named array A), used as a lookup table to determine estimated external trim loss; and
- These values are never updated.

The problems that can be identified with the method are:

- The unbounded nature of the two-dimensional knapsack function often causes the underestimates to be unrealistically low (section 6.3 highlights this fact with the use of example problem **EP1**); and
- Unnecessary builds are generated because the value of $h(n)$ is too optimistic as a result of the impractical underestimates.

As a result, the author proposes an extension to the modified Wang method. The previous section (section 6.3) introduced concepts used by the proposed extension to the modified Wang method, which is called the PSSP method. This section will aim at solving a larger example problem instance with this method and then at defining a formal PSSP algorithm based on the method. The PSSP method aims at updating the values in the two-dimensional array (array A) containing the underestimates of the external trim loss. This will be accomplished by solving a smaller instance of the given original C2DGC problem instance by using only part of the given stock sheet, but still using all the original demand rectangles. For instance, if the stock sheet has dimensions (70,40) any possible smaller part of the stock sheet may be used. Through empirical studies (refer to chapter 7, table 7.6, pages 161-163) it was found that half of the stock sheet length and the full width is normally a good choice, which translates to a stock sheet of dimensions

(35,40). The full stock sheet length and half of the width (ℓ , $(0.5).w$) could be an even better choice, but for the sake of uniformity the dimensions $((0.5).\ell, w)$ will once again be used as it was also used with problem instance **EP1**. Therefore, the original C2DGC problem will be solved with the original Wang (WA) method using the given demand rectangles and a chosen beta value, but a smaller $((0.5).\ell, w)$ stock sheet. While the smaller instance of the original problem is being solved, newly generated builds are constantly evaluated and if it contains the least internal trim loss for that specific dimension at that stage, it is stored. When the smaller instance of the original problem has been solved, a propagation algorithm must be run to ensure that no overestimates exist within the new underestimates. The least internal trim loss for any given dimension (x,y) can then be compared with the underestimate as generated by the unconstrained two-dimensional knapsack function at the same dimension. If the least internal trim loss value is greater than that of the knapsack function, the knapsack function's value can be replaced with the least total trim loss value for dimension (x,y). By repeating this comparison process for each dimension falling in the smaller stock sheet, it is possible to tighten the underestimates. The updated underestimates can be used with the WAM method to solve the original C2DGC problem instance with the chosen beta value, the original demand rectangles and a stock sheet of dimensions (70,40). Note that these updated underestimates are only valid for the beta value that was used to calculate them. When the PSSP algorithm is executed with a different beta value, the updated underestimates must be calculated again for the new beta value.

To demonstrate the idea behind the algorithm, a larger sample problem instance will be used than the one that was used to illustrate the basic concepts (problem **EP1**, figure 6.4, page 103). The sample problem consists of a stock sheet of dimensions 70x40 and ten demand rectangles¹. The optimal solution for this problem is found using the Wang or modified Wang algorithm (utilizing $h_3(n)^2$) with a beta (β) value of 0.02^3 . The original

¹ Refer to chapter 5, section 5.1, table 5.1, problem F3, page 56

² Refer to chapter 5, section 5.2.3.1.1, function 5.3, page 82

underestimates as given by the unbounded two-dimensional knapsack function for this problem instance are shown in table 6.8. Unfortunately, because of paper size constraints only part of the table is shown.

	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	
3	30	33	36	39	42	45	48	51	54	57	60	63	66	69	72	75	78	81	84	87	90	93	96	99	102	105	
4	40	44	48	52	56	60	64	68	72	76	80	84	0	4	8	12	16	20	24	28	32	36	40	44	48	52	
5	50	55	60	65	70	75	80	85	90	95	100	105	22	27	32	37	42	47	52	0	5	10	15	20	25	30	
6	60	66	72	78	84	90	96	102	108	114	120	126	44	50	56	62	68	74	80	29	35	41	47	53	59	65	
7	70	77	84	91	98	105	112	119	126	133	140	147	66	73	80	87	94	101	108	58	65	72	79	86	93	100	
8	80	88	96	104	112	120	128	136	144	152	160	168	0	8	16	24	32	40	48	56	64	72	80	88	96	104	
9	90	99	108	117	126	135	144	153	162	171	180	189	22	31	40	49	58	67	76	28	37	46	55	64	73	82	
10	100	110	120	130	140	150	160	170	180	190	200	210	44	54	64	74	84	94	104	0	10	20	30	40	50	60	
11	110	121	132	143	154	165	0	11	22	33	44	55	66	77	88	99	110	121	132	29	40	51	0	11	22	33	
12	120	132	144	156	168	180	16	28	40	52	64	76	0	12	24	36	48	60	72	58	70	82	32	44	56	68	
13	130	143	156	169	182	195	32	45	58	71	84	97	22	35	48	61	74	87	100	56	69	82	64	77	90	103	
14	140	154	168	182	196	210	48	62	76	90	104	118	44	58	72	86	100	114	128	0	14	28	42	56	70	84	
15	150	165	180	195	210	225	64	79	94	109	124	139	66	81	96	111	126	141	156	0	15	30	40	55	70	85	
16	160	0	16	32	48	64	80	96	112	0	16	32	0	16	32	48	64	80	96	29	0	16	15	0	16	32	
17	170	11	28	45	62	79	96	113	130	19	36	53	22	39	56	73	90	107	124	58	30	47	47	33	50	67	
18	180	22	40	58	76	94	112	130	148	38	56	74	44	62	80	98	116	134	152	28	46	64	79	66	84	102	
19	190	33	52	71	90	109	0	19	38	57	76	95	66	85	104	123	142	33	52	0	19	38	0	19	38	57	
20	200	44	64	84	104	124	16	36	56	76	96	116	0	20	40	60	80	60	80	0	20	40	32	44	64	84	
21	210	55	76	97	118	139	32	53	74	95	116	137	22	0	21	42	63	84	105	29	5	26	30	20	41	62	
22	44	66	0	22	44	66	0	22	44	66	0	22	44	23	0	22	44	23	0	22	35	23	0	22	35	23	
23	54	77	12	35	58	81	16	39	62	85	20	0	23	46	24	4	27	50	28	8	31	54	32	12	35	58	
24	64	88	24	48	72	96	32	56	80	104	40	21	0	24	48	29	8	32	56	0	16	40	15	8	24	48	
25	74	99	36	61	86	111	48	73	98	123	60	42	22	4	29	54	34	16	41	0	25	28	42	12	37	60	
26	84	110	48	74	100	126	64	90	116	142	80	63	44	27	8	34	60	43	24	29	10	36	40	40	26	52	
27	94	121	60	87	114	141	80	107	134	161	90	74	55	36	17	45	72	55	37	32	43	43	15	11	38	63	
28	104	132	72	100	128	156	96	124	152	180	100	84	65	46	28	56	41	24	52	80	0	28	56	47	24	52	80
29	0	29	58	56	0	0	29	58	28	0	0	29	22	8	0	0	29	22	0	0	0	29	22	0	0	0	
30	10	40	70	69	14	15	0	30	46	19	20	5	35	31	16	25	10	40	28	0	14	15	0	30	5	19	
31	20	51	82	82	28	30	16	47	64	38	40	26	23	54	40	28	36	33	56	29	15	46	32	55	39	25	
32	30	0	32	64	42	40	15	47	79	0	32	30	0	32	15	42	40	15	47	22	0	32	30	0	32	15	
33	40	11	44	77	56	55	0	33	66	19	44	20	22	12	8	12	40	11	24	0	30	55	0	33	20	19	
34	50	22	56	90	70	70	16	50	84	38	64	41	35	35	24	37	26	38	52	0	5	39	32	20	25	30	
35	60	33	68	103	84	85	32	67	102	57	84	62	23	58	48	40	52	53	80	0	19	25	15	19	30	49	
36	36	44	80	56	35	71	48	84	0	36	44	80	0	35	23	48	56	0	36	22	20	0	35	23	48	55	
37	46	55	92	69	49	86	64	101	18	55	64	101	22	16	16	24	56	27	48	8	5	31	25	20	33	45	
38	56	66	16	54	63	101	0	38	36	66	16	34	35	39	32	18	42	54	16	0	30	39	0	22	36	49	
39	39	77	28	67	50	0	16	55	0	39	36	28	23	50	0	16	55	0	39	0	0	16	32	0	16	36	
40	49	88	40	80	64	15	29	69	18	33	44	49	0	39	24	38	72	20	42	0	16	9	22	8	22	33	

Table 6.8: Original underestimates as stored in array A

³ Refer to chapter 7, section 7.2.2, table 7.2, page 147, for results when solving problem P3 with the Wang and modified Wang methods

The PSSP algorithm will then be implemented using the following steps:

- Firstly, allocate memory for a new two-dimensional array, named array B, (recall that the array used as a lookup table to determine estimated external trim loss is called array A and it contains the underestimates as calculated by the unconstrained two-dimensional knapsack function), with dimensions 35x40. Note that it is half of the stock sheet length by the stock sheet width of the example problem instance;
- Fill the new array (array B) with values calculated as follows:

$$Cell(x,y) = x.y$$

where x and y indicate index values for cells (stored elements) within the two-dimensional array B and at the index (x,y) the value of x.y should be stored;

- Replace the values in array B that is larger than $\lfloor 35.40.(0.02) \rfloor + 1 = 29$ with the integer value 29;

Table 6.9 represents the initialization values in array B.

- Solve the given problem instance with the original Wang method, using the original ten demand rectangle types and a beta (β) value of 0.02, but use only half of the original stock sheet dimensions (35x40);
- The values in array B are replaced with the internal trim loss of the best build found for any given dimension (x,y) with the original Wang method, if any build for dimension (x,y) was found;

Table 6.10 represents the values stored in array B after the original Wang method has been executed. The highlighted values in array B, table 6.10, represent builds that were generated and stored by the Wang method.

[illegible]

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	29	29	29	29	29	29	
2	2	4	6	8	10	12	14	16	18	20	22	24	26	28	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	
3	3	6	9	12	15	18	21	24	27	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	
4	4	8	12	16	20	24	28	29	29	29	29	29	29	29	29	29	29	29	29	29	29	0	29	29	29	29	29	29	29	29	29	29	29	29	29	
5	5	10	15	20	25	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	0	29	29	29	29	29	29	29	
6	6	12	18	24	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	
7	7	14	21	28	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	
8	8	16	24	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	0	29	29	29	29	29	29	29	29	29	29	29	29	29	
9	9	18	27	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	28	29	29	29	29	29	29	29	
10	10	20	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	
11	11	22	29	29	29	29	29	29	29	29	29	29	29	29	29	0	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	
12	12	24	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	0	29	29	29	29	29	29	29	29	29	29	29	29	29	
13	13	26	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	
14	14	28	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	0	29	29	29	29	29	29	29	
15	15	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	
16	16	29	29	29	29	29	29	29	29	29	0	29	29	29	29	29	29	29	0	29	29	0	29	29	29	29	29	29	29	29	0	29	29	29	29	29
17	17	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	
18	18	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	28	29	29	29	29	29	29	
19	19	29	29	29	29	29	29	29	29	29	29	29	29	29	29	0	29	29	29	29	29	29	29	29	29	29	29	29	0	29	29	29	29	29	29	
20	20	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	0	29	29	29	29	29	29	29	29	29	29	29	29	29	29
21	21	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	0	29	29	29	29	29	29	5	29	29	29	29	29	
22	22	29	29	0	29	29	29	0	29	29	29	0	29	29	29	0	29	29	29	0	29	29	29	0	29	29	23	0	29	29	23	29	29	29	23	
23	23	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	0	29	29	29	4	29	29	29	8	29	29	29	29	29	29	
24	24	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	0	29	29	29	8	29	29	29	29	29	29	29	29	29	
25	25	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	4	29	29	29	16	29	29	29	29	29	29	29	29	
26	26	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	8	29	29	29	29	29	29	29	29	29	29	29	29	
27	27	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	23	29	29	16	29	29	29	29	29	29	29	29	29	29	
28	28	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	0	29	29	29	29	29	29	0	29	29	29	29	29	29	
29	29	29	29	29	0	29	29	29	28	29	29	29	29	0	29	29	29	28	0	29	29	29	8	29	29	29	29	0	29	29	29	29	29	29	29	
30	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	0	29	29	29	29	5	29	29	29	29	29	29	29	29	14	29	29	29	29	19	
31	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	23	29	29	28	29	29	29	29	29	29	29	29	29		
32	29	29	29	29	29	29	29	29	29	29	0	29	29	29	15	29	29	0	29	29	29	29	15	29	29	29	29	28	0	29	29	29	29	15		
33	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	12	29	29	29	29	24	0	29	29	29	29	29		
34	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29		
35	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	23	29	29	29	29	29	29	19	29	29	29	29	29		
36	29	29	29	29	29	29	29	0	29	29	29	29	29	29	29	29	0	29	29	29	29	29	23	29	29	29	29	29	29	29	29	29	29	29	29	
37	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29		
38	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29		
39	29	29	29	29	29	29	29	9	29	29	29	29	29	15	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29		
40	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	18	29	29	29	29	29	24	29	29	27	29	29	29	29	29	29	29	29		

Table 6.10: Updated values of array B after running the original Wang method

All builds that were found with the original Wang method are highlighted in table 6.10. After this process is completed, some of the values in table 6.10 are still overestimates of the internal trim loss. The highlighted values represent exact builds with the least amount of internal trim loss that were found by the original Wang method on the specific dimensions, therefore these values are not overestimates. On the other hand, examine as an example cell(5,22), that contains the value of 29 (which is an overestimate) and is situated directly beneath (in the next row) a build found by the original Wang method with an internal trim loss of 0. To calculate the value that, say all cells(x,y) should contain (except for exact builds), the following propagation formula can be used:

$$\text{Min}\{\text{value}(\text{cell}(x,y)), \text{value}(\text{cell}(x-1,y))+y, \text{value}(\text{cell}(x,y-1))+x\} \quad (6.1)$$

This formula is intended to be applied in a recursive manner. Starting for example in cell(1,1), continuing in the row to cell(1,2), then to cell(1,3), constantly updating and adjusting the estimates in these cells according to formula 6.1. When the end of row 1 is reached, the algorithm moves to row 2 and so forth. Note that the minimum of three different cells are calculated when formula 6.1 is used, and that if, for instance, cell(1,1) is examined, only the first of the three possibilities refer to a valid cell (falling within the stock sheet dimensions). Whenever a cell is examined by formula 6.1, the algorithm must first determine which of the three cell references are valid, and only use the valid cell references for function 6.1.

The basic idea behind the recursive propagation algorithm, utilizing function 6.1, is that for any cell(i,j) that has to be allotted an underestimate that is valid for the current value of beta, the internal waste and resulting further waste for every build found by the Wang method (within the (i,j) dimension) is considered and the cell with minimal waste is chosen.

Therefore, the value of cell(5,22) will be calculated as:

$$\text{Min}\{\text{value}(\text{cell}(5,22)), \text{value}(\text{cell}(5,21))+5, \text{value}(\text{cell}(4,22))+22\} = 22$$

A further example could be cell(5,23):

$$\text{Min}\{\text{value}(\text{cell}(5,23)), \text{value}(\text{cell}(5,22))+5, \text{value}(\text{cell}(4,23))+23\} = 27$$

where in this case, the result of 27 is based on underestimates developed by the recursive formula 6.1 for cells (4,23) and (5,22).

When formula 6.1 is used to examine the values in table 6.10 and updates it if it is overestimates, the values in table 6.11 is produced. To complete the process, follow the following simple steps for every corresponding cell(x,y) in array A and array B:

- Compare the value of each cell(x,y) in array A (table 6.8) with the value of each cell(x,y) in array B (table 6.11); and
- If the value in array B is larger than the value in array A, replace the value in array A with the value in array B.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	28	29	29	29	29	29	29	
2	2	4	6	8	10	12	14	16	18	20	22	24	26	28	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	
3	3	6	9	12	15	18	21	24	27	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	
4	4	8	12	16	20	24	28	29	29	29	29	29	29	29	29	29	29	29	29	29	29	0	4	8	12	16	20	24	28	29	29	29	29	29	29	
5	5	10	15	20	25	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	22	27	29	29	29	29	29	0	5	10	15	20	25	29	
6	6	12	18	24	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	
7	7	14	21	28	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	
8	8	16	24	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	0	8	16	24	29	29	29	29	29	29	29	29	29	29	
9	9	18	27	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	22	29	29	29	29	29	29	28	29	29	29	29	29	29	
10	10	20	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	
11	11	22	29	29	29	29	29	29	29	29	29	29	29	29	29	0	11	22	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	
12	12	24	29	29	29	29	29	29	29	29	29	29	29	29	29	16	28	29	29	29	29	0	12	24	29	29	29	29	29	29	29	29	29	29	29	
13	13	26	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	22	29	29	29	29	29	29	29	29	29	29	29	29	29	
14	14	28	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	0	14	28	29	29	29	29		
15	15	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	
16	16	29	29	29	29	29	29	29	29	0	16	29	29	29	29	29	29	0	16	29	0	16	29	29	29	29	29	29	29	0	16	29	29	29	29	
17	17	29	29	29	29	29	29	29	29	11	28	29	29	29	29	29	29	19	29	29	22	29	29	29	29	29	29	29	29	29	29	29	29	29	29	
18	18	29	29	29	29	29	29	29	29	22	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	28	29	29	29	29	29	29	
19	19	29	29	29	29	29	29	29	29	29	29	29	29	29	29	0	19	29	29	29	29	29	29	29	29	29	29	29	0	19	29	29	29	29	29	
20	20	29	29	29	29	29	29	29	29	29	29	29	29	29	29	16	29	29	29	29	29	0	20	29	29	29	29	29	29	29	29	29	29	29	29	
21	21	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	22	0	21	29	29	29	29	29	5	26	29	29	29	29	29	
22	22	29	29	0	22	29	29	0	22	29	29	0	22	29	29	0	22	29	29	0	22	29	23	0	22	29	23	0	22	29	23	29	29	29	29	23
23	23	29	29	4	27	29	29	8	29	29	29	12	29	29	16	29	29	29	20	0	23	29	24	4	27	29	28	8	29	29	29	29	29	29	29	
24	24	29	29	8	29	29	29	16	29	29	29	24	29	29	29	29	29	29	29	21	0	24	29	29	8	29	29	29	29	29	29	29	29	29	29	
25	25	29	29	12	29	29	29	24	29	29	29	29	29	29	29	29	29	29	29	29	22	4	29	29	29	16	29	29	29	29	29	29	29	29	29	
26	26	29	29	16	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	27	8	29	29	29	29	29	29	29	29	29	29	29	29	
27	27	29	29	20	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	23	29	29	16	29	29	29	29	29	29	29	29	29	29	29	
28	28	29	29	24	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	0	28	29	29	29	29	29	0	28	29	29	29	29	29	29	
29	29	29	29	28	0	29	29	29	28	29	29	29	29	0	29	29	29	28	0	29	29	22	8	29	29	29	29	0	29	29	29	29	29	29	29	
30	29	29	29	29	5	29	29	29	29	29	29	29	29	14	29	0	29	29	19	29	5	29	29	29	29	29	29	28	29	14	29	29	29	29	19	
31	29	29	29	29	10	29	29	29	29	29	29	29	29	28	29	16	29	29	29	29	26	23	29	29	28	29	29	29	29	29	29	29	29	29	29	
32	29	29	29	29	15	29	29	29	29	0	29	29	29	29	15	29	29	0	29	29	29	29	15	29	29	29	29	28	0	29	29	29	29	15		
33	29	29	29	29	20	29	29	29	29	11	29	29	29	29	29	29	29	19	29	29	29	12	29	29	29	29	24	0	29	29	29	29	29	29		
34	29	29	29	29	25	29	29	29	29	22	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29		
35	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	23	29	29	29	29	29	29	29	19	29	29	29	29	29		
36	29	29	29	29	29	29	29	29	0	29	29	29	29	29	29	29	0	29	29	29	29	23	29	29	29	29	29	29	29	29	29	29	29	29	29	
37	29	29	29	29	29	29	29	29	9	29	29	29	29	29	29	29	18	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29		
38	29	29	29	29	29	29	29	29	18	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	
39	29	29	29	29	29	29	29	29	9	29	29	29	29	29	15	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	
40	29	29	29	29	29	29	29	29	18	29	29	29	29	29	29	29	18	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	

Table 6.11: Further updated values in array B after the propagation algorithm was run

Through this process we have increased the values of some of the underestimates in array A, but the updated values still remain underestimates (valid only for the beta value that was used to calculate them). The resulting updated values in array A is represented in table 6.12, and this array can now be used with the modified Wang method (WAM) as a lookup table for estimating external trim loss. Table 6.12 represents only half of the final array A (full size of array B), because only that part of array A is updated. A few important facts to note about the partial stock sheet propagation (PSSP) algorithm are:

- It uses the same or better underestimates than the modified Wang method, therefore the values of the heuristic evaluation function ($h(n)$) used by the PSSP algorithm will always be greater than or equal to the values of the heuristic evaluation function ($h(n)$) of the modified Wang method. Therefore the PSSP heuristic is more informed than the WAM heuristic and therefore the set of states examined by PSSP is a subset of those expanded by the modified Wang method (refer to chapter 4, section 4.2.9.3, page 51 for a detailed explanation of this statement);
- According to section 5.2.3.1.1 (page 82), when using the heuristic evaluation function 5.1 as proposed by Oliveira and Ferreira (1990) to calculate the values of $h(n)$, overestimates of the actual external trim loss may occur. Therefore, the PSSP algorithm should be implemented using the heuristic evaluation function $h_3(n)$ (refer to chapter 5, section 5.2.3.1.1, function 5.3, page 82), which is both admissible and monotone;
- The PSSP algorithm can be implemented using any partial stock sheet size. This implies that the partial stock sheet size (in our example 35x40) used in the first part of the algorithm may vary in dimensions. For instance, the dimensions 70x20 could have been used, or any other partial dimensions of the original stock sheet;
- The propagation of values will never extend further into array A than the dimensions of array B; and

- The PSSP algorithm will only be initialized for β values larger than 0.
- If the original WAM method cannot find a solution for a problem instance with a β value of 0, then the PSSP algorithm will be used.

	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
1	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
2	20	22	24	26	28	30	32	34	36	38	40	42	44	46	48	50	52	54	56	58	60	62	64	66	68	70
3	30	33	36	39	42	45	48	51	54	57	60	63	66	69	72	75	78	81	84	87	90	93	96	99	102	105
4	40	44	48	52	56	60	64	68	72	76	80	84	0	4	8	12	16	20	24	28	32	36	40	44	48	52
5	50	55	60	65	70	75	80	85	90	95	100	105	22	27	32	37	42	47	52	0	5	10	15	20	25	30
6	60	66	72	78	84	90	96	102	108	114	120	126	44	50	56	62	68	74	80	29	35	41	47	53	59	65
7	70	77	84	91	98	105	112	119	126	133	140	147	66	73	80	87	94	101	108	58	65	72	79	86	93	100
8	80	88	96	104	112	120	128	136	144	152	160	168	0	8	16	24	32	40	48	56	64	72	80	88	96	104
9	90	99	108	117	126	135	144	153	162	171	180	189	22	31	40	49	58	67	76	28	37	46	55	64	73	82
10	100	110	120	130	140	150	160	170	180	190	200	210	44	54	64	74	84	94	104	29	29	29	30	40	50	60
11	110	121	132	143	154	165	0	11	22	33	44	55	66	77	88	99	110	121	132	29	40	51	29	29	29	33
12	120	132	144	156	168	180	16	28	40	52	64	76	0	12	24	36	48	60	72	58	70	82	32	44	56	68
13	130	143	156	169	182	195	32	45	58	71	84	97	22	35	48	61	74	87	100	56	69	82	64	77	90	103
14	140	154	168	182	196	210	48	62	76	90	104	118	44	58	72	86	100	114	128	0	14	28	42	56	70	84
15	150	165	180	195	210	225	64	79	94	109	124	139	66	81	96	111	126	141	156	29	29	30	40	55	70	85
16	160	0	16	32	48	64	80	96	112	0	16	32	0	16	32	48	64	80	96	29	0	16	29	29	29	32
17	170	11	28	45	62	79	96	113	130	19	36	53	22	39	56	73	90	107	124	58	30	47	47	33	50	67
18	180	22	40	58	76	94	112	130	148	38	56	74	44	62	80	98	116	134	152	28	46	64	79	66	84	102
19	190	33	52	71	90	109	0	19	38	57	76	95	66	85	104	123	142	33	52	0	19	38	29	29	38	57
20	200	44	64	84	104	124	16	36	56	76	96	116	0	20	40	60	80	60	80	29	29	40	32	44	64	84
21	210	55	76	97	118	139	32	53	74	95	116	137	22	0	21	42	63	84	105	29	5	26	30	29	41	62
22	44	66	0	22	44	66	0	22	44	66	0	22	44	23	0	22	44	23	0	22	35	23	29	29	35	23
23	54	77	12	35	58	81	16	39	62	85	20	0	23	46	24	4	27	50	28	8	31	54	32	29	35	58
24	64	88	24	48	72	96	32	56	80	104	40	21	0	24	48	29	8	32	56	29	29	40	29	29	29	48
25	74	99	36	61	86	111	48	73	98	123	60	42	22	4	29	54	34	16	41	29	29	29	42	29	37	40
26	84	110	48	74	100	126	64	90	116	142	80	63	44	27	8	34	60	43	29	29	29	36	40	40	29	52
27	94	121	60	87	114	141	80	107	134	33	60	84	23	50	32	16	43	70	52	29	40	33	29	29	38	53
28	104	132	72	100	128	156	96	124	152	52	80	105	0	28	56	41	29	52	80	0	28	56	47	29	52	80
29	29	29	58	56	0	29	29	58	28	0	29	29	22	8	29	29	29	29	0	29	29	29	29	29	29	29
30	29	40	70	69	14	29	0	30	46	19	29	5	35	31	29	29	29	40	28	29	14	29	29	30	29	19
31	29	51	82	82	28	30	16	47	64	38	40	26	23	54	40	28	36	33	56	29	29	46	32	55	39	29
32	30	0	32	64	42	40	15	47	79	0	32	30	29	32	15	42	40	29	47	28	0	32	30	29	32	15
33	40	11	44	77	56	55	29	33	66	19	44	29	29	12	29	29	40	29	24	0	30	55	29	33	29	29
34	50	22	56	90	70	70	29	50	84	38	64	41	35	35	29	37	29	38	52	29	29	39	32	29	29	30
35	60	33	68	103	84	85	32	67	102	57	84	62	23	58	48	40	52	53	80	29	19	29	29	29	30	49
36	36	44	80	56	35	71	48	84	0	36	44	80	29	35	23	48	56	29	36	29	29	29	35	29	48	55
37	46	55	92	69	49	86	64	101	18	55	64	101	29	29	29	29	56	29	48	29	29	31	29	29	33	45
38	56	66	29	54	63	101	29	38	36	66	29	34	35	39	32	29	42	54	29	29	30	39	29	29	36	49
39	39	77	29	67	50	15	29	55	29	39	36	29	29	50	29	29	55	29	39	29	29	29	32	29	29	36
40	49	88	40	80	64	29	29	69	18	33	44	49	29	39	24	38	72	27	42	29	29	29	29	29	29	33

Table 6.12: Final values in array A after comparison with array B

To conclude the discussion of the PSSP method, an algorithm is given in figure 6.8.

To solve a C2DGC problem instance with dimensions length(L) and width(W) with the PSSP algorithm, follow these steps:

Initial steps:

- Choose a value for beta (greater than or equal to 0 and less than or equal to 1);
- Allocate memory for arrays named A and G with dimensions (L, W);
- Execute the unbounded two-dimensional knapsack function as proposed by Gilmore and Gomory (1966) and fill each dimension of array G with the value returned for that dimension by the knapsack function. The value represents an underestimation of internal trim loss for that dimension;
- Allocate memory for another two-dimensional array named B, with dimensions ($L * S, W$), where S indicates a certain percentage of the stock sheet (S is greater than 0 and less than or equal to 1);

Recursive steps:

- Set $A = G$;
- Fill array B with values indicating the maximum internal trim loss for a pattern in each dimension of array B. In other words $\text{cell}(x,y) = x * y$;
- Replace values in array B that is larger than that allowed by beta with $\lfloor \text{beta} * L * W \rfloor + 1$;
- Solve the C2DGC problem instance with the original Wang method, using the original demand pieces for the problem as well as the upper bounds placed on these pieces but only part of the stock sheet ($L * S, W$), effectively simplifying the problem; While the problem is being solved, compare each newly generated feasible ($l \in L, w \in W$) build's internal trim loss with the values stored in array B. If the internal trim loss of any newly generated build is lower than the value for that same dimension that is stored in array B, the value in array B should be replaced by the internal trim loss of the new build;
- When the Wang method has finished its iterations, execute a propagation algorithm, which must ensure that no values stored in array B are overestimations for the specific β ;
- For every corresponding $\text{cell}(x,y)$ in array A and array B:
 - * Compare the value of $\text{cell}(x,y)$ in array A with the value of $\text{cell}(x,y)$ in array B; and
 - * If the value in array B is larger than the value in array A, replace the value in array A with the value in array B.
- Solve the original problem instance with the WAM algorithm using the tighter lower bounds represented in array A;
- If an optimal solution was found for the problem instance, stop; otherwise restart at the recursive steps, but choose a larger value for beta (refer to paragraph 6.5 for a beta handling strategy).

Figure 6.8: The PSSP algorithm

6.4 Upper bounds and the waste gap

When delving further into the significance of the beta value and its inherent properties, it becomes obvious that although lower bounds are necessary to determine good estimations of future trim loss as well as initial beta values, the range of possible beta values that might be used remains huge. For

instance, if an initial beta value of 0.02 is calculated using lower bounds, the waste gap still remains at $0.02 \leq \beta \leq 1$. In the literature concerning the modified Wang method (WAM), a vast amount of work has been done regarding lower bounds and its usefulness. The author has also contributed in the lower bound field with the PSSP algorithm (section 6.3.1, pages 111-123), which aims at computing sharper lower bounds by updating the original lower bounds as given by Gillmore and Gomory (chapter 5, section 5.2.3.1.1.1, pages 83-87).

Two strategies will be utilized to determine upper bounds for C2DGC problems. Firstly, a method utilizing information from the Wang method's building process is discussed from which an upper bound could be calculated. Secondly, a beam search algorithm is used that calculates *good* solutions for C2DGC problems, which could in turn be used as upper bounds.

6.4.1 Upper bound propagation and the waste gap

As an extension to the PSSP algorithm research, the author proposes a method of using information obtained from the building process in the original Wang method by storing all builds (feasible and infeasible (internal trim loss exceeds the pruning criterion ($L.W.B$)), falling within the stock sheet area) and using these as upper bounds. If a specific build was not found that is equal to the stock sheet dimensions, a propagation algorithm is used to find the best cut with the same dimensions as the stock sheet. When this value has been acquired, the waste gap might be reduced.

Problem P8 (chapter 5, section 5.1, table 5.1, page 56) will be used to demonstrate the effectiveness of this algorithm addition. Firstly, the algorithm is executed with an initial beta value of 0.00. All builds (feasible and infeasible, falling within the stock sheet area) are stored. When two or more builds are found with the same dimensions, the build with the least internal trim loss will be used. Table 6.13 shows part of the upper bound array, as it is stored in memory after the algorithm has been executed with a beta value

of 0.00. Due to paper size constraints, the whole array can unfortunately not be displayed.

	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55
15	*	44	*	0	*	104	*	44	*	0	155	104	66	*	22	*	52
16	*	66	*	0	*	130	*	66	33	0	65	130	99	*	33	0	65
17	44	*	78	*	*	*	*	18	*	*	*	88	*	24	*	156	*
18	0	48	36	55	28	28	91	248	0	0	36	36	28	72	110	0	0
19	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
20	*	*	*	77	*	117	*	30	32	*	72	*	*	*	154	46	96
21	80	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
22	*	*	140	*	*	62	128	*	*	*	216	*	152	156	*	*	252
23	80	220	0	168	*	312	220	0	168	*	93	160	168	110	*	13	160
24	23	192	121	121	26	26	0	0	0	18	18	18	288	242	23	23	23
25	*	*	*	110	*	*	*	*	*	170	*	*	*	*	*	*	*
26	225	*	60	33	64	*	*	*	360	99	304	312	*	123	420	230	*
27	48	11	*	33	*	45	48	*	72	99	*	*	308	92	264	165	416
28	80	115	288	22	442	78	22	29	64	0	60	64	0	90	330	115	240
29	0	234	*	75	80	17	108	*	88	99	352	138	312	*	*	0	80
30	90	96	34	44	*	104	*	90	96	0	126	494	*	364	87	161	0
31	55	0	31	0	105	0	51	112	51	0	31	112	51	0	55	0	31
32	120	224	68	0	*	20	81	28	145	30	62	256	81	10	66	30	62
33	225	*	56	54	60	93	288	150	68	0	612	*	76	585	80	90	42
34	48	*	13	54	150	13	102	36	80	0	36	462	60	48	462	96	72
35	168	*	*	220	*	*	66	77	88	340	*	*	*	*	*	442	459
36	126	96	104	220	189	56	261	150	62	180	759	170	*	*	*	190	273
37	*	*	*	638	*	*	156	169	182	770	*	*	*	*	*	*	*
38	160	550	70	23	*	65	243	60	32	46	108	448	140	238	550	345	144
39	80	348	80	23	480	225	45	60	75	24	31	480	132	272	168	184	352
40	348	*	221	11	*	91	48	64	64	72	18	*	*	*	594	165	192
41	80	221	162	110	200	130	289	126	23	46	780	324	196	210	300	31	176
42	23	11	110	33	33	55	87	62	0	24	54	54	220	143	638	23	23
43	480	*	200	33	*	*	*	300	*	0	*	*	45	*	*	675	286
44	225	91	130	55	*	143	280	30	32	0	90	*	48	247	682	69	312
45	45	48	289	87	*	280	*	704	92	99	*	884	*	308	174	0	31
46	60	64	126	62	300	30	704	644	0	0	910	459	54	60	203	62	16
47	75	64	23	0	*	32	92	0	23	0	93	736	486	160	88	0	29
48	24	72	46	24	0	0	99	0	0	0	0	340	46	24	46	46	0
49	31	18	780	54	*	90	*	910	93	0	*	*	*	234	*	0	31
50	480	*	324	54	*	*	884	459	736	340	*	*	836	*	*	384	992

Table 6.13: Initial upper bounds before propagation for P8 and beta = 0.00

Table 6.13 shows all the builds (feasible and infeasible) that were found by the original Wang method. All cells with stars (*) indicate dimensions where builds were not found.

	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55
15	82	44	59	0	15	30	45	44	59	0	15	30	45	60	22	37	52
16	48	64	80	0	16	32	48	64	33	0	16	32	48	64	33	0	16
17	44	61	78	42	59	76	93	18	35	48	65	82	99	24	41	54	71
18	0	18	36	54	28	28	46	64	0	0	18	36	28	46	64	0	0
19	39	58	77	96	71	72	91	110	47	48	67	86	79	98	117	54	55
20	78	98	118	77	97	116	136	30	32	52	72	92	112	132	152	46	66
21	80	101	122	119	140	160	181	76	79	100	121	142	163	184	205	100	121
22	119	141	140	161	183	62	84	106	126	148	170	192	152	156	178	154	176
23	80	103	0	23	46	69	92	0	23	46	69	92	115	110	133	13	36
24	23	47	41	65	26	26	0	0	0	18	18	18	42	66	23	23	23
25	62	87	82	107	69	70	45	46	47	66	67	68	93	118	76	77	78
26	101	127	60	33	59	85	90	92	94	99	116	118	144	123	129	131	133
27	48	11	38	33	60	45	48	75	72	99	126	153	180	92	119	146	173
28	80	51	79	22	50	78	22	29	57	0	28	56	0	28	56	84	112
29	0	29	58	64	80	17	46	75	88	48	77	106	51	80	109	0	29
30	39	69	34	44	74	61	91	90	96	0	30	60	90	120	87	54	0
31	31	0	31	0	31	0	31	62	51	0	31	62	51	0	31	0	31
32	52	40	68	0	32	20	52	28	60	30	62	94	81	10	42	30	62
33	90	80	56	42	60	64	97	74	68	0	33	66	76	62	80	84	42
34	48	82	13	47	81	13	47	36	70	0	34	68	60	48	82	96	72
35	87	122	54	89	124	57	66	77	88	48	83	118	111	100	135	150	127
36	126	96	95	131	167	56	92	123	62	96	132	168	162	152	188	190	182
37	165	136	136	173	210	100	137	169	109	144	181	218	213	204	241	244	237
38	160	176	70	23	61	65	103	60	32	46	84	122	140	178	216	254	144
39	80	119	80	23	62	101	45	60	75	24	31	70	109	148	168	184	199
40	119	159	121	11	51	91	48	64	64	72	18	58	98	138	178	165	192
41	80	121	162	53	94	130	93	110	23	46	67	108	149	190	231	31	72
42	23	11	53	33	33	55	87	62	0	24	54	54	96	138	180	23	23
43	62	51	94	33	76	99	132	108	47	0	43	86	45	88	131	77	78
44	101	91	130	55	99	143	177	30	32	0	44	88	48	92	136	69	113
45	45	48	93	87	132	177	222	76	79	48	93	138	99	144	174	0	31
46	60	64	110	62	108	30	76	122	0	0	46	92	54	60	106	54	16
47	75	64	23	0	47	32	79	0	23	0	47	94	105	112	88	0	29
48	24	72	46	24	0	0	48	0	0	0	0	48	46	24	46	46	0
49	31	18	67	54	43	44	93	46	47	0	49	98	97	76	99	0	31
50	70	58	108	54	86	88	138	92	94	48	98	148	148	128	152	54	86

Table 6.14: Propagated upper bounds for P8 and beta = 0.00

Even though a build was found in the last cell (50,55), it is not necessarily the optimal upper bound, because in this instance, the cell (49,55) contains a build with internal trim loss of 31. This implies that cell (50,55) can contain a build with an internal trim loss of $31 + 55 = 86$.

For the above mentioned reason we run a propagation algorithm to fill the empty cells with values and let it determine the best upper bound for cell (50,55). Table 6.14 displays the array of updated, propagated upper bounds, and it should be kept in mind that with a beta value of 0.00 the optimal solution for **P8** was not found. After the original Wang algorithm was executed with a beta value of 0.00, a feasible build with a total trim loss of 104 was found (the build has an internal trim loss of 0, but is complete and therefore when it is placed on the stock sheet has an external trim loss of 104, and the total trim loss is internal trim loss plus external trim loss). This is not an optimal solution, because 104 is not less than or equal to $(0.00).55.50$, therefore the search has to continue with larger beta values. To minimize the waste gap, the value stored in cell (50,55), which is 86, is important, and the gap can now be written as:

$$0.00 < \beta \leq (86 / (55.50))$$

$$0.00 < \beta \leq 0.0313$$

Continuing to solve the problem will imply increasing the value of beta. From the above-stated waste gap, it is already clear that a beta value of more than 0.0313 is unnecessary. For the sake of the example, the beta value will be increased with an arbitrary value of 0.01. Therefore, after the original Wang method has been executed with a beta value of 0.01, table 6.15 is produced as the propagated upper bounds. The upper bound in cell (50,55) indicates a value of 34, implying a sharp drop in size of the waste gap. The best cutting pattern with a beta value of 0.01 for problem **P8** is again one with a total trim loss of 104, indicating that it is not an optimal solution. The waste gap can now be written as:

$$0.01 < \beta \leq (34 / (55.50))$$

$$0.01 < \beta \leq 0.0124$$

From this it can be deduced that any beta value of more than 0.0124 is redundant. Therefore, the value of beta will not be increased to 0.02, but rather kept down to 0.0124, which guarantees that an optimal solution will be found with this value of beta.

	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55
17	44	61	78	42	59	76	93	18	35	48	65	82	99	24	41	54	71
18	0	18	36	54	28	28	46	64	0	0	18	36	28	46	64	0	0
19	39	58	77	96	71	72	91	110	47	48	67	86	79	98	117	54	55
20	78	98	118	77	97	116	136	30	32	52	72	92	112	132	152	46	66
21	80	101	122	119	140	160	181	76	79	100	121	142	163	184	205	100	121
22	119	141	140	161	183	62	84	106	126	148	170	192	152	156	178	154	176
23	80	103	0	23	46	69	92	0	23	46	69	92	20	43	66	13	36
24	23	47	41	65	26	26	0	0	0	18	18	18	42	49	23	23	23
25	62	87	82	107	69	70	45	46	47	66	67	68	93	101	76	77	78
26	101	127	60	33	59	84	86	92	94	99	99	101	127	123	129	131	133
27	48	11	38	33	60	45	48	75	72	99	126	151	165	92	119	146	56
28	80	51	26	22	32	60	22	29	57	0	10	38	0	28	55	83	26
29	0	29	43	22	51	17	46	75	64	48	56	85	26	55	55	0	29
30	37	67	34	44	74	61	91	37	67	0	30	60	27	57	87	54	0
31	31	0	13	0	13	0	31	62	13	0	31	62	13	0	13	0	31
32	52	40	36	0	32	20	52	28	36	30	62	94	46	10	42	30	62
33	82	80	33	36	60	58	91	74	38	0	31	64	53	56	80	58	36
34	48	82	13	40	74	13	23	36	61	0	34	46	43	18	52	43	18
35	23	58	54	82	50	57	66	53	35	48	63	96	94	70	105	58	73
36	13	49	85	68	93	56	57	93	55	74	63	99	69	105	129	50	31
37	52	89	126	110	78	100	102	139	102	122	34	71	108	145	138	104	86
38	90	55	59	23	61	65	85	60	32	46	83	20	57	93	131	50	23
39	80	71	80	23	41	56	45	60	75	18	31	48	68	79	23	13	52
40	71	111	121	11	42	58	29	18	58	18	18	58	58	67	76	23	39
41	80	121	113	53	85	102	74	18	23	46	67	38	26	49	64	31	72
42	23	11	53	33	33	22	22	62	0	18	36	47	45	45	87	23	11
43	41	42	85	33	76	66	67	28	29	0	25	68	45	88	77	64	57
44	56	58	102	22	66	47	49	30	32	0	26	45	48	86	43	37	26
45	45	29	74	22	67	49	94	68	79	26	68	95	72	22	46	0	31
46	60	18	18	62	28	30	68	93	0	0	46	92	50	50	35	37	16
47	75	58	23	0	29	32	79	0	23	0	13	60	61	70	64	0	29
48	18	18	46	18	0	0	26	0	0	0	0	48	46	18	46	46	0
49	31	18	67	36	25	26	68	46	13	0	49	98	79	28	74	0	31
50	48	58	38	47	68	45	95	92	60	48	98	148	130	56	50	52	34

Table 6.15: Propagated upper bounds for P8 and beta = 0.01

The original Wang method is now executed for a third time with a beta value of 0.0124. Upon completion of the execution, an optimal cutting pattern is found with an internal trim loss of 34 ($34 < (0.0124) \cdot 55.50$). This will then terminate the search process.

6.4.2 Beam search, upper bounds and the waste gap

A second method that could be used to determine an upper bound for any given C2DGC problem is a greedy search method. In its simplest form a greedy search would usually follow the single best path down a search tree (hence the name greedy) and terminate when the deepest node in the path cannot be expanded any further. This will then terminate the search. Even though greedy searches execute very fast, the quality of the solution obtained by them is usually not very good. In order to obtain a better result but still inhibit the exponential growth of C2DGC problem search spaces, a beam search algorithm is used that limits the number of nodes expanded per level (a specific depth in the search space) to the value w .

For this reason the author implemented a beam search algorithm that can be combined with the Wang and modified Wang methods, where the value of w can be specified. The algorithm computes a possible solution for any given C2DGC problem instance, and depending on the size of w (beam width) the quality of the solution may vary. As an example, problem **P8** (chapter 5, section 5.1, table 5.1, page 56) will be used to demonstrate the use of this algorithm in determining upper bounds and the waste gap.

The first step is to choose a value for the beam width w . For this example, the problem will be solved five times with five separate beam widths (10, 15, 20, 25 and 30) starting each time with a beta value of 0.00, which is systematically increased. The results of the solution process are summarized in table 6.16.

Beta (β)	Beam width (m)						
		Wang (no beam)	10	15	20	25	30
	0.00	104	212	212	212	212	110
	0.01	104	212	128	128	128	110
	0.02	34	194	86	86	86	86

Table 6.16: Least total trim loss of solution patterns using beam search

The first observation that can be made when examining table 6.16 is that Wang's method will always find either the same or better quality solution than beam search. A beam width of 30 seems to generate good quality solutions when compared with the Wang method, but it should be taken into account that execution times of the beam search algorithm increases as the beam width increases. After the first iteration of the beam search algorithm with m equal to 30, the waste gap can be defined as:

$$0.00 < \beta \leq (110 / (55.50))$$

$$0.00 < \beta \leq 0.04$$

showing that any value of beta greater than 0.04 will generate unnecessary patterns. After the second iteration of the beam search algorithm with m equal to 30, the waste gap can be defined as:

$$0.01 < \beta \leq (110 / (55.50))$$

$$0.01 < \beta \leq 0.04$$

which implies that the value of beta lies in-between 0.01 and 0.04. After the third iteration of the beam search algorithm with m equal to 30, the waste gap can be defined as:

$$0.02 < \beta \leq (86 / (55.50))$$

$$0.02 < \beta \leq 0.0313$$

Upon completion of the iterative search process, it seems as if the calculation of the waste gap with the upper bound propagation method, as described in section 6.4.1, performs better. Still, the beam search method to calculate upper bounds and the waste gap remains a simple and effective implementation.

6.5 Strategies for handling the value of beta (β)

An important aspect of all algorithms based on Wang's method is choosing an initial starting value for β . Many researchers have proposed different strategies to compute an initial value, and some of these methods will shortly be described.

The most popular method is choosing an initial starting value of 0.00, and if the optimal solution is not found, the value of β is gradually increased by a constant value (usually 0.01) until the optimal solution is found (Daza *et al*, 1995:642). Therefore, this method uses 0 as a lower bound on the value of β . A problem with this method is that some problem instances may require a large value for β before the optimal solution is found, and through this lower bound process, a great amount of unnecessary work may be done to iterate through small β values.

Another method to calculate a lower bound for the value of β was proposed by Zissimopoulos (Zissimopoulos, 1984) and Hifi (Hifi, 1994), and Hifi (Hifi, 1997:730-732) introduced an improved version of the method a few years later. This method involves the implementation of a one-dimensional bounded knapsack for creating a set of horizontal and vertical strips and then combines them for obtaining a feasible cutting pattern for a given problem instance. If the realized solution does not satisfy the demand constraints then solving approximately two sets of packing problems creates a feasible cutting pattern. From this pattern the initial value of β is then derived.

Vasko introduced the idea of calculating an upper bound on the value of β , and then to gradually decrease (diminish) this value as the search progressed (refer to section 5.2.2.3, pages 76-77, for more information on dynamically diminishing the β value). The upper bound on β is calculated using an algorithm called SPAM, which quickly generates solutions to the constrained two-stage cutting stock problem. These solutions are then used to obtain an initial upper bound for the minimum trim waste of the general (non-staged) constrained guillotine cutting stock (C2DGC) problem (Vasko, 1988:109).

6.5.1 Lower bound using the WAM lookup table

The author proposes a new method to find an initial lower bound on the value of β . A sample problem with stock sheet dimensions of 70x40 and twenty demand rectangles (refer to chapter 5, section 5.1, table 5.1, page 56, problem P2) will be used to demonstrate the concept. The optimal solution for this problem is found with a beta (β) value of 0.02. The method derives a lower bound from the lookup table (array A) that was constructed using the unbounded two-dimensional knapsack function of Gilmore and Gomory (Gilmore & Gomory, 1966). Part of the resulting lookup table for the given problem instance is displayed in table 6.17.

The highlighted value in table 6.17 is the last entry in array A at cell(40,70). This value is an underestimate of the minimum total trim loss of the best feasible cutting pattern for the given problem instance. Therefore, it is a viable value that can be used to calculate β :

$$\beta = \text{value}(\text{cell}(\mathcal{W}, \mathcal{L})) / (\mathcal{L} \cdot \mathcal{W}) \quad (6.2)$$

where \mathcal{L} is equal to the stock sheet length and \mathcal{W} is equal to the stock sheet width. For this specific problem instance, the initial value of β is:

$$\beta = \text{value}(\text{cell}(40,70)) / 70.40 = 25 / (70.40) = 0.008$$

The value of β is incremented with the value of 0.01. Therefore, the calculated value will be rounded up to the nearest 0.01 increment and the initial β value for this specific problem instance is then $\beta = 0.01$.

	61	62	63	64	65	66	67	68	69	70
26	76	64	78	57	71	57	46	32	39	25
27	36	63	72	83	56	45	32	0	27	32
28	0	28	17	45	17	45	65	38	0	28
29	32	23	0	29	37	28	51	59	50	53
30	29	47	0	30	0	18	21	21	45	33
31	18	18	0	0	12	24	15	15	21	18
32	34	54	37	42	21	0	21	0	0	18
33	0	12	21	22	0	12	25	0	0	0
34	19	9	0	12	23	0	19	9	0	18
35	18	53	0	35	22	9	0	18	27	27
36	18	0	0	0	0	0	0	0	0	0
37	19	53	15	12	36	44	28	35	9	14
38	0	9	21	0	9	0	0	9	0	14
39	36	26	12	17	31	20	0	21	11	14
40	19	22	12	18	41	22	19	15	19	25

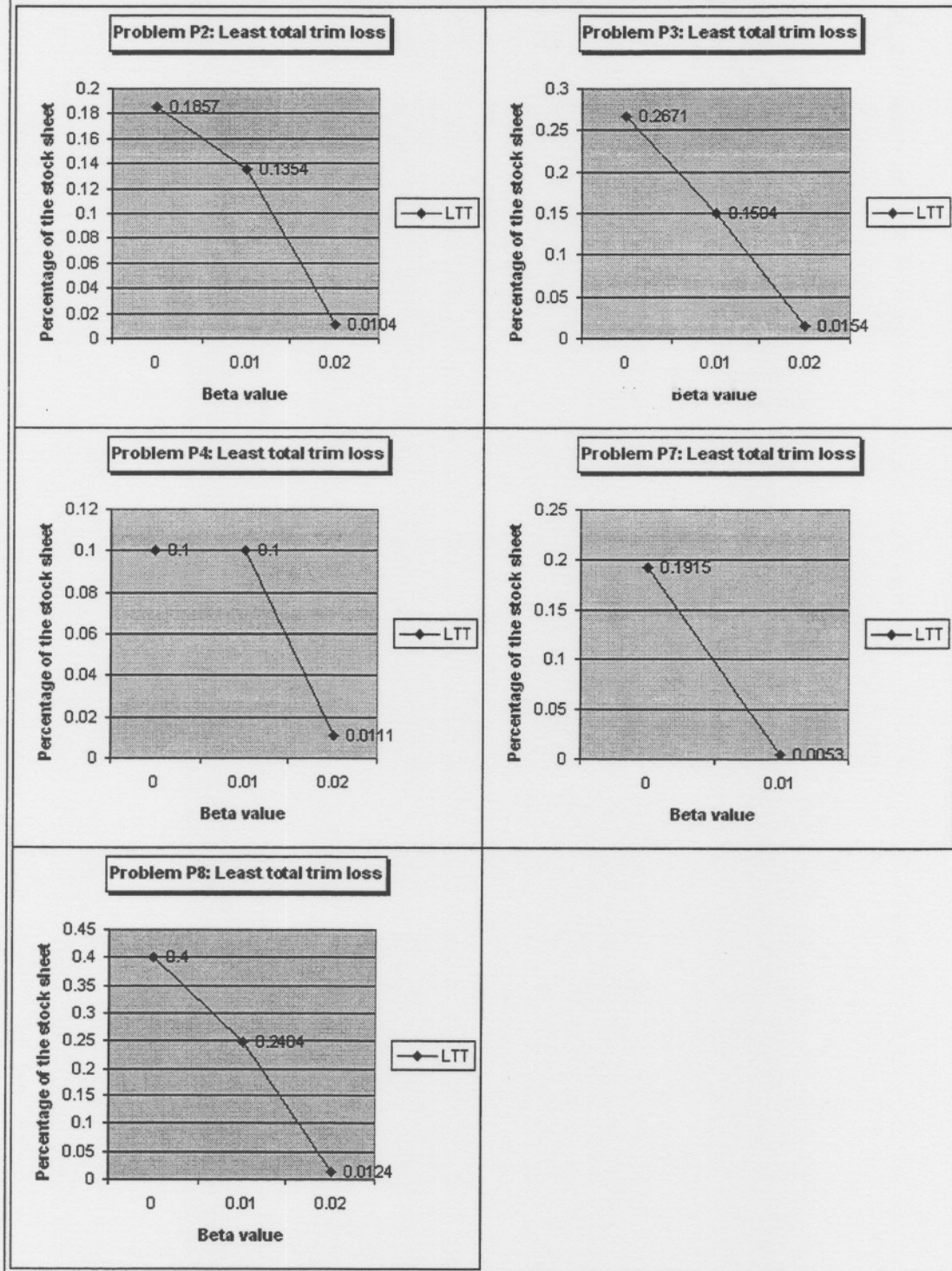
Table 6.17: Initial lower bound on beta (β)

This method therefore generates an initial β value that might be less than the optimal β value. Therefore, if the initial value is too small, it will gradually be increased by 0.01 until the optimal solution is found.

6.5.2 Increasing the value of beta (β)

Whenever an initial value for beta is chosen and the optimal solution for the problem is not found using that beta value, the value of beta is gradually increased by a constant value (usually 0.01) until the optimal cutting pattern is found. This process of increasing the value of beta by the arbitrary value of 0.01 is arguably not the best method to determine the next beta value, because in some cases a very large beta value is required to solve the problem optimally.

Chart 6.1: Change in total trim loss as the value of beta increases



The author proposes that the solving process for any given C2DGC problem instance be started with an initial beta value as calculated in section 6.5.1.

When all patterns have been constructed using the initial beta value and the optimal solution were not generated, the value of beta should be incremented by a specific fraction. To calculate this increment fraction, chart 6.1 shows specific problem instances (**P2, P3, P4, P7** and **P8** from chapter 5, section 5.1, table 5.1, page 56) that were solved with the original Wang method using a starting beta value of 0 and an increment value for beta of 0.01. The idea is that when the problem is solved with an initial beta value (in these cases 0.00, but it can be any other starting value), the pattern with the least amount of total trim loss is identified. This total trim loss is then converted to a percentage value (in terms of the stock sheet area) and multiplied by a certain fractional value to obtain the next value for beta. Chart 6.1 will be used to determine a trend for the change in total trim loss value as the value of beta increases by 0.01, and from this a fractional increase value will be determined.

The least total trim loss (LTT) line for all problems basically shows the same characteristics. If the value of beta is 0 (the initial value), the value of the least total trim loss is relatively high. As the value of beta increases (leading to more, and better patterns being generated) the value of the least total trim loss for the best pattern found using the given beta value decreases. This trend continues until the optimal solution for the specific problem instance is found, and then the LTT line will flatten and remain on the same stock sheet percentage value.

Table 6.18 summarizes the values given in chart 6.1. The table also gives the percentage values needed to reach the beta value required to generate the optimal solution.

Problem	0.00	0.01	0.02	Fraction
P2	0.1857	0.1354	0.0104	0.0560
P3	0.2671	0.1504	0.0154	0.0577
P4	0.1000	0.1000	0.0111	0.1110
P7	0.1915	0.0053	-	0.0277
P8	0.4000	0.2484	0.0124	0.0310
Average fraction				0.0567

Table 6.18: Calculation of beta (β) increase fraction

The fraction value for each problem (P2, P3, P4, P7 and P8) was calculated using the value of the least total trim loss value of the optimal pattern divided by the least total trim loss value of the best pattern found with the initial beta value. For instance, for P2 the value of fraction was calculated as follows:

$$0.0104 / 0.1857 = 0.0560$$

This implies that the value of beta should be set to 5.60% of the value of the least total trim loss as generated by a beta of 0.00 to find the optimal solution. The fraction values were then calculated for all other problem instances and an average fraction value was produced from these. The average fraction value is then 5.67%, meaning that if the optimal solution is not found with the initial beta value, it should be set to the current beta value plus 5.67% of the least total trim loss found with the initial beta. This process should then be iterated until the optimal solution is reached. Refer to chapter 7, section 7.2.7, pages 168-170, for numerical results when the increase fraction is implemented to solve problem instances.

6.6 Summary

Chapter 6 delved into the problems that were identified in chapter 5 concerning algorithms based on the Wang and modified Wang methods. It started off by introducing the reader to optimization techniques that could be

used to enhance the performance of C2DGC problem solving algorithms. The optimization techniques included pattern coding and elimination of symmetrical duplicate patterns. Chapter 6 continued with a discussion of lower bounds and the problems encountered by the modified Wang method when these bounds are not tight enough. It then introduced the PSSP algorithm that was proposed by the author, which attempted to calculate better lower bounds and improve the efficiency of the WAM algorithm. Lastly, the importance of upper bounds was discussed and how it can be used in accordance with other techniques to construct a strategy to manage the value of beta.

Chapter 7 supplies numerical results that were obtained by practically implementing the ideas that were represented in this chapter and then solving problems **P1** through **P8**.

CHAPTER 7: Numerical tests and results

7.1 Introduction

By practically implementing existing algorithms and the author's proposed PSSP algorithm, numerical results were obtained and these results will now be discussed. Firstly, a comparison will be made between the results obtained by the author's implementation of the Wang algorithms¹ (WA) and modified Wang algorithms (WAM) and the WA and WAM algorithms of Daza et al (Daza et al, 1995:643). Secondly, the results obtained by implementing algorithm enhancements (symmetrical duplicate pattern removal, partial stock sheet propagation algorithm) will be discussed.

7.2 Numerical results

Numerical tests were done on the eight C2DGC problem instances given in table 5.1 (chapter 5, section 5.1, page 56). These problems were also solved by Daza et al (Daza et al, 1995:643) and the different results will be compared. All problems were solved on the same personal computer, with the following specifications:

- Intel Gigabyte motherboard and an Athlon XP 1.8 GHz CPU;
- 256 MB of RAM;
- 40 gigabyte hard disk drive; and
- running Microsoft Windows 2000 Service Pack 2.

The programming language that was used is Borland C++ Builder 6.0 using Microsoft's DirectX application programming interface (API).

¹ Refer to chapter 5, section 5.2, pages 57-90, for information pertaining the author's implementation of the Wang and modified Wang algorithms.

7.2.1 AWA and AWAM algorithms versus DWA and DWAM algorithms

In this section, results obtained by solving the eight C2DGC problem instances given in chapter 5, are summarized. No algorithm enhancements were added (partial stock sheet propagation algorithm or symmetrical duplicate pattern removal). The WA and WAM algorithms, as was implemented by Daza et al, will be referred to as the DWA and DWAM algorithms. The results as given in table 7.1 for these two algorithms are taken directly from Daza's article (Daza et al, 1995:643). On the other hand, the algorithms as implemented by the author will be referred to as the AWA and AWAM algorithms. The AWA and AWAM algorithms utilise the improvements made by Vasko (refer to chapter 5, section 5.2.2, pages 75-77) to the original Wang method. The modified Wang methods (both the author's and Daza's) use the heuristic evaluation function as presented by Oliveira and Ferreira (refer to chapter 5, section 5.2.3.1.1, function 5.1, page 81). Section 7.2.4 gives numerical results when the admissible, monotone heuristic function 5.3 (refer to chapter 5, section 5.2.3.1.1, function 5.3, page 82) is used instead of function 5.1.

Table 7.1 summarizes the results obtained by Daza for the DWA and DWAM algorithms and by the author with the AWA and AWAM algorithm. The following codes are used as headings for the various columns in the table:

- **Problem:** Indicates which problem instance (refer to table 5.1, chapter 5, section 5.1, page 56) was solved;
- **Algorithm:** The specific algorithm that was used to solve the different **problem** instances;
- **β:** The value of beta that was used by the specified **algorithm** to solve a specific **problem** instance;
- **N:** The exact number of nodes that were **generated** by each **algorithm**. This implies that every pattern that is generated by Wang's method is counted, even if it does not represent a feasible pattern to be stored;

- **L:** The exact number of nodes that were **stored** in memory by each **algorithm**. This value represents only the feasible patterns that were generated by Wang's method and that were stored;
- **Processing time:** The time taken by each **algorithm** to solve a specific **problem** instance, written in the form (minutes'seconds.milliseconds"); and
- **Trim loss:** The least amount of total trim loss for the optimal cutting pattern generated by a specific **algorithm** for a specific **problem** instance.

As this chapter progresses, a large number of algorithms will be introduced, each of which will be assigned a specific abbreviation. As new algorithms are introduced, a reference table will be updated with the new abbreviations and a short description of the characteristics of the new algorithm. Reference table 7.1 introduces the first four algorithms, named DWA, DWAM, AWA and AWAM.

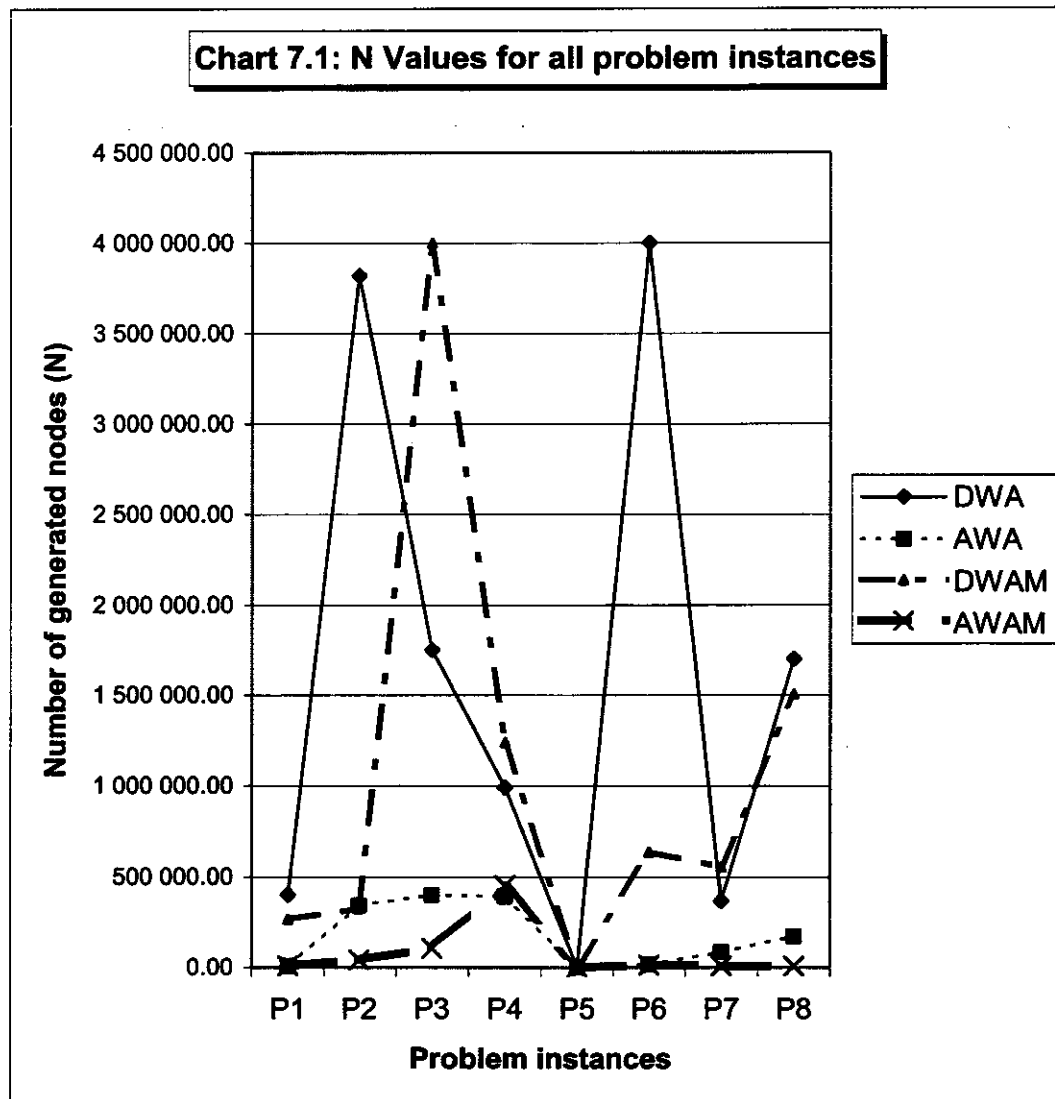
Algorithm	Description
DWA	Daza's implementation of the original Wang method. This algorithm utilizes a breadth-first search method combined with Wang's rectangle building method and pruning criteria (using a proportion parameter Beta and internal trim loss).
DWAM	Daza's implementation of the modified Wang method. This algorithm utilizes a breadth-first search method combined with Wang's rectangle building method and Oliveira and Ferreira's heuristic function ($h(n)$) and pruning criteria (using a proportion parameter Beta and internal trim as well as estimated external trim loss).
AWA	The author's implementation of the original Wang method. This algorithm utilizes a breadth-first search method combined with Wang's rectangle building method and pruning criteria (using a proportion parameter Beta and internal trim loss). This algorithm also implements the improvements made by Vasko (complete builds) to the Wang method.
AWAM	The author's implementation of the modified Wang method. This algorithm utilizes a breadth-first search method combined with Wang's rectangle building method and Oliveira and Ferreira's heuristic function ($h(n)$) and pruning criteria (using a proportion parameter Beta and internal trim as well as estimated external trim loss). This algorithm also implements the improvements made by Vasko (complete builds) to the Wang method.

Reference table 7.1: DWA, DWAM, AWA and AWAM algorithms

Problem	Algorithm	β	N	L	Processing Time	Trim Loss
P1	DWA	0.00	400 952	546	04'25.22"	0
	AWA	0.00	8 931	386	00'03.33"	0
	DWAM	0.00	267 168	446	03'56.01"	0
	AWAM	0.00	8 465	376	00'02.11"	0
P2	DWA	0.02	3 819 692	1 693	35'54.01"	29
	AWA	0.02	343 470	628	01'49.90"	29
	DWAM	0.05	324 942	485	04'14.69"	29
	AWAM	0.03	42 808	222	00'14.09"	29
P3	DWA	0.02	1 750 218	1 165	16'57.76"	43
	AWA	0.02	399 440	825	02'09.90"	43
	DWAM	0.06	*	*	*	*
	AWAM	0.03	105 255	396	00'34.09"	43
P4	DWA	0.02	989 954	868	09'29.95"	31
	AWA	0.02	393 210	1 144	02'17.30"	31
	DWAM	0.06	1 238 464	971	15'08.40"	100
	AWAM	0.04	451 784	1 423	02'38.90"	31
P5	DWA	0.00	1 400	31	00'00.93"	0
	AWA	0.00	273	34	00'00.18"	0
	DWAM	0.00	926	25	00'01.31"	0
	AWAM	0.00	273	34	00'00.18"	0
P6	DWA	0.00	*	*	*	*
	AWA	0.00	15 828	389	00'05.44"	0
	DWAM	0.00	637 298	693	09'10.24"	0
	AWAM	0.00	12 226	311	00'04.33"	0
P7	DWA	0.01	367 692	526	03'39.58"	8
	AWA	0.01	85 625	352	00'29.48"	8
	DWAM	0.05	555 912	644	12'00.98"	11
	AWAM	0.01	9 402	125	00'03.52"	8
P8	DWA	0.02	1 700 076	1 138	16'00.91"	34
	AWA	0.02	171 873	446	00'59.12"	34
	DWAM	0.08	1 502 852	1 081	29'00.00"	34
	AWAM	0.02	6 961	81	00'02.56"	34

Table 7.1: Results for AWA and AWAM versus DWA and DWAM

The results displayed in the table show that for problem **P1** the AWA and AWAM algorithms perform much better than the DWA and DWAM algorithms in terms of generated nodes and stored nodes. In terms of generated nodes for **P1**, the AWA generates only 8 929 nodes compared to the 400 952 of the DWA algorithm.



This is an impressive 392 023 saving in terms of number of generated nodes. The number of stored nodes for the AWA amounts to 386 where the DWA stores 546 nodes. Where the AWAM algorithm is concerned in problem **P1**, the results are equally impressive. The AWAM algorithm generates only

8 465 nodes to find the optimal solution where the DWAM algorithm generates 267 168. This is nearly 31 times more generated nodes for the DWAM algorithm than for the AWAM algorithm. In terms of processing time, the AWA and AWAM algorithms perform much better than the DWA and DWAM algorithms. This, in part, can be ignored because of the fact that the problems were solved on a much slower computer, but then again the AWA and AWAM algorithms do generate, in virtually all cases, a lot less nodes than the DWA and DWAM algorithms.

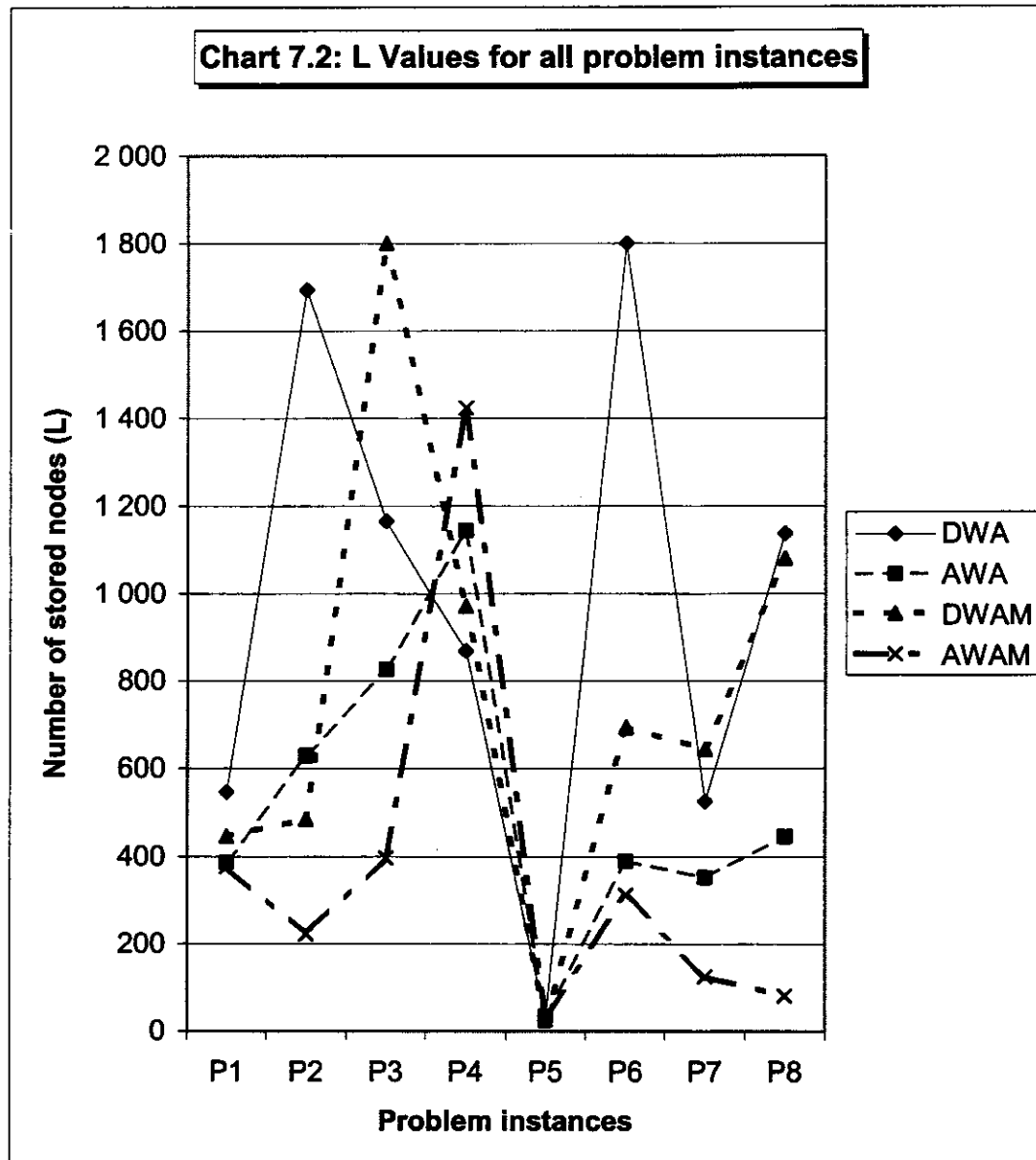
Chart 7.1 graphically demonstrates the results. The chart displays the number of generated nodes (**N**) as generated by every algorithm for each problem instance. It is clearly visible that the AWA and AWAM algorithms perform better for all problem instances than the DWA and DWAM algorithms. In most cases, the performance improvement is quite drastic.

Chart 7.2 displays the number of stored nodes (**L**) as stored by every algorithm for each problem instance. It is obvious that the DWA and DWAM algorithms do not perform much better where stored nodes are concerned. Both the AWA and AWAM algorithms perform better than the DWA and DWAM algorithms for 6 of the 8 problem instances that were solved where stored nodes are concerned.

Furthermore it should be noted that the DWA algorithm fails to find an optimal solution for problem **P3** (indicated by an asterisk (*) in table 7.1), but the AWA algorithm does manage to efficiently find the optimal solution for **P3**. Lastly, the DWA algorithm fails to find a solution for problem **P6**, but the AWA algorithm once again manages to efficiently solve the problem and delivers an optimal solution.

With all these facts considered, and the results showing that the AWA and AWAM algorithms outperform the DWA and DWAM algorithms, it is accepted that the AWA and AWAM algorithms are indeed suitable algorithms for which more efficient algorithmic enhancements may now be considered. From this

point on, the results obtained from the implementation of the AWA and AWAM algorithms to solve the problem instances P1 to P8 will be used for reference.



7.2.2 AWA and AWAM algorithms versus the MAWAM algorithm

In chapter 5, a detailed description was given concerning the admissibility and monotonicity of heuristic functions. In short, the original modified Wang method as proposed by Oliveira and Ferreira utilised a heuristic evaluation function to determine estimated external trim loss. Although this function worked reasonably well, the method sometimes required a larger β value to find the optimal solution than the original Wang method (WA) did (this is evident when studying table 7.1 above, for instance, problem P2 requires a beta value of 0.02 for the AWA method and a beta value of 0.03 for the AWAM method). This was because in some cases, the heuristic function overestimated the value of $h(n)$, resulting in critical patterns not being generated. Furthermore, these overestimates could result in a situation where the algorithm fails to find the optimal solution for a problem instance even if the value of beta is increased to 1. Daza *et al* (Daza *et al*: 1995:639) provides a solution to this problem in the form of an admissible, monotone heuristic function² that never overestimates the value of $h_3(n)$. It therefore only requires the same β value as the original Wang method to reach optimal cutting patterns.

Reference table 7.2 introduces the MAWAM algorithm and shows that this algorithm differs from the AWAM algorithm in that it uses a monotone and admissible heuristic function $h_3(n)$ instead of the heuristic function $h(n)$ as introduced by Oliveira and Ferreira.

Algorithm	Description
DWA	Daza's implementation of the original Wang method. This algorithm utilizes a breadth-first search method combined with Wang's rectangle building method and pruning criteria (using a proportion parameter Beta and internal trim loss).
DWAM	Daza's implementation of the modified Wang method. This algorithm utilizes a breadth-first search method combined with Wang's rectangle building method and Oliveira and Ferreira's heuristic function ($h(n)$) and pruning criteria (using a proportion parameter Beta and internal trim as well as estimated external trim loss).

² Refer to chapter 5, section 5.2.3.1.1, function 5.2, page 82.

AWA	The author's implementation of the original Wang method. This algorithm utilizes a breadth-first search method combined with Wang's rectangle building method and pruning criteria (using a proportion parameter Beta and internal trim loss). This algorithm also implements the improvements made by Vasko (complete builds) to the Wang method.
AWAM	The author's implementation of the modified Wang method. This algorithm utilizes a breadth-first search method combined with Wang's rectangle building method and Oliveira and Ferreira's heuristic function ($h(n)$) and pruning criteria (using a proportion parameter Beta and internal trim as well as estimated external trim loss). This algorithm also implements the improvements made by Vasko (complete builds) to the Wang method.
MAWAM	The author's implementation of an enhanced modified Wang method. This algorithm utilizes a breadth-first search method combined with Wang's rectangle building method and Daza's heuristic function ($h_3(n)$) and pruning criteria (using a proportion parameter Beta and internal trim as well as estimated external trim loss). This algorithm also implements the improvements made by Vasko (complete builds) to the Wang method.

Reference table 7.2: MAWAM algorithm

Table 7.2 summarizes the results obtained by solving the eight example problem instances with the MAWAM algorithm using an admissible heuristic function ($h_3(n)$), and it also gives results obtained from the AWA (author's original Wang method) and AWAM (author's original modified Wang method) algorithms for comparative purposes.

It is evident when studying table 7.2 that Daza's heuristic function holds true to its promises. It always finds the optimal solution for all the problem instances utilizing the same β value as the original Wang method. One important point to note is that if the AWAM and the MAWAM algorithms find the optimal solution at the same β value, it is possible for the AWAM algorithm to outperform the MAWAM algorithm, as illustrated in problems **P6**, **P7** and **P8**. The reason for this is that even though the AWAM algorithm sometimes overestimates the $h(n)$ value, it may for some problems, not discard critical builds that lead to the optimal solution. For any problem instance, the AWAM algorithm is not guaranteed to find the optimal solution using the same β value as the MAWAM method, and is indeed "lucky" if it does. The fact remains however, that if they do find the optimal solution with the same β value, the AWAM algorithm could fare better than the MAWAM algorithm.

Problem	Algorithm	β	N	L	Processing Time	Trim Loss
P1	AWA	0.00	8 931	386	00'03.33"	0
	AWAM	0.00	8 465	376	00'02.11"	0
	MAWAM	0.00	8 465	376	00'02.11"	0
P2	AWA	0.02	343 470	628	01'49.90"	29
	AWAM	0.03	42 808	222	00'14.09"	29
	MAWAM	0.02	38 466	183	00'12.47"	29
P3	AWA	0.02	399 440	825	02'09.90"	43
	AWAM	0.03	105 255	396	00'34.09"	43
	MAWAM	0.02	96 837	357	00'31.62"	43
P4	AWA	0.02	393 210	1 144	02'17.30"	31
	AWAM	0.04	451 784	1 423	02'38.90"	31
	MAWAM	0.02	76 858	495	00'25.95"	31
P5	AWA	0.00	273	34	00'00.18"	0
	AWAM	0.00	273	34	00'00.18"	0
	MAWAM	0.00	273	34	00'00.18"	0
P6	AWA	0.00	15 828	389	00'05.44"	0
	AWAM	0.00	12 226	311	00'04.33"	0
	MAWAM	0.00	13 549	337	00'04.58"	0
P7	AWA	0.01	85 625	352	00'29.48"	8
	AWAM	0.01	9 402	125	00'03.52"	8
	MAWAM	0.01	19 020	159	00'06.52"	8
P8	AWA	0.02	171 873	446	00'59.12"	34
	AWAM	0.02	6 961	81	00'02.56"	34
	MAWAM	0.02	23 161	144	00'07.97"	34

Table 7.2: Admissible heuristic function

**Chart 7.3: N Values for all problem instances -
Admissible heuristic function**

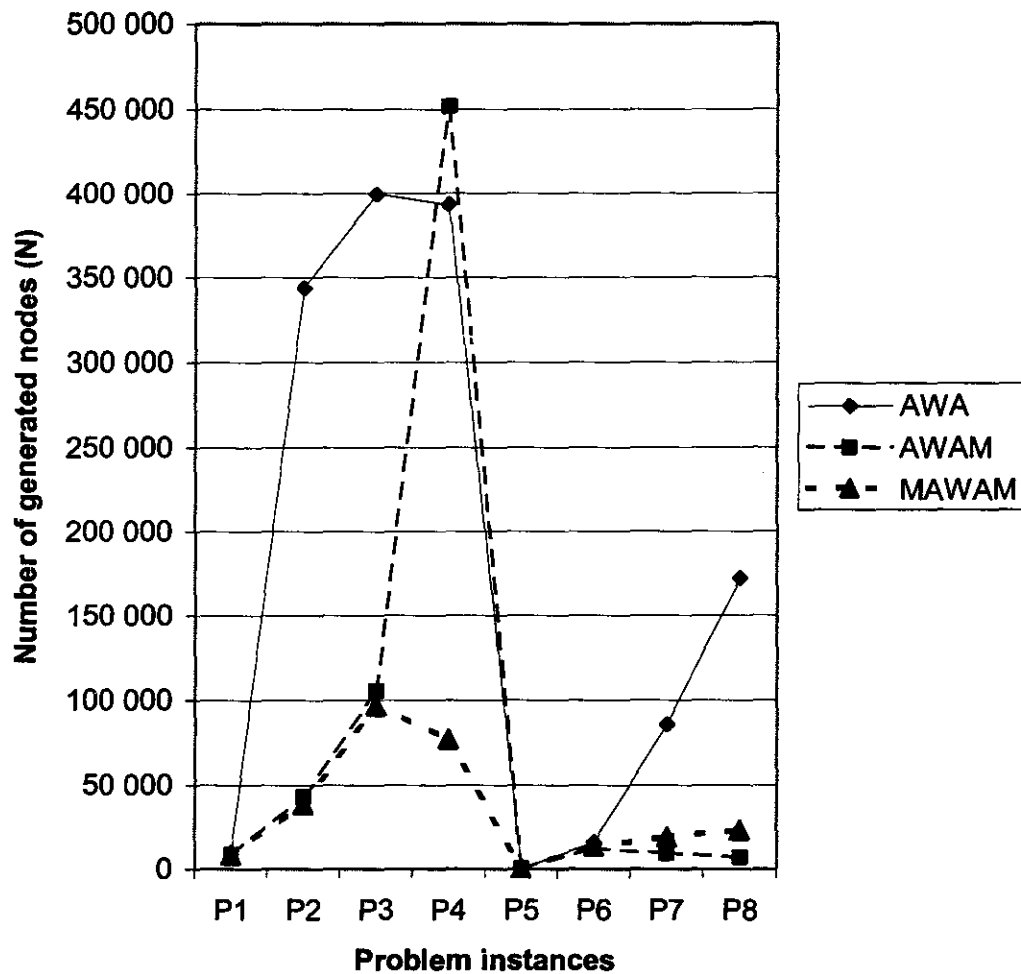


Chart 7.3 shows the consistent and reliable performance of the MAWAM algorithm when compared to the AWA and AWAM algorithms. It performs better in most instances where the number of generated nodes (**N**) are concerned. It should be noted that the **N** value represents the number of generated nodes for the three algorithms (AWA, AWAM and MAWAM) at the beta (β) value where the optimal pattern was found for each individual algorithm.

Chart 7.4: L Values for all problem instances - Admissible heuristic function

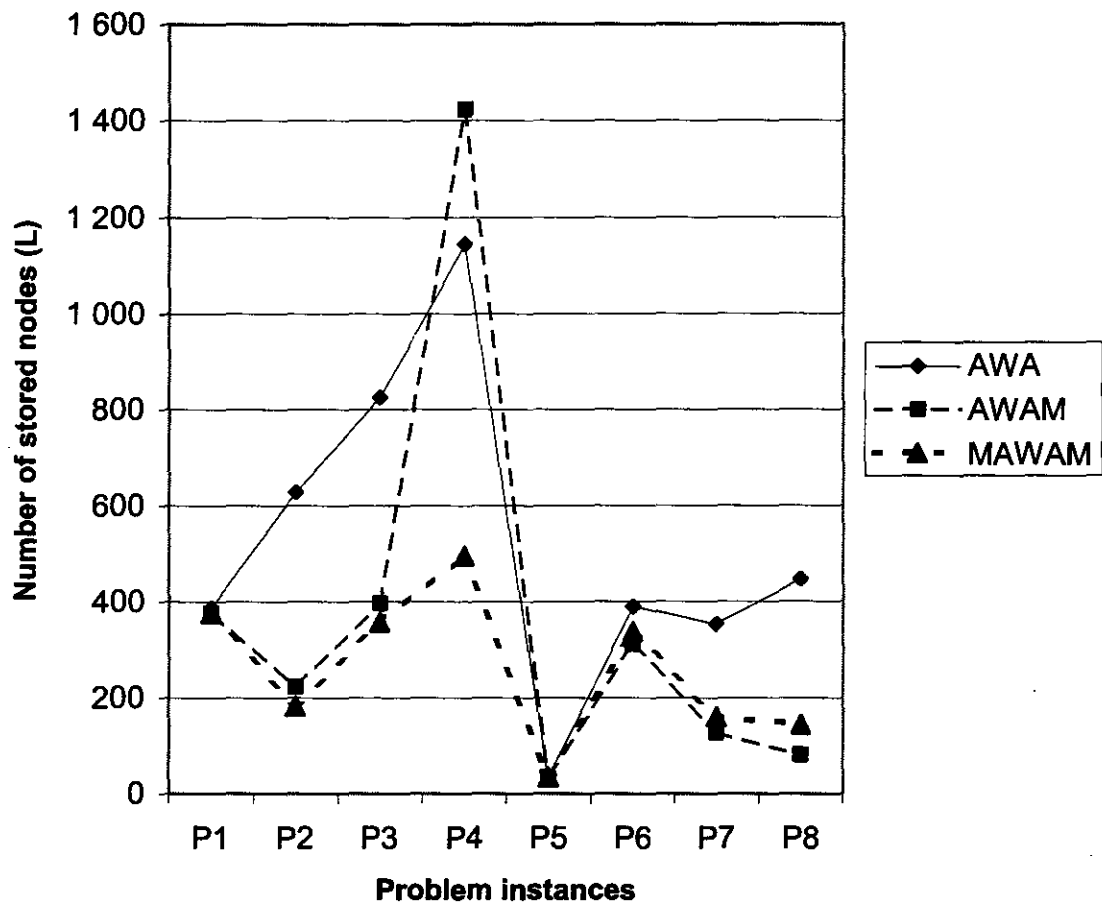


Chart 7.4 shows that where stored nodes (L) are concerned, the MAWAM algorithm once again performs well with less high peaks than the AWA and AWAM algorithms.

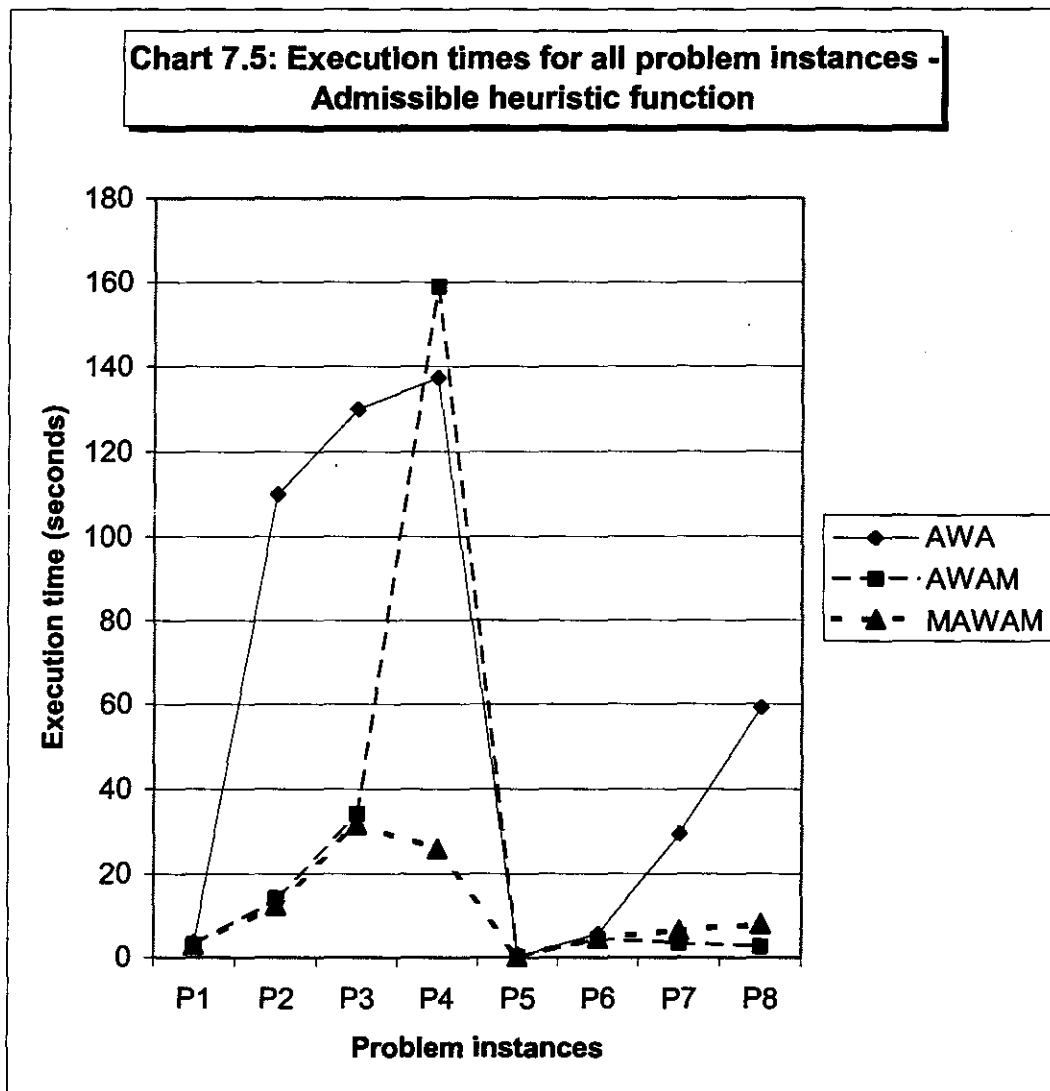


Chart 7.5 shows nearly the same structure and properties as chart 7.3. This can be expected when the heuristic function that is used do not require excessive computational times.

7.2.3 AWA and AWAM algorithms versus A*WA and A*WAM algorithms

Up to this point, the implemented algorithms (both Daza's (DWA and DWAM) and the author's (AWA, AWAM and MAWAM)) algorithms utilized Wang's method of rectangle generation, combined with a breadth-first search

algorithm³. The DWA and AWA algorithms used the original Wang method's pruning criteria to prune away unnecessary branches in the search tree (where internal trim loss is greater than the value of $(\beta) \cdot (\text{stock sheet length}) \cdot (\text{stock sheet width})$). The DWAM, AWAM and MAWAM algorithms used the modified Wang method's pruning criteria combined with a breadth-first search algorithm to prune away unnecessary branches in the search tree (where estimated total trim loss is greater than the value of $(\beta) \cdot (\text{stock sheet length}) \cdot (\text{stock sheet width})$).

These algorithms can be improved upon by replacing the breath-first search algorithm with an A* search algorithm (which implies a branch-and-bound search combined with the dynamic programming principle⁴). This implies that nodes will be sorted as they are expanded and the most promising node (least cost node) will always be expanded next. The most promising node is the node with the lowest internal trim loss for the original Wang method and the node with the lowest estimated total trim loss for the modified Wang method. Furthermore, the search process can be terminated as soon as the least cost node's cost (the one to be expanded next) is higher than that of the best solution found up to that point.

The enhanced AWA and MAWAM algorithms are called the A*WA and A*WAM algorithms. The A*WA and A*WAM algorithms are introduced in reference table 7.3

³ Refer to chapter 5, section 5.2.1.6.1, pages 67-75, for a detailed discussion on the implementation of a breadth-first search method combined with the Wang method.

⁴ Refer to chapter 4, sections 4.2.6-4.2.9, pages 43-54, for detailed discussions of branch-and-bound search, branch-and-bound search with underestimations, branch-and-bound using the dynamic programming principle and the A* search method.

Algorithm	Description
DWA	Daza's implementation of the original Wang method. This algorithm utilizes a breadth-first search method combined with Wang's rectangle building method and pruning criteria (using a proportion parameter Beta and internal trim loss).
DWAM	Daza's implementation of the modified Wang method. This algorithm utilizes a breadth-first search method combined with Wang's rectangle building method and Oliveira and Ferreira's heuristic function ($h(n)$) and pruning criteria (using a proportion parameter Beta and internal trim as well as estimated external trim loss).
AWA	The author's implementation of the original Wang method. This algorithm utilizes a breadth-first search method combined with Wang's rectangle building method and pruning criteria (using a proportion parameter Beta and internal trim loss). This algorithm also implements the improvements made by Vasko (complete builds) to the Wang method.
AWAM	The author's implementation of the modified Wang method. This algorithm utilizes a breadth-first search method combined with Wang's rectangle building method and Oliveira and Ferreira's heuristic function ($h(n)$) and pruning criteria (using a proportion parameter Beta and internal trim as well as estimated external trim loss). This algorithm also implements the improvements made by Vasko (complete builds) to the Wang method.
MAWAM	The author's implementation of an enhanced modified Wang method. This algorithm utilizes a breadth-first search method combined with Wang's rectangle building method and Daza's heuristic function ($h_3(n)$) and pruning criteria (using a proportion parameter Beta and internal trim as well as estimated external trim loss). This algorithm also implements the improvements made by Vasko (complete builds) to the Wang method.
A*WA	The author's implementation of an enhanced original Wang method. This algorithm utilizes an A* search method combined with Wang's rectangle building method and pruning criteria (using a proportion parameter Beta and internal trim loss). This algorithm also implements the improvements made by Vasko (complete builds) to the Wang method.
A*WAM	The author's implementation of an enhanced modified Wang method. This algorithm utilizes an A* search method combined with Wang's rectangle building method and Daza's heuristic function ($h_3(n)$) and pruning criteria (using a proportion parameter Beta and internal trim as well as estimated external trim loss). This algorithm also implements the improvements made by Vasko (complete builds) to the Wang method.

Reference table 7.3: A*WA and A*WAM algorithms

The eight example problem instances were once again solved using these new algorithms based on the A* search method and the results are compared with that of the AWA and MAWAM algorithms. Note that the MAWAM algorithm is used instead of the AWAM algorithm, this is because using an admissible and monotone heuristic function ($h_3(n)$) is preferred and because the A*WAM method also uses $h_3(n)$. Table 7.3 summarizes the results obtained by solving the example problem instances.

Problem	Algorithm	β	N	L	Processing Time	Trim Loss
P1	AWA	0.00	8 931	386	00'03.33"	0
	MAWAM	0.00	8 465	376	00'02.11"	0
	A*WA	0.00	8 931	386	00'03.33"	0
	A*WAM	0.00	8 465	376	00'02.11"	0
P2	AWA	0.02	343 470	628	01'49.90"	29
	MAWAM	0.02	38 466	183	00'12.47"	29
	A*WA	0.02	140 449	534	00'45.80"	29
	A*WAM	0.02	17 923	170	00'05.88"	29
P3	AWA	0.02	399 440	825	02'09.90"	43
	MAWAM	0.02	96 837	357	00'31.62"	43
	A*WA	0.02	400 349	1 055	02'11.20"	43
	A*WAM	0.02	75 342	397	00'24.83"	43
P4	AWA	0.02	393 210	1 144	02'17.30"	31
	MAWAM	0.02	76 858	495	00'25.95"	31
	A*WA	0.02	131 184	963	00'43.53"	31
	A*WAM	0.02	25 369	279	00'08.63"	31
P5	AWA	0.00	273	34	00'00.18"	0
	MAWAM	0.00	273	34	00'00.18"	0
	A*WA	0.00	273	34	00'00.18"	0
	A*WAM	0.00	273	34	00'00.18"	0
P6	AWA	0.00	15 828	389	00'05.44"	0
	MAWAM	0.00	13 549	337	00'04.58"	0
	A*WA	0.00	15 828	389	00'05.44"	0
	A*WAM	0.00	13 549	337	00'04.58"	0
P7	AWA	0.01	85 625	352	00'29.48"	8
	MAWAM	0.01	19 020	159	00'06.52"	8
	A*WA	0.01	44 395	324	00'14.95"	8
	A*WAM	0.01	10 627	144	00'03.81"	8
P8	AWA	0.02	171 873	446	00'59.12"	34
	MAWAM	0.02	23 161	144	00'07.97"	34
	A*WA	0.02	91 015	442	00'32.17"	34
	A*WAM	0.02	12 935	138	00'04.63"	34

Table 7.3: A* search algorithms

Chart 7.6: N Values for all problem instances - A* search algorithms

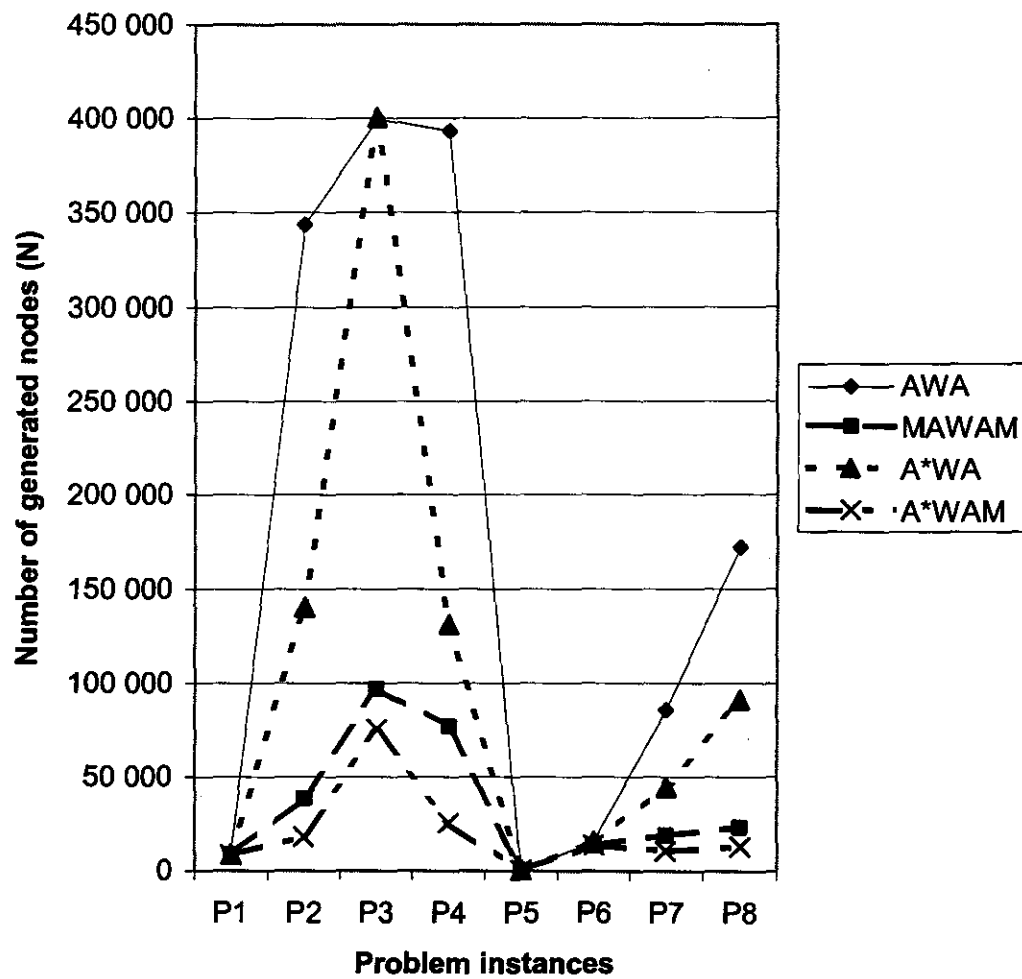
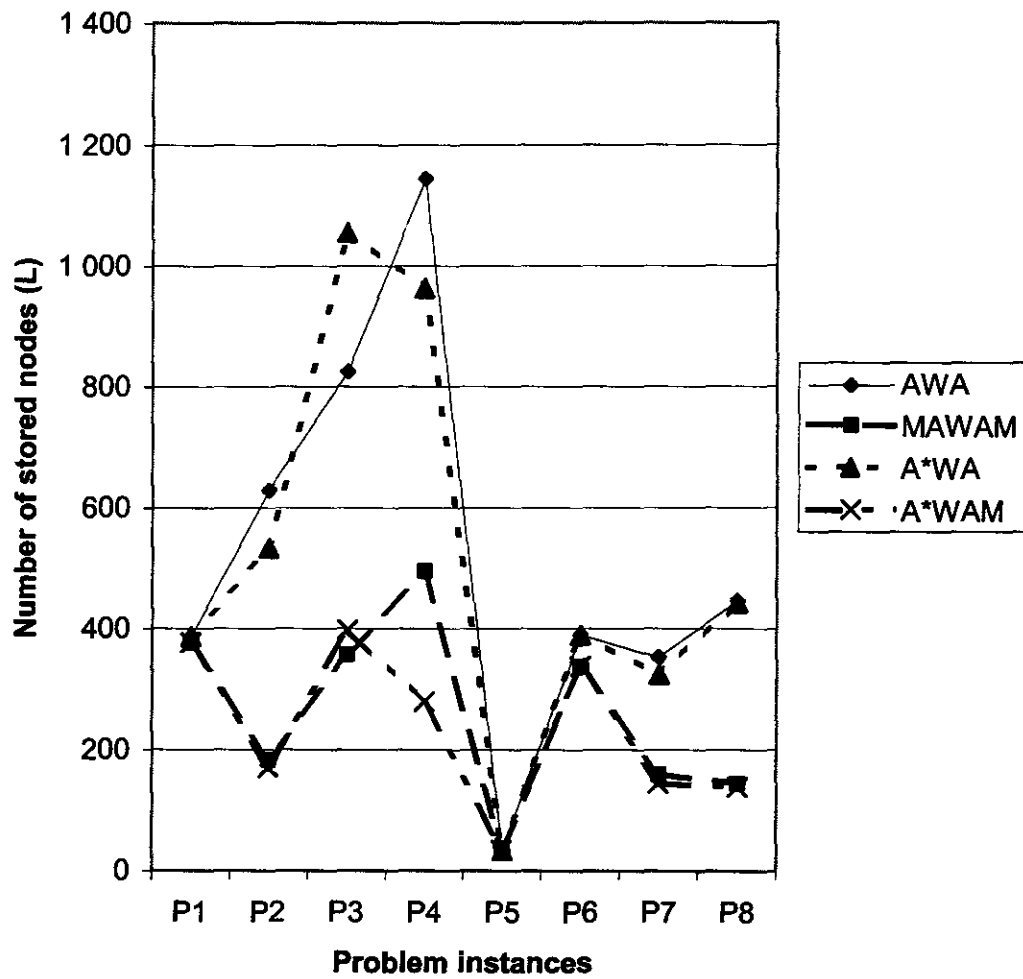
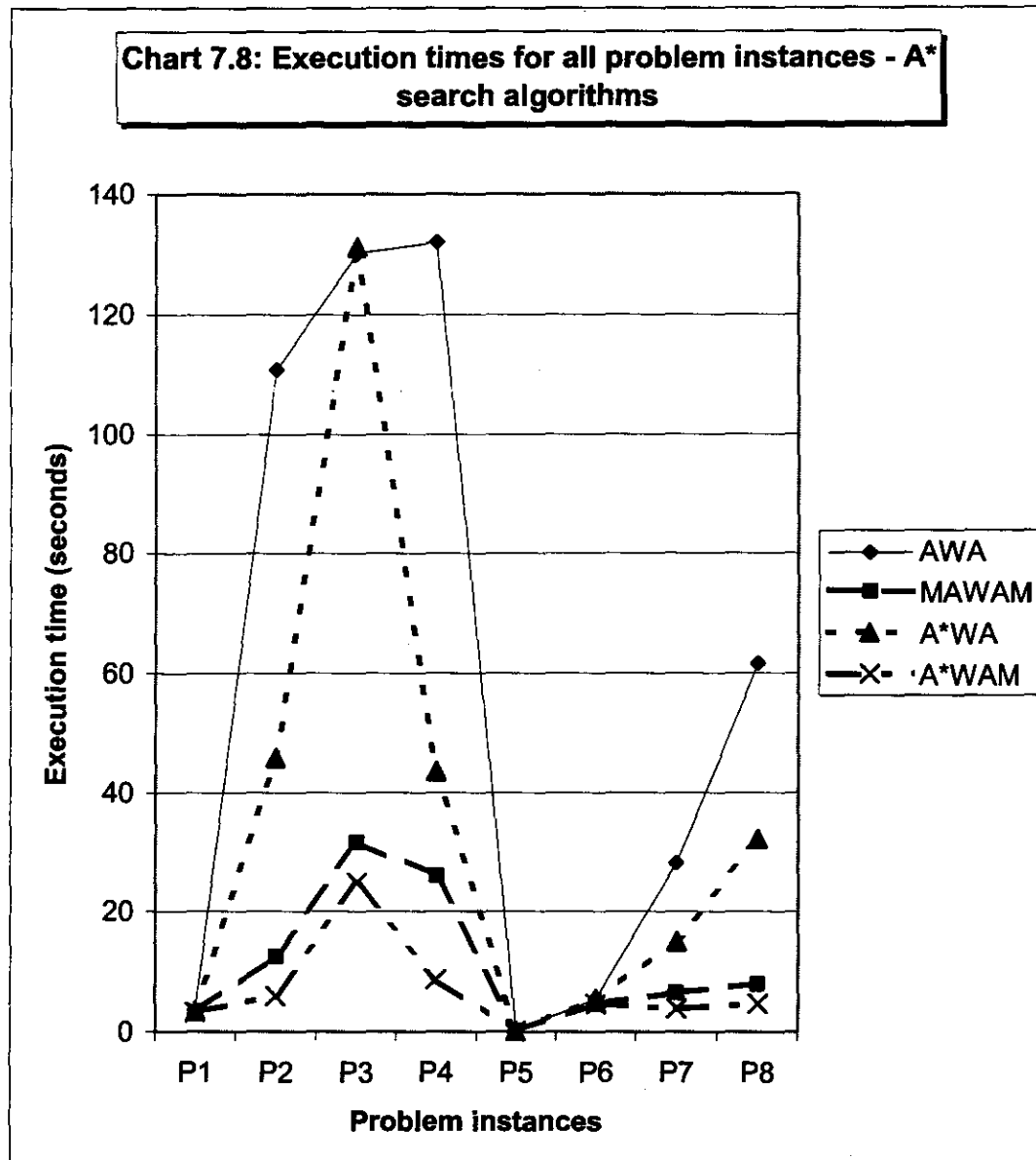


Chart 7.7: L Values for all problem instances - A* search algorithms





7.2.4 Adding symmetrical duplicate pattern removal to the A*WA and A*WAM algorithms

Cung *et al* (Cung *et al*, 2000:196) studied symmetrical patterns occurring in search lists of cutting problems and devised a pattern coding scheme to identify these duplicates and remove them from the list. They are called the SA*WA and SA*WAM algorithms and reference table 7.4 introduces them.

Algorithm	Description
DWA	Daza's implementation of the original Wang method. This algorithm utilizes a breadth-first search method combined with Wang's rectangle building method and pruning criteria (using a proportion parameter Beta and internal trim loss).
DWAM	Daza's implementation of the modified Wang method. This algorithm utilizes a breadth-first search method combined with Wang's rectangle building method and Oliveira and Ferreira's heuristic function ($h(n)$) and pruning criteria (using a proportion parameter Beta and internal trim as well as estimated external trim loss).
AWA	The author's implementation of the original Wang method. This algorithm utilizes a breadth-first search method combined with Wang's rectangle building method and pruning criteria (using a proportion parameter Beta and internal trim loss). This algorithm also implements the improvements made by Vasko (complete builds) to the Wang method.
AWAM	The author's implementation of the modified Wang method. This algorithm utilizes a breadth-first search method combined with Wang's rectangle building method and Oliveira and Ferreira's heuristic function ($h(n)$) and pruning criteria (using a proportion parameter Beta and internal trim as well as estimated external trim loss). This algorithm also implements the improvements made by Vasko (complete builds) to the Wang method.
MAWAM	The author's implementation of an enhanced modified Wang method. This algorithm utilizes a breadth-first search method combined with Wang's rectangle building method and Daza's heuristic function ($h_3(n)$) and pruning criteria (using a proportion parameter Beta and internal trim as well as estimated external trim loss). This algorithm also implements the improvements made by Vasko (complete builds) to the Wang method.
A*WA	The author's implementation of an enhanced original Wang method. This algorithm utilizes an A* search method combined with Wang's rectangle building method and pruning criteria (using a proportion parameter Beta and internal trim loss). This algorithm also implements the improvements made by Vasko (complete builds) to the Wang method.
A*WAM	The author's implementation of an enhanced modified Wang method. This algorithm utilizes an A* search method combined with Wang's rectangle building method and Daza's heuristic function ($h_3(n)$) and pruning criteria (using a proportion parameter Beta and internal trim as well as estimated external trim loss). This algorithm also implements the improvements made by Vasko (complete builds) to the Wang method.
SA*WA	The author's implementation of an enhanced original Wang method. This algorithm utilizes an A* search method combined with Wang's rectangle building method and pruning criteria (using a proportion parameter Beta and internal trim loss). This algorithm also implements the improvements made by Vasko (complete builds) to the Wang method as well as Cung's pattern coding scheme to remove symmetrical duplicate patterns.
SA*WAM	The author's implementation of an enhanced modified Wang method. This algorithm utilizes an A* search method combined with Wang's rectangle building method and Daza's heuristic function ($h_3(n)$) and pruning criteria (using a proportion parameter Beta and internal trim as well as estimated external trim loss). This algorithm also implements the improvements made by Vasko (complete builds) to the Wang method as well as Cung's pattern coding scheme to remove symmetrical duplicate patterns.

Reference table 7.4: SA*WA and SA*WAM algorithms

Table 7.4 shows the results when the 8 example problem instances are solved with the SA*WA and SA*WAM algorithms that utilize Cung's method of pattern coding to remove symmetrical duplicate patterns.

Problem	Algorithm	β	N	L	Processing Time	Trim Loss
P1	A*WA	0.00	8 931	386	00'03.33"	0
	A*WAM	0.00	8 465	376	00'02.11"	0
	SA*WA	0.00	8 852	382	00'03.30"	0
	SA*WAM	0.00	8 388	372	00'02.01"	0
P2	A*WA	0.02	140 449	534	00'45.80"	29
	A*WAM	0.02	17 923	170	00'05.88"	29
	SA*WA	0.02	139 348	530	00'45.63"	29
	SA*WAM	0.02	17 823	169	00'05.83"	29
P3	A*WA	0.02	400 349	1 055	02'11.20"	43
	A*WAM	0.02	75 342	397	00'24.83"	43
	SA*WA	0.02	388 310	1 031	02'09.22"	43
	SA*WAM	0.02	73 589	391	00'23.92"	43
P4	A*WA	0.02	131 184	963	00'43.53"	31
	A*WAM	0.02	25 369	279	00'08.63"	31
	SA*WA	0.02	125 987	889	00'40.67"	31
	SA*WAM	0.02	25 132	273	00'08.23"	31
P5	A*WA	0.00	273	34	00'00.18"	0
	A*WAM	0.00	273	34	00'00.18"	0
	SA*WA	0.00	272	33	00'00.17"	0
	SA*WAM	0.00	272	33	00'00.17"	0
P6	A*WA	0.00	15 828	389	00'05.44"	0
	A*WAM	0.00	13 549	337	00'04.58"	0
	SA*WA	0.00	15 275	355	00'05.28"	0
	SA*WAM	0.00	13 015	304	00'04.46"	0
P7	A*WA	0.01	44 395	324	00'14.95"	8
	A*WAM	0.01	10 627	144	00'03.81"	8
	SA*WA	0.01	43 797	321	00'14.57"	8
	SA*WAM	0.01	10 627	144	00'03.81"	8
P8	A*WA	0.02	91 015	442	00'32.17"	34
	A*WAM	0.02	12 935	138	00'04.63"	34
	SA*WA	0.02	90 442	440	00'31.47"	34
	SA*WAM	0.02	12 860	138	00'04.51"	34

Table 7.4: A* search algorithms with symmetrical duplicate pattern removal

The results show that this algorithmic enhancement does not greatly enhance the performance of the A*WA and A*WAM algorithms, but Cung does state that it is possible to generate more efficient and competitive strategies using the pattern coding scheme to reject more complex symmetrical duplicate combinations.

7.2.5 Partial stock sheet propagation (PSSP) algorithm

In the previous section, the A*WAM algorithm was found to be one of the best performing algorithms tested so far (without symmetrical duplicate pattern removal). The PSSP algorithm, as described in chapter 6 (section 6.3.1, pages 111-123), will utilize the A*WAM algorithm. One of the reasons is that an admissible heuristic function is needed if the PSSP algorithm, as proposed by the author, is to be complete and exact.

Table 7.5 summarizes the numerical results that were obtained by practically implementing the PSSP algorithm. The table contains data about the solving of the partial sub-problem with the original Wang method (**Sub AWA**). Furthermore, it also displays data for the complete problem that were solved using the A*WAM algorithm with updated underestimate values (**PSSP**). These values are then added to show the total number of generated (**N**) and stored (**L**) nodes as well as the combined execution time. (For more information on the PSSP algorithm, refer to chapter 6, section 6.3.1, pages 111-123).

The partial sub-problems were solved using the original demand rectangles and a stock sheet of the dimensions $(L, (0.5) \cdot W)$. Although any dimensions could be used for the sub-problem sheet, it was found that the $(L, (0.5) \cdot W)$ dimension performed well (refer to table 7.6, pages 161-163). Recall that this partial sub-problem is solved in order to update the modified Wang reference table containing underestimate values. Lastly, as stated in chapter 6, section 6.3.1, pages 111-123, the partial sub-problem will only be solved if the value of β is greater than 0, otherwise only the results as obtained from the

A*WAM algorithm alone will be used. For all problem instances (where the beta value is not equal to 0) the PSSP algorithm performs better than the A*WAM algorithm, both in terms of execution time and the number of nodes that were generated.

Problem	Algorithm	β	N	L	Processing Time	Trim Loss
P1	A*WAM	0.00	8 465	376	00'02.11"	0
P2	A*WAM	0.02	17 923	170	00'05.88"	29
	Sub AWA	0.02	71	9	00'00.08 "	
	PSSP	0.02	17 410	167	00'05.74 "	29
	Total		17 481	176	00'05.82"	
P3	A*WAM	0.02	75 342	397	00'24.83"	43
	Sub AWA	0.02	1 473	44	00'00.64 "	
	PSSP	0.02	73 553	385	00'23.64 "	43
	Total		75 026	429	00'24.28"	
P4	A*WAM	0.02	25 369	279	00'08.63"	31
	Sub AWA	0.02	859	27	00'00.42 "	
	PSSP	0.02	22 511	256	00'07.80 "	31
	Total		23 370	283	00'08.22"	
P5	A*WAM	0.00	273	34	00'00.18"	0
P6	A*WAM	0.00	13 549	337	00'04.58"	0
P7	A*WAM	0.01	10 627	144	00'03.81"	8
	Sub AWA	0.01	26	13	00'00.04 "	
	PSSP	0.01	10 133	139	00'03.72 "	8
	Total		10 159	142	00'03.76"	
P8	A*WAM	0.02	12 935	138	00'04.63"	34
	Sub AWA	0.02	270	16	00'00.14 "	
	PSSP	0.02	11 760	137	00'04.16 "	34
	Total		12 030	153	00'04.30"	

Table 7.5: PSSP algorithm results

To conclude the results for the PSSP algorithm, table 7.6 displays numerical results obtained while testing which percentage of the stock sheet should be used to solve the partial problem. Only problems P2, P3, P4, P7 and P8 are included, as the other problem instances require a beta value of only 0 to generate optimal solutions.

Problem	Algorithm	β	N	L	Processing Time	Trim Loss
P2	50% L Sub	0.02	3 666	52	00'01.125 "	29
	PSSP	0.02	<u>17 564</u>	<u>168</u>	<u>00'05.343 "</u>	
	Total		21 230	220	00'06.468"	
	40% L Sub	0.02	347	18	00'00.172 "	29
	PSSP	0.02	<u>17 816</u>	<u>168</u>	<u>00'05.343 "</u>	
	Total		18 163	186	00'05.515"	
	30% L Sub	0.02	63	6	00'00.047 "	29
	PSSP	0.02	<u>17 923</u>	<u>170</u>	<u>00'05.578 "</u>	
	Total		17 986	176	00'05.625"	
	50% W Sub	0.02	71	9	00'00.082 "	29
	PSSP	0.02	<u>17 410</u>	<u>167</u>	<u>00'05.740 "</u>	
	Total		17 481	176	00'05.822"	
	40% W Sub	0.02	5	2	00'00.032 "	29
	PSSP	0.02	<u>17 767</u>	<u>169</u>	<u>00'05.625 "</u>	
	Total		17 772	171	00'05.657"	
	30% W Sub	0.02	4	2	00'00.047 "	29
	PSSP	0.02	<u>17 923</u>	<u>170</u>	<u>00'05.672 "</u>	
	Total		17 927	172	00'05.719"	
P3	50% L Sub	0.02	7 810	89	00'02.563 "	43
	PSSP	0.02	<u>58 718</u>	<u>341</u>	<u>00'18.970 "</u>	
	Total		66 528	430	00'21.533"	
	40% L Sub	0.02	1 808	44	00'00.625 "	43
	PSSP	0.02	<u>71 395</u>	<u>389</u>	<u>00'23.060 "</u>	
	Total		73 203	433	00'23.685"	
	30% L Sub	0.02	378	18	00'00.172 "	43
	PSSP	0.02	<u>75 151</u>	<u>396</u>	<u>00'23.780 "</u>	
	Total		75 529	414	00'23.952"	
	50% W Sub	0.02	1 473	44	00'00.641 "	43
	PSSP	0.02	<u>73 553</u>	<u>385</u>	<u>00'23.640 "</u>	
	Total		75 026	429	00'24.281"	
	40% W Sub	0.02	716	51	00'00.406 "	43
	PSSP	0.02	<u>74 419</u>	<u>392</u>	<u>00'25.970 "</u>	
	Total		75 135	443	00'26.376"	
	30% W Sub	0.02	42	11	00'00.406 "	43
	PSSP	0.02	<u>75 268</u>	<u>398</u>	<u>00'25.610 "</u>	
	Total		75 301	409	00'25.767"	

Problem	Algorithm	β	N	L	Processing Time	Trim Loss
P4	50% L Sub	0.02	5 394	76	00'01.984 "	31
	PSSP	0.02	<u>19 539</u>	<u>250</u>	<u>00'06.844 "</u>	
	Total		24 933	326	00'08.828"	
	40% L Sub	0.02	1 140	30	00'00.454 "	31
	PSSP	0.02	<u>24 252</u>	<u>270</u>	<u>00'08.437 "</u>	
	Total		25 392	300	00'08.891"	
	30% L Sub	0.02	172	11	00'00.094 "	31
	PSSP	0.02	<u>24 392</u>	<u>275</u>	<u>00'08.375 "</u>	
	Total		24 564	286	00'08.469"	
	50% W Sub	0.02	859	27	00'00.422 "	31
	PSSP	0.02	<u>22 511</u>	<u>256</u>	<u>00'07.797 "</u>	
	Total		23 370	283	00'08.219"	
P7	40% W Sub	0.02	1 213	97	00'00.578 "	31
	PSSP	0.02	<u>24 293</u>	<u>263</u>	<u>00'08.453 "</u>	
	Total		25 506	360	00'09.031"	
	30% W Sub	0.02	16	3	00'00.046 "	31
	PSSP	0.02	<u>25 369</u>	<u>279</u>	<u>00'08.750 "</u>	
	Total		25 385	282	00'08.796"	
	50% L Sub	0.01	2 186	41	00'00.796 "	8
	PSSP	0.01	<u>10 003</u>	<u>131</u>	<u>00'03.844 "</u>	
	Total		12 189	172	00'04.640"	
	40% L Sub	0.01	21	8	00'00.047 "	8
	PSSP	0.01	<u>10 047</u>	<u>138</u>	<u>00'03.610 "</u>	
	Total		10 068	146	00'03.657"	
	30% L Sub	0.01	12	5	00'00.047 "	8
	PSSP	0.01	<u>10 625</u>	<u>147</u>	<u>00'03.891 "</u>	
	Total		10 637	152	00'03.938"	
	50% W Sub	0.01	26	13	00'00.040 "	8
	PSSP	0.01	<u>10 133</u>	<u>139</u>	<u>00'03.719 "</u>	
	Total		10 159	152	00'03.759"	
	40% W Sub	0.01	4	4	00'00.032 "	8
	PSSP	0.01	<u>10 311</u>	<u>139</u>	<u>00'03.782 "</u>	
	Total		10 315	143	00'03.814"	
	30% W Sub	0.01	3	3	00'00.094 "	8
	PSSP	0.01	<u>10 627</u>	<u>144</u>	<u>00'03.891 "</u>	
	Total		10 630	147	00'03.985"	

Problem	Algorithm	β	N	L	Processing Time	Trim Loss
P8	50% L Sub	0.02	1 394	34	00'00.484 "	34
	PSSP	0.02	<u>12 383</u>	<u>135</u>	<u>00'04.297 "</u>	
	Total		13 777	169	00'04.781"	
	40% L Sub	0.02	22	5	00'00.094 "	34
	PSSP	0.02	<u>12 935</u>	<u>138</u>	<u>00'04.515 "</u>	
	Total		12 957	143	00'04.609"	
	30% L Sub	0.02	3	1	00'00.001 "	34
	PSSP	0.02	<u>12 935</u>	<u>138</u>	<u>00'04.406 "</u>	
	Total		12 938	139	00'04.407"	
	50% W Sub	0.02	270	16	00'00.141 "	34
	PSSP	0.02	<u>11 760</u>	<u>137</u>	<u>00'04.156 "</u>	
	Total		12 030	153	00'04.297"	
	40% W Sub	0.02	6	2	00'00.047 "	34
	PSSP	0.02	<u>12 016</u>	<u>134</u>	<u>00'04.109 "</u>	
	Total		12 022	136	00'04.156"	
	30% W Sub	0.02	8	4	00'00.047 "	34
	PSSP	0.02	<u>12 655</u>	<u>135</u>	<u>00'04.359 "</u>	
	Total		12 663	139	00'04.406"	

Table 7.6: Results for other partial areas to solve the sub-problem

The results in table 7.6 show that the 50% w Sub ($\mathcal{L},(0.5).(\mathcal{W})$) dimension performs well as well as the 40% w Sub ($\mathcal{L},(0.4).(\mathcal{W})$) dimension. It is therefore recommended that one of these two dimensions be used for the sub-problem dimensions when using the PSSP algorithm.

7.2.6 Normalized results

To summarize the results of sections 7.2.1 – 7.2.5, a table consisting of the original values for the number of generated nodes (N), the number of stored nodes (L) and the execution times for all problem instances (P) will be given in table 7.7. Table 7.8 gives normalized results using the AWA algorithm's results as the norm.

Problem	Algorithm	β	N	L	Processing Time	Trim Loss
P1	AWA	0.00	8 931	386	00'03.33"	0
	A*WA	0.00	8 931	386	00'03.33"	0
	SA*WA	0.00	8 852	382	00'03.30"	0
	AWAM	0.00	8 465	376	00'02.11"	0
	MAWAM	0.00	8 465	376	00'02.11"	0
	A*WAM	0.00	8 465	376	00'02.11"	0
	SA*WAM	0.00	8 388	372	00'02.01"	0
	PSSP	0.00	8 465	376	00'02.11"	0
P2	AWA	0.02	343 470	628	01'49.90"	29
	A*WA	0.02	140 449	534	00'45.80"	29
	SA*WA	0.02	139 348	530	00'45.63"	29
	AWAM	0.03	42 808	222	00'14.09"	29
	MAWAM	0.02	38 466	183	00'12.47"	29
	A*WAM	0.02	17 923	170	00'05.88"	29
	SA*WAM	0.02	17 823	169	00'05.83"	29
	PSSP	0.02	17 481	176	00'05.82"	29
P3	AWA	0.02	399 440	825	02'09.90"	43
	A*WA	0.02	400 349	1 055	02'11.20"	43
	SA*WA	0.02	388 310	1 031	02'09.22"	43
	AWAM	0.03	105 255	396	00'34.09"	43
	MAWAM	0.02	96 837	357	00'31.62"	43
	A*WAM	0.02	75 342	397	00'24.83"	43
	SA*WAM	0.02	73 589	391	00'23.92"	43
	PSSP	0.02	75 026	429	00'24.28"	43
P4	AWA	0.02	393 210	1 144	02'17.30"	31
	A*WA	0.02	131 184	963	00'43.53"	31
	SA*WA	0.02	125 987	889	00'40.67"	31
	AWAM	0.04	451 784	1 423	02'38.90"	31
	MAWAM	0.02	76 858	495	00'25.95"	31
	A*WAM	0.02	25 369	279	00'08.63"	31
	SA*WAM	0.02	25 132	273	00'08.23"	31
	PSSP	0.02	23 370	283	00'08.22"	31

Problem	Algorithm	β	N	L	Processing Time	Trim Loss
P5	AWA	0.00	273	34	00'00.18"	0
	A*WA	0.00	273	34	00'00.18"	0
	SA*WA	0.00	272	33	00'00.17"	0
	AWAM	0.00	273	34	00'00.18"	0
	MAWAM	0.00	273	34	00'00.18"	0
	A*WAM	0.00	273	34	00'00.18"	0
	SA*WAM	0.00	272	33	00'00.17"	0
	PSSP	0.00	273	34	00'00.18"	0
P6	AWA	0.00	15 828	389	00'05.44"	0
	A*WA	0.00	15 828	389	00'05.44"	0
	SA*WA	0.00	15 275	355	00'05.28"	0
	AWAM	0.00	12 226	311	00'04.33"	0
	MAWAM	0.00	13 549	337	00'04.58"	0
	A*WAM	0.00	13 549	337	00'04.58"	0
	SA*WAM	0.00	13 015	304	00'04.47"	0
	PSSP	0.00	13 549	337	00'04.58"	0
P7	AWA	0.01	85 625	352	00'29.48"	8
	A*WA	0.01	44 395	324	00'14.95"	8
	SA*WA	0.01	43 797	321	00'14.57"	8
	AWAM	0.01	9 402	125	00'03.52"	8
	MAWAM	0.01	19 020	159	00'06.52"	8
	A*WAM	0.01	10 627	144	00'03.81"	8
	SA*WAM	0.01	10 627	144	00'03.81"	8
	PSSP	0.01	10 159	142	00'03.76"	8
P8	AWA	0.02	171 873	446	00'59.12"	34
	A*WA	0.02	91 015	442	00'32.17"	34
	SA*WA	0.02	90 442	440	00'31.47"	34
	AWAM	0.02	6 961	81	00'02.56"	34
	MAWAM	0.02	23 161	144	00'07.97"	34
	A*WAM	0.02	12 935	138	00'04.63"	34
	SA*WAM	0.02	12 860	138	00'04.51"	34
	PSSP	0.02	12 030	153	00'04.30"	34

Table 7.7: Summary of results before normalization

Problem	Algorithm	β	N	L	Processing Time	Trim Loss
P1	AWA	0	1.000	1.000	1.000	0
	A*WA	0	1.000	1.000	1.000	0
	SA*WA	0	0.991	0.990	0.991	0
	AWAM	0	0.948	0.974	0.634	0
	MAWAM	0	0.948	0.974	0.634	0
	A*WAM	0	0.948	0.974	0.634	0
	SA*WAM	0	0.939	0.964	0.604	0
	PSSP	0	0.948	0.974	0.634	0
P2	AWA	0.02	1.000	1.000	1.000	29
	A*WA	0.02	0.409	0.850	0.417	29
	SA*WA	0.02	0.406	0.844	0.415	29
	AWAM	0.03	0.125	0.354	0.128	29
	MAWAM	0.02	0.112	0.291	0.113	29
	A*WAM	0.02	0.052	0.271	0.054	29
	SA*WAM	0.02	0.052	0.269	0.053	29
	PSSP	0.02	0.051	0.280	0.053	29
P3	AWA	0.02	1.000	1.000	1.000	43
	A*WA	0.02	1.002	1.279	1.010	43
	SA*WA	0.02	0.972	1.250	0.995	43
	AWAM	0.03	0.264	0.480	0.262	43
	MAWAM	0.02	0.242	0.433	0.243	43
	A*WAM	0.02	0.189	0.481	0.191	43
	SA*WAM	0.02	0.184	0.474	0.184	43
	PSSP	0.02	0.188	0.520	0.187	43
P4	AWA	0.02	1.000	1.000	1.000	31
	A*WA	0.02	0.334	0.842	0.317	31
	SA*WA	0.02	0.320	0.777	0.296	31
	AWAM	0.04	1.149	1.244	1.157	31
	MAWAM	0.02	0.195	0.433	0.189	31
	A*WAM	0.02	0.065	0.244	0.063	31
	SA*WAM	0.02	0.064	0.239	0.060	31
	PSSP	0.02	0.059	0.247	0.060	31

Problem	Algorithm	β	N	L	Processing Time	Trim Loss
P5	AWA	0	1.000	1.000	1.000	0
	A*WA	0	1.000	1.000	1.000	0
	SA*WA	0	0.996	0.971	0.944	0
	AWAM	0	1.000	1.000	1.000	0
	MAWAM	0	1.000	1.000	1.000	0
	A*WAM	0	1.000	1.000	1.000	0
	SA*WAM	0	0.996	0.971	0.944	0
	PSSP	0	1.000	1.000	1.000	0
P6	AWA	0	1.000	1.000	1.000	0
	A*WA	0	1.000	1.000	1.000	0
	SA*WA	0	0.965	0.913	0.971	0
	AWAM	0	0.772	0.799	0.796	0
	MAWAM	0	0.856	0.866	0.842	0
	A*WAM	0	0.856	0.866	0.842	0
	SA*WAM	0	0.822	0.781	0.822	0
	PSSP	0	0.856	0.866	0.842	0
P7	AWA	0.01	1.000	1.000	1.000	8
	A*WA	0.01	0.518	0.920	0.507	8
	SA*WA	0.01	0.511	0.912	0.494	8
	AWAM	0.01	0.110	0.355	0.119	8
	MAWAM	0.01	0.222	0.452	0.221	8
	A*WAM	0.01	0.124	0.409	0.129	8
	SA*WAM	0.01	0.124	0.409	0.129	8
	PSSP	0.01	0.119	0.403	0.128	8
P8	AWA	0.02	1.000	1.000	1.000	34
	A*WA	0.02	0.530	0.991	0.544	34
	SA*WA	0.02	0.526	0.987	0.532	34
	AWAM	0.02	0.041	0.182	0.043	34
	MAWAM	0.02	0.135	0.323	0.135	34
	A*WAM	0.02	0.075	0.309	0.078	34
	SA*WAM	0.02	0.075	0.309	0.076	34
	PSSP	0.02	0.070	0.343	0.073	34

Table 7.8: Normalized results using the AWA algorithm as the norm

The original values are then converted to normalized values, using the AWA algorithm's results as the norm. The formula to calculate the normalized values are given below:

$$\text{normalized value} = \text{Any algorithm's value} / \text{AWA value}$$

The normalized values are given in table 7.8. The normalized results show that the PSSP, A*WAM and SA*WAM algorithms are three of the best-performing algorithms. It should be noted that symmetrical duplicate removal could also be added to the PSSP algorithm, which will effectively enhance the PSSP algorithm even more.

7.2.7 Increasing the beta (β) value

In chapter 6, section 6.5.2, pages 133-136, a fractional increase value for beta was calculated. This value will now be used in the solving process to modify the value of beta whenever an optimal solution is not found with a beta value of 0.00. Table 7.9 summarizes the results found when using the increase fraction with the AWA algorithm. In the same table, the results obtained by solving the problems with a constant beta increase value of 0.01 are also displayed.

It should be noted that for this example the AWA algorithm was used, but it could be substituted with any one of the other algorithms that were tested in sections 7.2.1 to 7.2.5.

Problem	Algorithm	β	N	L	Processing Time	Trim Loss
P1	AWA	0.0000	8 931	386	00'03.33 "	0
P2	AWA(Constant beta increase)	0.0000	12 014	98	00'03.19 "	520
		0.0100	65 106	258	00'18.81 "	379
		0.0200	<u>343 470</u>	<u>628</u>	<u>01'49.90 "</u>	29
			420 590	984	02'11.90"	
	AWA(Fractional beta increase)	0.0000	12 014	98	00'03.19 "	520
		0.0105	<u>68 185</u>	<u>265</u>	<u>00'20.33 "</u>	29
			80 199	363	00'23.52"	
P3	AWA(Constant beta increase)	0.0000	7 375	73	00'01.60 "	748
		0.0100	72 012	323	00'22.12 "	421
		0.0200	<u>399 440</u>	<u>825</u>	<u>02'09.90 "</u>	43
			478 827	1 221	02'33.62"	
	AWA(Fractional beta increase)	0.0000	7 375	73	00'01.60 "	748
		0.0125	133 726	477	00'45.41 "	311
		0.0208	<u>452 566</u>	<u>866</u>	<u>02'20.13 "</u>	43
			593 667	1 416	03'07.14"	
P4	AWA(Constant beta increase)	0.0000	9 469	89	00'03.45 "	280
		0.0100	58 647	355	00'18.24 "	280
		0.0200	<u>393 210</u>	<u>1 144</u>	<u>02'17.30 "</u>	31
			461 326	1 588	02'38.99"	
	AWA(Fractional beta increase)	0.0000	9 469	89	00'03.45 "	280
		0.0057	15 140	123	00'04.67 "	280
		0.0113	<u>95 601</u>	<u>478</u>	<u>00'29.45 "</u>	31
			120 210	690	00'37.62"	

Problem	Algorithm	β	N	L	Processing Time	Trim Loss
P5	AWA	0.0000	273	34	00'00.18 "	0
P6	AWA	0.0000	15 828	389	00'05.44 "	0
P7	AWA(Constant beta increase)	0.0000	23 556	199	00'07.78 "	288
		0.0100	<u>85 625</u>	<u>352</u>	<u>00'29.48 "</u>	8
			109 181	551	00'37.26"	
	AWA(Fractional beta increase)	0.0000	23 556	199	00'07.78 "	288
		0.0109	<u>110 400</u>	<u>393</u>	<u>00'35.45 "</u>	8
			133 956	592	00'43.23"	
P8	AWA(Constant beta increase)	0.0000	6 918	71	00'07.39 "	1 100
		0.0100	29 807	169	00'10.21 "	683
		0.0200	<u>171 873</u>	<u>446</u>	<u>00'59.12 "</u>	34
			208 598	686	01'16.72"	
	AWA(Fractional beta increase)	0.0000	6 918	71	00'07.39 "	1 100
		0.0227	<u>217 432</u>	<u>509</u>	<u>01'18.36 "</u>	34
			224 350	580	01'25.75"	

Table 7.9: Constant versus fractional beta (β) value increases

The results in table 7.9 show that in some cases the constant increase of the beta value performs better, and in other cases the fractional increase fares better. A possible reason for this is that the beta values required to find the optimal solutions for these problem instances are small. As is mentioned in chapter 6, section 6.5.2, pages 133-136, if a problem requires a large beta value to generate the optimal solution, a lot of unnecessary work might be done while the value of beta is gradually increased with a constant value of 0.01 (or some other arbitrary value). This method is therefore a good alternative to a constant increase in the value of beta.

7.2.8 Industry-sized problem instances

Throughout the thesis, reference has been made to the eight C2DGC problems presented in chapter 5, section 5.1, table 5.1, page 56. These problem instances are all small textbook-sized problems that are easy to solve and are useful for research purposes since many authors refer to them in papers. (Daza et al., 1995, Christofides and Whitlock (1977)). These problems, however, do not show whether the Wang and modified Wang methods scale well when used to solve large industry-sized problem instances. This last section in the numerical test chapter provides some results obtained from solving these larger problem instances with the Wang and modified Wang methods. The problem instances were obtained from a large local corporation (PGGlass Pty. Ltd.) that cuts and sells glass sheets. Table 7.10 summarizes these problem instances.

Problem	Stock plate length (L) and width (W)	Demand rectangles' length (l), width (w) and upper bound (b)
PG1	(2000,2800)	(290,1440,3); (585,955,1); (925,560,17); (950,290,12) (956,1195,2); (1440,1195,5); (1490,1440,1)
PG2	(2550,3210)	(1130,1150,108); (894,1130,162); (889,1264,108) (979,1332,108); (1064,1086,108); (804,1264,108) (753,1330,54)
PG3	(1500,2125)	(290,1440,3); (585,955,1); (925,560,17); (950,290,12) (956,1195,2); (1140,1195,5); (1490,1440,1)
PG4	(1000,1500)	(290,129,20); (585,355,4); (925,560,17); (950,290,12) (555,395,2); (650,796,5); (200,324,10)

Table 7.10: Set of four C2DGC problem instances from PGGlass Pty. Ltd.

These problem instances were solved with the algorithm based on Wang's first algorithm with Vasko's improvements (AWA) as well as with the algorithm based on the modified Wang method with Daza's heuristic function

(MAWAM). The results obtained from these numerical tests are summarized in table 7.11.

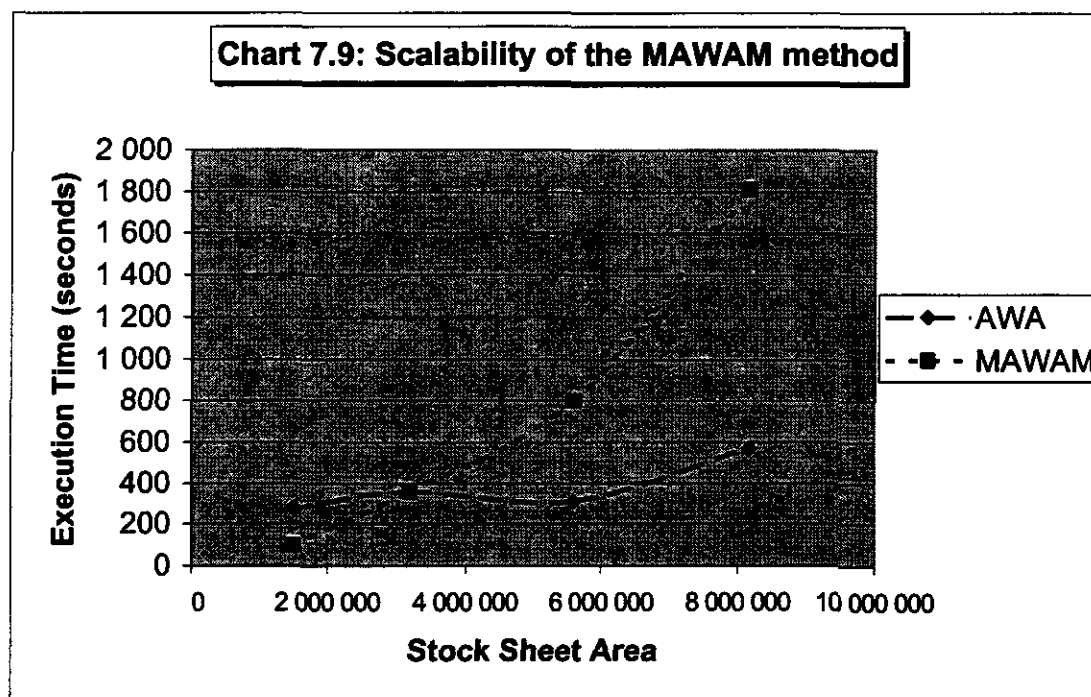
Problem	Algorithm	β	N	L	Processing Time	Trim Loss
PG1	AWA	0.00	5 282	63	00'01.203 "	1 928 960
		0.01	129 130	361	00'29.190 "	938 000
		0.02	<u>1 330 528</u>	<u>1 232</u>	<u>04'40.600 "</u>	96 525
			1 464 940	1 656	05'10.993"	
	TOTAL					
	MAWAM Table MAWAM				13'18.400 "	
		0.00	514	18	00'00.141 "	2 619 200
		0.01	1 274	32	00'00.343 "	2 102 225
		0.02	<u>15 125</u>	<u>122</u>	<u>00'03.547 "</u>	96 525
			16 913	172	13'22.431"	
	TOTAL					
PG2	AWA	0.00	1 963	35	00'00.050 "	4 144 620
		0.01	10 060	81	00'02.390 "	2 087 964
		0.02	61 579	215	00'13.910 "	2 087 964
		0.03	215 833	396	00'48.160 "	2 087 964
		0.04	893 804	903	03'17.000 "	327 586
		0.05	<u>1 356 179</u>	<u>1 002</u>	<u>05'06.400 "</u>	327 586
	TOTAL					
	MAWAM Table MAWAM				29'52.000 "	
		0.00	502	17	00'00.156 "	4 144 620
		0.01	645	20	00'00.218 "	4 144 620
		0.02	783	22	00'00.219 "	4 144 620
		0.03	1 089	26	00'00.281 "	4 144 620
		0.04	17 249	221	00'04.094 "	327 586
		0.05	<u>75 535</u>	<u>331</u>	<u>00'16.880 "</u>	327 586
			95 803	637	30'13.848"	
	TOTAL					

Problem	Algorithm	β	N	L	Processing Time	Trim Loss
PG3	AWA	0.00	916	25	00'00.218 "	1 466 700
		0.01	4 593	54	00'01.063 "	624 300
		0.02	11 546	96	00'02.640 "	624 300
		0.03	41 148	219	00'09.250 "	517 975
		0.04	108 146	403	00'24.440 "	517 975
		0.05	185 157	554	00'42.060 "	517 975
		0.06	251 763	635	00'57.060 "	517 975
		0.07	383 881	816	01'28.230 "	289 200
		0.08	<u>607 608</u>	<u>1 089</u>	<u>02'12.300 "</u>	242 300
	TOTAL		1 594 758	3 891	05'57.261"	
	MAWAM Table MAWAM				05'04.300 "	
		0.00	241	12	00'00.062 "	1 466 700
		0.01	335	14	00'00.082 "	1 041 900
		0.02	335	14	00'00.082 "	1 041 900
		0.03	496	17	00'00.109 "	1 041 900
		0.04	1 654	44	00'00.406 "	517 975
		0.05	12 901	160	00'02.953 "	517 975
		0.06	34 922	286	00'07.921 "	517 975
		0.07	63 928	425	00'15.880 "	289 200
		0.08	<u>114 403</u>	<u>543</u>	<u>00'25.780 "</u>	242 300
	TOTAL		229 215	1 515	05'57.575"	
PG4	AWA	0.00	18 147	118	00'03.890 "	336 810
		0.01	<u>1 227 340</u>	<u>2 584</u>	<u>04'32.400 "</u>	12 900
	TOTAL		1 245 487	2 702	04'36.290"	
	MAWAM Table MAWAM				01'25.000 "	
		0.00	425	16	00'00.078 "	982 000
	TOTAL	0.01	<u>62 234</u>	<u>1 040</u>	<u>00'13.800 "</u>	12 900
			62 659	1 056	01'38.878"	

Table 7.11: Numerical results for larger problem instances

When studying the results in table 7.11, it becomes obvious that the MAWAM (modified Wang method) does not fare well for problem instances

with large stock sheets. The reason for this is that if, for instance, a lookup table of underestimates must to be calculated for problem **PG2**, one underestimates for each dimension within the stock sheet must be calculated. This translates to a staggering $2550 \times 3210 = 8\,185\,500$ underestimates! Table 7.11 shows that to compute these values for **PG2**, approximately 1792 seconds (29'52.00") of processing time is required. To completely solve problem **PG2** with the AWA method, requires but 568.36 seconds (09'28.36"). This shows that the MAWAM method does not scale well and its performance will deteriorate even further for larger stock sheets. It should be noted, though, that when the underestimates have been calculated, the solving process completes very fast with the MAWAM algorithm (much faster than with the AWA algorithm). Therefore, if a method could be devised to calculate underestimates quicker, the MAWAM method might scale better. Chart 7.9 shows how the MAWAM algorithm's performance gets worse as the size of the stock sheet increases.



7.2.8.1 PSSP algorithm with initial underestimates of zero

As shown in chart 7.9, the modified Wang method's (MAWAM) lookup table is calculated at a great computational cost for problem instances with large stock sheets. This becomes such a bother that the original Wang method could rather be used for such problem instances. Although the original Wang method fares better for such problems than the modified Wang method, it still requires a great deal of processing time to reach an optimal solution. A possible solution to this problem is to use the PSSP algorithm to solve these instances. However, instead of utilizing a two-dimensional knapsack function (Gilmore & Gomory) to calculate initial underestimates for the MAWAM method, set all the values in the lookup table initially to zero. This then eliminates the cost incurred by calculating underestimates, and with the PSSP algorithm, builds found by the original Wang method will be used as underestimate and also be propagated further.

Table 7.12 displays the results obtained by solving the four larger, industry sized problems (**PG1 – PG4**) with the AWA, MAWAM and PSSP (with initial underestimates of zero) algorithms:

Problem	Algorithm	β	N	L	Processing Time	Trim Loss
PG1	AWA	0.00	5 282	63	00'01.203 "	1 928 960
		0.01	129 130	361	00'29.190 "	938 000
		0.02	<u>1 330 528</u>	<u>1 232</u>	<u>04'40.600 "</u>	96 525
			1 464 940	1 656	05'10.993"	
	MAWAM Table MAWAM	0.00	514	18	13'18.400 "	2 619 200
		0.01	1 274	32	00'00.141 "	2 102 225
		0.02	<u>15 125</u>	<u>122</u>	<u>00'03.547 "</u>	96 525
			16 913	172	13'22.431"	
	PSSP	0.00	1 545	45	00'00.466 "	3 454 400
		0.01	70 005	299	00'20.188 "	938 000
		0.02	<u>306 536</u>	<u>659</u>	<u>01'30.685 "</u>	96 525
			378 086	1 003	01'51.339	
PG2	AWA	0.00	1 963	35	00'00.050 "	4 144 620
		0.01	10 060	81	00'02.390 "	2 087 964
		0.02	61 579	215	00'13.910 "	2 087 964
		0.03	215 833	396	00'48.160 "	2 087 964
		0.04	893 804	903	03'17.000 "	327 586
		0.05	<u>1 356 179</u>	<u>1 002</u>	<u>05'06.400 "</u>	327 586
	TOTAL		2 539 418	2 632	09'28.360"	
	MAWAM Table MAWAM	0.00	502	17	29'52.000 "	4 144 620
		0.01	645	20	00'00.156 "	4 144 620
		0.02	783	22	00'00.219 "	4 144 620
		0.03	1 089	26	00'00.281 "	4 144 620
		0.04	17 249	221	00'04.094 "	327 586
		0.05	<u>75 535</u>	<u>331</u>	<u>00'16.880 "</u>	327 586
	TOTAL		95 803	637	30'13.848"	
	PSSP	0.00	694	23	00'00.225 "	5 938 108
		0.01	7 042	72	00'03.176 "	2 141 028
		0.02	19 935	130	00'07.354 "	2 087 964
		0.03	58 800	221	00'19.719 "	2 087 964
		0.04	147 292	388	00'48.543 "	327 586
		0.05	<u>292 223</u>	<u>577</u>	<u>01'42.122 "</u>	327 586
	TOTAL		525 986	1 411	02'21.139"	

Problem	Algorithm	β	N	L	Processing Time	Trim Loss
PG3	AWA	0.00	916	25	00'00.218 "	1 466 700
		0.01	4 593	54	00'01.063 "	624 300
		0.02	11 546	96	00'02.640 "	624 300
		0.03	41 148	219	00'09.250 "	517 975
		0.04	108 146	403	00'24.440 "	517 975
		0.05	185 157	554	00'42.060 "	517 975
		0.06	251 763	635	00'57.060 "	517 975
		0.07	383 881	816	01'28.230 "	289 200
		0.08	607 608	1 089	02'12.300 "	242 300
	TOTAL		1 594 758	3 891	05'57.261"	
	MAWAM Table MAWAM	0.00	241	12	05'04.300 "	
		0.01	335	14	00'00.062 "	1 466 700
		0.02	335	14	00'00.082 "	1041 900
		0.03	496	17	00'00.109 "	1041 900
		0.04	1 654	44	00'00.406 "	517 975
		0.05	12 901	160	00'02.953 "	517 975
		0.06	34 922	286	00'07.921 "	517 975
		0.07	63 928	425	00'15.880 "	289 200
		0.08	114 403	543	00'25.780 "	242 300
	TOTAL		229 215	1 515	05'57.575"	
	PSSP	0.00	427	21	00'00.148 "	1 466 700
		0.01	3 311	50	00'01.000 "	1 041 900
		0.02	7 290	81	00'02.053 "	624 300
		0.03	14 918	226	00'05.157 "	624 300
		0.04	33 244	198	00'10.825 "	517 975
		0.05	76 882	354	00'20.053 "	517 975
		0.06	129 141	470	00'31.122 "	517 975
		0.07	205 015	642	00'50.142 "	289 200
		0.08	254 168	709	01'37.058 "	242 300
	TOTAL		724 396	2 751	02'57.558"	
PG4	AWA	0.00	18 147	118	00'03.890 "	336 810
		0.01	1 227 340	2 584	04'32.400 "	12 900
	TOTAL		1 245 487	2 702	04'36.290"	
	MAWAM Table MAWAM	0.00	425	16	01'25.000 "	
		0.01	62 234	1 040	00'00.078 "	982 000
	TOTAL		62 659	1 056	00'13.800 "	12 900
					01'38.878"	
	PSSP	0.00	9 340	117	00'01.423 "	
		0.01	480 291	1 804	01'11.015 "	12 900
	TOTAL		489 631	1 921	01'12.438"	

Table 7.12: PSSP algorithm with initial underestimates of zero

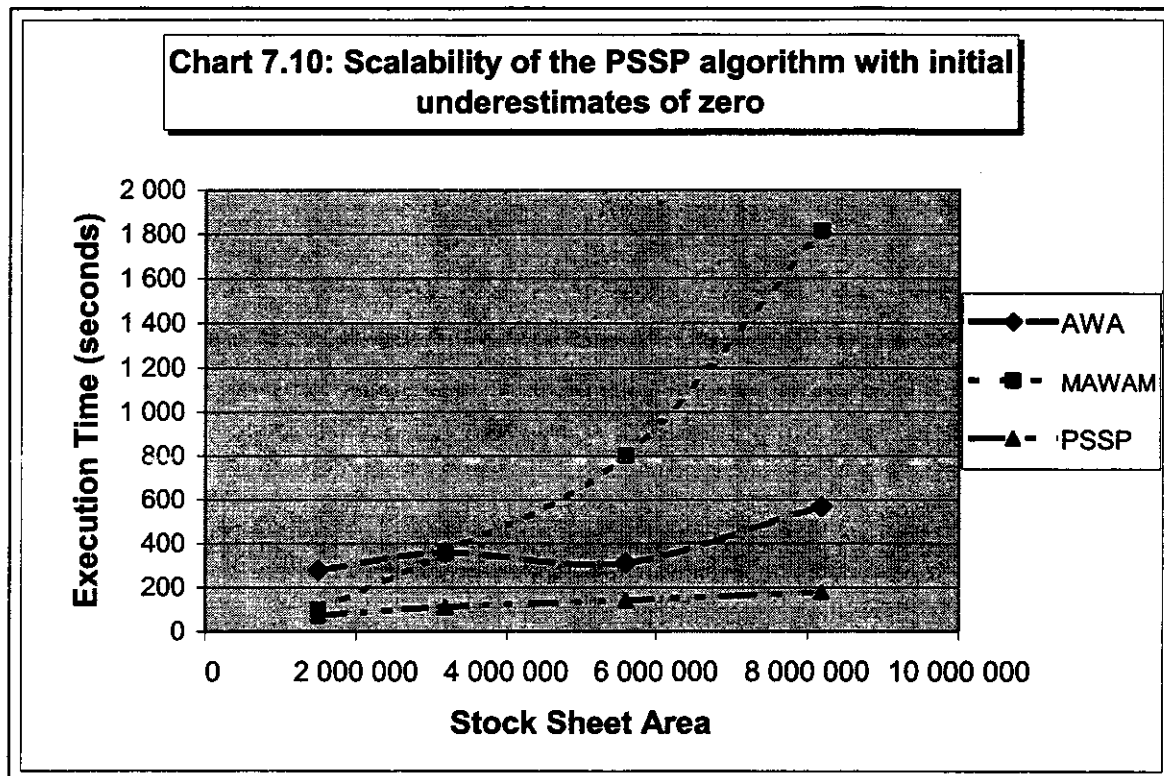


Chart 7.10 shows that the PSSP algorithm with initial underestimates of zero does indeed scale well for problem instances where larger stock sheets are concerned.

7.3 Summary

Chapter 7 was an exciting path of discovery, and it showed that the theoretical foundations of artificial intelligence search methods are well laid. Furthermore, the new PSSP algorithm as proposed by the author was implemented and the results obtained from solving sample problems with it seem promising, especially for larger, industry-sized problem instances. Increasing the value of beta by an arbitrary value and the scalability of the modified Wang method were also investigated.

Chapter 8 concludes the thesis by giving a short summary of how the objectives set for the study were reached and it also discusses the possibilities that exist for further study.

CHAPTER 8: Conclusion

8.1 Introduction

The research within this thesis concentrated on the investigation of exact methods and algorithms that are capable of solving C2DGC problem instances. Firstly, a theoretical study was undertaken concerning artificial intelligence search methods. Secondly, existing exact problem solving approaches for C2DGC problems were examined, with emphasis on methods proposed by Wang (1983), Vasko (1989), Oliveira and Ferreira (1990) and Daza et al (1995). These existing methods were then combined with different artificial intelligence search methods.

Through a theoretical study and empirical tests it was found that the Wang method is effective in solving C2DGC problem instances, but usually it does a great deal of unnecessary work and generates redundant patterns if the choice and management of the beta (β) value are not handled with care. Even with a meticulously conceived strategy for handling the beta value and implemented improvements made by Vasko to the original Wang method, the Wang method is still not efficient in its calculation of solutions.

Oliveira and Ferreira (1990) conceived a method that they called the modified Wang method (WAM), which utilised underestimates to guide the search process. These underestimates were based on work done by Gilmore and Gomory (1966), where an unbounded knapsack problem is solved using a dynamic programming procedure, resulting in underestimates to be used by the WAM method. By solving small-sized textbook problems it was shown that the WAM method was very effective and efficient in solving C2DGC problems as opposed to the Wang method.

Algorithmic enhancements undertaken in the thesis included a method that strived to find more informed heuristic functions for the WAM method (PSSP algorithm). This resulted in a more efficient WAM algorithm as the search

space was pruned with more accuracy. However, further empirical work on a system developed for this study using industry-sized problem instances exposed a weakness in the WAM and PSSP methods. As the stock sheet size became larger, the number of underestimates that needed to be calculated by the Gilmore and Gomory method became unrealistically high. As a result the calculation of the underestimate lookup table turned into a daunting task, translating into a method that did not scale well as problem sizes increased. For this reason, the PSSP algorithm was enhanced, eliminating the need for the lookup table calculation by starting with initial underestimates of 0. This algorithm was tested against the Wang (WA) and modified Wang (WAM) algorithms and it was showed that the PSSP algorithm using underestimates of 0 scaled well as the problem sizes increased.

Further algorithmic enhancements included the definition of a waste gap as well as methods to determine the waste gap by using upper bounds. The first method used data as generated by the Wang method in the rectangle building process as the problem was solved. This data was then refined through a propagation process to determine an upper bound for the waste gap. Secondly, a beam search algorithm was used to generate a solution quickly (which was not necessarily exact but acceptable) for a C2DGC problem instance and then use that solution as an upper bound for the waste gap.

Lastly, generating initial values for beta and handling these values as searches were undertaken with the algorithms based on the Wang and modified Wang methods, were undertaken. Firstly, a method to calculate initial beta values using the table of underestimates as generated for the WAM method was devised. Secondly, a fractional increase in the value of beta was proposed as a search was done, as opposed to a steady increase of for example 0.01 in the value of beta.

At the beginning of the thesis, in chapter one, five main objectives for the study were identified. These objectives form the foundation of the study and in section two of this chapter, a summary of how these objectives were reached,

is given. Section three highlights new research, new ideas and new concepts that were introduced in the thesis. The fourth section deals with new research opportunities that presented themselves during the course of the study.

8.2 Objectives of the study

The objectives can be summarised as follows:

- Gaining an understanding of what artificial intelligence search methods are and how they function;
- Gaining an understanding of what the C2DGC problem models and algorithms entail;
- Developing algorithms that solve C2DGC problems;
- Investigate the effectiveness and efficiency of these algorithms; and
- To develop an integrated software package implementing these algorithms.

Each individual objective will now be discussed to show how it was reached.

8.2.1 Gaining an understanding of what artificial intelligence search methods are and how they function

This objective was achieved by firstly defining the concept artificial intelligence (chapter 2, section 2.1.1) as well as search methods (chapter 2, section 2.2.2). Chapters 3 and 4 dealt exclusively with search methods. Chapter 3 discussed uninformed artificial intelligence search methods and chapter 4 discussed informed artificial intelligence search methods. These uninformed and informed search methods were then individually analysed and evaluated. The effectiveness and efficiency of each method was measured in terms of four criteria, namely:

- *Completeness*: is the method guaranteed to find a solution when one exists for the problem?

- *Time complexity*: how long does it take to find a solution?
- *Space complexity*: how much memory does it need to perform the search?
- *Exactness*: does the strategy find the highest-quality solution when there are several different solutions?

Studying this material provided us with a better understanding of the functioning of artificial intelligence search methods.

8.2.2 Gaining an understanding of what the C2DGC problem models and algorithms entail

This objective was reached by firstly giving a theoretical definition (chapter 2, section 2.2.3) as well as a mathematical formulation (chapter 2, section 2.2.3.1) of the problem. Furthermore, research efforts related to the C2DGC problem were listed in chapter 2, section 2.2.3.2 as well as a discussion on previous research done on the C2DGC problem (chapter 2, section 2.3).

Throughout chapters 3 and 4, where the different artificial intelligence search methods were discussed, reference was made to the effectiveness and efficiency of those algorithms when solving C2DGC problems.

8.2.3 Developing algorithms that solve C2DGC problems

The process of algorithm development was started in chapter 5, sections 5.2 and 5.3, where different approaches were identified by which the C2DGC problem might be solved. An appropriate approach was chosen, namely the Wang method and section 5.2.1 dealt with this method. Concepts connected to this method were also discussed, such as trim loss (chapter 5, section 5.2.1.2) and acceptable waste percentages (chapter 5, section 5.2.1.3). Furthermore, the modified Wang method was also discussed as a possible method from which algorithms could be derived. The PSSP algorithm that was developed by the author, was introduced in chapter 6 (section 6.3.1).

8.2.4 Investigate the effectiveness and efficiency of these algorithms

Chapter 7 accomplished this task in that the implemented algorithms were used to solve a set of 8 C2DGC problem instances. Daza (Daza et al, 1995:642) presented these problem instances in their article and by solving these problems data was recorded. Tables and graphs were derived from the data, which showed the effectiveness and efficiency of the derived algorithms, and it also highlighted which algorithms perform better than others.

8.2.5 To develop an integrated software package implementing these algorithms

The fourth objective could only have been achieved by doing empirical work using some form of computer program to solve the given problem instances. The fifth objective aims at transforming the computer program that had to be written into an integrated piece of software that is both useful and professionally engineered. This software can be downloaded from the Internet along with a complete user manual at the following URL:

<http://www.puk.ac.za/studentelewe/scientiae/itweb/1st/1st.htm>

8.3 New research

New concepts have been introduced in the thesis. These include:

- The PSSP (Partial Stock Sheet Propagation) algorithm was developed that updates the modified Wang lookup table to provide better underestimates of internal trim loss. It was proven that the PSSP algorithm using the updated WAM lookup table will always search through a smaller space than the standard WAM method (chapter 6, section 6.3.1);

- A beam search algorithm was developed, implemented and tested. Tests revealed that it was difficult to manage the beam width and the beta parameters simultaneously resulting in a search method that generates mostly non-exact solutions. Therefore, the beam search was used to generate upper bounds in the definition of the waste gap, as discussed in chapter 6 (chapter 4, section 4.2.4 and chapter 6, section 6.4.2);
- A new method was devised to calculate an initial lower bound on the value of beta by utilizing the last entry in the modified Wang lookup table. This method is simple, yet very effective, and requires very little extra processing time (chapter 6, section 6.5.1);
- An alternative method to fractionally increase the value of beta was devised (chapter 6, section 6.5.2);
- A new strategy to handle the beta value was presented that manages the waste gap and helps in determining the next beta value (chapter 6, section 6.4); and
- Finally, experiments were conducted with some industry-sized problems that exposed weaknesses in the WAM method. By altering the PSSP algorithm slightly in order for it to use initial underestimates of 0, the need to calculate a lookup table was eliminated. Then these underestimates were updated with data obtained by solving a portion of the original problem instance with the Wang method. This was shown to be an effective and efficient substitution for underestimates as calculated by the Gilmore and Gomory two-dimensional knapsack function. In fact, the PSSP method with initial underestimates of 0 scaled very well for larger problem instances as compared to the WAM method.

8.4 Further research

It has been established, through the numerical tests done in chapter 7, that the PSSP algorithm is indeed a plausible algorithm with which C2DGC problems can be solved, and therefore further research will center on ways to

refine the search. In addition, new algorithms have been developed which adds new data and facts to the scientific community's pool of knowledge.

Furthermore, a better heuristic function could be derived to calculate a more accurate value for h . Concepts introduced by Daza et al are a good starting point and refinements to h should also improve on the algorithms' execution time as well as the number of generated and stored nodes.

Bibliography

Bagchi, A. Mahanti, A. 1983. Search algorithms under different kinds of heuristics: A comparative study. Journal of the ACM, 30: 1-21.

Baker, B.M. 1999. A spreadsheet modelling approach to the assortment problem. European Journal of Operational Research, 114: 83-92.

Barr, A. Feigenbaum, E.A. 1981. The handbook of artificial intelligence. Reading, Massachusetts. Addison-Wesley Publishing Company Inc.

Beasley, J.E. 1985. Algorithms for unconstrained two-dimensional guillotine cutting. Journal of the Operational Research Society, 36: 297-306.

Bortfeldt, A. Gehring, H. 2001. A hybrid genetic algorithm for the container loading problem. European Journal of Operational Research, 131: 143-161.

Bundy, A. 1997. Artificial Intelligence techniques: a comprehensive catalogue. Fourth, revised edition. Springer-Verlag Berlin Heidelberg, Berlin.

Chao, H. Harper, M.P. Quong, R.W. 1995. A tight lower bound for optimal bin packing. Operations Research Letters, 18: 133-138.

Charniac, E. McDermott, D. 1985. Introduction to artificial intelligence. Addison-Wesley, Reading, Massachusetts.

Christofides, N. Hadjiconstantinou, E. 1995. An exact algorithm for orthogonal 2-D cutting problems using guillotine cuts. European Journal of Operational Research, 83(1): 21-38, 18 May.

Christofides, N. Whitlock, C. 1977. An algorithm for two-dimensional cutting problems. Operations Research, 25(1): 30-44.

Ciesielski, V. 2001. Artificial Intelligence Search. Department of Computer Science, RMIT University.

[Available on the Internet:]

<http://www.cs.rmit.edu.au/AI-Search/>

Cung, V. Hifi, M. Le Cun, B. 2000. Constrained two-dimensional cutting stock problems: a best-first branch-and-bound algorithm. International Transactions in Operational Research, 7(3): 185-210, 1 May.

Daza, V.P. De Alvarenga, A.G. De Diego, J. 1995. Exact solutions for constrained two-dimensional cutting problems. European Journal of Operations Research, 84(3): 633-644, 3 August.

Eppen, G.D. Gould, F.J. Schmidt, C.P. Moore, J.H. Weatherford, L.R. 1998. Introductory management science. Fifth, International Edition. Prentice Hall Inc., Upper Saddle River, New Jersey.

Fayard, D. Zissomopoulos, V. 1995. An approximation algorithm for solving unconstrained two-dimensional knapsack problems. European Journal of Operational Research, 84(3): 618-632, August.

Gau, T. Wäscher, G. 1995. CUTGEN1: A problem generator for the Standard One-Dimensional Cutting Stock Problem. European Journal of Operational Research, 84(3): 572-579, August.

Gilmore, P.C. Gomory, R.E. 1966. The theory and computation of knapsack functions. Operations Research, 15:1045-1075.

Haugeland, J., editor. 1985. Artificial intelligence: The very idea. MIT Press, Cambridge, Massachusetts.

Heckmann, R. Lengauer, T. 1998. Computing closely matching upper and lower bounds on textile nesting problems. European Journal of Operational Research, 108: 473-489.

Held, M. Karp, R.M. 1971. The travelling salesman problem and minimal spanning trees.: Part II. Mathematical Programming, 1: 6-25.

Herz, J.C. 1972. A recursive computing procedure for two-dimensional stock cutting. IBM Journal of Research and Development, 16:462-469.

Hifi, M. 1994. Study of some combinatorial optimisation problems: cutting stock, packing and set covering problems. PhD thesis, University of Paris, 1 Pantheon-Sorbonne.

Hifi, M. 1997. An improvement of Viswanathan and Bagchi's exact algorithm for constrained two-dimensional cutting stock. Computers & Operations Research, 24(8): 727-736.

Hinxman, A.I. 1976. Problem reduction and the two-dimensional trim-loss problem. Artificial Intelligence and Simulation: Summer Conference, University of Edinburgh, 158-165.

Kendall, G. 2000. Applying Meta-Heuristic Algorithms to the Nesting Problem Utilising the No Fit Polygon. Nottingham: University of Nottingham. (Thesis – D.Phil.) 242 p.

Korf, R.E. 1996. Artificial Intelligence Search Algorithms. Computer Science Department, University of California, Los Angeles.

[Available on the Internet:]

<http://citeseer.nj.nec.com/92777.html>

Kurzweil, R. 1990. The age of intelligent machines. MIT Press, Cambridge, Massachusetts.

Li, H. Tsai, J. 2001. A fast algorithm for assortment optimisation problems. Computers & Operations Research, 28: 1245-1252.

Luger, G.F. Stubblefield, W.A. 1991. Artificial Intelligence: structures and strategies for complex problem solving. Second edition. The Benjamin/Cummings Publishing Company Inc, Redwood City, California.

Morabito, R.N. Arenales, M.N. Arcaro, V.F. 1992. An And-Or-graph approach for two-dimensional cutting problems. European Journal of Operational Research, 58:263-271.

Morabito, R. Garcia, V. 1998. The cutting stock problem in a hardboard industry: a case study. Computers & Operations Research, 25(6): 469-485, June.

Nilsson, N. 1980. Principles of artificial intelligence. Springer-Verlag, Berlin.

Oliveira, J.F. Ferreira, J.S. 1990. An improved version of Wang's algorithm for two-dimensional cutting problems. European Journal of Operational Research, 44: 256-266.

Pearl, J. 1984. Heuristics. Addison Wesley, New York.

Polya, G. 1945. How to solve it. Princeton: Princeton University Press.

Preiss, B.R. 1999. Data structures and algorithms with object-oriented design patterns in Java. John Wiley & Sons, 605 Third Avenue, New York.

Rich, E. Knight, K. 1991. Artificial Intelligence. International Edition. McGraw-Hill Book Co, Singapore.

Russel, S. Norvig, P. 1995. Artificial Intelligence: a modern approach. Prentice Hall, New Jersey.

Schalkoff, R.J. 1990. Artificial intelligence: An engineering approach. McGraw-Hill, New York.

Sen, A. Bagchi, A. 1989. Fast recursive formulations for best-first search that allow controlled use of memory. Proceedings of the IJCAI-89. International Joint Conference on Artificial Intelligence, 297-302.

Vasko, F.J. 1989. A computational improvement to Wang's two-dimensional cutting stock algorithm. Computers and industrial engineering, 16: 109-115.

Viswanathan, K.V. Bagchi, A. 1993. Best-first search methods for constrained two-dimensional cutting stock problems. Operations Research, 41(4): 768-776, July-August.

Wang, P.Y. 1983. Two algorithms for Constrained Two-Dimensional Cutting Stock Problems. Operations Research, 31(3): 573-586, May-June.

Winston, P.H. 1977. Artificial Intelligence. Addison-Wesley Publishing Company, Inc. Philippines.

Winston, P.H. 1992. Artificial Intelligence: Third Edition. Addison-Wesley Publishing Company, Reading, Massachusetts.

Yuret, D. De la Maza, M. 1993. Dynamic Hill Climbing: Overcoming the limitations of optimisation techniques. Massachusetts Institute of Technology, Cambridge.

[Available on the Internet:]

<http://citeseer.nj.nec.com/yuret93dynamic.html>

Zissimopoulos, V. 1985. Heuristic methods for solving (un)constrained two-dimensional cutting stock problems. Methods of Operations Research, 49: 345-357.

