

https://dspace.org/

A fully three-dimensional heuristic algorithm for container packing

Aspoas, Sean Graham

2016
http://hdl.handle.net/10539/21336

Downloaded from a DSpace repository. https://dspace.org/

A FULLY THREE-DIMENSIONAL HEURISTIC ALGORITHM FOR CONTAINER PACKING

Sean Graham Aspoas

A research report submitted to the Faculty of Science, University of the Witwatersrand, Johannesburg, in partial fulfilment of the requirements for the degree of Master of Science.

Johannesburg, 1996

Degree awarded with distinction on 4 December 1996.

Declaration

I declare that this research report is my own work. It is being submitted for the Degree of Master of Science in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other University.

Sean Graham Aspoas

The 22nd day of May 1996.

Declaration

I declare that this research report is my own work. It is being submitted for the Degree of Master of Science in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other University.

Sean Graham Aspoas

The 22nd day of May 1996.

Abstract

We present a new three-dimensional container-packing algorithm. The algorithm is truly three-dimensional, thus overcoming the limitations of layering algorithms, especially when a large number of parcel types is used. The algorithm sequentially places parcels into the container using localised heuristic information, and makes use of a balanced tree to store potential packing positions. The result is an algorithm with time complexity $O(k n \log n)$, where k is the number of parcel types, and n the maximum number of parcels that can be placed. Test results, including a comparative test, are very favourable, and show that the algorithms performance actually increases as the number of parcel types is increased. This is a direct result of the three-dimensional algorithm facilitating the utilisation of all useful packing positions using the variety of parcel sizes available.

To my wife Jenny

Acknowledgements

I would like to thank my supervisors, David Lubinsky and Hanoch Neishlos, for their invaluable contributions, and both the Foundation for Research and Development and the University of the Witwatersrand for their financial support. I would also like to thank Compu-Clearing (Pty) Ltd for furthering my interest in container packing.

Contents

DECLARATION i

ABSTRACT ii

ACKNOWLEDGEMENTS iv

CONTENTS v

LIST OF FIGURES vii

LIST OF TABLES viii

CHAPTER 1 - INTRODUCTION 1

1.1 General Introduction 1

1.2 Literature Survey 2

1.2.1 One-Dimensional Packing 3

1.2.2 Two-Dimensional Packing 4

1.2.3 Three-Dimensional Packing 5

1.3 Research Directions 12

CHAPTER 2 - MODIFYING THE DROP-SEARCH ALGORITHM 15

2.1 Removing the Drop-Search Heuristic's Discrete Restriction 16

2.2 Creating a Compound Heuristic 17

2.2.1 Basic Heuristic Value Calculation 17

2.2.2 Stage One 18

2.2.3 Stage Two 20

2.3 Performance of the Non-discrete Drop-search Algorithm 20

2.4 Towards a true 3D Heuristic 21

CHAPTER 3 - ALGORITHM DEVELOPMENT 24

3.1 SPATIAL REPRESENTATION 24

3.1.1 Terminology and Concepts 24

3.1.2 Useful Potential Position Identification 24

3.1.3 Spatial Data Structures 26

3.2 $O(N^2)$ ALGORITHM 28

3.2.1 The Algorithm 29

3.2.2 Algorithm Complexity 30

3.3 THE $O(N^2 \log N)$ ALGORITHM 32

3.3.1 The Heuristic Tree 32

3.3.2 Maintaining the Data Structures 32

3.3.3 The Algorithm 33

3.3.4 Algorithm Complexity 34

CHAPTER 4 - HEURISTIC DEVELOPMENT 35

CHAPTER 5 - TESTING 40

5.1 TEST DATA GENERATION 40

5.2 GENERAL TEST RESULTS 41

5.3 TIME PERFORMANCE 43

5.4 COMPARATIVE TEST RESULTS 44

CHAPTER 6 - FUTURE WORK 47

6.1 MORE EFFICIENT BALANCED TREE IMPLEMENTATION 47

6.2 MORE READABLE STOWAGE PLANS 47

6.3 A BLOCKING APPROACH 47

6.4 MODIFYING THE HEURISTIC 48

6.5 A COMPOUND HEURISTIC 49

6.6 HUMAN-COMPUTER CO-OPERATIVE WORK 49

CHAPTER 7 - CONCLUSION 51

REFERENCES 52

List of Figures

Figure 1 : An example showing the use of the Put-Flat heuristic 18
Figure 2 : An example showing positions considered unnecessarily by the modified drop-search algorithm 22
Figure 3 : Example packing (without container dimension prioritisation) showing irregular, inefficient packing layout 38
Figure 4 : Example packing (using container dimension prioritisation) showing regular, grouped layout 38
Figure 5 : General test results - packing performance 41
Figure 6 : General test results - time performance 42
Figure 7 : Time performance with increasing container size 44
Figure 8 : Comparative test results - problem type I 46
Figure 9 : Comparative test results - problem type II 46

List of Tables

Table 1 : A summary of the three-dimensional packing algorithms found in the literature. 13

CHAPTER 1 - INTRODUCTION

1.1 General Introduction

Packing is the partitioning of a resource into a number of pre-defined portions. Three-dimensional container packing usually concerns placing as many pre-defined parcel types into a storage container as possible.

This report discusses a new truly three-dimensional container packing algorithm. The algorithm is based on the author's Drop-Search Heuristic algorithm [Aspoas 1992]. The existing algorithm is modified to allow real-valued container and parcel dimensions. The new algorithm is truly three-dimensional, instead of '2+1' dimensional (where the algorithm is based on a two-dimensional approach). This addresses the problem of unusable covered space and also allows the simplification of the heuristic evaluation system used. The time complexity of the algorithm is shown to be $O(kn\log n)$, where k is the number of parcel types, and n the maximum number of parcels that can be placed[1].

The algorithm sequentially places parcels into the container using localised heuristic information, and makes use of a balanced tree to store potential packing positions. Test results, including a comparative test, are very favourable, and show that the algorithm's performance actually increases as the number of parcel types is increased. This is a direct result of the fully three-dimensional algorithm facilitating the utilisation of all useful packing positions using the variety of parcel sizes available.

The remainder of this chapter includes a review of the relevant literature and the resulting research directions. Chapter 2 discusses the development of the Drop-Search algorithm

[1] The maximum number that can be placed of the smallest parcel type present is used as n

into a non-discrete algorithm for commercial application, as well as the subsequent ideas which lead to the new algorithm. Chapter 3 fully details the new algorithm for the efficient identification of potential parcel positions within the container. Chapter 4 discusses the alterations and simplifications made to the heuristic evaluation system as a result of these changes. In Chapter 5, test results, including a comparative test, are presented, while Chapter 6 discusses future research directions. Finally, Chapter 7 concludes the report.

1.2 Literature Survey

Interest in cutting and packing problems has grown steadily from as early as the 1950s [Sweeney and Paternoster 1992]. The topic has attracted researchers from disciplines ranging from computer science and engineering to logistics and production, and includes cutting-stock, trim-loss, bin-packing, pallet-loading, knapsack, and container packing problems [Dyckhoff 1990].

One and two-dimensional cutting problems have received the most attention, although few optimal reasonable time solution algorithms have been found. This brief review is not meant to constitute a complete survey of the packing and cutting literature. The focus of the review is on packing items into a single container, using orthogonal item positioning only. The reader is referred to [Dowsland and Dowsland 1992] and [Sweeney and Paternoster 1992] for a broader view of the research area.

1.2.1 One-Dimensional Packing

Perhaps the simplest one-dimensional cutting or knapsack problem involves cutting a given length into a number of required smaller stock lengths, so as to minimise the length left over. Practical applications include cutting a length of wood or piping into required lengths, and memory allocation in multi-process systems.

Hirschberg and Wong present a polynomial time algorithm for solving this problem when there are only two stock lengths [Hirschberg and Wong 1976]. Further, Coffman et al show that as long as the number of different stock lengths is fixed, then the problem is solvable in polynomial time, although the degree of the polynomial may be too high to be practically useful [Coffman et al 1984].

In general, however, this problem is NP-complete (proof by transformation from partition [Garey and Johnson 1979]). If the number of significant digits used in the length representation is restricted, then the problem is solvable in pseudo-polynomial time by dynamic programming [Garey and Johnson 1979]. Additional constraints and factors such as multiple initial lengths further increase the complexity of such approaches [Goulimis 1990].

Due to the formidable complexity of solving such problems, and many related, more difficult one-dimensional problems, most work has focused on finding approximation or heuristic approaches for obtaining practically useful results.

Since the knapsack problem can be formulated as an integer-programming problem (for which the general case is NP-complete [Garey and Johnson 1979]), much of this work has focused on approximation techniques for integer programming. This usually involves the 'integerisation' of linear programming using branch and bound and cutting plane techniques. Practically useful results, in terms of time taken and quality of result, have been obtained using these approaches [Stadtler 1990][Goulimis 1990].

Heuristics based more on common sense than mathematics, such as the First Fit Decreasing algorithm [Garey and Johnson 1979] and its derivatives, have also been shown to provide practically useful solutions [Vasko et al 1994].

1.2.2 Two-Dimensional Packing

Two-dimensional packing problems are a generalisation of the one-dimensional case, and include pallet loading, bin packing and stock cutting [Dyckhoff 1990].

Pallet loading concerns placing as many identical rectangular items onto a rectangular loading area (pallet) as possible. It is usually assumed that items must be placed orthogonally to the sides of the pallet.

Various heuristic methods have been developed. Bischoff and Dowsland find patterns which provide a good fit along the perimeter of the pallet, and then extend these patterns towards the centre, filling the remaining space [Bischoff and Dowsland 1982]. Interestingly, a comprehensive comparison has shown the Bischoff and Dowsland algorithm above to be optimal in over 95% of the tested cases [Dowsland and Dowsland 1992]. This is a strong indication that heuristic approaches have a definite role to play in tackling packing problems.

Another heuristic approach used is to reduce the problem to one dimension [Dowsland and Dowsland 1992]. This is done by dividing the pallet into strips using a heuristic approach, and then packing each strip using a one-dimensional algorithm (e.g. [Baker et al 1981]).

Integer programming is another solution approach commonly used. The algorithms used to solve the integer programming problem are similar to those used for the one-dimensional problem. Since the number of variables and constraints is greatly increased, Beasley improves on the standard branch and bound algorithm using Lagrangian relaxation, to solve moderately sized problems [Beasley 1985].

Dowsland describes an exact algorithm which runs in reasonable time for problems with sizes up to around 50 items [Dowsland 1987]. Dowsland equates the pallet-loading

problem to a graph theory problem. The resulting search tree is then heavily pruned, allowing reasonably sized problems to be solved optimally.

1.2.3 Three-Dimensional Packing

The increase in the complexity of the packing problem when moving from two to three dimensions means that exact methods are unlikely to provide practically useful algorithms. Integer programming models can theoretically be extended to the three-dimensional case, although, as reported by Dowsland and Dowsland [1992], little work has been done in this area.

The rest of this section examines the development of three-dimensional packing heuristics. The items to be packed into the container will be referred to as parcels.

George and Robinson developed an algorithm for packing parcels of different types (sizes) into a single container [George and Robinson 1980]. The container is packed one layer at a time, and in turn, each layer is packed by placing columns of parcels across the layer's width. This wall-building approach reduces the three-dimensional problem to a two-dimensional one.

The parcel types are each given a ranking according to several criteria, including the size of the smallest dimension and the size of the largest dimension. The depth of each new layer is determined by the highest ranked remaining parcel type, and the layer is packed with parcels of the same type using a regular packing pattern with each parcel in the same orientation. Remaining space within the layer, or space made up by combining this with previously remaining space, is then packed using the same algorithm and using other parcel types. This approach is clearly suited to packing parcel lists which consist of a relatively low number of parcel types, since this should lead to each layer being filled efficiently.

Test results show that the algorithm performs well on actual packing problems from industry. Variations in the algorithm result from varying the parcel-type ranking rules. Different ranking rules performed better in different situations, suggesting that a number of variations should be tried and the best result used.

Bischoff and Marriott proposed a similar algorithm to that of George and Robinson [Bischoff and Marriott 1990]. Here, each layer is packed with only one type of parcel, regardless of wasted space in a layer. The parcels are packed into the layer using Bischoff and Dowsland's two-dimensional packing (pallet loading) procedure discussed above. Each layer is only one parcel deep, with the depth of a layer being determined by the 'most useful' orientation of the parcel type being packed. Orientations can be ranked according to a number of criteria, including percentage of the layer filled and the greatest number of parcels accommodated. All remaining parcels which cannot be accommodated in a 'full' layer are packed using the George and Robinson algorithm.

Since no parcels are placed crossing layers, the layers may be placed in the container in any order. This means that if the distribution of the parcels is a factor (e g. for balancing weight distribution) then the order can be changed accordingly.

Due to the 'one parcel type only' layering technique, it is apparent once again that the algorithm is best suited to a parcel list consisting of a relatively low number of parcel types.

Bischoff and Marriott's comparative study of variations of the George and Robinson algorithm and the Bischoff and Marriott algorithm show that the latter algorithm generally performs better. [Bischoff and Marriott 1990]. This is as expected due to the more sophisticated two-dimensional algorithm used.

No single variation tested proved significantly better than the rest in all situations. Bischoff and Marriott do, however, identify that certain variations perform better in certain

situations. This leads to the suggestion that a compound heuristic be used. Here, each problem situation is examined and a few suitable solution variations used, with the best providing the final result. This approach leads to useful packing results with parcels lists of around 500 parcels within minutes on an 'advanced micro'.

Bischoff et al discuss an algorithm which, given a parcel list of a number of different parcel types, attempts to maximise the volume packed and at the same time provide a high degree of stability [Bischoff et al 1995]. In order to achieve stability, the algorithm packs the container from the ground up, never placing a parcel in a position in which its entire base is not supported. Viewed from above, the lowest available flat rectangular packing surface is packed with a layer of up to two types of parcels using a relatively simple two dimensional packing algorithm. One of the objectives is to not split the resulting packing surface into too many new packing surfaces, as viewed from above. The algorithm is essentially reducing the problem to a number of two-dimensional problems, although the divisions into layers is more flexible than before.

Bischoff et al provide test results showing that their algorithm consistently out-performs George and Robinson's algorithm. They identify 'fragmentation' of the packing surfaces as a problem area. As the packing areas become more fragmented, towers of parcels tend to result, leading to less stable packings, and the wasting of the spaces between the towers. The test results also indicate that the algorithm is more suitable for parcel lists with relatively few parcel types, since fragmentation is reduced.

Bischoff et al attempt to overcome the fragmentation problem by employing various methods of merging available packing surfaces. This leads to a compound heuristic which only partially deals with the problem, producing a small improvement in the test results. The compound heuristic packs around 100 parcels in a few minutes on a 33 MHz 486 PC.

Haessler and Talbot discuss the problem of packing parcels into rail trucks [Haessler and Talbot 1990]. Once more, the idea of reducing the dimensionality of the problem is

employed. The parcels are arranged into stacks, not higher than allowed by the available space, using a one-dimensional algorithm. These stacks are then treated as individual rectangular blocks, each having the 'footprint' of the box with the largest horizontal area in that particular pile. A two-dimensional algorithm is then applied to fit each footprint into the floor of the railcar. This heuristic takes into account a number of practical constraints, including stability, proximity of similar cargo and weight distribution. Resulting packings consist of pairs of columns running the length of the rail truck. The algorithm has been put to practical use with favourable results. The main drawback is that the algorithm is specifically designed for an suited to a particular application, and not suitable for general packing problems.

Gehring et al provide the first general heuristic suitable for packing a parcel list consisting of any number of parcel types [Gehring et al 1990]. Their approach also divides the container up into layers. Firstly, the parcel list is ordered by decreasing parcel volume. The depth of a layer is determined by a layer determining box (LDB). The LDB is placed into the far corner of the container in a certain orientation. All other parcels placed in this layer must fit into the spaces above and to the side of the LDB. These two rectangular spaces are placed into a spare-space list (which operates like a stack).

All spaces on the spare-space list are now packed in turn. The pair of parcels which take up the greatest volume and which fit into the top space on the spare space list is chosen. The remaining spaces beside, in front of and above the placed parcels are placed onto the spare space list (they are pushed in reverse order). This seems to be an attempt to keep the packing more stable or 'bottom heavy'. When all the spaces within the current layer have been used, the next layer is determined using another LDB.

If a spare space is too small for the smallest parcel, an attempt is made to amalgamate this 'unusable' space with other available spaces, although no details are given [Gehring et al 1990]. As with the Bischoff and Marriott algorithm, no boxes straddle the layers, allowing the order of the layers to be changed. Gehring et al leave the utilisation of larger spare

spaces resulting from the combination of adjacent spare spaces (which are in different layers) to the packing 'overseer'. No mention is made of possible computer aided support for the overseer.

The stowage plan generated can be varied by changing the orientations of the LDBs or choosing LDBs in different orders. Once more it is appropriate to run a number of these variations as a compound heuristic. The algorithm packs parcel lists of length around 50 in a few minutes on an IBM PC AT.

It is uncertain how the Gehring et al algorithm will perform with parcels lists consisting of relatively few parcel types since the algorithm will probably produce layers packed uniformly with mostly the same parcel type, the results are likely to be similar to those obtained using the George and Robinson algorithm. The Haessler and Talbot algorithm will probably perform better since it uses a more sophisticated algorithm for packing each layer.

All of the three-dimensional heuristic-based packing algorithms discussed so far have relied on approaches used to solve similar problems of a lower dimension. The heuristic-based algorithm of Ivancic et al makes a break from this, and considers the container as a truly three dimensional space [Ivancic et al 1989]. The heuristic sequentially places each parcel into the container. At each step, all available useful packing positions and parcel orientations are considered, with the exception of any positions which are covered (as viewed from above). Useful positions are determined by the positions of the previously placed parcels and the edges of the container. Only positions in which the parcel touches sides in the x, y and z dimensions are considered.

Since the algorithm was developed to take the economic value of the parcels into consideration, individual positions are evaluated in terms of the total volume used (parcel volume plus unusable 'trapped' space below the parcel) and the value of the parcel. This approach could be simplified to deal with the packing problem being discussed by giving

each parcel a value equal to its own volume. Since the algorithm was developed for use as a step in a multi-container packing algorithm, few results for the single container algorithm are available. Those that are available are favourable, although Ivancic et al indicate that a number of improvements on the heuristic could be made. The relative simplicity of this 'simplified' heuristic, which only considers volume covered, also suggests that further heuristic values would be needed.

Aspoas presents the drop-search heuristic algorithm aimed at reducing the inter-layer problem discussed above [Aspoas 1992]. Similarly to the algorithm of Ivancic et al, each available useful position is considered at each step. In order to reduce the complexity of the algorithm, the container is viewed from above, with the parcels being 'dropped' into the container. The drawback of this approach is that any covered space is eliminated from consideration, as in the approach of Ivancic et al.

Each position and orientation is evaluated in terms of three factors (in order of importance) : minimising the amount of covered space, minimising the height of the base of the parcel, and maximising the parcel surface area which comes into contact with other parcels and the container. These heuristics are intended to produce tight packings and leave large empty spaces.

Another drawback of the drop-search algorithm is that it can only be used for discrete sized containers and parcels, due to the discrete-space representation used. The container space is represented by a three-dimensional array, with each entry representing a unit-cube. A further two-dimensional array, called the 'view', represents the view from above the container by storing the packed height at each unit-square. The result of this discrete representation is that the container and parcel dimensions are restricted to discrete values.

When a parcel is being placed, all possible discrete x-y positions are considered as starting points for the drop-search. The resting place of the dropped parcel is then calculated using the view.

The algorithm was designed to facilitate interaction in the packing process by the user, if desired. The heuristic attempts to place parcels is a similar fashion to a human packer, allowing the user to understand the general packing strategy easily, and therefore successfully interact with the algorithm. The human-intervention system developed by Dorman allows the user to remove and place parcels in the container at will, with the heuristic continuing with the packing when desired [Dorman 1992].

Test results show an average packed volume of around 83 percent on various randomly generated test cases. The time taken per packing, on a 33MHz 486 PC, averages around seven seconds with an average of 22 parcels per packing. Human intervention has been shown to improve on the packings by around five percent. Importantly, the interaction of the human user with the heuristic generally produced better results than either one of the two [Dorman 1992] [Aspoas 1992].

Ngoi et al make use of a spatial representation strategy (not restricted to discrete sizes) to represent the container and the placed parcels [Ngoi et al 1994]. The data structure consists of a number of two-dimensional arrays, representing the state of the container at the various heights corresponding to the tops of placed parcels. This representation facilitates the search for 'empty volumes'. An empty volume is an empty 'space bounded by the vertical upward projection of a rectangular horizontal face'. The horizontal faces are the tops of parcels or the floor of the container. This is similar to the approach of Bischoff et al, where the packing surfaces themselves are stored as rectangles, as opposed to the entire container being represented. Unlike the previous two approaches, spaces covered by parcels are not lost to the algorithm.

At each step, a single parcel is placed in one of the empty volumes. The most suitable parcel and empty volume pair are picked using a number of ranking rules which attempt to fill available spaces as completely as possible, to leave usable spaces and not split horizontal faces into smaller horizontal faces. This last criterion is similar to that of

Bischoff et al, where the packing surfaces are packed in such a way as to avoid splitting the surfaces unnecessarily.

Test results show above 80 percent container utilisation. The time taken on a 33MHz 386 PC is up to a few minutes for packings of 200 boxes.

The algorithm never places a larger parcel over a smaller one, eliminating the problem of covered space found in Aspoas' drop-search algorithm. The drawback of this is that packings will tend to form towers of parcels (as the example packing in [Ngoi et al 1994] indicates). This leads to inter-tower spaces being wasted, as well as potential unstable packings. This was also the main drawback of the algorithm of Bischoff et al, and seems to be indicative of the approach of isolating packing surfaces (as viewed from above).

1.3 Research Directions

Due to the complexity of the three-dimensional packing problem, only heuristic-based approaches are currently feasible. Most current approaches adapt one or two-dimensional packing methods. This tends to waste space due to 'between-layer' spaces ([George and Robinson 1980], [Bischoff and Marriott 1990], [Gehring et al 1990]) or lead to 'tower-building' which may waste space and produce unstable packings ([Bischoff et al 1995]).

Three of the approaches, however, make use of heuristics which are closer to being 'full' three-dimensional algorithms, and so avoid the inherent between-layer spaces which result from layering techniques. Each of the three algorithms makes use of some form of three-dimensional space representation. Firstly, the approach of Ivancic et al is promising despite the problem of unusable covered space, but needs further development. Secondly, Aspoas' drop-search heuristic only works for discrete cases, and also suffers from unusable covered space. Finally, the approach of Ngoi et al tends to revert back to tower-building due to inflexibility in the parcel placement rules.

Table 1 summarises the algorithms discussed, including their basic data representation and packing technique, as well as their main drawbacks.

Algorithm	Data Representation / Packing technique	Parcel types	Drawbacks
George and Robinson	Layer by layer	Few	Inter-layer space
Bischoff and Marriott	Layer by layer	Few	Inter-layer space
Bischoff et al	'Region' layering	Few	Inter-region space / Tower building
Haessler and Talbot	Stack by stack	Restricted	Application specific
Gehring et al	Layer by layer	Many	Inter-layer space / Not suited to few parcel types
Ivancic et al	3D representation	Many	Loses covered space / Needs further development
Aspoas	Discrete 3D representation	Many	Loses covered space / Discrete sizes only
Ngoi et al	3D representation	Many	Between-tower space / Tower building

Table 1 : A summary of the three-dimensional packing algorithms found in the literature.

The above discussion suggests that an algorithm representing three-dimensional space (i.e. not using layering) should be used to avoid the problem of between-layer spaces. The algorithm should allow covered spaces to be created, in order to avoid the tower-building problem encountered by Ngoi et al. At the same time, the algorithm should make use of covered spaces as potential packing volumes.

An aspect of algorithm design missing from all the papers discussed is some form of complexity analysis. All the papers which do give results, merely provide the time taken for some test examples. The test examples and the machines used always differ, leaving no reliable form of comparison. An important part of designing any algorithm is ensuring that the complexity of the algorithm is as 'low' as possible. Designing an algorithm which runs in under a minute on small sized examples, but which takes hours to run when the problem size is increased to a more useful size, should be avoided.



CHAPTER 2 - MODIFYING THE DROP-SEARCH ALGORITHM

The desired features set out in the previous section are all met to some extent by the drop-search heuristic algorithm, except for the use of covered spaces, the need for real-valued container and parcel dimensions, and a low order of complexity. This suggests that the Drop-Search algorithm should be modified to address these three requirements.

In removing the covered spaces and integer-dimension restrictions from the algorithm, the primary problem is an increase in the computational complexity of the algorithm. Taking covered space into account means that there will be more placement positions to consider. The drop-search algorithm could be used for 'real-valued' dimensions if the number of significant digits used is restricted, with each unit-cube representing, for example, a one millimetre cube. The obvious problem with this is the enormous number of unit-cubes needed and possible drop positions to be considered. It is clear that an alternative method for representing the container space is needed.

The problem of changing the algorithm to use real valued dimensions was taken as the primary concern in modifying the drop-search algorithm to make it practically useful for commercial application.

2.1 Removing the Drop-Search Heuristic's Discrete Restriction

The discrete version of the algorithm assumes that every possible 'grid-line' (lines created by the unit-cubes) represents a useful packing position. If the algorithm were changed so that only grid-lines corresponding to placed parcels and the container walls were used, then all useful positions would still be considered, with the number of search positions drastically reduced.

The above observations lead to the development of a system of two lists, called the x-grid and y-grid. The x-grid stores, in order, the list of x-positions corresponding to the sides of placed parcels on the x-axis. The y-grid operates similarly for the y-axis. The two grids are used to find every potentially useful x-y position by looping through the y-grid for every entry in the x-grid.

Once the x-y position is found, the parcel is dropped by searching through the list of placed parcels, which is sorted in descending order of height. To calculate the heuristic value for a found position, the placed parcel list has to be searched to find touching parcels.

The grid system is extended to include a z-grid for the height dimension to facilitate manual movement of a parcel around the container. The grid system means that only potentially useful positions (positions in line with other parcels) need to be considered by the user, and is therefore a strong extension of the human-computer interface.

2.2 Creating a Compound Heuristic

A number of modifications were made to the basic heuristic calculation method used in the drop-search algorithm, to make it more suitable for practical use. The final result was a compound heuristic, consisting of two stages, each based directly on the basic drop-search heuristic value calculation method [Aspoas 1992].

2.2.1 Basic Heuristic Value Calculation

The calculation of the heuristic values used, as discussed in section 1.2.3, is based on three factors (in order of importance) : minimising the amount of covered space, minimising the height of the base of the parcel, and maximising the parcel surface area which comes into contact with other parcels and the container (or 'touching area').

The covered space c is calculated by taking the percentage of the lower surface area of the parcel being placed that does not touch a parcel below (represented by a value from 0 to 1). This is a two-dimensional approximation of the three-dimensional space found below and covered by the parcel. The height h of the parcel is represented by the percentage of the full height of the container at which the base of the parcel rests. The touching area t is calculated as the percentage of the total surface area of the parcel which is touching the container or other parcels. Percentages are used in an attempt to normalise the values for different parcel sizes.

The final heuristic value is calculated as a weighted sum of the three values :

$$heuristic_value = c_{weight} \times (1-c) + h_{weight} \times (1-h) + t_{weight} \times t$$

Note that the covered and height measures have been 'inverted' to give values to be maximised. The actual weights used were arrived at after experimenting with various values, and are 50, 25 an 10 respectively. The final heuristic equation is therefore :

$$heuristic_value = 50(1-c) + 25(1-h) + 10t$$

The no-cover value is given the highest priority since this effectively adds the third dimension to the packing algorithm. The height value is next since this also contributes to eliminating covered space as well as maintaining a large, useful volume at the upper end of the container.

2.2.2 *Stage One*

The first stage of the packing algorithm attempts to pack each parcel, in order of decreasing volume, using the updated drop-search algorithm and the heuristic value calculation method discussed above. If the parcel being placed is of the same type as the previously placed parcel, then before performing the drop-search the positions around the previously placed parcel are first examined. If the greatest of the associated heuristic values is greater than or equal to the previously placed parcel's heuristic, then that position is used. This serves two purposes. The first is to speed up the performance of the algorithm. The second is to maintain the locality of box types. This tends to result in packing layouts which are more coherent, efficient and easier to read. Further, locality is often desirable in container transport, especially if a container is to be shared between a number of different clients.

A further modification to the first stage heuristic is a mechanism which acts as a tie-breaker between positions with the same heuristic value. The modification is based on the observation that when parcels are being packed into a 'block' up to the side of the

container, it is sometimes more efficient to change the orientation of the last row of parcels (see figure below). The decision is based on a simple calculation to decide which orientation allows the most parcels in the remaining space that the block of parcels will take up. This modification, called the 'put-flat' heuristic, is applied even if there aren't enough parcels of the same type to fill up the space on which the calculation is based. Further, the space may not even be free, since it is assumed that the packing is proceeding from low co-ordinate to high co-ordinate values along each axis.

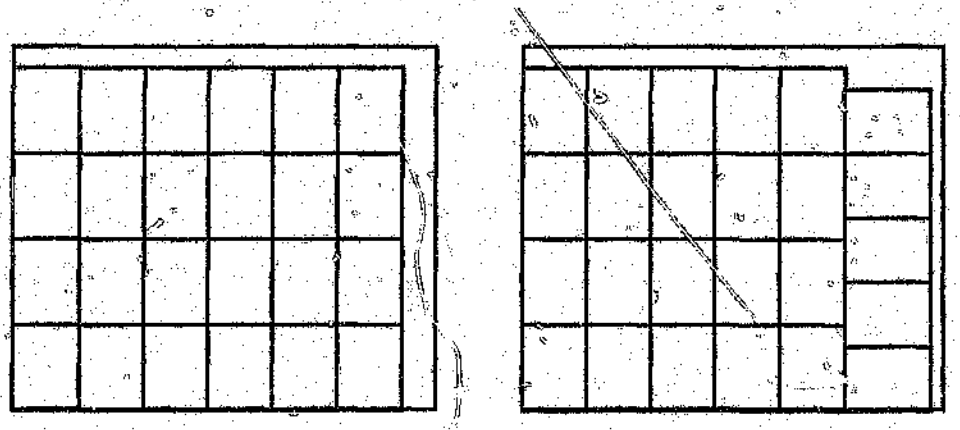


Figure 1 : An example showing the use of the Put-Flat heuristic

Figure 1 is an example of the use of the put-flat heuristic. In the left container 24 parcels have been placed. In the right, the last column of parcels is rotated allowing an extra parcel to be placed.

The put-flat heuristic can be seen as a primitive attempt to perform one-dimensional packing. The heuristic could be extended to take the last two rows into account, or the last three, or four, etc. This would eventually result in a one-dimensional packing problem, and a solution algorithm such as that of Hirschberg and Wong could be used to find how many columns of each parcel to use.

The put-flat heuristic is applied to the comparison of any two positions with the same heuristic value, and comes into effect if one or both positions would result in a space between the placed parcel and the container wall that is too small for another parcel of its type.

2.2.3 Stage Two

The second stage packs each parcel type in turn. Instead of trying each parcel in every possible orientation, as many parcels as possible are packed in one orientation before another is attempted. All possible orientation orderings are attempted. The best resulting packing for the parcel type (most parcels placed, followed by highest sum of individual heuristic values) is used, and the next parcel type attempted.

The modified drop-search algorithm is used to place each parcel (without the put-flat heuristic additions). Placing parcels using fixed orientation tends to form large blocks of parcels, which aids locality and more coherent packing plans. Due to the emphasis placed on finding efficient groupings of a particular parcel type, this approach is perhaps more suitable for problems with fewer parcel types.

2.3 Performance of the Non-discrete Drop-search Algorithm

The revised Drop-Search algorithm has complexity $O(n^4)$, where n is the total number of parcels placed. For each parcel (n) and each x-y position (n^2) the parcel is dropped and the heuristic calculated ($2n$).

The compound heuristic used makes use of the $O(n^4)$ algorithm a bounded number of times, and tests show that the final algorithm runs in a few minutes for test cases of moderate size (up to a couple of hundred) on a 486DX-33. For larger test cases (up to around 800 parcels with many different parcel types) the algorithm can take a number of hours to run. This time complexity and execution time is unacceptable in a practically orientated algorithm, and needs to be improved.

The results achieved using the compound heuristic are very favourable compared to previous methods used by the company concerned. The remaining problem is that of the time taken for larger problems.

2.4 Towards a true 3D Heuristic

The two main drawbacks of the algorithm developed above are its speed and the fact that covered space is still not utilised.

The existing algorithm could be extended to include covered spaces if the z-grid were also utilised when searching for potential positions. The 'dropping' routine would still be needed, but only to check if the position is legal in terms of placed parcels. The complexity of the algorithm would increase to $O(n^5)$, which is clearly not desirable.

The algorithm would consider all combinations of x, y and z-grid points as possible positions, resulting in $O(n^3)$ positions being considered. In fact many of these positions may not correspond to useful positions. This occurs when a new entry into one of the grids results in a multitude of new potential positions corresponding to every combination of points from the other two grids. In Figure 2, the circled positions are considered unnecessarily by the modified Drop-Search algorithm (the container is viewed from above

with 9 parcels already placed, and grid lines completed using dotted lines). Further, positions may correspond to corners in the packing which are not viable due to the presence of other parcels. Eliminating these positions would clearly improve the performance of the algorithm. In fact, associated with each parcel placed, there is a bounded number of useful potential positions. These positions result from the bounded number of parcels in the immediate area which, together with the parcel being placed, result in a bounded number of new potential positions. This means that the number of positions searched can be reduced to $O(n)$, since the number of parcels placed is $O(n)$. The resulting algorithm, which takes covered space into account, would have an expected complexity of $O(n^3)$.

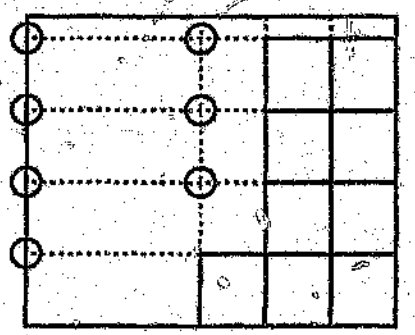


Figure 2 : An example showing positions considered unnecessarily by the modified drop-search algorithm

So, for each parcel, $O(n^2)$ work would be performed. One of these factors of n is a result of calculating whether the parcel can physically fit in the potential position and calculating the heuristic value if it does. Both of these calculations depend only on *local* information. Only parcels in the immediate vicinity are needed when calculating if the current parcel intersects or touches any other parcels. If this fact could be taken advantage of, then both calculations would take constant time, and the resulting overall complexity would be $O(n^2)$.

Each time a parcel is placed the heuristic value for each position is recalculated, even if this has already been done for most positions for the same parcel type in the previous step. Clearly, storing the heuristic values associated with each position would improve on the performance of the algorithm. This can be done by making use of a balanced tree to keep track of the heuristic values in order. Each time a parcel is placed, certain covered positions (a bounded number) would have to be removed, while new positions (again, a bounded number) would have to be considered. At each step the position with greatest heuristic value would be chosen (if the parcel type is the same as the previous parcel's type). If there were k different parcel types, this would lead to an overall time complexity of $O(kn\log n)$.

Maintaining a list of all possible placement positions and a sorted list of the best heuristic positions available could provide useful tools for manual parcel placement. Firstly, the human packer could browse through the list of all useful positions when making a placement, instead of having to place the parcel and then test whether the position is feasible and useful. Secondly, the human packer could browse through the best positions as seen by the algorithm, and would therefore be guided in placement decisions.

CHAPTER 3 - ALGORITHM DEVELOPMENT

This chapter details the development of the full three-dimensional searching algorithm, based on the above discussion.

3.1 Spatial Representation

3.1.1 Terminology and Concepts

The container and parcels are all rectangular boxes. One of the corners of the container rests at the origin, with the length, width and height corresponding to the x, y and z axes respectively. The position of a parcel in the container, in a given orientation, can be identified uniquely by its starting x,y,z co-ordinates.

Conceptually, each parcel in the container (and indeed the container itself) has edges which include the lower bound on the interval, but exclude the upper. For example a cubic parcel with side length one, starting at the origin, extends over the interval [0,1) on each axis. This allows two parcels to be placed side by side without any intersection between them.

3.1.2 Useful Potential Position Identification

The packing algorithm needs to identify useful potential positions for placing parcels. It is important that the algorithm used for this does not eliminate any final packing result. It should be possible for any packing layout to be arrived at through a particular sequence of choices from the potential positions presented by the algorithm. It should be noted that different final packings can be considered equivalent. An example of this is two packings

which are identical except for the position of a single parcel, which is slightly displaced into an open space. Any packing can be 'reduced' to a form in which every parcel in the container rests against an edge (either of another parcel or of the container itself) in each of the three dimensions. This is equivalent to tilting the container up onto one of its bottom corners, and allowing gravity to pull the parcels downwards until each is touching an edge in all three dimensions.

Any position in which a parcel is 'supported' (touches another side) in all three dimensions will be considered a useful potential placement position. Further, in the z-dimension the parcel must be supported from below, to eliminate 'stalactite' parcels. This definition of a useful potential position ensures that any (reduced) possible final packing result can be reached.

It is important to note that when a parcel is placed within the container, that all new resulting potential positions can be found using locally placed parcels only. Any new position will result from a parcel resting against one of the edges of the newly placed parcel and against two other 'local edges'.

A pseudocode algorithm for finding all new potential positions resulting from a newly placed parcel is given below. A test parcel is placed against one of the newly placed parcels sides. Support in the remaining two dimensions is then searched for in the locally positioned parcels in the container. If support is found, a check is performed to ensure that the new position is physically possible - i.e. that the parcel would not intersect any other parcel (only local parcels need to be checked) and that the entire parcel fits into the container. Finally, if the position is not already stored as a potential position, the new position is stored.

Assuming a bounded number of 'local' parcels, steps (1) to (4) in the algorithm below each take bounded time. The time complexity of step (5) will have to wait until the data structures used have been discussed in the next section. Note that in step (5) the heuristic

value found in step (4) is stored with the position. The position itself is stored as the starting x,y,z point (or 'root') of the parcel and the parcel orientation (of which there are at most six).

```
MODULE AddNewPositions (p1 : newly placed parcel)
  p2 : parcel used to test for new positions
    FOR each orientation of p2
      FOR each face of p1
        place p2 against face of p1
        check 'local' parcels for support in other two dimensions        (1)
        IF support found in both dims
          IF position 'possible' (no intersections with local parcels)   (2)
            IF  position NOT already stored                              (3)
              Calculate position's heuristic using 'local' parcels       (4)
              Add new position                                           (5)
```

3.1.3 Spatial Data Structures

Each time a parcel is placed within the container a number of maintenance actions are necessary: new potential placement positions must be identified, existing positions which are no longer feasible must be excluded, and the heuristic values associated with the new positions and some of the existing positions (since the newly placed parcel may alter a positions heuristic value) must be calculated. As discussed in section 3.1.2, new position identification requires only locally placed parcels. The same applies to calculating the heuristic value of a parcel, since only parcels which touch the parcel, or the container itself, need to be considered. This also applies to removing potential positions which are no longer spatially feasibly due to the newly placed parcel.

In order to achieve the desired order of complexity, the spatial representation used must take advantage of the locality inherent in the algorithm. This is done by dividing the container down into a number of cubes, called p-cubes. The size of the p-cubes should be roughly equal to that of the largest parcel in the parcel list[1]. The idea is to restrict any change caused by the addition of a parcel to a bounded number of these cubes. Clearly, all maintenance following the placement of a parcel mentioned above can be made by examining the parcels placed in a bounded number of p-cubes around the placed parcel.

The lower and upper bounds of each p-cube operate in the same way as those for parcels. A parcel whose corner with lowest x, y and z co-ordinates rests within a p-cube will be referred to as 'in' that p-cube, regardless of whether the parcel extends into neighbouring p-cubes.

The container is represented directly by a three-dimensional array of p-cubes. Following from the above discussion, the parcels placed in a given p-cube, as well as the potential positions within that p-cube, must be stored for each p-cube. This is achieved by giving each p-cube two linked lists - one for placed parcels rooted in the p-cube (called the *placed list*), and one for potential positions similarly rooted (called the *position list*)[2]. Note that the position list stores the position as well as its associated heuristic value. Also, the length of each list is bounded since each length depends on the number of parcels placed in the local area.

In module AddNewPositions above, step (5), which adds a newly found position, will have a bounded time complexity since adding to the position list will require a bounded number of operations. It follows that the time complexity of module AddNewPositions is bounded.

[1] See section 3.2.2

[2] The name 'p-cube' results from the storage of the positions and placed lists

3.2 O(n^2) Algorithm

When a new parcel is placed within the container, the state of the container array must be updated to reflect the change. Consider the following module :

Module MaintainContainer (pUsed : position that has just been used;
parcel : parcel just placed in pUsed position)

Remove pUsed form relevant p-cube (1)

Remove 'covered' positions from neighbouring p-cubes (2)

Update 'touched positions' heuristics in neighbouring p-cubes (3)

AddNewPositions(parcel) (4)

The following observations refer to the numbered steps above. (1) simply searches through the p-cubes position list, and removes the used position from the position list. The time needed for this operation is bounded. (2) removes positions which are no longer feasible due to the new parcel being placed. Only certain of the neighbouring p-cubes need to be search for this step, and the time complexity for this step is also bounded. It should be noted that step (2) makes step (1) redundant, and that step (1) is included separately for clarity. Since the newly inserted parcel may affect the heuristic values of some of the local potential positions, these values must be updated. (3) performs this task by looking at all the positions in certain neighbouring p-cube position lists, and recalculating the heuristic values (again using only local p-cube data). Once more, the time needed for this operation is bounded. Finally, in step (4) the new potential positions created by the addition of the new parcel are added to the relevant p-cube position lists, as discussed in section 3.1.2, taking bounded time due to the bounded length of the position list. It follows that module MaintainContainer requires a bounded amount of time.

Step (2) above searches through all potential positions in a number of neighbouring p-cubes. This process is simplified by joining the position lists from each required p-cube,

and then searching through the resulting single list. Steps (3) and (4) require similar lists of positions and placed parcels. The creation of these 'chains' is facilitated using a doubly linked list for the position and placed parcel lists in each p-cube. A chain is created by linking a list's last element's 'next' pointer to the next list's first element, etc. This creates a forward linked chain of the required elements. When the chains are broken down after use, the changed forward pointers (one per p-cube involved) are restored using the list's first element's 'previous' pointer. The creation or destruction of each chain requires only one pointer operation per p-cube.

The edges of the container are neatly dealt with by simply adding 'false' parcels onto the end of these temporary chains, representing the outside of the container. For example, the outside of the container at one of the corners is represented using three parcels - one 'blocking' each dimension. Conceptually, the outside of the parcel is filled space, and therefore not usable.

3.2.1 The Algorithm

The searching and maintenance aspects of the $O(n^2)$ algorithm have now been dealt with. The actual heuristic values given to each potential position, will be discussed in Chapter 4. The algorithm outline is as follows :

```
Module PackN²
    initialise p-cubes                                          (1)
    sort parcel types by descending volume                      (2)
    WHILE still parcels to pack                                 (3)
      IF new parcel type THEN                                   (4)
        Clear all position lists                                (5)
        FOR all placed parcels p AddNewPositions(p)             (6)
          Add corner positions                                  (7)
      Find position pBest with greatest heuristic value         (8)
```

IF found (9)

Place *parcel* at position *pBest* (10)

MaintainContainer(*pBest*,*parcel*) (11)

ELSE (12)

Move to next parcel type (13)

At step (4), if a new type of parcel is encountered, the potential positions must be cleared (5) and recalculated (6) for the new parcel type. Potential positions must be recalculated since new positions may arise due to a smaller parcel dimension, or different supporting parcels being in reach of a larger parcel dimension. Also, old positions may fall away due to supporting parcels being out of reach or the new parcel type not fitting the available space. In addition, each time a new parcel type is encountered, the positions which may be available at the bottom corners of the container must be added (7) since these positions are not necessarily supported by any of the placed parcels. This completes the notion of the outside of the container being represented by false parcels, since at the intersections of the false parcels, potential positions may be found (except, of course, at the upper corners). This step also provides the initial positions in the empty container at the start of a packing.

3.2.2 Algorithm Complexity

Let n be the volume of the container divided by the volume of the smallest (in volume) parcel type. Note that the maximum number of parcels placed is always bounded above by n. Let k be the number of parcel types to be packed. It should be noted that this definition for n does not necessarily correspond to the input length for the algorithm. An example is where the input is simply one parcel type and the number of such parcels. This means that the complexity analysis below is not conventional, and measures how the complexity

changes as the ratio between the container size and the size of the smallest parcel increases, rather than as the input length increases.

In order to achieve the desired complexity, the number of p-cubes affected by a parcel's placement is restricted to 'local' parcels only. The length of each p-cube's side is set equal to the maximum parcel dimension found in the parcel list. The number of p-cubes is therefore bounded above by n. This choice also means that a parcel cannot stretch over more than two p-cubes in any dimension, and proves convenient and efficient when accessing local parcels.

In Module PackN^2 above, step (1) takes O(n) time, while step (2) takes O($k\log k$). The IF statement at step (4) is true a maximum of k times. In step (5), O(n) lists, each of bounded length, are cleared. Step (6) repeats module AddNewPositions O(n) times, which results in an overall time complexity of O(n) for step (6). Step (7) adds the potential positions corresponding to the lower four corners of the container, and takes a bounded amount of time. It follows that the IF statement contributes O(kn) overall to the algorithm complexity.

Step (8) requires O(n) operations, since there are O(n) position lists to search. Steps (10) and (13) each take constant time, while step (11) takes a bounded amount of time as discussed in the previous section. The body of the WHILE loop therefore takes time O(n), and is repeated a maximum of n times.

The resulting time complexity of the algorithm is O($n + k\log k + n^2 + kn$). Note that the IF statement contributes O(kn) in total, as mentioned above, and so does not need to be considered as part of the WHILE loop for the complexity analysis. If the number of parcel types remains constant, the overall complexity is O(n^2), as expected.

3.3 The O(*kn*log*n*) Algorithm

The next step in the algorithm development is to reduce the algorithm complexity by making use of a balanced tree to store the heuristic values.

3.3.1 The Heuristic Tree

The heuristic tree is a balanced red-black tree used to store all heuristic values associated with the current potential positions. The total number of potential positions generated over the duration of the algorithm is O(*n*). This follows from the fact that in each p-cube there are a bounded number of potential positions produced from the bounded number of parcels that can be placed in that p-cube. This means that the cost of each insertion performed on the heuristic tree is O(log*n*). When a position is no longer viable, it is removed from the tree by simply marking the node as used. Each node in the tree also stores how many of its children are not marked empty, allowing the position with greatest heuristic value to be easily found. The search is performed by moving through the tree, taking the right branch as long as the right child is non-empty or has non-empty children (indicated by a non-empty child count greater than zero). If this is not the case, then the left child must be taken. This means that O(log*n*) moves are required to locate the largest heuristic value.

The maintenance of a red-black tree is simplified if no items of equal value are to be stored. This is one of the reasons no two potential positions are allowed to have the same heuristic value, as discussed in Chapter 4 below.

3.3.2 Maintaining the Data Structures

Now that the operation of the heuristic tree has been discussed, the tree must be linked into the existing data structures. When the best heuristic value has been found, the

associated potential position must be allocated. This is facilitated using a pointer to the position node in its position list (in turn in a p-cube), stored with the heuristic value in the tree node. Consider the MaintainContainer module repeated below:

```
Module MaintainContainer (   pUsed : position that has just been used;
                             parcel : parcel just placed in pUsed position )
    Remove pUsed form relevant p-cube                                    (1)
    Remove 'covered' positions from neighbouring p-cubes                 (2)
    Update 'touched positions' heuristics in neighbouring p-cubes        (3)
    AddNewPositions(parcel)                                              (4)
```

Steps (1) to (4) are made possible by the inclusion of the extra pointer in each tree node. In step (3), the tree nodes corresponding to the affected positions cannot simply be updated, since this may compromise the ordering of the tree. Instead, the tree nodes must be removed (marked as empty) and the updated heuristic values re-inserted into the tree.

3.3.3 The Algorithm

The algorithm outline for the O($kn\log n$) algorithm is very similar to that of the O(n^2) algorithm (except for the steps marked in bold) :

```
Module PackKNLogN
    initialise p-cubes                                   (1)
    initialise heuristic tree                            (2)
    sort parcel types by descending volume               (3)
    WHILE still parcels to pack                          (4)
      IF new parcel type THEN                            (5)
```

```
    Empty heuristic tree                          (6)
    Clear all position lists                      (7)
    Add corner positions                          (8)
    FOR all placed parcels p AddNewPositions(p)   (9)
  Find position pBest in heuristic tree           (10)
  IF found                                        (11)
    Place parcel at position pBest                (12)
    MaintainContainer(pBest,parcel)               (13)
  ELSE                                            (14)
    Move to next parcel type                      (15)
```

Step (2) simply initialises the heuristic tree. Step (6) actually removes all nodes from the tree, as opposed to marking all nodes empty. In step (9), module AddNewPositions must add each new position to the heuristic tree. Step (10) makes use of the modified tree search procedure discussed above.

3.3.4 Algorithm Complexity

The complexity analysis for the $O(kn\log n)$ algorithm is similar to that of the $O(n^2)$ algorithm. Step (2) is a simple constant time initialisation, while step (6) empties the heuristic tree, requiring $O(n)$ operations. In step (9), module AddNewPositions requires $O(\log n)$ operations, since a bounded number of new positions is added to the heuristic tree per placed parcel. Overall, step (9) requires $O(kn\log n)$ operations, since there are $O(n)$ placed parcels and k parcel types. Step (10) requires $O(\log n)$ operations, as discussed above. This leads to an overall complexity of $O(n + k\log k + n\log n + kn\log n) = O(kn\log n)$ as expected. For a fixed number of parcels, this reduces to $O(n\log n)$.

CHAPTER 4 - HEURISTIC DEVELOPMENT

As mentioned in section 3.3.1, the heuristic algorithm used never equates two positions. This simplifies the operation of the balanced red-black heuristic tree. Perhaps more importantly, this means that the algorithm always identifies a 'best' position out of all the current potential positions. In this way, no arbitrary decision has to be made to choose between a number of ties, and nothing is left up to the chance ordering of the positions. A heuristic algorithm which provides a strong ordering means that the algorithm has more control over the positions used.

The heuristic algorithm used is based primarily on that developed for the Drop-Search algorithm [Aspnas 1992], and subsequently extended for the non-discrete Drop-Search algorithm (see section 2.2). During the development of the full three-dimensional algorithms, a number of changes were made to the heuristic, following observation of the performance of the algorithms under various conditions.

The first such change was to remove the no-cover heuristic measure. This was initially built into the heuristic measure as an attempt to compensate for the loss of all covered space (as viewed from above the container). Since the new algorithm is fully three-dimensional, covered space is not lost to the algorithm. Early test results clearly indicated that the no-cover measure actually reduced the packing performance.

The second change was to remove the height measure, which was originally included to encourage the utilisation of space closer to the bottom of the container, so as to leave large usable spaces higher up, and also to avoid the creation of covered space. The latter reason clearly no longer holds, while the former still has some merit. The question which needs to be asked is : Why give the height of the container, or the z-dimension, greater priority than the other two dimensions? Perhaps the dimensions should be prioritised, with

parcels being packed tightly in high-priority dimensions. This observation is put to use in the tie-breaking measures discussed later in this section.

All that is left of the original heuristic is the side-touch component, which measures how much of the surface area of the parcel being placed touches either the side of the container, or other parcels. This helps create tight, uniform packings with large usable spaces, and is clearly still a desirable heuristic measure. Since the other heuristic components have been removed, the need for a weight falls away, and the heuristic value used is simply the surface area of the parcel which is in contact with the container or other parcels :

$$heuristic_value = touching\ area = t$$

Reducing the heuristic measure to this single component clearly indicates that the new algorithm overcomes some of the unnecessary complications found in the original algorithm, and partially compensated for by the additional measures. Importantly, the side-touch measure makes use of local information only, as required for the $O(kn\log n)$ complexity.

The drawback of having only one measure making up the heuristic is that many ties will result. For example, there may be a number of potential positions in which the same two surfaces of the parcel are fully up against other parcels or the container. As discussed above, this is undesirable.

To act as a tie-breaker, and in an attempt to improve the packing performance of the algorithm, the idea of encouraging tighter packing in a priority dimension has been introduced. The container dimensions are prioritised in inverse order of size. The idea is that the smallest container dimension is the most precious commodity. When two potential positions have the same side-touch heuristic value, the better of the two positions is determined by how well each utilises the prioritised dimension.

If the x-dimension is deemed to have the highest priority, then the position which results in the parcel having the smallest upper extent in the x-dimension, is considered the better of the two. If both positions are still tied, then the position which results in the parcel having the smallest *lower* extent in the x-dimension, is considered the better of the two. If a tie still exists, then the parcel extents in the dimension with second highest priority are considered. Finally, the third dimension is also considered. Note that no two positions can result in a tie, since this would mean that the positions being compared are the same, and repeated potential positions are not stored.

Prioritising the container dimensions also helps to promote more regular packing layouts, since layering or the formation of blocks of parcels in the same orientation is promoted due to the tendency to push all parcels as far as possible in a prioritised dimension. Regular packing layouts are beneficial in a number of ways. Firstly, the number of potential packing positions is reduced due to entire regions being filled. Secondly, a regular block of parcels makes 100% utilisation of the space concerned, and the idea is that this local optimisation should help the overall optimisation. Finally, from a practical point of view, regular layouts are easier to pack and may be useful if all of a particular component of a shipment is to be unloaded first.

Figure 3 is an example of a packing, with 5 parcel types, in which no dimension prioritisation has been applied. The result is a packing layout which is clearly 'fragmented' and not suitable for practical use. The same test case, this time using dimension prioritisation, is shown in figure 4. The layout is clearly more regular and usable, while the percentage container utilisation is increased from 81.1 to 84.4 percent[1].

[1] The test data generation technique used is the same as that discussed in the next chapter

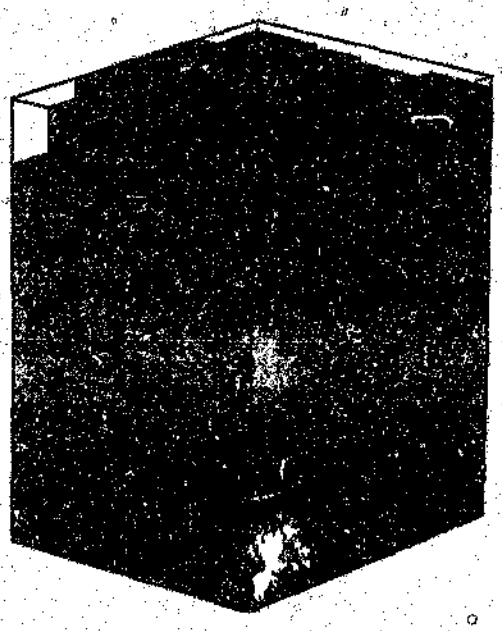


Figure 3 : Example packing (without container dimension prioritisation) showing irregular, inefficient packing layout[1]

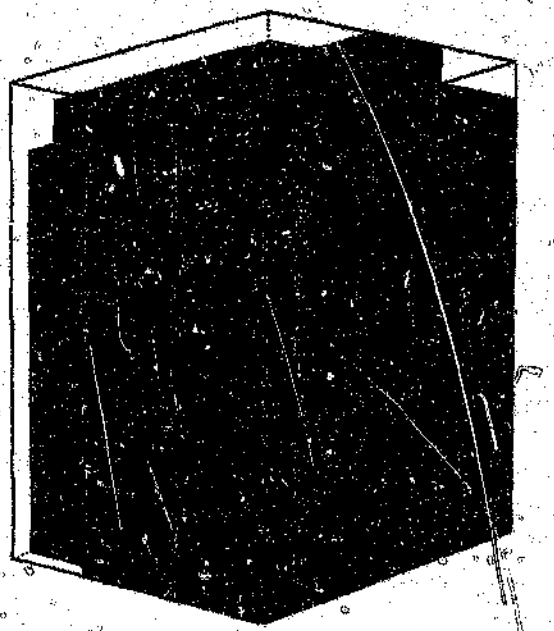


Figure 4 : Example packing (using container dimension prioritisation) showing regular, grouped layout

[1]Graphics generated using Dorman's container packing graphics tool [Dorman 1992]

Testing showed that the prioritisation technique discussed above performed better than both the put-flat heuristic and the method of searching the immediate area around the previously placed parcel used in the non-discrete drop-search algorithm as tie-breaking mechanisms (see section 2.2). For this reason both of these tie-breaking mechanisms are excluded from the new heuristic.

CHAPTER 5 - TESTING

All testing was performed on a Silicon Graphics Indy machine with a MIPS R4400 processor.

5.1 Test Data Generation

The test data generation technique used is based on that found in [Bischoff et al 1995]. The basic parameters used are the container size (c_x by c_y by c_z), the number of parcel types (p_{types}), and the range in which the parcel type dimensions must lie (x-dimension in $[x_{low}, x_{high}]$, y-dimension in $[y_{low}, y_{high}]$ and z-dimension in $[z_{low}, z_{high}]$). The first step is to generate the actual dimensions for each parcel type, choosing random numbers in the relevant range.

Next the number of parcels of each type is determined using an iterative approach. The number of each parcel type is initially set to zero. At each step a parcel type is chosen randomly, and the number of that type is incremented. This process continues until the addition of the new parcel would result in the total volume of all parcels being greater than the volume of the container. This signals the end of the process, and the counter for that parcel type is not incremented.

The final result is a list of parcels whose summed volume is less than the volume of the container by less than the volume of the largest parcel type. This means that a result showing what percentage of the container has been packed should be a fair reflection of the performance of the algorithm, and if anything gives the algorithm slightly less credit than it deserves.

The test data generation method is especially useful if comparison is to be made concerning the packing performance as the number of parcel types is varied, since the exact number of parcel types can be set for each test data set.

5.2 General Test Results

The first test data set is designed to test the performance of the algorithm with a wide variety of parcel lists and container sizes. For this purpose, a separate test data set was generated for each of the following values of p_{types} : 1, 5, 10, 15, 20, 25, 30, 35, 40, 45 and 50. Each test set (one for each value of p_{types}) consists of 1000 test cases. The container sizes used were randomly regenerated for each test case, using the ranges [900,1300] for x, [1300,1700] for y, and [900,1700] for z. The parcel dimensions were also randomly regenerated for each test case, using the ranges [50,350] for x, [250,450] for y, and [150,300] for z.

The result is a good variety of container and parcel sizes, and a sizeable number of test cases per set. The range of values for p_{types} is also broad. Taking the average container size and average parcel size, the average number of parcels per test case is 136. With 50 parcels types, the average number of parcels per parcel type is thus less than 3. The result is an average number of parcels per parcel type which ranges from 136 to less than 3, providing a broad test range, and testing the algorithm under extreme conditions. Figure 5 presents the results of the 'general' test set.

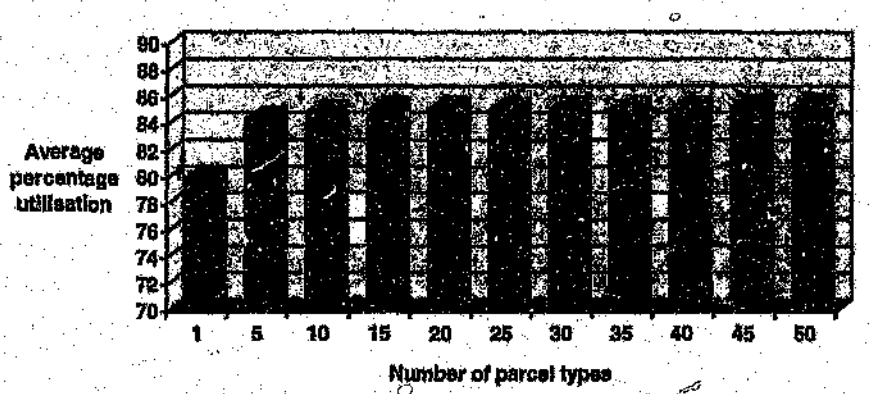


Figure 5 : General test results - packing performance

It is significant that the high percentage utilisation achieved for 5 parcel types is maintained and in fact improved upon all the way up to 50 parcel types. The result for one parcel type is noticeably lower, but still shows a useful container utilisation of 80%. Overall, the utilisation results are remarkably consistent over a wide range of values for p_{types}.

The algorithm is clearly well suited to problems ranging from a few parcel types to a very high number. Different heuristic parameters or a different heuristic approach may be better suited to problems with a low number of parcel types, since it seems unlikely that it is inherently more difficult to pack fewer parcel types.

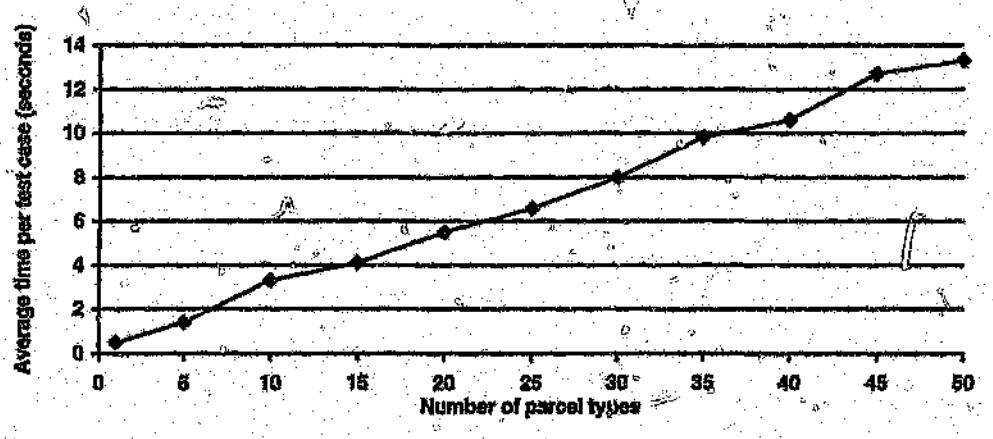


Figure 6 : General test results - time performance

Figure 6 shows the average execution time per test case for the general test results in Figure 5. Clearly, the more parcel types there are the longer each packing takes. There are two factors which contribute towards this phenomenon. Firstly, a higher number of potential positions can result from a higher number of parcel types due to a more irregular packing layout. Regular layouts tend to generate fewer positions. An example of this is a block of parcels all in the same orientation - no potential positions occur within the block. The other, and more significant cause is the initialisation of the potential position lists and

heuristic tree each time a new parcel type is encountered. In section 3.3.4 the initialisation step was shown to be O($kn\log n$), and the increase in time should be linear in k (assuming that n remains constant), as verified by Figure 6.

5.3 Time Performance

The second data set is designed to demonstrate the O($n\log n$) performance of the algorithm with a constant number of parcel types. For this purpose, a set of five parcel types was generated using the same dimension range as before. To show the time performance of the algorithm as the value of n changes, the size of the container must be varied, since n is the volume of the container divided by the volume of the smallest parcel. The ratio of the sides of the container were kept constant at 4 : 2 : 3 (x:y:z). For each value of n, the required container dimensions were calculated in this ratio based on a container volume of n multiplied by the volume of the smallest parcel.

Figure 7 demonstrates the time performance of the algorithm as the value of n is increased, and its proximity to the graph of ($n\log n$ / 800). The deviations in the graph result from the changing resultant packing layouts. Different layouts generate different numbers of potential positions and different heuristic tree sizes. Importantly, the container utilisation achieved starts at 81.7 % and climbs above 90% at n=3500, where it remains for the higher values of n.

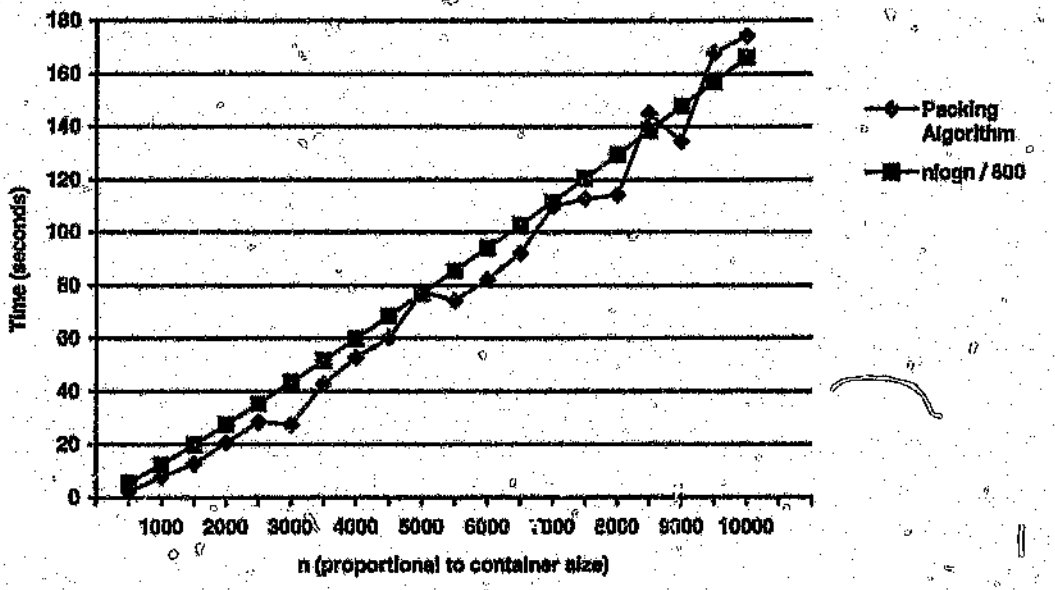


Figure 7 : Time performance with increasing container size

5.4 Comparative Test Results

The final test data set uses the same data set parameters used in [Bischoff et al 1995], and provides comparative performance information. Two problem types are used, both using a fixed container size of 1200 by 1000 by 1500. Problem type I uses parcel dimension ranges of [200,400] for *x*, [150,350] for *y* and [100,300] for *z*, while problem type II uses the wider ranges of [150,450] for *x*, [100,400] for *y* and [50,350] for *z*. For each problem type, the following values for p_{types} are used : 3, 5, 8, 10, 12, 15 and 20. The test results presented in [Bischoff et al 1995] are based on a test set of 100 test cases per variation. Tests showed that different randomly generated sets of 100 cases using the same parameters gave performance results differing by a few percentage points. To reduce this problem, data sets of 1000 cases were used instead.

Figures 8 and 9 show the comparative results. It should be noted that the results in [Bischoff et al 1995] are given in graph format, and that the estimates of these results shown in Figures 8 and 9 may be slightly inaccurate, but should be accurate to within one percentage point. For further comparison, the [Bischoff et al 1995] results show that their algorithm outperforms the George and Robinson algorithm in all cases.

Both graphs clearly show that the presented algorithm outperforms the Bischoff et al algorithm in all but one case. It is important to note that the Bischoff et al results become worse as the number of parcel types increases, whereas the presented algorithm's results become better. This is a clear indication that the presented algorithm is better suited to cases where a large number of parcel types is being used. This is as expected, since the Bischoff et al algorithm employs a layering technique in which layers consisting of up to two parcel types are formed. A greater variety of parcel types makes it difficult to 'form large blocks/walls of one single box type' [Bischoff et al 1995]. Also more parcel types increase the likelihood of the packing surfaces becoming fragmented.

Another important observation is that the Bischoff et al results become worse when moving from type I to type II problems. Once more this is a result of the layering technique, since a greater variety of parcel types makes it more difficult to build efficient layers, and also results in packing surface fragmentation. The presented algorithm's results, however, are better for type II problems. This is an indication that the algorithm is using the potential which a greater variety of parcels has for utilising awkward remaining empty space.

The average run time per test case ranged from 0.8 seconds for n=2 up to 5.5 seconds for n=20, an increase by a factor of less than 7. This is compared to the [Bischoff et al 1995] algorithm which ranges from 2.0 seconds up to 90 seconds, an increase in time by a factor of 45. This demonstrates the importance of a low order of complexity.

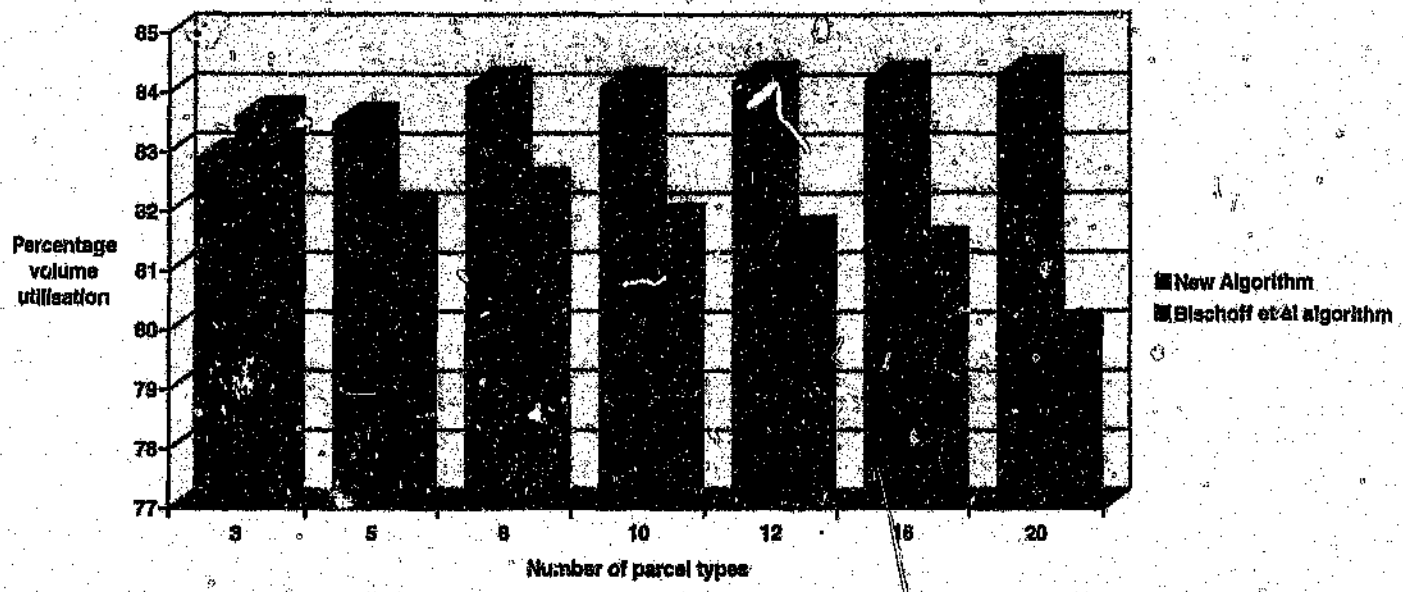


Figure 8 : Comparative test results - problem type I

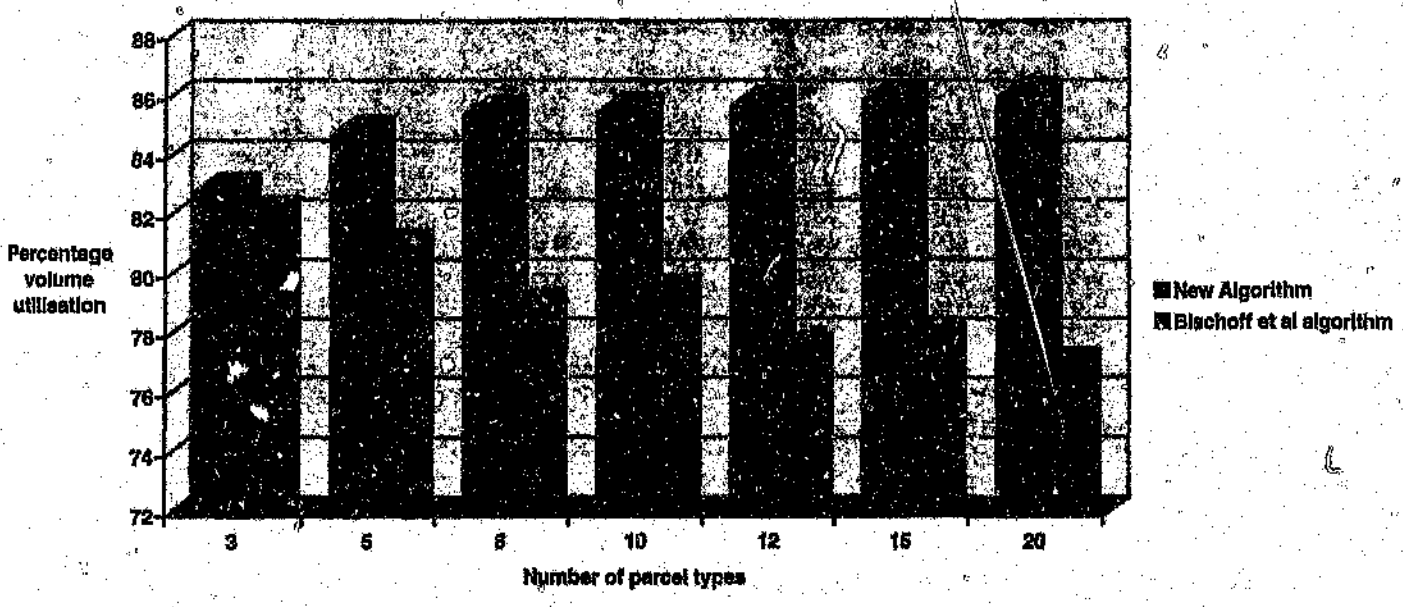


Figure 9 : Comparative test results - problem type II

CHAPTER 6 - FUTURE WORK

This chapter outlines ideas for future research resulting from the work performed in developing the algorithm presented, and the subsequent discussion above.

6.1 More efficient balanced tree implementation

The red-black balanced-tree implementation used deals with deletions by simply marking the relevant node as deleted. This creates many useless nodes in the tree, and a more sophisticated deletion method would improve the efficiency of the algorithm in terms of memory usage and time.

6.2 More readable stowage plans

The stowage plans generated by the algorithm are not always suitable for practical use, since they may be too complicated or 'untidy' to be understood or seriously considered. It these situations, it may be preferable to sacrifice volume utilisation for a more regular, neater result. One possible approach is to give higher heuristic values to positions in which the parcel shares corner positions with and has the same orientation as other parcels of the same type. Another approach is discussed in the next section.

6.3 A blocking approach

The second stage of the non-discrete drop-search algorithm discussed in Section 2.2 can be adapted and applied to the new algorithm. This would require many deletions from the heuristic tree at regular intervals during the operation of the algorithm. With reference to the discussion in section 6.1, a more sophisticated balanced tree dealing with deletions in a more efficient way should be implemented before this stage is attempted.

The second stage of the non-discrete drop-search algorithm aims to place parcels of the same type in large blocks to produce tight, efficient packings, and readable stowage plans that are more practical in real situations. Due to this block formation approach, the second stage is more likely to perform in a similar manner to the layering approaches of Bischoff et al and Gehring et al discussed above. Thus, this stage would probably perform better for fewer parcel types, since the spaces generated tend to be less fragmented and more suitable for large blocks of parcels. In this way, the second stage may bring the results for fewer parcel types up to the standard achieved for higher numbers of parcel types.

6.4 Modifying the heuristic

A number of modifications to the algorithm have been briefly experimented with. One such modification is to always include the positions immediately around a placed parcel as potential positions, regardless of whether support is found in the x and y dimensions (support must still be present in the z dimension). The idea here is to promote tight blocks of parcels, which should improve the performance of the algorithm for fewer parcel types, and help promote more readable stowage plans. A similar idea is to always choose a position around the most recently placed parcel, as long as the heuristic value associated with the best of these is greater or equal to the position's just used. The 'put-flat' heuristic discussed in section 2.2 is a further modification that could be made. Once more this heuristic is specifically aimed at packing lists of fewer parcel types.

Clearly there are a number of modifications that could be made which should improve the performance of the algorithm for fewer parcel types.

6.5 A compound heuristic

The tie-breaking methods used by the algorithm are open to variation, and a number of additional measures have been discussed above. There was some indication that different combinations of these measures produced significantly different results for individual test cases. This suggests that a parcel list be packed two or three times using different heuristic variations, with the best result being chosen. Another possibility, more suitable when response time is important, is to choose suitable heuristic parameters according to the parcel list (e.g. use the blocking approach discussed in section 6.3 if there are fewer parcel types).

6.6 Human-computer co-operative work

The algorithm of Gehring et al relies on the packing overseer to pack parcels into spaces resulting from the combination of adjacent spare spaces [Gehring et al 1990]. In this way the overseer and the algorithm work together to provide the final solution. Dowsland and Dowsland conclude that any good solution technique for packing must incorporate a decision support system (DSS) [Dowsland and Dowsland 1992]. This will allow individual users to tune the performance of the algorithm according to specific requirements and special cases.

The work of Aspoas and Dorman incorporates human intervention and a decision support system in the form of a heuristic value reported to the user when attempting to place a parcel manually, and shows how the user and the heuristic can work together to provide better solutions than either can by itself [Dorman 1992][Aspoas 1992]. A further aspect of this system is a visualisation tool which incorporates graphical representations of the current packing, parcels being moved around the container by the user, and of empty space which may be difficult for the user to visualise [Dorman 1992].

The new algorithm can be extended and integrated with the existing system discussed above. The fact that the algorithm places one parcel at a time means that the human would be able to intervene at any time, with the algorithm continuing after modifications have been made. The state of the p-cubes and heuristic tree would have to be maintained during the human intervention phase, so that the algorithm could continue at any time. The algorithms developed above contain the required functionality to perform this task. A further extension of the decision support system could allow the user to search through all the possible positions stored in the heuristic tree, in order of preference, and to make the desired choice.

CHAPTER 7 - CONCLUSION

The new packing algorithm presented has clearly met its objective of being an efficient, fully three-dimensional algorithm suited to 'many parcel type' problems. The results for few parcels types are also very favourable, and the algorithm can be considered as suitable for any number of parcel types. The comparative test results, using recently published results, show that the algorithm provides excellent container utilisation.

The complexity of the algorithm has been reduced to $O(kn\log n)$, and actual run times have been greatly reduced making the algorithm suitable for large packing problems.

Probably the greatest drawback of the algorithm is that 'untidy' packing layouts may result for problems with many parcel types. To what extent this is inherent in the problem, is unsure.

Although suitable as a stand-alone system, the algorithm has maintained it suitability for human interaction, due to the sequential placement of parcels and the ability to continue from any given point during a packing activity.

REFERENCES

1) Aspoas S G, The Drop-Search Heuristic Algorithm for Container Packing, *Honours Research Report*, Computer Science Department, University of the Witwatersrand, South Africa, 1992.

2) Baker B S, Brown D J and Katseff H P, A 5/4 Algorithm for Two-Dimensional Packing, *Journal of Algorithms*, Vol. 2, pp. 348-368, 1981.

3) Beasley J E, An exact two-dimensional non-guillotine cutting tree search procedure, *Operations Research*, Vol. 33, No. 1, pp. 49-64, 1985.

4) Bischoff E E, Janetz F and Ratcliff M S W, Loading pallets with non-identical items, *Euro. J. Opl. Res.*, Vol. 84, pp. 681-692, 1995.

5) Bischoff E E and Dowsland W B, An application of the micro to product design and distribution, *J. Opl. Res. Soc.*, Vol. 33, pp. 271-280, 1982.

6) Bischoff E E and Marriott M D, A comparative evaluation of heuristics for container loading, *Euro. J. Opl. Res.*, Vol. 44, pp. 267-276, 1990.

7) Bond A H and Gasser L, *An Analysis of Problems and Research in DAI*, Chapter 1 of Readings in Distributed Artificial Intelligence, Bond A H and Gasser L (eds.), Morgan Kaufmann, California, 1988.

8) Coffman Jr, Garey M R and Johnson D S, Approximation algorithms for bin packing - An updated survey, in : Austello G, Lucertini M and Serafini P (eds.), Algorithm Design for Computer Systems Design, Springer-Verlag, New York, pp. 49-106, 1984.[1]

9) Dorman R H, Interactive 3D Visualisation Tool For Nesting, *Honours Research Report*, Computer Science Department, University of the Witwatersrand, South Africa, 1992.

10) Dowsland K A, An exact algorithm for the pallet loading problem, *Euro. J. Opl. Res.*, Vol. 31, pp. 78-84, 1987.

11) Dowsland K A and Dowsland W B, Packing problems, *Euro. J. Opl. Res.*, Vol. 56, pp. 2-14, 1992.

12) Dowsland W B, Three-dimensional packing - solution approaches and heuristic development, *Int. J. Prod. Res.*, Vol. 29, No. 8, pp. 1673-1685, 1991.

[1] Referenced in [Stadtler 1990]

13) Dyckhoff H, A typology of cutting and packing problems, *Euro. J. Opl. Res.*, Vol. 44, pp. 145-159, 1990.

14) Garey M R and Johnson D S, *Computers And Intractability - A Guide to the Theory of NP-Completeness*, W. H. Freeman And Co., 1979.

15) Gehring H, Menschner K and Meyer M, A computer-based heuristic for packing pooled shipment containers, *Euro. J. Opl. Res.*, Vol. 44, pp. 278-288, 1990.

16) George J A and Robinson D F, A heuristic for packing boxes into a container, Computer and Operations Research, Vol. 7, pp. 147-156, 1980.

17) Georgeff M P, Planning, *Ann. Rev. Comput. Sci.* 1987. 2: pp.359-400.

18) Goulimis C, Optimal solutions for the cutting stock problem, *Euro. J. Opl. Res.*, Vol. 44, pp. 197-208, 1990.

19) Haessler R W and Talbot F B, Load planning for shipments of low density products, *Euro. J. Opl. Res.*, Vol. 44, pp. 289-299, 1990.

20) Han C P, Knott K and Egbelu P J, A heuristic approach to the three-dimensional cargo-loading problem, *Int. J. Prod. Res.*, Vol. 27, No. 5, pp. 757-774, 1989.

21) Hirschberg D S and Wong C K, A Polynomial-Time Algorithm for the Knapsack Problem with Two Variables, *J. ACM*, Vol. 23, No. 1, pp. 147-154, 1976.

22) Ivancic N, Mathur K and Mohanty BB, An Integer Programming Based Heuristic Approach to the Three-dimensional Packing Problem, J. Mfg. Oper. Mgt. 2, pp. 268-298, 1989.

23) Li K and Cheng K, On Three-Dimensional Packing, *Siam J. Computing*, Vol. 19, No. 5, pp. 847-867, 1990.

24) Ngoi B K A, Tay M L and Chua E S, Applying spatial representation techniques to the container packing problem, *Int. J. Prod. Res.*, Vol. 32, No. 1, pp. 111-123, 1994.

25) Stadtler H, A one-dimensional cutting stock problem in the aluminium industry and its solution, *Euro. J. Opl. Res.*, Vol. 44, pp. 209-223, 1990.

26) Sweeney P E and Paternoster E R, Cutting and Packing Problems: A Categorized Application-Orientated Research Bibliography, *J. Opl. Res. Soc.*, Vol. 43, No. 7, pp. 691-706, 1992.

27) Vasko F J, McNamara J A, Newhart D D and Wolf F E, A Practical Solution to a Cargo Loading Problem at Bethlehem Steel, *J. Opl. Res. Soc.*, Vol. 45, No. 11, pp. 1285-1292, 1994.



Author: Aspoas Sean Graham.
Name of thesis: A fully three-dimensional heuristic algorithm for container packing.

PUBLISHER:
University of the Witwatersrand, Johannesburg
©2015

LEGALNOTICES:

Copyright Notice: All materials on the University of the Witwatersrand, Johannesburg Library website are protected by South African copyright law and may not be distributed, transmitted, displayed or otherwise published in any format, without the prior written permission of the copyright owner.

Disclaimer and Terms of Use: Provided that you maintain all copyright and other notices contained therein, you may download material (one machine readable copy and one print copy per page)for your personal and/or educational non-commercial use only.

The University of the Witwatersrand, Johannesburg, is not responsible for any errors or omissions and excludes any and all liability for any errors in or omissions from the information on the Library website.