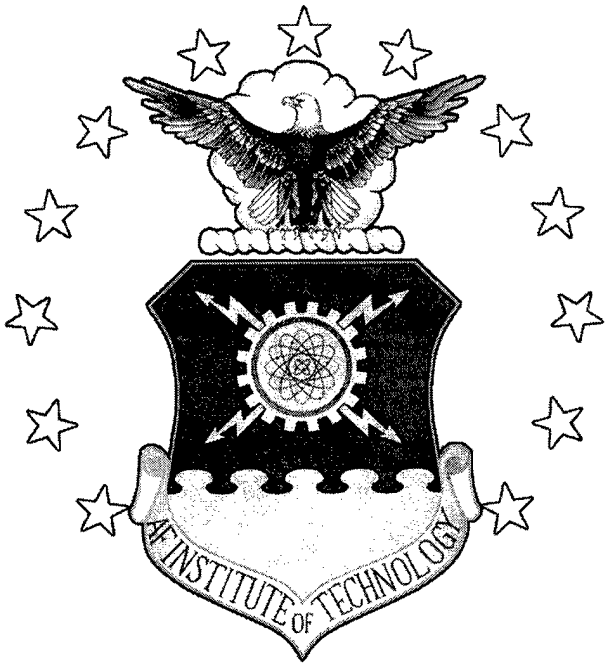


THE DISTRIBUTER'S THREE-DIMENSIONAL PALLET-PACKING PROBLEM: A HUMAN INTELLIGENCE-BASED HEURISTIC APPROACH

THESIS

Erhan BALTACIOĞLU, First Lieutenant, TUAF

AFIT/GOR/ENS/01M-02

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUDE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U. S. Government.

AFIT/GOR/ENS/01M-02

THE DISTRIBUTER'S THREE-DIMENSIONAL PALLET-PACKING PROBLEM: A HUMAN INTELLIGENCE-BASED HEURISTIC APPROACH

THESIS

Presented to the Faculty

Department of Operational Sciences

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillments of the Requirements for the

Degree of Master of Science

Erhan BALTACIOGLU, B.S.

First Lieutenant, TUAF

March 2001

AFIT/GOR/ENS/01M-02

THE DISTRIBUTOR'S THREE-DIMENSIONAL PALLET-PACKING PROBLEM: A HUMAN INTELLIGENCE-BASED HEURISTIC APPROACH

Erhan BALTACIOGLU,
First Lieutenant, TUAF

Approved:

James T. Moore, Lt Col, USAF (RET) (Advisor)	date
Raymond R. Hill, Jr., Lt Col, USAF (Reader)	date

Acknowledgements

I would like to thank my thesis advisor, Dr. James T. Moore and my thesis reader LtCol Raymond Hill for their guidance, support and being fully open to innovative ideas through this research. Their experience and knowledge helped me to have continuous progress on my thesis and to develop a successful model.

I also would like to thank the Turkish Air Force for providing me such a quality Master's program opportunity. I will take full advantage of the experience that I got in this program, in my whole life to serve my country better.

Mr. and Mrs. Graziers also deserve a lot of thanks for the support they provided me during these trying months.

Lastly, I am extra grateful to my parents and my family for all the values they taught me, which helped to complete this program.

Table of Contents

List of Figures

List of Tables

Page

Abstract

The Distributor's Pallet Packing Problem is to load a set of distinct boxes with given dimensions on pallets or in containers to maximize volume utilization. This problem is still in its early stages of research, but there is a high level of interest in developing effective models to solve this NP-hard problem to reduce the time, energy and other resources spent in packing pallets.

In its search to improve operations, the Air Force is also making an effort to solve this problem. Building an analytical model and developing a genetic algorithm approach have been tried, but the problem requires additional research and there is a need to produce realistic solutions in a reasonable amount of time.

We develop a special heuristic algorithm and code it in the C programming language. In our model, we used powerful heuristic tools and dynamic data structure to mimic human intelligence, providing a new solution approach to pallet packing. We created another program to visualize packing results. Tests on hundreds of problems show that our model makes the most of volume utilization in minimal time making it a leader among presented and published works.

THE DISTRIBUTER'S THREE-DIMENSIONAL PALLET-PACKING PROBLEM: A HUMAN INTELLIGENCE-BASED HEURISTIC APPROACH

Chapter 1 - Background and Statement of the Problem

1.1 Introduction

Everyday many items are shipped from one place to another. These items are put in containers or pallets. To ship more items while spending less energy, time and money, the items should be packed optimally, or at least near optimally. This problem becomes even more important when we start to talk about air shipping.

The Air Force uses standard HCU-6/E (463L) pallets in air shipping. The length and the width of the pallets are 88 inches (7 feet 4 inches) and 108 inches (9 feet), respectively. However, only 84 inches of the length and 104 inches of the width are actually available. Loadmasters are required to leave the outside two inches of the pallet unpacked so a cargo net can securely fit around the packed boxes. The maximum height of a pallet is 96 inches (8 feet) for pallets loaded in the main compartment and 76 inches (6 feet 4 inches) for pallets loaded on the ramp (Taylor, 1994).

By eyeballing the items to be packed, experienced Air Force loadmasters can efficiently pack Air Force standard HCU-6/E (463L) pallets. However, the questions are "How efficiently do they pack?" or "Can those items be packed more efficiently and even provide a reduction in the number of sorties required?" A scientific approach is required to answer these questions.

For these reasons, the Air Force is in search of a model that will help efficiently pack the pallets and provide loadmasters with a report stating where the boxes should be

placed on the pallet to minimize unused space. This will save time and resources. If the pallets are more efficiently packed, the number of sorties flown may decrease and aircraft may be freed to carry other items in large-scale mobilizations (Ballew, 2000).

1.2 Background

This research is a follow-on of Ballew's (2000) research. He developed an integer programming mathematical formulation of a simplified version of the three-dimensional pallet-packing problem. Some of the problem's constraints were not included in the formulation. Unfortunately, the solver package found a local optimum to a simplified and small problem (just 3 boxes). The formulation of a bigger problem with more boxes is not practical because the number of variables and constraints increase incredibly fast as the number of boxes increase. Ballew did employ a simple genetic algorithm to solve a slightly larger but still a small problem (11 boxes) but found no reasonable solution within 45 minutes.

The Air Force has sponsored research in this area on multiple occasions in search of a better way to pack the pallets and load the aircraft. These include an early effort by Taylor (1994), research on an airlift-loading model by Chocolaad (1998) and Romaine (1999), and a three-dimensional packing problem approach by Manship and Tilley (1998). None of the developed techniques are able to pack hundreds of various sized boxes on an HCU-6/E (463L) pallet while considering realistic constraints. They also do not have acceptable solution times for this problem.

Bischoff, Janetz, and Ratcliff (1995) developed a three-dimensional heuristic approach to pack multiple sized boxes on a pallet. Their algorithm packs the boxes in

layers allowing up to two different box types per layer; however, it gives preference to layers filled by a single box type. The algorithm did produce stable loads, but as the number of various sized boxes increased, the packing efficiency declined.

Martello, Pisinger and Vigo (2000) developed a branch-and-bound algorithm to solve a three-dimensional bin-packing problem. Their solution however, is not strictly three-dimensional. They first construct bin slices having width W, height H, and different depths. The slices are then combined into three-dimensional bins. However, they do not include various important constraints, and they assume that no items can be rotated. Under these relaxed conditions, their approach had good results.

Real world packing problems are complex and have many constraints. The boxes have different weights, volumes, and dimensions. The varied dimensions of the boxes can cause gaps in the usable packing space. Additional wasted space can be caused by weight balancing and box placement restrictions. Hence, in most cases, the total volume of the boxes packed is considerably less than the available volume of the container. Most research has focused on packing a certain number of boxes in a container. But in a real world problem, this is not always the case. One tries to pack as many boxes as a container can hold before moving to another container. Because this changes the domain space, considering this important point changes the approach that must be taken to get a good solution

The three-dimensional packing problem is a natural generalization of the classical one- and two-dimensional problems, and therefore it is NP-hard (Reeves 1995). This means that, in general, optimal solutions are computationally impractical to achieve. For this reason, most of the studies have focused on the practical aspects of loading a

container and developing heuristic solutions based on the concept of filling out the container with boxes organized in layers, walls, and columns. In other cases, two-dimensional pallet packing heuristics are applied to the general three-dimensional container-loading problem. These heuristics are, in general, on-line packing algorithms, which means they pack boxes one-by-one in a given order. More precisely, when the algorithm is packing a box, it has information only about the boxes previously packed, and once a box is packed, it cannot be moved to another place. This technique is not efficient and is also not applicable, when applying the load balance and other constraints.

Since the pallet-packing problem has a large solution space, it is extremely difficult to prove a solution is the global optimum. Only with many different sets of boxes can an algorithm be tested and its performance evaluated.

1.3 Statement of the Problem

The problem is a three-dimensional pallet-packing problem. There are basically two general types of pallet packing problems. They are the "manufacturer's pallet packing problem" and the "distributor's pallet packing problem." The 'manufacturer's pallet packing problem' is easier to solve since it seeks the optimum layout of identical rectangular boxes on a rectangularly shaped pallet.

For the "distributor's pallet packing problem," the objective is to load boxes of varying dimensions onto as few pallets as possible (Askin and Standridge, 1993). This problem is more difficult to solve than the manufacturer's problem. For the case in which only one pallet is loaded, the objective is to minimize unused pallet space since Air Force 463L pallets are usually "cubed out" before they are "grossed out" (Taylor, 1994).

This means that, in general, the total available volume of a pallet is filled before its weight limit is reached. Most of the time, the items packed are rectangular in shape, and this property makes the problem easier to solve, compared to trying to pack items with different shapes.

The problem is to pack as many boxes as possible from a given set of rectangular-shaped items into a three-dimensional rectangular bin. The objective is to minimize the unused bin volume while considering many different kinds of constraints. These constraints are explained in the Scope and Methodology Section. The problem is strongly NP-hard and extremely difficult to solve in practice (Martello, Pisinger and Vigo, 2000).

The purpose of this research is to develop a three-dimensional pallet-packing algorithm and an executable written code employing the developed algorithm.

1.4 Scope and Methodology

In our problem, all items are rectangular boxes. We solve this problem as a single-pallet packing problem, not as a multiple-pallet packing problem where an entire aircraft is loaded. Trying to pack all pallets to be loaded on an aircraft is more difficult to solve. The reason is that the balance of the aircraft has to be considered, and this means that the problem has to be considered as a single hierarchical problem, instead of being considered as a combination of several single-pallet packing problems.

Our single pallet problem has many constraint types. The first one is that every box takes a unique space in the pallet, with no overlapping allowed. A second type prevents packing beyond the dimension limits of the pallet. Another constraint type

actually accounts for several different restrictions. The packing must be realistic. It must be stable. No overhang is allowed when the boxes are packed. For a box to be packed, its entire base must be on top of either the pallet or other boxes. Boxes can be rotated and packed with one of six different options. Heavy boxes should be packed below lighter ones. For flight safety reasons, the Air Force prefers the center of gravity of a loaded pallet to be within four inches of the center of the pallet. (TASC, 1998)

The problem is extremely hard to solve with all the constraints stated above. Most approaches start simple and then add other constraints. Some simplifying assumptions are employed in our approach. In our approach, we initially include the first two constraint types. So that every box takes a unique space in the pallet (no overlapping allowed) and packing beyond the dimension limits of the pallet is prevented. We also allow packing of the boxes in all six orientations. Before a pallet is loaded onto a plane, the loadmaster secures the pallet by tying down cargo nets around the load. Thus, in our approach the top of the load is close to level to accomodate the cargo net. When the cargo net is thrown over a load, level at the top, boxes do not shift or fall.

We also allow overhang and unstable packing to simplify the problem, so some boxes might not have a complete foundation under them. Since Air Force 463L pallets are usually "cubed out" before they are "grossed out" (Taylor, 1994), we omitted all constraints dealing with weight and center of gravity.

We do not employ an existing heuristic technique such as genetic algorithms, tabu search, or simulated annealing. A genetic algorithm is not sufficient because it does not appear practical to consider all of the stated constraints while performing a multiple crossover and expect termination in a reasonable time. Representing a solution in a

genotype form with hundreds of boxes turns out to be a very long gene, which excessively increases the storage space and the computational time. Tabu search is not applicable because one cannot define the move that we need to apply to our problem. Since the boxes have various dimensions, it is impractical to define the move as a swapping of locations of two boxes. Simulated annealing is a stochastic heuristic technique which has been employed by Faina (Faina, 2000) and it is not, in our opinion, intelligent enough for this problem type. When we try to pack a large number of boxes, we have a very large solution space and this renders stochastic techniques very inefficient.

We create our own problem-specific heuristic technique using a composition of the tools that other heuristic methods use. We develop an adaptive heuristic algorithm modeling human intelligence. We employ an algorithm that combines an adaptation of the human intelligence with extensive data analysis applied to the candidate boxes by taking the advantages of some smart programming tools and data structures. We write the three-dimensional pallet-packing algorithm in the C programming language. We test it with several different sets of boxes, and apply a validation and verification process.

1.5 Overview

Chapter Two presents a detailed review of past work and some solution techniques developed to solve three-dimensional packing problems. Additionally it presents information about the most promising commercial three-dimensional pallet packing software packages.

Chapter Three describes in detail the heuristic algorithm we developed along with the data structure and other programming tools used. Chapter Four presents solutions generated by our approach for different sized and different featured problems while discussing and comparing the qualities of each solution with previously published solutions. Chapter Five provides conclusions and recommendations for future work.

Chapter 2 – Literature Review

2.1 Introduction

The packing problem is a problem with many different variants. The early form of this type of problem was the one-dimensional packing or partitioning problem, in which a set of n positive values w_j, e.g. weight values, must be partitioned into the minimum number of subsets so that the total value in each subset does not exceed a given bin capacity W.

The two-dimensional bin-packing problem extends the one-dimensional bin-packing problem. Instead of considering only one set of positive values, we consider two different sets of positive values, namely two different dimensions, e.g. width and length of the rectangular pieces to be cut out of big industrial plastic film. As expected, this problem is harder to solve than the one-dimensional bin-packing problem.

These packing problems are NP-hard problems. NP stands for 'non-deterministic polynomial'. NP-hard means the solution time increases exponentially as the size of the problem increases. The three-dimensional bin-packing problem is strongly NP-hard because the three-dimensional bin-packing problem is a special case of the one-dimensional bin-packing problem (Martello, Pisinger and Vigo, 2000).

2.2 Previous Research

Packing problems have been considered by a number of researchers, but past work was largely restricted to the one- or two-dimensional cases. It is widely agreed that, due to its complexity, any analytical solution to this problem is unlikely in the

foreseeable future. As a result, most those successful approaches to the problem have taken the form of heuristics.

The volume of published work discussing three-dimensional solutions is still very limited due in part, one suspects, to the level of complexity involved, and there appear to be no clear measures of success (or failure). This might be expected in light of a survey of such analysis for the two-dimensional problem by Coffman and Shor (1990). This survey concludes that the field is still in the early stages of development—algorithms and probability models tend to be simplistic, and estimates of performance are far more common than exact measures. As the three-dimensional case is less well-studied and more complex, it is not surprising that the published work generally presents successful implementations, but it fails to provide the reader with any clear measure of scientific success. They do, however, provide some interesting insights into the various views on how successful packings are best achieved. Most of the approaches are based on a basic wall-building concept although, as described below, this is achieved in a variety of different ways.

Among the earliest publications are those of Tinarelli and Addonizio (1978) and George and Robinson (1980). The former of these papers addresses the problem of minimizing the number of containers used for transporting a given cargo. Identical items are grouped together and layers are developed. These generally take a simple block form. George and Robinson (1980), in the first English language paper to directly address the container loading problem, describe in detail an algorithm developed to load a container with cargo consisting of a number of distinct types (sizes) of boxes. They utilize a fairly sophisticated 'wall building' approach in which sections of the container across the full

width and height are packed. Such an approach in many ways mirrors how the real-life packing of containers is carried out, and ensures that cargo of the same type is largely kept together. They describe the implementation of their algorithm for a cargo of 20-box types. A variety of solutions may be obtained by commencing packing with different box types and utilizing different orientations. When there are insufficient boxes to complete a wall utilizing one box type, spaces are generated above and to the right of the wall and these are packed utilizing a space filling procedure. At all times, the method attempts to retain a flat forward packing face. The procedure endeavors to keep boxes of like type together by defining 'open' and 'closed' box types, and it also accounts for box orientation constraints. Open type boxes are certain box types that are already started to be packed, while closed type boxes are the boxes of the types that are yet to be used.

Bischoff and Dowsland (1982) had an approach also based on the principle of filling the container by building layers across its width. However, there are two main differences between their procedure and that of George and Robinson: first, each layer is constructed only from a single type of box; and second, the arrangement of boxes within a layer is determined through a two-dimensional packing procedure, which aims to maximize the area utilization of the cross-section (i.e. of the rectangle formed by the width and height of the container). This two-dimensional pallet-packing procedure is a heuristic and was originally proposed as an approach for calculating efficient layout patterns for boxes on a pallet.

The filling of spaces in a layer is not considered in the procedure proposed by Bischoff and Dowsland (1982) and therefore the order in which the layers are formed has no influence on the packing efficiency achieved. The criterion used to decide the depth

dimension of a layer, however, is of crucial importance. Each of the three sides of a box is examined in turn as a potential depth dimension for a layer. With this dimension fixed, the maximum number of boxes, which can be accommodated in the cross-section, is then determined by means of the two-dimensional packing routine. In other words, if the container width and height are denoted by W and H, and the three box dimensions are *w*, *l* and *h*, respectively, with, say, *w* being currently considered as the depth of the layer, the figure calculated is the number of rectangles *l* X *h* which fit into the rectangle W X H. If this is greater than the number of boxes of that particular type still to be loaded, a full layer cannot be formed and the depth dimension concerned is consequently dropped from further consideration. If, on the other hand, there is more than one possible depth-wise orientation yielding a complete layer, a choice needs to be made between them. One obvious criterion for making this choice is the percentage fill of the cross-section (in either volume or area terms). Following another rationale, however, it might be desirable to attempt to maximize the number of boxes which are accommodated in complete layers. It could therefore be advantageous to select the orientation, which leaves the least number of boxes unpacked once as many identical layers as possible have been constructed.
At some stage in this procedure, a point will be reached where either all boxes are packed in complete layers or -as is more likely- it is not possible to form further complete layers of any box type. In this case the remaining boxes are packed using the George and Robinson (1980) approach.

Mention has already been made of the fact that in many cases several different variants of the same heuristic can be devised. George and Robinson (1980) explicitly point out the two alternative rules for choosing among open box types. In a set of trial

runs discussed in George and Robinson's (1980) paper, the results obtained by using the second rule -which orders open types according to the same principle as closed types- were slightly better, but the authors make no general suggestions as to which criterion should be used for more efficient packings. Test runs with a different priority key for unopened box types are also referred to in their paper. Here the authors conclude, "that the original priority structure more often gives the best results". However, no precise details are given of how much better the results obtained were and, vice versa, how much worse they were when another ranking rule produced the best solution.

A wall building approach is a natural simplification of the problem. It forms an important component of the algorithm described by Bischoff and Marriott (1990), and has been adopted by a number of other authors. Liu and Chen (1981) also present a wall-based algorithm in which they consider the different ways in which valid box orientations may be used to maximize the widthwise utilization of the container. Having assigned the wall base, a similar approach is applied to the container height. Gehring, Menschner and Meyer (1990) also present a heuristic for packing non-identical items within a container. They, like George and Robinson (1980), utilize the idea of packing sections of the container across the full width and height. They utilize an ordering based on decreasing volume, and having placed the first block in a section (layer), the layer determining box (LDB), they develop a packing across the container floor first and then upwards. This tends at first to produce something of a decreasing wedge across the width of the container. The authors report that good solutions are obtained, but they only present results for its application on two problems. They too ensure that they retain a flat front packing wall but differ from George and Robinson (1980) in that they prohibit boxes

from straddling adjacent layers. As they state, this approach does ensure that cargo sections can be moved around so as to provide appropriate weight redistribution, but it will clearly lead in some instances to reduced volume utilization. A further aspect associated with not allowing boxes to straddle sections is that of load stability. Packings where boxes do straddle between layers can produce a more cohesive load.

Han, Knott and Egbelu (1989) show that the idea of walls need not be restricted to the vertical sides of the container. They describe an algorithm in which the container (major prism) is packed with identical boxes (minor prisms). The algorithm as described is designed for only a single box type that is constant in both size and shape and no practical constraints are considered. The approach is to produce packings of L-shaped modules, with the initial module considered spanning the whole of the container base, and one of the container walls. The arrangement within the 'L' is determined by dynamic programming (similar to the approach of Steudel, 1979), which maximizes the edge utilization. The idea of building walls along any of the six faces of the container is an interesting one; however, the example they use fits one less box than that obtained by stacking multiples of two different 'wall' arrangements on the floor of the container. The weakness in the approach of Han *et al.* (1989) is a result of maximizing the utilization of the perimeter of the 'L' module. No evidence is presented to suggest why an L-shaped module approach should be adopted. (Their example consists of packing a container of size 48" by 42" and 40" with boxes 11" by 6" by 6". They are able to fit 195 boxes, a 95.16% volume utilization of the container. They quote the US General Services Administration whose published results (1966) for the same problem only provide 82.5% utilization).

Bischoff and Marriott (1990) did a case study on the development of composite heuristics, without proposing any solution procedures. The case study has reported and discussed the results of an analysis of 14 heuristic procedures for producing efficient container loading patterns, based on a two-dimensional packing technique. It has been shown that the performance of such heuristics may be very domain-dependent and, more particularly, may vary crucially with the number of different items in a load. The approach described was demonstrated to be superior, but only for certain types of problems.

The Bischoff and Marriott study used an idealized problem formulation in which cargo weight, materials handling aspects and other such factors were not considered and the objective was defined as minimizing the container length needed to accommodate a given cargo. The situation handled in the paper is different from merely determining a feasible arrangement for stowing all of a given cargo. The results of the study suggest that the most efficient order is likely to depend on the number of different box types in the cargo under consideration.

The heuristic method suggested by Haessler and Talbot (1990) is based primarily on the assignment of boxes to stacks, these then being placed across the container. Although this is done in part in order to simplify the problem, an approach based upon sectioning the packing face into two or more components, 'sub-walls', which fall between the ideas of stack and 'full-walls', may be worth pursuing. Haessler and Talbot's heuristic also utilizes the fact that they allow the adjustment of order quantities so as to make best use of the space available.

Mohanty, Mathur and Ivancic (1994) proposed a multi-dimensional knapsack problem approach to the three-dimensional packing problem dealing with filling up various containers with boxes. Their objective was to maximize utilization of the space in the containers or the value of the contents of the containers. They present a column generating procedure which heuristically uses a 'greedy approach' to generate columns one at a time, without considering any constraints other than overlapping and dimensions of the containers. Since they use a 'greedy approach', their approach is not robust and is strongly affected by the number of different items to be packed.

Chen, Lee, and Shen (1995) presented a zero-one mixed integer linear programming model for the general three-dimensional container-loading problem. The problem involves packing a set of non-uniform cartons into unequal-sized containers. The model considers the issues of carton orientations, multiple carton sizes, multiple container sizes, avoidance of carton overlapping, and space utilization. Several special container loading problems such as selecting one container from several alternatives, weight balance, and variable container length were addressed. The modifications to the general model needed for these situations were also provided. Very small-scale example problems (with only 6 cartons) were illustrated to validate the models. For further development, additional constraints can be introduced to the models to include other concerns in the container-loading problem such as the stability of the packing pattern, stackability, the integrity of each carton type, and weight restriction. Unfortunately, this work presented only an analytical model. Using this model, it is impossible to solve a real world problem, since the number of variables and the number of constraints increase quadratically as the number of cartons increases (it grows as $2n^2$, where n is the number

of cartons). Also, in the conclusion section, the author says that a more efficient solution procedure is needed to solve large-scale container loading problems.

Bischoff, Janetz and Ratcliff (1995) developed a model producing a high degree of stability. The basic concept underlying their algorithm is simple: The loading pattern is constructed from the bottom upwards using single layers of up to two different box types at a time. The choice of box type(s) and orientation(s) is governed by the resulting utilization of the loading surface onto which a layer is to be placed. They developed and tested three algorithms with slight modifications: the first one packs up to two different box types per layer, the second one packs up to two different box types but of the same height, and the third one allows packing only one box type per layer. They solved 1400 different problems and compared three algorithms' solution efficiencies. Their most efficient model was the first one, but as the number of different box types increases, the solution quality declines.

Terno, Scheithauer, Sommerweiß and Riehme (1997) employed a different heuristic algorithm. In addition to the dimension and overlapping constraint, they take total weight limit of the pallet and the stability constraints into account. They basically employed a layering approach while packing each layer by using a branch and bound solution method. They solved 700 problem sets among the problems that Bischoff *et al.* (1995) solved and made comparisons with past work. Their solutions were better than Bischoff *et al.*'s solutions, but since their model was mainly designed for the "Manufacturer's Pallet Packing Problem", as the number of different items increases, the volume utilization declines.

Martello, Pisinger and Vigo (2000) present a branch-and-bound method to solve the three-dimensional packing problem. They tried to orthogonally pack all the items into the minimum number of bins. A computational test is presented showing that problems with the number of boxes less than 30 and 50 were solved. When the average number of items per bin gets bigger, the problem becomes harder to solve. Another weakness was that they assumed that the items may not be rotated. They considered only basic type of constraints (overlapping and bin dimension limits).

Faina (2000) developed a geometrical model that reduces the general three-dimensional packing problem to a finite enumeration scheme. Cartons were loaded only on volume restrictions; no other restrictions were considered, and the boxes can assume any of the six possible orthogonal orientations. He pointed out that adding more constraints could possibly result in the algorithm giving worse results. He also says that the use of an approximation algorithm, which derives from a truncation of a global optimization algorithm, is, from all points of view, better than a mere heuristic procedure, but he does not substantiate this statement. He developed a simulated annealing algorithm called $zone_{3d}$. The results of many statistical tests were given with different numbers of boxes. He had many approximations in his algorithm and due to these approximations, he explained that the algorithm is not guaranteed to find a global minimum of the wasted container volume.

Faina (2000) starts his algorithm with the first box placed at the origin. By induction, it is supposed that the i^{th} box has been located; now the method of zones locates at most 2n+1 points where the $(i+1)^{th}$ box could be located, and this point is chosen at random; and so on. In this way the initial configuration is obtained. Then the

algorithm performs a small perturbation to this initial configuration by altering slightly the order of the boxes, for instance by changing the positions of any two boxes at random, and by constructing a new configuration.

With up to 32 boxes, Faina could choose cooling schedules which support at best the quality of the final placings; but with over 64 boxes, the effort to improve the final placing is too unfavorable from a computational point of view. Therefore, the results obtained get worse as the number of boxes increases.

Ballew (2000) developed a mathematical formulation similar to the analytical method of Chen, Lee, and Shen (1995), by using nonlinear integer programming on a simplified version of this problem. He presented a general mathematical formulation. Unfortunately, when implemented, the solver package Hyperlingo found a local optimum to a very simplified and small problem of just three boxes without considering several important constraints. The formulation of a bigger problem with more boxes was unrealistic because the number of variables and constraints increase incredibly fast as the number of boxes increase.

Alternatively, Ballew employed a simple genetic algorithm with single crossover to solve a small sample (only 11 boxes) problem using the genetic algorithm software library Genesis. The length of the genotype for such a small problem was 1,232 bits. Unfortunately, Genesis did not show any signs of convergence in a reasonably amount of time. Ballew concluded that one reason for this might be that Genesis only allows for single-point crossover, which is too simplistic for this problem. After 1,000 generations, which took about 45 minutes, the best solution did not come close to a feasible packing of 11 boxes (Ballew, 2000).

Ballew speculated that one potential method for determining the number of crossover points is to base it on the number of boxes. We doubt this approach would succeed because, increasing the number of crossover points does not necessarily mean that a genetic algorithm will provide better solutions. There is always a strong possibility of achieving a deteriorated solution set if we increase the number of crossover points.

In Chapter One, we pointed out the importance of the efficient packing of a pallet especially for the Air Force. For those reasons, along with the research done at the Air Force Institute of Technology (AFIT), the Air Force has contracted research on problem tthrough Computer Science Corporation (1997) and TASC, Inc. (1998).

In its research, the Computer Science Corporation concluded that an analytical model giving a global optimum might not be possible, but with heuristic techniques, it is possible to develop an algorithm giving near optimum solutions in a reasonable time. Therefore, they stated that a software package using such a heuristic algorithm could be developed (Computer Science Corporation, 1997).

At the end of TASC's research, they developed a software package in C++ language considering some of the required constraints of the problem to test the performance of the algorithm they developed. After performing some tests, they concluded that they could develop a software package including other required constraints, providing better solutions than those software packages found on the market. They suggested it would require about six months to develop and the cost of such a product would be $150,000 (TASC, Inc., 1998).

2.3 Existing Commercial Software Packages

There are a few companies that have developed packing software packages. We found two companies with commercial products.

One of the companies is CAPE Systems, Inc. (2000) offers two different types of products. One is called Truckfill. The program provides help on planning, creating, editing, viewing, printing and maintaining Multi-Product load plans for trucks, container loads and railcars. On their website, it said that the package uses the latest optimization techniques and can quickly build truck and container loads based on realistic loading rules and restrictions, but there is no other specific information. We sent an e-mail asking about the technique they have used to develop the package, but they replied that all available information is on their web site. Also Truckfill recalculates load counts, load weight, load dimensions and the center of gravity of the total load after any changes are made to the Load Plan.

Cape Systems, Inc. (2000) other product is Cape Pack'99, which is a packaging design and pallet loading software. There are basically three main groups in the software package: Design Group, Arrange Group and Pallet Group. The design group resizes an existing primary pack or designs a completely new product. Starting with a proposed size and specifying the scope for dimensional and/or volumetric changes, the program can establish what it calls the best possible primary pack size. The arrange group creates new case sizes for existing primary pack sizes. These first two program groups are not appropriate for our problem.

The third program group in Cape Pack '99 is the one that is related to our problem; the Pallet Group. It generates a range of numerical and graphical solutions for

loading objects onto pallets and into trucks. Pallet layers and cases can be added or deleted and pattern layouts can be rearranged with the drag and drop Layer Editor. This group also includes Display Pallet for loading different size products onto the same pallet. Using basic information about the size and weight of the outer packaging, the type of pallet and the maximum height and weight of the pallet load, these programs generate numerical and graphical solutions for loading packages onto pallets and into trucks. In this program group, one can calculate up to three different pallets simultaneously and work with cases, trays, bags, cylinders, ovals and trapezoidal shapes. This module allows the user to load products of up to 40 different sizes. Unfortunately, there is a restriction on the dimensions of the container: each dimension must be less than 100 inches, which is less than the length of the 463L pallet of 108 inches. Depending on the number of different sized boxes to be packed, the solution time is around 1 minute. Although we do not know what kind of optimization technique it uses, the program uses some algorithms for packing loads in layers or columns.

Another commercial packing software is Cube-IQ, which is the product of Magic Logic Optimization, Inc (2000). This software package is a product of Remarkable Software Company, and from its specifications, it appears it might be of use to the Air Force.

Cube-IQ is very flexible, and allows the containers (trucks, pallets, crates) to be either rectangular, or have a non-flat roof or floor, such as sliced-off corners of airline containers. An overall weight limit is taken into account and the system automatically handles axle weight limits. There is also an option for the correct positioning of the center of gravity. Cube-IQ optimizes over multiple containers, optionally in multiple

sizes. It also allows the items to be rectangular, cylinder and 'sofa' (3D L-shapes). Other box data include switches for 'turnable', 'allowed on its side', 'allowed on its end', 'bottom-only', and 'top-only' (possibly in maximum number of layers). Box weight is taken into account. Cube-IQ supports loading and stacking rules for each orientation of the package separately. This allows the user to set up more complex loading rules, such as 'flat only if on top' (for large, but flat boxes), and 'straight up unless on top' (for boxes that can only support other boxes if they are loaded upright).

It produces an output of the volume and weight capacity utilization for all loaded containers, and for each loaded package the container number and, within that container, the 3-D loading coordinates. Cube-IQ has a graphics window in which the user can see the container as it is loaded. The picture can be rotated, and it will build up one block of boxes at a time. It gives a print out of complete manifests and loading instructions, showing for each block of similar boxes exactly where it is to be loaded, including pictures. Its runtime is fairly short: From one second on a Pentium III-500 PC for a case with 25 loaded parcels, to one minute to load hundreds of boxes.

It seems like it is a good optimization package but there is little insight into the techniques Cube-IQ uses to pack items. Since it finds a good solution, not the optimal solution, it likely uses a heuristic algorithm. It is also not clear whether or not Cube-IQ ensures heavier boxes are packed towards the bottom.

Chapter 3 – Methodology

3.1 Introduction

We have developed a robust pallet-packing model that solves challenging problems in short times while finding near optimal to optimum solutions. In this chapter we present our model in detail; the input and output data, the data structure, and the heuristic method that we developed are discussed.

Ravindran *et al.* (1986) present the principles of modeling. The first four principles they present are key to successful model building:

1. Do not build a complicated model when a simple one will suffice,
2. Beware of modeling the problem to fit the technique,
3. The deduction phase of modeling must be conducted rigorously, and
4. Models should be validated prior to implementation.

We followed these principles while building our model. We started simple but challenging enough to solve big problems in a reasonable amount of time while relaxing some constraints as we pointed out in Section 1.4.

As the second principle advises, we did not try to force fit an existing solution technique onto our problem. We examined past related work in depth to determine the efficiencies of existing solution techniques.

3.2 Overview

Based on this start, we developed a special heuristic for our problem and implemented it in the C programming language. We performed debugging, verification,

and validation to produce a software package that packs a number of given boxes into a given container considering only the restrictions presented in Section 1.4. Our model packs as many boxes as possible in a given container while selecting the suitable boxes from a given box set. This property makes our approach more realistic. The model is also able to pack rectangular boxes in any orientation. Actually, our model not only packs rectangular boxes in any orientation, it also packs according to different orientations of the pallet. In other words, it builds walls or layers along any of the six faces of the given container if all three pallet dimensions are not the same.

The model basicly builds a new packing during each iteration. Our approach does not limit the number of different boxes in each layer. It may pack any number of different boxes within a layer if their surfaces make a good match to reduce the unpacked gaps within the layer. This property makes it robust.

Our heuristic employs both layer packing and wall building approaches. There are some other important methods that our program uses to pack boxes efficiently and quickly. One of them is packing a sublayer into any of the available unused space in the last packed layer, which we call a layer-in-layer packing approach. Another new, and the most important, feature of our heuristic is an adaptation of human behavior and intelligence to decide which box to pack. Considerable improvement also comes from the data structure we employ. For verification, validation and debugging purposes, we also have written a program to visualize the best solution found. Since the output of our program contains x, y, z coordinates and x, y, z dimensions of the packed orientation of each packed box, it is difficult to manually check a solution to see if there is any

constraint violation or numerical error. To make that process easier and to increase the presentability of solutions, we display the best solution in a box-by-box format.

3.2.1 Human Intelligence

Tabu search developed by Glover (1986) was motivated by Glover's observation of how people approach problem solving. Faced with the tough pallet-packing problem, we sought, and found similar motivation. We call our approach an adaptation of human intelligence because we pack boxes onto the pallet just like a human, from bottom to top or by building walls. This actually ensures a level top of the load for the cargo net. Humans prefer to pack boxes to reduce packing surface irregularities. We eyeball the dimensions of any gap to be packed and pick the most suitable boxes to keep the topology as smooth as possible. Our heuristic does the same thing. While packing a layer, it attempts to retain a flat forward packing face. In each step, the dimensiones of the gaps to be filled are determined, all the eligible boxes and their orientations are analyzed, and the best fitting box is selected and packed. Before starting to pack any layer, it analyzes all unpacked boxes to pick the most suitable layer thickness to reduce wasted volume. However, the selected layer thickness is flexible and might be increased to accomodate the height of the selected box. Finally, a human will also fill uneven layers, if possible. This led us to our layer-in-layer approach.

Our layer building technique is so effective that after a few iterations, the model can be stopped with the solution quality only about 1-8% less than the volume utilization of the best solution. This feature is effective especially when we deal with a large number of boxes and computing time is scarce.

3.2.2 Attributive Memory

An important tool we use is attributive memory. We use attribute-based memory to avoid using a huge amount of memory to save evey solution produced during the solution process. Our model keeps only three parameters of the solution found during iterations. These attributes are the pallet orientation of the solution, another one is the starting layer thickness value, and the last one is the volume utilization of the solution. The attributes of the best solution are then used to reconstruct that best solution.

3.3 Input Data

All box dimensions and the pallet dimensions are read from a text file specified by the user. (.txt ectension assumed). This file must be in the same hard drive folder where the program is executing. The user must follow the necessary format of the input file.

Figure 3-1 provides an example of an input file:

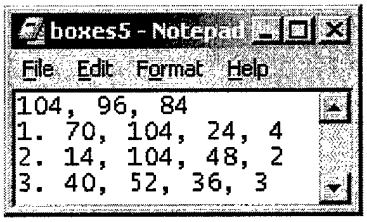


Figure 3-1: Input File Format

In the first line, the three numbers are the x, y, and z dimensions of the pallet. The order of the dimensions is important because at the end, we convert the output data format of the best solution to the orientation of the pallet entered in the input file.

All subsequent lines contain box information. In each line, the first number is the box label and has no affect on the solution since the program does not take these labels into consideration. Box labels merely provide an organized way to input the data file. The second, third, and fourth numbers are the x, y, and z dimensions of each box type, respectively. Since the program tries all possible orientations of each box, the order of the box dimensions actually has no importance. The fifth number represents the number of boxes of the same type. Although commas between the numbers are not required, at least one space character must delimit the numbers. No error handling is included in the current program so the input criteria should be followed.

3.4 Data Structure

Data structure is a critical component of any program. Choosing the proper data structure affects both performance and solution time. For our program, it is important to reach data quickly We use two different arrays and a double linked list to accomplish this.

The first array is the Boxlist[] array, which keeps all box dimensions, coordinates of packed boxes in the container, and other necessary data. There is a total of twelve fields in each record of this array:

Element	Name	Description
1.	Packst	: Status of packing (0: Not packed; 1: Packed),
2.	N	: The number of boxes that have the same dimensions,
3.	Dim1	: The length of one of the three dimensions,
4.	Dim2	: The length of another of the three dimensions,

Element	Name	Description
5.	Dim3	: The length of the other of the three dimensions,
6.	Cox	: X-coordinate of the location of the packed box,
7.	Coy	: Y-coordinate of the location of the packed box,
8.	Coz	: Z-coordinate of the location of the packed box,
9.	Packx	: X-dimension of the orientation of the box as it has been packed,
10.	Packy	: Y-dimension of the orientation of the box as it has been packed,
11.	Packz	: Z-dimension of the orientation of the box as it has been packed,
12.	Vol	: Volume of the box (Dim1*Dim2*Dim3)

We also store the volume of each box so the model does not have to calculate it each time it needs the box volume. Fields 6-11 are meaningless if Packst value is zero, but they provide the packing information once the box is packed and Packst is set to one. Each box has a record in the Boxlist[] array.

The other array is the Layers[] array. This array stores all the different lengths of all box dimensions. Each Layerdim value in this array represents a different layer thickness value with which each iteration can start packing. Before starting iterations, all different lengths of all box dimensions along with evaluation values are stored in this array. The evaluation values (Layereval values) are calculated by the Listcanditlayers function as explained in Section 3.6.1. There are two different data fields for each record in this array:

Element	Name	Description
1.	Layerdim	: A dimension value,
2.	Layereval	: Evaluation weight value for the corresponding layerdim value.

The double linked list we use keeps the topology of the edge of the current layer under construction. We keep the x and z coordinates of each gap's right corner. The program looks at those gaps and tries to fill them with boxes one at a time while trying to keep the edge of the layer even. Each entry in the double linked list has these data fields:

Element	Name	Description
1.	*pre	: Pointer that keeps the address of the previous entry,
2.	Cumx	: Keeps the x-coordinate of the gap's right corner,
3.	Cumz	: Keeps the z-coordinate of the gap's right corner,
4.	*pos	: Pointer that keeps the address of the following entry.

During execution of each iteration, this double link list may have only one entry, or may have hundreds of entries based on the box sizes of the box set being packed. Most of the time, new entries are inserted while useless entries are removed. Therefore, a double linked list is used to handle these needs dynamically and efficiently.

3.5 Numerical Limits

These model limitations were decided considering both memory limitations for an average computer memory capacity and the nature of the realistic packing problems:

Maximum number of boxes in a box set : 5000

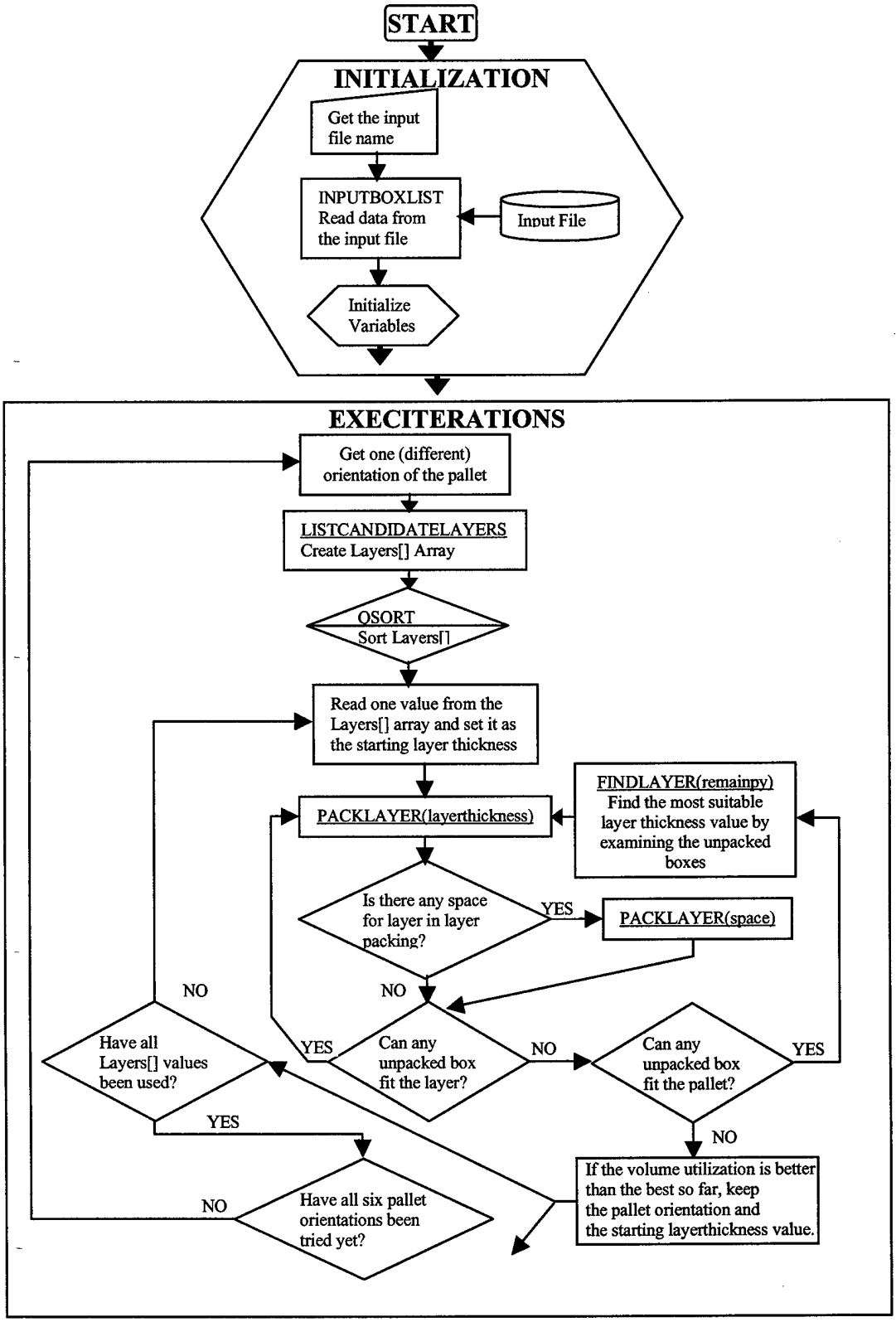
Number of total different dimension values : 1000

Max dimension length for either pallet or any box : 32,767

All dimension values must be integer numbers.

3.6 Flow Chart of The Algorithm

Figure 3-2 depicts the flow chart of how the iterations are performed. The pseudo-codes of the functions are provided in Appendix A. The program itself is in Appendix B.



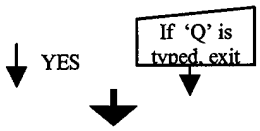
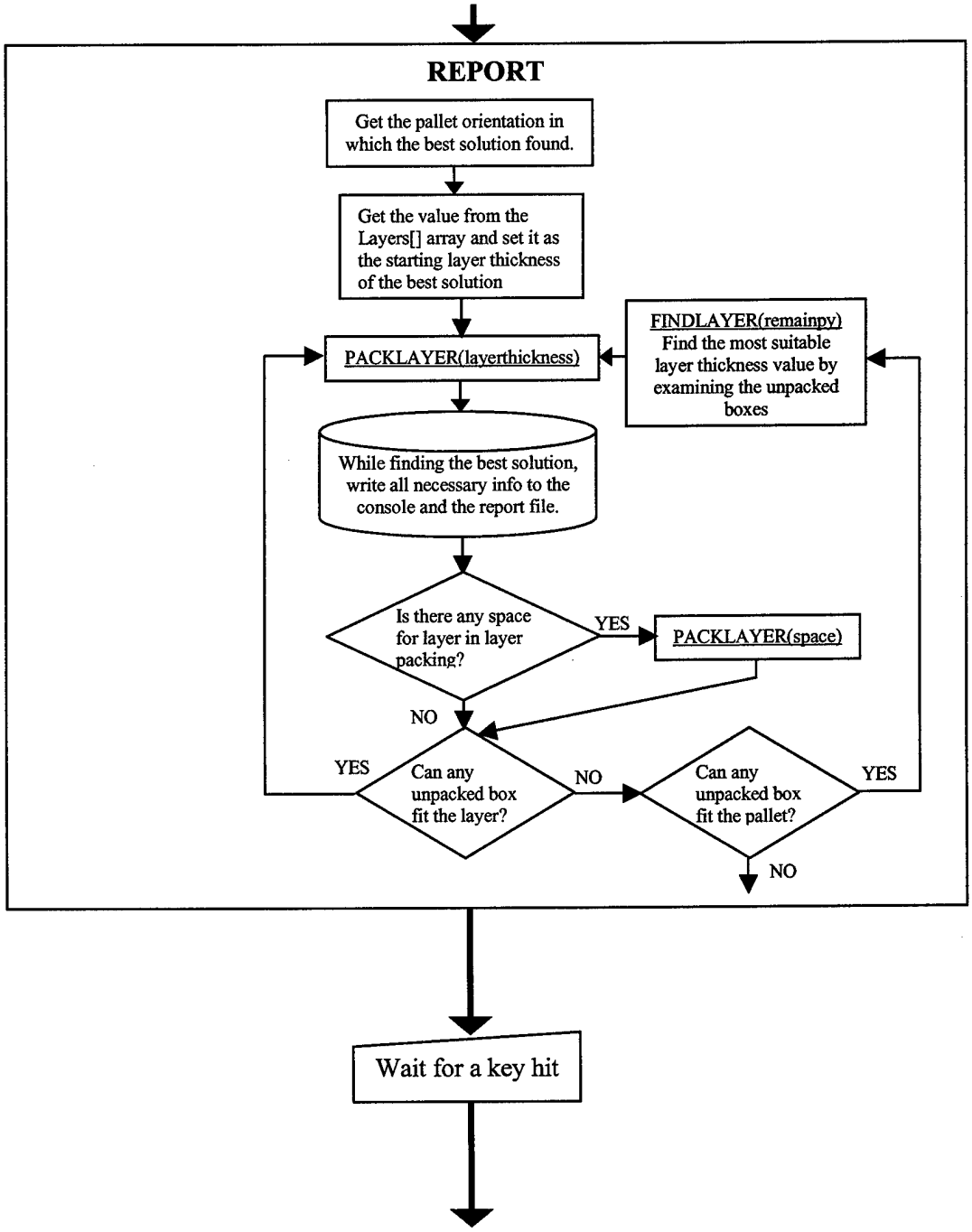


Figure 3-2: Flow Chart of the Algorithm



END

Figure 3-2: Flow Chart of The Algorithm

3.7 How Does The Heuristic Work?

3.7.1 Preparation For Iterations

Assume Figure 3-1 presents the problem to solve. Figure 3-3 is the corresponding BOXLIST[] array.

Variable assignments: XX=104; YY=96; ZZ=84;

Boxlist[X]=(Packst, N, Dim1, Dim2, Dim3, Cox, Coy, Coz, Packx, Packy, Packz, Vol)

Boxlist[1]=(	0,	4,	70,	104,	24,	0,	0,	0,	0,	0,	174720)
Boxlist[2]=(	0,	4,	70,	104,	24,	0,	0,	0,	0,	0,	174720)
Boxlist[3]=(	0,	4,	70,	104,	24,	0,	0,	0,	0,	0,	174720)
Boxlist[4]=(	0,	4,	70,	104,	24,	0,	0,	0,	0,	0,	174720)
Boxlist[5]=(	0,	2,	14,	104,	48,	0,	0,	0,	0,	0,	69888)
Boxlist[6]=(	0,	2,	14,	104,	48,	0,	0,	0,	0,	0,	69888)
Boxlist[7]=(	0,	3,	40,	52,	36,	0,	0,	0,	0,	0,	74880)
Boxlist[8]=(	0,	3,	40,	52,	36,	0,	0,	0,	0,	0,	74880)
Boxlist[9]=(	0,	3,	40,	52,	36,	0,	0,	0,	0,	0,	74880)

Figure 3-3: Creating the Boxlist[] Array

After the input process is complete, the model creates a layer thickness array named LAYERS[]. This array contains every unique dimension of the boxes less than the y dimension of the current orientation of the pallet with their individual evaluation values. The Layers[] array is created for each orientation of the pallet. Each entry is a possible layer thickness value for iterations with the current orientation of the pallet to start the packing.

The evaluation value of a layerdim represents how close all other boxes are to this layer height if we selected this value as a layer thickness for the packing. The model calculates these evaluation values as follows:

Retrieve a box and one of its dimensions. Examine the previously set layerdim values in the array. If this is a different length and less than the y dimension of the current orientation of the pallet, store the length in a new element in the layerdim array. Then it goes through every other box retrieving its dimension closest to the layerdim value, and adds up the absolute value of the differences between that dimension and the layerdim value. The layerdim value with the smallest layereval weight value is the most suitable layer thickness value; this value should yield the smoothest layer height.

Continuing our example, calculations for the Layers[] array for the pallet orientation X=104, Y=96, Z=84 are:

Layers[X]=(Layerdim, Layereval)

Abs(70-70)+Abs(70-70)+Abs(70-70)+Abs(70-48)+Abs(70-48)+Abs(70-52)+Abs(70-52)+Abs(70-52)=98

Layers[1]=(70, 98)

Abs(24-24)+Abs(24-24)+Abs(24-24)+Abs(24-14)+Abs(24-14)+Abs(24-36)+Abs(24-36)+Abs(24-36)=56

Layers[2]=(24, 56)

Abs(14-24)+Abs(14-24)+Abs(14-24)+Abs(14-24)+Abs(14-14)+Abs(14-40)+Abs(14-40)+Abs(14-40)=106

Layers[3]=(14, 106)

Abs(48-70)+Abs(48-70)+Abs(48-70)+Abs(48-70)+Abs(48-48)+Abs(48-40)+Abs(48-40)+Abs(48-40)=100

Layers[4]=(48, 100)

Abs(40-24)+Abs(40-24)+Abs(40-24)+Abs(40-24)+Abs(40-48)+Abs(40-48)+Abs(40-40)+Abs(40-40)=80

Layers[5]=(40, 80)

Abs(52-70)+Abs(52-70)+Abs(52-70)+Abs(52-70)+Abs(52-48)+Abs(52-48)+Abs(52-52)+Abs(52-52)=80

Layers[6]=(52, 80)

Abs(36-24)+Abs(36-24)+Abs(36-24)+Abs(36-24)+Abs(36-48)+Abs(36-48)+Abs(36-36)+Abs(36-36)=72

Layers[7]=(36, 72)

There are 8 different dimension values but since the dimension value 104 is larger than the y dimension of the current pallet orientation 96, we do not evaluate it as a possible layer thickness. After having such a Layers[] array prepared, it is sorted ascending order with respect to its layereval values:

Layers[X]=(Layerdim, Layereval):	Layers[1]=(24, 56)
	Layers[2]=(36, 72)
	Layers[3]=(52, 80)
	Layers[4]=(40, 80)
	Layers[5]=(70, 98)
	Layers[6]=(48, 100)
	Layers[7]=(14, 106)

Figure 3-4: Creating the Layers[] Array

Since the smallest layereval value potentially may be the most suitable layer thickness value, having that list sorted and starting to pack from the most promising layer thickness values would be an important factor to reduce the solution time, especially if we consider packing a large number of different box types. However, this greedy approach does not always hold. Sometimes an iteration starting with a larger layerdim value yields the best solution.

3.7.2 Execution Of An Iteration

Iterations tie closely to the six possible orientations of a pallet. During iterations, all six orientations of the pallet are packed. Each unique orientation of the pallet is treated as a pallet to pack. Obviously, if a pallet has three identical dimensions, it has only one orientation. In general, we have 1, 2 or 6 orientations for 1, 2 or 3 unique dimensions, respectively. In each iteration, each orientation of the pallet is packed once for each element in the Layers[] array. Each iteration begins packing with an initial layer thickness taken from layerdim value in the Layers[] array. Thus, if we have 7 different dimension values in our Layers[] array and the pallet has 3 unique dimensions, the program potentially performs 6*7=42 iterations. Thus, the solution time of our program is effected by both the number of different dimension values and the number of total boxes to be packed. The number of different box types does not have a direct affect on the solution time. It is always possible to terminate the program prematurely by pressing the 'Q' key whenever a sufficient 'best so far' value has appeared on the console. Since the layer packing approach of the program is really effective, to get a very high volume utilization in a very short time is strongly possible.

Before explaining the details of the heuristic, we need to mention its computational complexity. If we have n boxes in our box set and d different dimension values for all boxes, the worst solution time is given by:

$$\vartheta(t)=6\ n\ d\ P(t) \tag{1}$$

where P(t) is the time spent to find and pack any box, which can be defined:

$$P(t)=6\ n\ E(t) \tag{2}$$

where E(t) is time to examine an orientation of a box. E(t) depends upon the computer. We ran all the test problems on a Pentium III 750 MHz computer, and for that computer, $E(t)\approx 0.18$ microseconds (10^{-6}).

Therefore the worst-case solution time performance is:

$$\vartheta(t)=36\ n^2\ d\ E(t) \tag{3}$$

Each iteration starts with the pallet empty, all boxes available, a pallet orientation and a layer thickness from the Layers[] array. Subsequent iterations change the starting layer thickness or the pallet orientation. The parameters of the best packing found, based on volume packed, are saved as the current packing solution.

We perform layer packing or wall building thus reducing the problem to a systematic series of two-dimensional packing problems. As Figure 3-5 depicts, we pack along the x- and z-axis. To track the current topology, each right corner coordinate data is maintained in a doubly linked list. As boxes are packed, this coordinate data will change. The doubly linked list facilitates the change to the coordinate data. This approach means we only need to track the current edge being packed, and we avoid overlaps of layers and pallet edges.

Each packing action begins by finding the smallest z-value in the coordinate data list and from that list finding the current gap in the layer to fill. The candidate boxes are examined to find one that fills the gap, with the correct layer thickness, and to find a box that fills the gap yet exceeds the current layer thickness by as small an amount as possible. If no box is found to fill the gap, the gap is ignored. Each box is examined in each orientation.

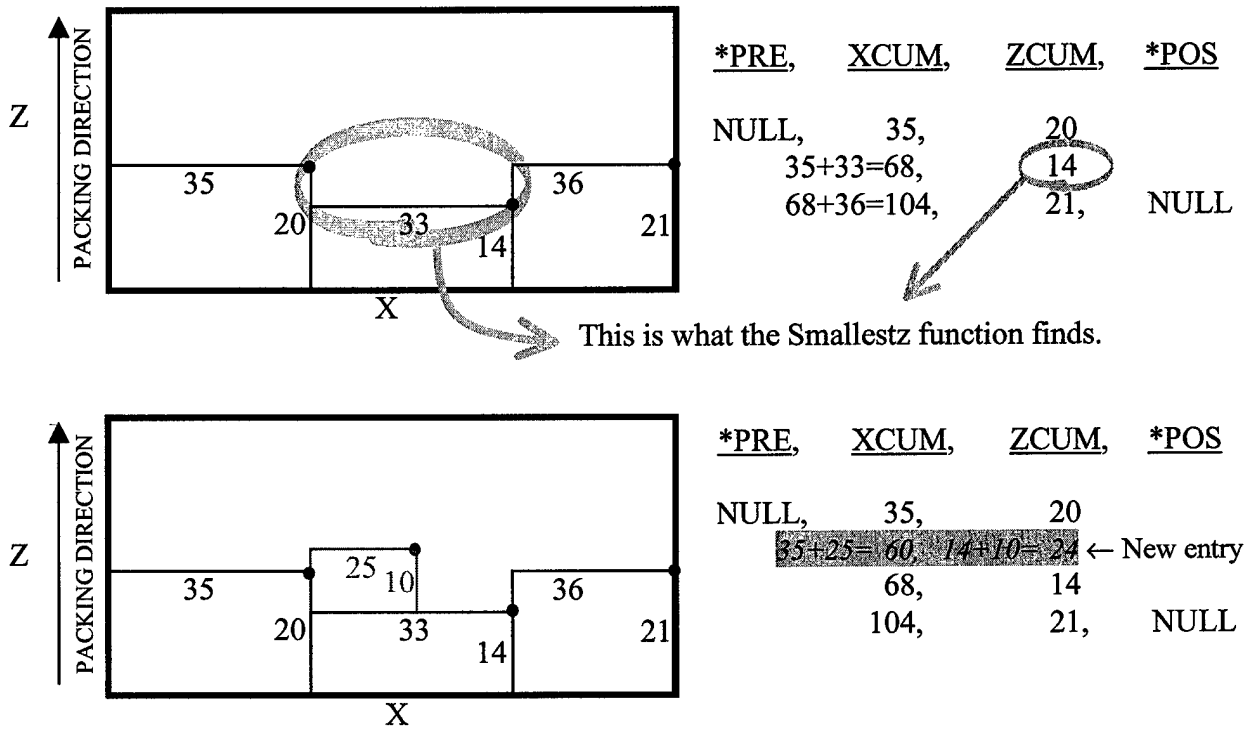


Figure 3-5: Packing a layer

Table 3-1: Key Packing Program Functions

FUNCTION	PURPOSE
Packlayer	Update the linked list and the Boxlist[] array as a box is packed.
Findsmallestz	Determine the gap with the smallest z value in the current layer.
Findbox	Find the box that best fits to the current gap.
Analyzebox	Used by Findbox to analyze box dimensions.

Checkfound	Determine which box to pack.
Execiterations	Execute iterations by calling proper functions.
Report	Duplicate best-so-far packing.
Outputboxlist	Writes packing information to file.
Graphunpackedout	Writes packing order for visualization program.

The Findbox function analyzes the unpacked boxes using the Analyzebox function. For each different orientation of the unpacked boxes, the Analyzebox parameters are:

Hmx : Maximum available x-dimension of the current gap to be filled.
Hy : Current layer thickness value.
Hmy : Maximum available y-dimension of the current gap to be filled.
Hz : Z-dimension of the current gap to be filled.
Hmz : Maximum available z-dimension to the current gap to be filled.
Dim1 : X-dimension of the orientation of the box being examined.
Dim2 : Y-dimension of the orientation of the box being examined.
Dim3 : Z-dimension of the orientation of the box being examined.

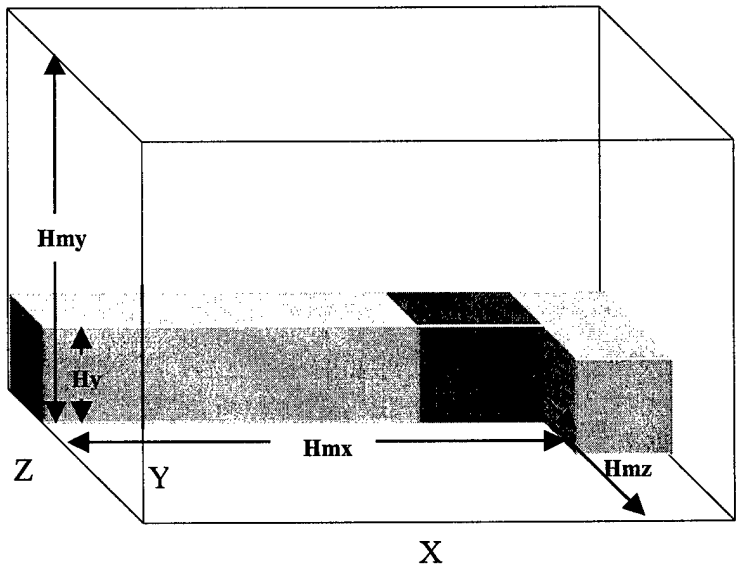


Figure 3-6: Findbox Function Parameters

Analyzebox seeks, in precedence order, a box with a y-dimension closest to Hy but not more than Hmy, with an x-dimension closest to, but not more than Hmx, and a z-dimension closest to Hz, but not more than Hmz. This means it considers the y-dimension, then among boxes having the same y-dimension, it looks at the x-dimension, and finally among the boxes having the same y- and x-dimension, it looks at the z-dimension. It calculates the differences between the gap's dimensions and the box dimensions for each box and picks the box with the least differences to be the best fitting one. It also finds a second box with a y-dimension bigger than the current layer thickness, but closest to the current layer thickness. Boxes that fit and are the proper thickness (y-values) are packed. Uneven boxes require extra consideration.

We developed a layer-in-layer packing to accommodate packing of uneven heights. A layerinlayer variable determines if there is any unevenness in the current packing layer.

If there is no box to fit the current gap, the gap under consideration is left unpacked. Figure 3-7 depicts this situation.

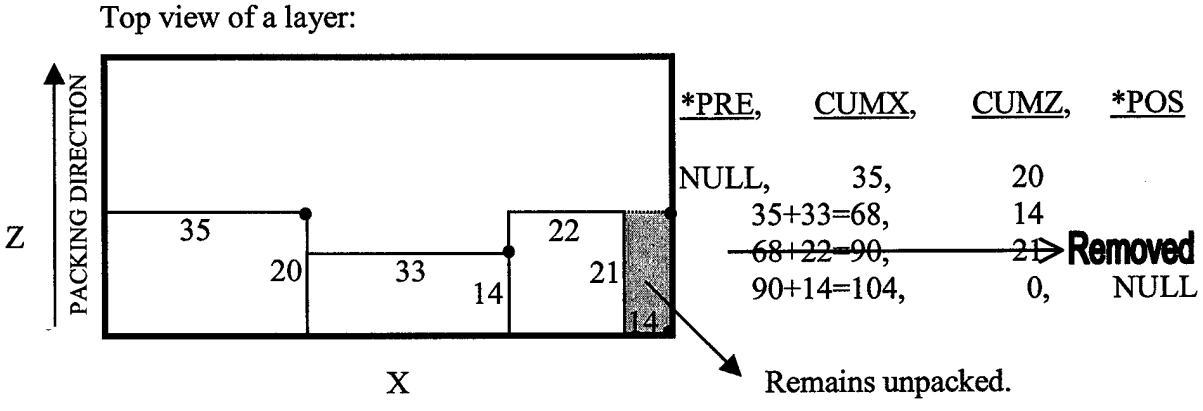


Figure 3-7: Leaving a Gap Unpacked

If it is required; the current layer thickness is increased to the y-dimension of the taller box found by the Findbox function. When the current layer thickness is increased, the total increment of the layer, from the beginning to the end of the current layer packing, is saved in the layerinlayer variable. After finishing the packing of the current layer, if the layerinlayer variable is greater than zero, another packing within the layer, for that layerinlayer thickness value, is performed.

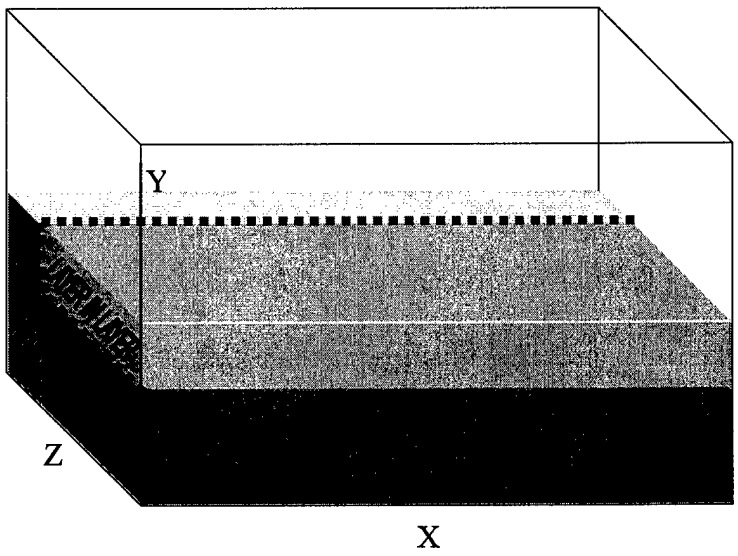


Figure 3-8: Layer in Layer Packing

Layer-in-layer packing means our approach can utilize more space in the pallet. Although not used often in the problem set we examined, when used, layer-in-layer packing gave much better results.

While the Layers[] array entries are used to start the first layer of each iteration, subsequent layers require calculation to determine good layer thickness values. These

calculations duplicate these described for the Layers[] array entries, but apply only to the as yet unpacked boxes. The best layer value is used as the layer thickness for subsequent packing layers. Layering continues until the pallet can no longer accommodate further layers. The pallet is then considered packed.

Each packing tracks the volume of boxes packed and volume of boxes not packed to derive a pallet utilization measure and percentage of packed box volume. Each packing also has an associated pallet orientation and Layers[] array index. The best packing found becomes the final solution of the model.

After getting the best solution's parameters, the function Report is called. The Report function re-performs the packing with the parameters of the best solution found, but now calls the Outputboxlist function to generate the report file and the Graphunpackedout function to generate the visualization program input file. Information about unpacked boxes is included at the end of the report file.

3.8 Output Data

The model has three output streams. One is to the console, another to the report file, and the final to "visudat" file. Both the console and the report file report the overall numerical summary of the solution as well as the packing coordinates and the orientation of each packed box. The list of any unpacked boxes, is appended to the end of the report file. The dimensions of the pallet along with the packing coordinates and the orientation of each packed box are output to the file "visudat". The graphical interface program uses the "visudat" file to visualize the solution.

The name of the report file uses the same root name of the input file but with the extension "out". Figure 3-9 presents a "visudat" file format, and Figure 3-10 presents a report file format.

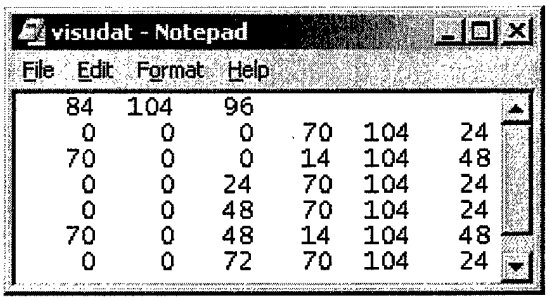


Figure 3-9: Visudat File Format

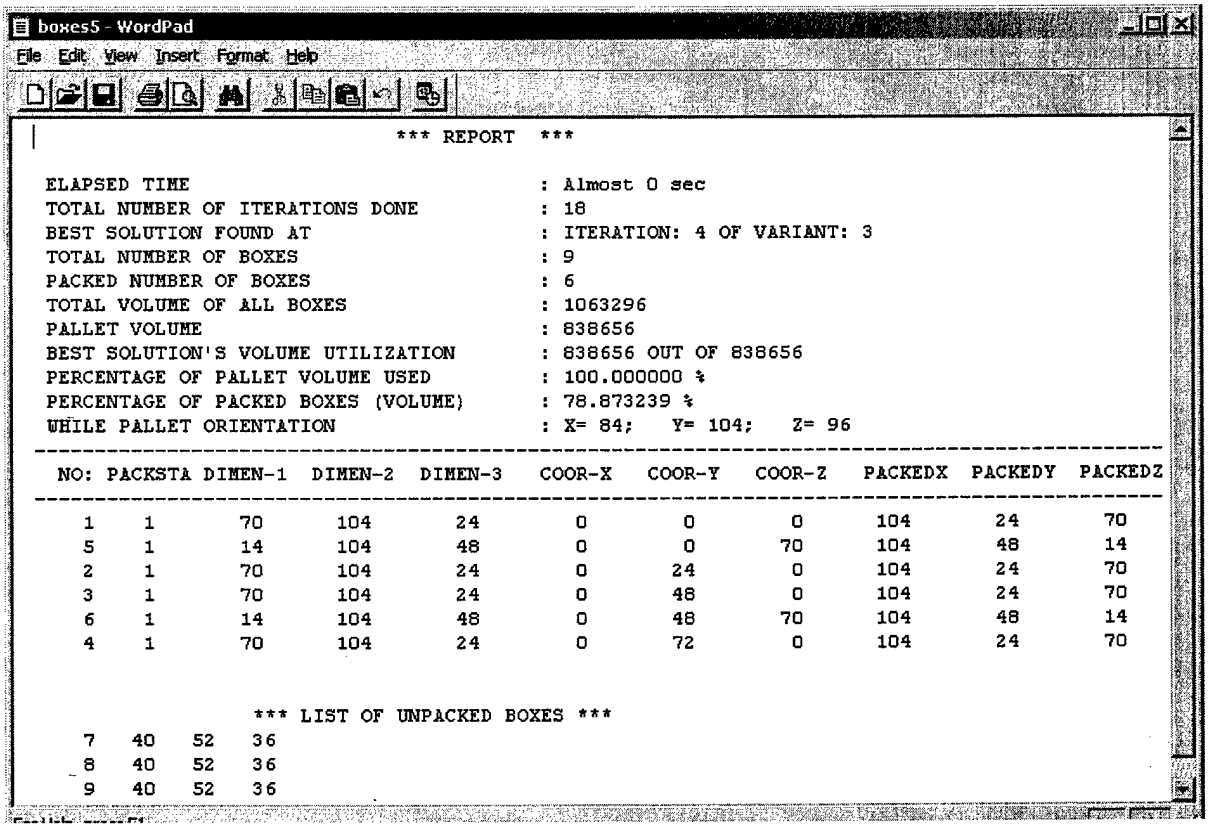


Figure 3-10: Output File Format

3.9 Graphical Interface Program

The graphical interface program, which is available in Appendix C, aids verification, checks for possible errors, and promotes the presentability of the best solution found. It reads the pallet dimensions and the box packing information from the "visudat" file and displays them on the screen interactively starting from the far end towards the user. This ensures a clear view of the box as it is packed. The program scales all dimension values and corresponding coordinates to properly fit the screen.

3.10 Summary

We developed a very efficient and robust heuristic technique to solve the three-dimensional pallet-packing problem. We implemented it in a program and our testing shows good results. In Chapter Four, we present the test problem solutions and some comparisons with other pallet-packing approaches.

Chapter 4 – Results

4.1 Introduction

We employed several different techniques to test our model. We created special test problem sets and examined performance on readily available, standard problem sets. In this chapter, we present testing results to learn about the important characteristics of our model.

4.2 Numerical Tests and Comparisons

To test our model, we created many different box sets. Prior works tested their methods with randomly generated box sets. While we also used random problems, it is impossible to evaluate the solution qualities of these random problems because the optimum solution is unknown. Thus, we developed a technique to generate problem sets with known solutions.

We used the Exel software to produce uniformly distributed random dimension values between certain numerical limits and then we arranged the box quantities to be close to the pallet volume. Table 4-1 summarizes a sample of those randomly generated problem solutions.

Table 4-1: Randomly generated problem solutions

Box Set	# of Boxes	# of Box Types	% Vol. Utilization	Solution Time(sec)
Set #1	307	5	89.5	2
Set #2	1728	5	97.5	189
Set #3	637	11	92.4	44
Set #4	1493	21	96.4	255
Worst Case(#5)	31	31	68.7	<1

For all randomly generated problems, the pallet dimensions are 104, 96, 84. The volume utilization of the pallet is fairly high, but the optimum solutions are unknown. We do know our solution times are quite low. In the very last line of Table 4-1, we present results for a problem that is extremely difficult to solve because all boxes are unique. The volume utilization is very low, but seems reasonable given the unique size of each box. We also obtained the solution very quickly. We packed 10 different realizations of the difficult problem type and found the worst-case solution was greater than 65%.

Since we wanted to evaluate our solutions by comparing them with optimal solutions, we developed a technique to generate problem sets. In our technique, we divide the container into many different sized boxes. These boxes can then be listed in any order, even rotated, to yield a packing problem with a known, optimal solution of 100% pallet volume utilization. All problem sets that we generated are in Appendix D.

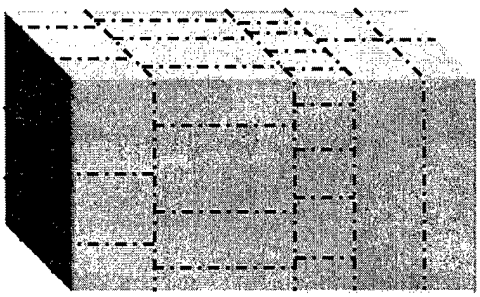


Figure 4-1: Creating Box Sets

The box sets that we generated with this problem generation technique and our solutions are presented in Table 4-2. Overall, our method does quite well.

Table 4-2: Specially Generated Problem Solutions

Box Set	# of Boxes	# of Box Types	% Vol. Utilization	Solution Time(sec)
Set #6	6	2	100	<1
Set #7	10	3	100	<1
Set #8	12	4	100	<1
Set #9	18	6	89.7	<1
Set #10	86	7	91.6	<1
Set #11	39	15	84.5	1

We developed our model as a distributor's pallet packing problem but wanted to test it on the manufacturer's pallet packing test problems (MPPP). Since the distributor's pallet packing problem is much harder than the MPPP, we expected our model to yield very good results on the MPPP. We created these problem sets using the same problem generation technique and and solved the problems. Our results are presented in Table 4-3. As expected, the model does very well on the MPPP.

Table 4-3: Specially Designed MPPP Solutions

Box Set	# of Boxes	# of Box Types	% Vol. Utilization	Solution Time(sec)
Set #12	576	1	100	4
Set #13	1152	1	100	18
Set #14	4992	1	100	228
Set #15	1152	2	100	8
Set #16	2784	2	98	315

There are also 700 problems generated by Bischoff *et al.* (1995) readily available on the Imperial College Management School web page (2001). These problems are randomly generated and come in 7 files. Each of the 100 problems in 7 files contain 3, 5,

8, 10, 12, 15, and 20 box types, respectively. All problems use the same pallet dimensions of 587, 233, and 220. We solved these 700 problems and obtained the results presented in Table 4-4. Solution detailes are presented in Appendix E.

Table 4-4: Solution Summary Of The Given Problem Set

Box Set	# of box types (n)	# of Boxes			% of Pallet Utilization			Utilization Std. Dev.	Solution Time(sec)		
		min	ave	max	min	ave	max		min	ave	max
BR-3	3	69	150	476	78.9	89.0	95.3	2.93	<1	1.07	12
BR-5	5	81	137	266	84.8	89.0	94.0	2.03	<1	1.23	5
BR-8	8	80	134	232	84.5	88.4	92.1	1.56	<1	1.85	5
BR-10	10	75	133	233	84.4	88.2	91.9	1.52	1	2.19	6
BR-12	12	84	133	218	84.0	87.6	89.9	1.33	1	2.73	8
BR-15	15	85	131	203	84.3	87.4	91.3	1.28	1	3.39	9
BR-20	20	90	130	172	84.3	87.1	90.2	1.15	2	4.47	9

We compared our solutions with the solutions of Bischoff, and Ratcliff (1995) (denoted by B/R), and Gehring and Bortfeld (1996) (denoted by G/B); Bortfeld and Gehring (1997) (denoted by B/G); Terno, Scheithauer, Sommerweiß, and Riehme (1997) (denoted by T/S/S/R) in Table 4-5. Information about the G/B and the B/G results come from the Terno, *et al.* (1997). We know that G/B uses a Genetic Algorithm model and B/G uses a Tabu Search model.

Table 4-5: Comparisons for the Bischoff/Ratcliff Examples

Set #	B/R			G/B			B/G			T/S/S/R			OURS		
	min	ave	max	min	ave	max	min	ave	max	min	ave	max	min	ave	max
BR-3	73.7	85.4	94.4	76.7	85.8	94.3	78.7	89.0	95.7	75.7	89.9	95.9	78.9	89.0	95.3
BR-5	73.8	86.3	93.8	78.4	87.3	95.2	79.7	88.7	95.0	81.9	89.6	94.7	84.8	89.0	94.0
BR-8	75.3	85.9	92.6	81.1	88.1	92.9	82.4	88.2	94.0	83.2	89.2	93.0	84.5	88.4	92.1
BR-10	78.4	85.1	90.1	82.7	88.0	91.6	80.9	87.4	92.0	83.1	88.9	92.7	84.4	88.2	91.9
BR-20	75.7	83.0	88.3	84.4	87.7	90.7	79.9	83.9	88.4	80.6	86.3	89.0	84.3	87.1	90.2

The best competitor to our model seems to be T/S/S/R. However, it appears that our model is more dependable. All of the minimum values of our results are 1 to 4% better than T/S/S/R minimum values while average and maximum values are very comparable. Solution time is another important factor of interest. Our model solves any of 700 box sets in 1 to 12 seconds on a Pentium III 750 MHz computer. The T/S/S/R model uses 150 to 600 seconds on a PentiumPro 200 MHz computer. Their computer is about 4 times slower than ours, but our solution time is about 50-100 times faster than theirs. Thus, it appears we may have some processing time advantage.

Another box set has been used by Loh and Nee (1992) (denoted by L/N); Ngoi, Tay, and Chua (1994) (denoted by N/T/C); Bischoff, Janetz, and Ratcliff (1995) (denoted by B/J/R); Bischoff, and Ratcliff (1995) (denoted by B/R); Gehring and Bortfeld (1996) (denoted by G/B); Bortfeld and Gehring (1997) (denoted by B/G); Terno, Scheithauer, Sommerweiß, and Riehme (1997) (denoted by T/S/S/R). In the Table 4-6 we compare the volume utilization percentages for 4 instances in that problem set. The other 11 instances are not presented since all boxes could be packed in the container by our model. Again our results compare favorably.

Table 4-6: Comparisons for the Loh/Nee Examples

Set #	# of Boxes	# of Box Types	L/N 1992	N/T/C 1994	B/J/R 1995	B/R 1995	G/B 1996	B/G 1997	T/S/S/R 1997	OURS 2001	Solution time (sec)
LN #2	200	8	76.8	80.7	89.7	90.0	89.5	96.6	93.9	93.3	3
LN #6	200	8	88.6	88.7	89.5	83.1	91.1	91.2	91.6	91.7	3
LN #7	200	8	78.2	81.8	83.9	78.7	83.3	84.7	84.7	84.7	1
LN #13	130	7	77.0	84.1	82.3	78.1	85.6	84.3	85.1	85.6	<1

In Section 2.2, we stated that the model developed by Han *et al.* (1989) was able to pack 195 boxes of 11" by 6" by 6" into a 48" by 42" by 40" container. Their volume utilization of the container was 95.16%. They also quoted the US General Services Administration whose published results in 1966 for the same problem only provide 82.5% utilization. We solved the same problem. Our model packed 196 boxes, with a utilization of 96.25% in less than a second.

Another test problem that we solved was created and solved by Chen *et al.* (1995) while developing an analytical model. Faina (2000) solved the same problem by using a special simulated annealing algorithm. They both solved this small instance with some fixed orientation of each box. The problem is provided in Table 4-7.

Table 4-7: Chen *et al.* and Faina's Example Box Set

Box Number	Dimensions Of Boxes		
	x	y	z
1	25	8	6
2	20	10	5
3	16	7	3
4	15	12	6
5	22	8	3
6	10	20	4

They were able to pack these six boxes into a container with dimensions of 10, 20, 35, by packing the boxes in certain defined orientations. Their solution is the optimal solution. Since our model rotates each box, we tested our model on this small problem

allowing full rotation of all boxes. We actually improved the packing using a container with dimensions of 10, 20, 33.

We wanted to make further comparisons with Faina's (2000) solutions, but the problem sets were not presented in the paper. He provides but emphasizes that beyond 64 boxes, the effort to improve the final placing is too expensive computationally. Therefore it is hard to make further comparisons. Faina also states that the results obtained are worse as the number of boxes increases. With our method, we see that increasing the number of the boxes has little affect on the solution quality, but it affects solution time.

Recall our model's worst case solution time formulation with $E(t) \approx 0.18$ microseconds (10^{-6}) for our computer, a Pentium III 750 MHz, 256 Mb, Windows 2000 Professional. For Problem #2 presented in Table 4-1; with n=1728 and d=15, $\vartheta(t)$=290 seconds, greater than actual solution time of 189 seconds. For very large problems of say 5000 boxes with 100 box types, $\vartheta(t)$=16200 seconds=4.5 hours. Since these times are large, we examined the impact of stopping our model early after a few iterations. A comparison is shown in Table 4-8. Notice our approach gives very good, possibly acceptable, results after just a couple of iterations.

Table 4-8: Comparisons with Premature Solution Times

BOX SET	NUMBER OF BOXES IN SET	SOLUTION UTILIZATION FULL %	FULL SOLUTION TIME (sec)	SOLUTION UTILIZATION PREMATURE %	PREMATURE SOLUTION TIME (sec)
Set #2	1728	97.5	189	93.2	6
Set #3	637	92.4	44	87.4	3
Set #4	1493	96.4	255	88.2	4
Set #12	576	100	4	93.8	1
Set #13	1152	100	18	99.5	2
Set #14	4992	100	228	97.8	93
Set #15	1152	100	8	98.2	2
Set #16	2784	98	315	91.7	13
BR-3 #65	476	94.4	12	92.9	1

BR-5 #39	266	93.8	5	89.5	1
BR-8 #39	232	89.7	5	85.9	1
BR-10 #56	233	91	6	89	1
BR-12 #56	218	88.1	8	86	1
BR-15 #13	203	88.4	9	85.2	1
BR-20 #51	166	88.1	9	86	1

4.3 Summary

An algorithm is an abstraction best evaluated by experimenting with a specific implementation. A heuristic is a special type of algorithm. We implemented our packing heuristic and solved various sets of problems. Our tests demonstrate the validity of the model and its performance. Our model proved to be an innovative and quick running algorithm, producing extremely good results.

Chapter 5 – Conclusions and Recommendations

5.1 Research Results

This research developed a solution approach to the three dimensional pallet-packing problem. Ballew (2000) developed a mathematical formulation and explored a heuristic approach using a genetic algorithm. His research found that it is not practical to try to solve this problem by using an analytical approach. Based on not only Ballew but the research of others, the complexity of the three-dimensional pallet-packing problem appears to rule out any type of search heuristic such as genetic algorithms or simulated annealing. These problems do not exploit enough problem-specific knowledge.

We built a new heuristic method. Our heuristic technique tries to mimic human intelligence and behavior, in particular how one would build layers, fill gaps in a layer, and examine various box orientations. We wrote a C program to test our algorithm, developed another program to visualize the best solution found, and used attributive memory and dynamic data structures, such as arrays and double linked lists, to improve processing efficiency.

Our heuristic not only solves large problems in a small amount of time, but it also gives very good or optimal solutions. It is also robust and high solution quality is not affected by different problem characteristics. We make these claims based on empirical results from literature problems, test bank problems, and problems we created with a known optimal solution.

5.2 Recommendation for Future Research

Our research showed us that powerful heuristic techniques would solve the distributor's three-dimensional pallet packing problem in a reasonable amount of time. Thus, we recommend working on employing other powerful heuristic tools or using those stated tools more extensively.

The primary objective should be adding other constraints rather than increasing the volume utilization of the pallet. The first essential constraint to add would be the load stability. Other necessary constraints such as loading boxes in some certain orientations and weight and balance should be added one by one.

Another avenue is to implement our heuristic within some other approach to solve multi-pallet packing problem. As Choocolaad (1998) and Romaine (1999) mentioned, the multi-pallet problem is important for Air Force mobility and deployment planning studies.

A final avenue of research might be to add some sort of local improvement heuristic. Given some solutions, based on some pallet orientation and layer thickness, can minor changes yield better packings.

Appendix A - Pseudo-codes of The Functions

MAIN

Perform initialization by calling INITIALIZE();
Get time(START);
Execute iterations and find the parameters of the best solution by calling
    EXECITERATIONS();
Get time(FINISH);
Using the parameters found, pack the best solution found, report to the console
    and to an output text file by calling REPORT();
Wait until a keystroke entered by the user;
End;

FUNCTION INITIALIZE()

Get the input FILENAME from the user;
Get the pallet and box set data entered by the user from the input file by calling the
    function INPUTBOXLIST();
Calculate the volume of the pallet;
Calculate the total volume of all boxes;
Create a node and call it SCRABFIRST. Each of these double linked list nodes keeps X and Z coordinates of each gap in the layer currently being packed.
SCRABFIRST.PRE=NULL; SCRABFIRST.POS=NULL;
Initialize variables those keep the best so far and its parameters.

FUCTION INPUTBOXLIST()

If exists, open the file FILENAME;
    Else {Tell the user "Cannot open the file FILENAME"; end;}
Read the first line of the input file and set the pallet dimension variables XX, YY, ZZ;
Read every other lines in the input file and fill each field in the array BOXLIST[] .
Now the variable TBN is already set to the total number of boxes input from the file;
Close the file FILENAME;
RETURN;

```
FUNCTION EXECITERATIONS();

For VARIANT=1 to 6 {
        For each value of VARIANT get a different orientation of the pallet to the
                variables PX, PY, PZ;
        List all possible candidate values by calling LISTCANDITLAYERS();
        Sort the array LAYERS in respect to its LAYEREVAL fields in increasing order
                by using QSORT;
        For each layer values in the LAYERS[] array, perform another iteration starting
                with that layer value as the starting layer thickness:
        For LAYERSINDEX=1 to LAYERLISTLEN {
                Get the first value of the LAYERS array as the starting
                        LAYERTHICKNESS value:
                        LAYERTHICKNESS=LAYERS[LAYERSINDEX].LAYERDIM
                Set all boxes' packed status to 0:
                        For X=1 to TBN do BOXLIST[X].PACKST=0;
                do {
                        Set the variable that shows remaining unpacked potential second
                                layer height in the current layer: LAYERINLAYER=0;
                        Set the flag variable that shows packing of the current layer is
                                finished or not: LAYERDONE=0;
                        Call PACKLAYER(), to pack the layer, and if a memory error is
                                responded, exit the program;
                        If there is a height available for packing in the current layer,
                                perform another layer packing in the current layer:
                        If LAYERINLAYER≠0 do{
                                Get the height available for packing in the current layer as
                                        the layer thickness to be packed:
                                        LAYERTHICKNESS=LAYERINLAYER;
                                Call PACKLAYER(), to pack the layer, and if a memory
                                        Error is responded, exit the program;
                        }
                        Call FINDLAYER(REMAINPY) to determine the most suitable
                                layer height fitting in the remaining unpacked height of the
                                pallet;
                } While PACKING≠0;
                If the volume utilization of the current iteration is better than the best so
                        far, and the iterations were not quit, keep the parameters:
                        (Pallet orientation, utilization, and the index of the initial layer
                        height in the LAYERS array);
                If a hundred percent packing was found, exit doing iteration and
        RETURN;
        }
}
```

```
FUNCTION LISTCANDITLAYERS();

LAYERLISTLEN=0;
For X=1 to TBN {
        Get each dimension of each box, one at a time by doing:
        For Y=1 to 3 {
                If Y=1 do {
                        EXDIM=BOXLIST[X].DIM1;
                        DIMEN2=BOXLIST[X].DIM2;
                        DIMEN3=BOXLIST[X].DIM3;
                }
                If Y=2 do {
                        EXDIM=BOXLIST[X].DIM2;
                        DIMEN2=BOXLIST[X].DIM1;
                        DIMEN3=BOXLIST[X].DIM3;
                }
                If Y=3 do {
                        EXDIM=BOXLIST[X].DIM3;
                        DIMEN2=BOXLIST[X].DIM1;
                        DIMEN3=BOXLIST[X].DIM2;
                }
                If any of the dimensions of the box being examined cannot fit into the
                        pallet's respective dimensions, exit this loop and continue with the
                        next loop;
                If EXDIM is the same as any of previously examined dimension lengths,
                        exit this loop and continue with the next loop;
                Set the evaluation value of the EXDIM to 0 by doing LAYEREVAL=0;
                For Z=1 to TBN do{
                        Get the closest dimension value of each box to the EXDIM by
                                looking at the absolute values of differences between each
                                dimension and EXDIM, and selecting the smallest value;
                                and set the variable DIMDIF to this value;
                        Add those values cumulatively by doing:
                                LAYEREVAL=LAYEREVAL+DIMDIF;
                }
                LAYERLISTLEN=LAYERLISTLEN+1;
                LAYERS[LAYERLISTLEN].LAYEREVAL=LAYEREVAL;
                LAYERS[LAYERLISTLEN].LAYERDIM=EXDIM;
        }
}
RETURN;
```

FUNCTION COMPLAYERLIST(i,j);

This function is required for the compiler built in function QSORT().
It returns the difference between the values i and j.

FUNCTION PACKLAYER();

If LAYERTHICKNESS=0 do { PACKING=0; RETURN;};
Initialize the first and only node to the layer's X and Z values:
SCRAPFIRST.CUMX=PX; SCRAPFIRST.CUMZ=0;
Perform an infinite loop unless 'Q' is typed to quit {
Check the keyboard input, if 'Q' is hit, exit the loop and RETURN;
To find the gap with the least Z value in the layer call FINDSMALLEST();

SITUATION-1:
If there is no box on both sides of the gap do {
Calculate the gap's X and Z dimensions;
To find the most suitable boxes to the gap found, by looking at;
the X-dimension of the gap: LENX,
layerthickness of the gap: LAYERTHICKNESS,
maximum available thickness to the gap: REMAINPY,
maximum available Z dimension to the gap: LPZ;
call FINDBOX(LENX, LAYERTHICKNESS, REMAINPY, LPZ, LPZ);
Check on the boxes found by the FINDBOX() function by calling
CHECKFOUND();
If the packing of the layer is finished, exit the loop;
If the edge of the layer is evened, go to the first line of the next loop;
Add a new node to the linked list showing the topology of the edge of the currently being packed layer after packing a new box, and set all the necessary variables and pointers properly;
To check the hundred percent packing condition,
call VOLUMECHECK();
}

SITUATION-2:

If there is no box on the left side of the gap do {
    Calculate the gap's X and Z dimensions;
    To find the most suitable boxes to the gap found, by looking at;
        the X dimension of the gap: LENX,
        layerthickness of the gap: LAYERTHICKNESS,
        maximum available thickness to the gap: REMAINPY,
        the Z dimension of the gap: LENZ,
        maximum available Z dimension to the gap: LPZ;
        call FINDBOX(LENX, LAYERTHICKNESS, REMAINPY,
        LENZ, LPZ);
    Check on the boxes found by the FINDBOX() function by calling
        CHECKFOUND();
    If the packing of the layer is finished, exit the loop;
    If the edge of the layer is evened, go to the first line of the next loop;
    Set all the necessary variables and pointers properly to represent the
        current topology of the edge of the layer that is currently being
        packed;
    If the edge of the current layer is evened, set all the necessary variables
        and pointers properly and dispose the unnecessary node;
    To check the hundred percent packing condition, call VOLUMECHECK()
}

SITUATION-3:

If there is no box on the right side of the gap do {
    Calculate the gap's X and Z dimensions;
    To find the most suitable boxes to the gap found, by looking at;
        the X dimension of the gap: LENX,
        layerthickness of the gap: LAYERTHICKNESS,
        maximum available thickness to the gap: REMAINPY,
        the Z dimension of the gap: LENZ,
        maximum available Z dimension to the gap: LPZ;
        call FINDBOX(LENX, LAYERTHICKNESS, REMAINPY,
        LENZ, LPZ);
    Check on the boxes found by the FINDBOX() function by calling
        CHECKFOUND();
    If the packing of the layer is finished, exit the loop;
    If the edge of the layer is evened, go to the first line of the next loop;
    Set all the necessary variables and pointers properly to represent the
        current topology of the edge of the layer that is currently being
        packed;
    If the edge of the current layer is evened, set all the necessary variables
        and pointers properly and dispose the unnecessary node;
    To check the hundred percent packing condition, call VOLUMECHECK()
}

SITUATION-4: IF THERE ARE BOXES ON BOTH SIDES OF THE GAP

SUBSITUATION-4A

If the Z dimensions of the gap is the same on both sides {

Calculate the gap's X and Z dimensions;

To find the most suitable boxes to the gap found, by looking at;

the X dimension of the gap: LENX,
layerthickness of the gap: LAYERTHICKNESS,
maximum available thickness to the gap: REMAINPY,
the Z dimension of the gap: LENZ,
maximum available Z dimension to the gap: LPZ;
call FINDBOX(LENX, LAYERTHICKNESS, REMAINPY, LENZ, LPZ);

Check on the boxes found by the FINDBOX() function by calling CHECKFOUND();

If the packing of the layer is finished, exit the loop;

If the edge of the layer is evened, go to the first line of the next loop;

Set all the necessary variables and pointers properly to represent the current topology of the edge of the layer that is currently being packed;

While updating the edge of topology information, if a part of the topology is evened, dispose unnecessary nodes, and update the others properly;

While updating the edge of topology information, if another gap is added to the topology, add a new node to keep this gap, and update the others properly;

To check the hundred percent packing condition,
call VOLUMECHECK();

}

SUBSITUATION-4B

If the Z dimension of the gap is different on both sides {

Calculate the gap's X and Z dimensions;

To find the most suitable boxes to the gap found, by looking at;

the X dimension of the gap: LENX,
layerthickness of the gap: LAYERTHICKNESS,
maximum available thickness to the gap: REMAINPY,
the Z dimension of the gap: LENZ,
maximum available Z dimension to the gap: LPZ;
call FINDBOX(LENX, LAYERTHICKNESS, REMAINPY, LENZ, LPZ);

Check on the boxes found by the FINDBOX() function by calling

CHECKFOUND();

If the packing of the layer is finished, exit the loop;

If the edge of the layer is evened, go to the first line of the next loop;

Set all the necessary variables and pointers properly to represent the current topology of the edge of the layer that is currently being packed;

While updating the edge of topology information, if another gap is added to the topology, add a new node to keep this gap, and update the others properly;

To check the hundred percent packing condition,

call VOLUMECHECK();

}

}

```
FUNCTION FINDLAYER();

Set the overall evaluation value to a big number: EVAL=1000000;
For X=1 to TBN {
        If the box number X has already been packed continue with the next loop:
                If BOXLIST[X].PACKST≠0 continue;
        Get each dimension of each box, one at a time by doing:
        For Y=1 to 3 {
                If Y=1 do {
                        EXDIM=BOXLIST[X].DIM1;
                        DIMEN2=BOXLIST[X].DIM2;
                        DIMEN3=BOXLIST[X].DIM3;
                }
                If Y=2 do {
                        EXDIM=BOXLIST[X].DIM2;
                        DIMEN2=BOXLIST[X].DIM1;
                        DIMEN3=BOXLIST[X].DIM3;
                }
                If Y=3 do {
                        EXDIM=BOXLIST[X].DIM3;
                        DIMEN2=BOXLIST[X].DIM1;
                        DIMEN3=BOXLIST[X].DIM2;
                }
                If any of the dimensions of the box being examined cannot fit into the
                        pallet's respective dimensions, exit this loop and continue with the
                        next loop;
                Set the evaluation value of the EXDIM to 0 by doing LAYEREVAL=0;
                For Z=1 to TBN do{
                        Get the closest dimension value of each box to the EXDIM by
                                looking at the absolute values of differences between each
                                dimension and EXDIM, and selecting the smallest value;
                                and set the variable DIMDIF to this value.
                        Add those values cumulatively by doing:
                                LAYEREVAL=LAYEREVAL+DIMDIF;
                }
                If the dimension that has just been examined has a smaller evaluation
                        value, keep that dimension:
                        If (LAYEREVAL<EVAL) do { EVAL=LAYEREVAL;
                                LAYERTHICKNESS=EXDIM};
        }
}
RETURN;
```

```
FUNCTION FINDBOX (HMX: Maximum X dimension of the gap;
        HY: Y dimension of the gap; HMY: Maximum Y dimension of the gap;
        HZ: Z dimension of the gap; HMZ: Maximum Z dimension of the gap);

Set all evaluation values to big numbers:
        For the box type fitting in the current layerthickness:
                BFX=32767; BFY=32767; BFZ=32767;
        For the box type that cannot fit in the current layerthickness, but the closest one:
                BFX=32767; BFY=32767; BFZ=32767;
For Y=1 to TBN with step BOXLIST[Y].N do{ (Examines only different boxes)
        If the box that is being examined has been packed before, continue with the next
                loop;
        X=The index of the box that has not been packed before among a certain type of
                box;
        Analyze all six possible orientations of the box being examined:
                ANALYZEBOX (HMX, HY, HMY, HZ, HMZ, BOXLIST[X].DIM1,
                        BOXLIST[X].DIM2, BOXLIST[X].DIM3);
                ANALYZEBOX (HMX, HY, HMY, HZ, HMZ, BOXLIST[X].DIM1,
                        BOXLIST[X].DIM3, BOXLIST[X].DIM2);
                ANALYZEBOX (HMX, HY, HMY, HZ, HMZ, BOXLIST[X].DIM2,
                        BOXLIST[X].DIM1, BOXLIST[X].DIM3);
                ANALYZEBOX (HMX, HY, HMY, HZ, HMZ, BOXLIST[X].DIM2,
                        BOXLIST[X].DIM3, BOXLIST[X].DIM1);
                ANALYZEBOX (HMX, HY, HMY, HZ, HMZ, BOXLIST[X].DIM3,
                        BOXLIST[X].DIM1, BOXLIST[X].DIM2);
                ANALYZEBOX (HMX, HY, HMY, HZ, HMZ, BOXLIST[X].DIM3,
                        BOXLIST[X].DIM2, BOXLIST[X].DIM1);
}
RETURN;
```

```
FUNCTION ANALYZEBOX (HMX, HY, HMY, HZ, HMZ, DIM1, DIM2, DIM3);

(If all dimensions of the given box fit the maximum space in the gap:)
If (DIM1<=HMX and DIM2<=HMY and DIM3<=HMZ) do{
        (If the y-dimension of the current orientation of the box fits to the gap's
                layer thickness:)
        If (DIM2<=HY) do {
                If the current box is a better fit in respect to its y-dimension compared to
                        the one selected before, keep the index of the current box in the
                        variable BOXI;
                If the current box has the same y-dimension as the y-dimension of the
                        selected one before, and the current box is a better fit in respect to
                        its x-dimension compared to the selected one before, keep the
                        index of the current box in the variable BOXI;
                If the current box has the same y and x-dimensions as the y and x
                        dimensions of the selected one before, and the current box is a
                        better fit in respect to its z-dimension compared to the one selected
                        before, keep the index of the current box in the variable BOXI;
        }
        (If the y-dimension of the current orientation of the box is bigger than the gap's
                layer thickness:)
        If (DIM2>HY) do {
                If the current box is a better fit in respect to its y-dimension compared to
                        the one selected before, keep the index of the current box in the
                        variable BBOXI;
                If the current box has the same y-dimension as the y-dimension of the
                        selected one before, and the current box is a better fit in respect to
                        its x-dimension compared to the selected one before, keep the
                        index of the current box in the variable BBOXI;
                If the current box has the same y and x-dimensions as the y and x-
                        dimensions of the selected one before, and the current box is a
                        better fit in respect to its z-dimension compared to the one selected
                        before, keep the index of the current box in the variable BBOXI;
        }
}
RETURN;
```

```
FUNCTION FINDSMALLEST();

Get the first node of the linked list representing the edge topology of the current layer:
        SCRAPMEMB=SCRAPFIRST;
Assign it to the variable which will keep the node with a smallest z-value:
        SMALLESTZ=SCRAPMEMB;
While SCRAPMEMB.POS≠NULL do {
        If (SCRAPMEMB.POS).CUMZ < SMALLEST.CUMZ then
                SMALLESTZ=SCRAPMEMB.POS;
        SCRAPMEMB=SCRAPMEMB.POS;
}
RETURN;


FUNCTION CHECKFOUND ();

(If a box fitting in the current layer thickness has been found, keep its index and
        orientation for packing:)
If BOXI≠0 then do{ CBOXI=BOXI; CBOXX=BOXX; CBOXY=BOXY;
        CBOXZ=BOXZ};
Else {
        If a box with a bigger y-dimension than the current layer thickness has been found
                and the edge of the current layer is even then select that box and set
                LAYERINLAYER variable for a second layer packing in the current layer
                and update the LAYERTHICKNESS=BBOXY;
        Else {
                If there is no gap in the edge of the current layer, packing of the layer is
                        done: LAYERDONE=1;
                Else: Since there is no fitting box to the currently selected gap, skip that
                        gap and even it by arranging and updating the necessary nodes and
                        variables;
        }
}
RETURN;
```

```
FUNCTION VOLUMECHECK ();

Mark the current box as packed: BOXLIST[CBOXI].PACKED=1;
Keep the orientation of the current box as it is packed:
        BOXLIST[CBOXI].PACKX=CBOXX;
        BOXLIST[CBOXI].PACKY=CBOXY;
        BOXLIST[CBOXI].PACKZ=CBOXZ;
Update the total packed volume:
        PACKEDVOLUME=PACKEDVOLUME+BOXLIST[CBOXI].VOL;
Update the number of boxes packed: PACKEDNUMBOX=PACKEDNUMBOX+1;
(If performing the best so far packing after being done with the iterations:)
If PACKINGBEST=1 do {
        To write the information of the packed box to the visualization data file named
                "VISUAL", call GRAPHUNPACKEDOUT;
        To write the information of the packed box to the report file,
                call OUTPUTBOXLIST;
}
Else if a hundred percent packing of the pallet has been reached or the total volume of the
        packed boxes is equal to the total volume of the input box set {
        Packing is finished: PACKING=0;
        A hundred percent packing has been reached: HUNDREDPERCENT=1;
}
RETURN;


FUNCTION GRAPHUNPACKEDOUT();

If this function is called for a visualization data out, write the necessary information to
        the file "VISUAL";
Else merge the unpacked box information to the end of the report file;
RETURN;


OUTPUTBOXLIST ();

Transform the coordinate system and orientation of every box from the best solution
        format to the pallet orientation entered by the user in the input text file by looking
        at the value of the variable BESTVARIANT;
Write the transformed box information (coordinates and the dimensions as is has been
        packed) to the REPORT file;
RETURN;
```

```
FUNCTION REPORT ();

Set the necessary variables to start the best packing found properly;
According to the BESTVARIANT value, determine the orientation of the pallet;
To tell other functions that the best packing found is being performed:
        PACKINGBEST=1;
Write the header information about the best solution found to the visualization data file
        "VISUAL";
Write the header information about the best solution found to the report data file;
List all possible candidate values by calling LISTCANDITLAYERS();
Sort the array LAYERS in respect to its LAYEREVAL fields in increasing order by
        using QSORT;
Set the starting layer thickness value to the best solution's starting layer thickness value:
        LAYERTHICKNESS= LAYERS[BESTITE].LAYERDIM;
Set all boxes' packed status to 0: For X=1 to TBN do BOXLIST[X].PACKST=0;
do {
        Set the variable that shows remaining unpacked potential second layer height in
                the current layer: LAYERINLAYER=0;
        Set the flag variable that shows packing of the current layer is finished or not:
                LAYERDONE=0;
        Call PACKLAYER(), to pack the layer, and if a memory error is responded, exit
                the program;
        If there is a height available for packing in the current layer, perform another
                layer packing in the current layer:
        If LAYERINLAYER≠0 do{
                Get the height available for packing in the current layer as the layer
                        thickness to be packed: LAYERTHICKNESS=LAYERINLAYER;
                Call PACKLAYER(), to pack the layer, and if a memory error is
                        responded, exit the program;
        }
        Call FINDLAYER(REMAINPY) to determine the most suitable layer height
                fitting in the remaining unpacked height of the pallet;
} While PACKING≠0;
Get the difference of the start time and the finish time;
Close both the visualization data file and the report file;
Write all the information about packing to the console;
RETURN;
```

Appendix B – The C Program Code of the Model

```
//**************************
// INCLUDED HEADER FILES
//**************************

#include <time.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <malloc.h>
#include <conio.h>


//**************************
// FUNCTION PROTOTYPES
//**************************

void initialize(void);
void inputboxlist(void);
void execiterations(void);
void listcanditlayers(void);
int complayerlist(const void *i, const void *j);
int packlayer(void);
int findlayer( short int thickness);
void findbox(short int hmx, short int hy, short int hmy, short int hz, short int hmz);
void analyzebox (short int hmx, short int hy, short int hmy, short int hz, short int hmz,
                                    short int dim1, short int dim2, short int dim3);
void findsmallestz(void);
void checkfound(void);
void volumecheck (void);
void graphunpackedout(void);
void outputboxlist(void);
void report(void);
```

```
//*********************************************************
// VARIABLE, CONSTANT AND STRUCTURE DECLARATIONS
//*********************************************************

char strpx[5], strpy[5], strpz[5], strcox[5], strcoy[5], strcoz[5], strpackx[5], strpacky[5],
        strpackz[5], filename[12], strtemp[]="", packing, layerdone, evened, variant,
        bestvariant, packingbest, hundredpercent, graphout[]="visudat", unpacked, quit;

short int tbn, x, n, layerlistlen, layerinlayer, prelayer, lilz, itenum, hour, min, sec,
        layersindex, remainpx, remainpy, remainpz, packedy, prepackedy, layerthickness,
        itelayer, boxx, boxy, boxz, boxi, bboxx, bboxy, bboxz, bboxi, preremainpy, cboxi,
        cboxx, cboxy, cboxz, bfx, bfy, bfz, bbfx, bbfy, bbfz, bestite, packednumbox,
        bestpackednum, xx, yy, zz, px, py, pz;

double packedvolume, bestvolume, totalvolume, totalboxvol, temp, percentageused,
        percentagepackedbox, elapsedtime;

struct boxinfo {
        char packst;
        short int dim1, dim2, dim3, n, cox, coy, coz, packx, packy, packz;
        long int vol;
} boxlist[5000];

struct layerlist{
        long int layereval;
        short int layerdim;
} layers[1000];

struct scrappad{
        struct scrappad *pre, *pos;
        short int cumx, cumz;
};

struct scrappad *scrapfirst, *scrapmemb, *smallestz, *trash;

time_t start, finish;

FILE *ifp, *ofp, *gfp;
```

```
//*****************
// MAIN PROGRAM
//*****************

int main(int argc, char *argv[]) {
        if (argc==1){
                printf ("(ASSUMED TO HAVE '.TXT' EXTENSION; UP TO 8
                        CHARACTERS LONG)\n");
                printf ("PLEASE ENTER THE NAME OF THE INPUT FILE : ");
                scanf ("%s",filename);
        }
        else {
                printf("%s", argv[1]);
                strcpy(filename, argv[1]);
        }
        initialize();
        time(&start);
        printf("\nPRESS Q TO QUIT AT ANYTIME AND WAIT\n\n");
        execiterations();
        time(&finish);
        report();
        getch();
        return 0;
}


//*************************
// PERFORMS INITIALIZATIONS
//*************************

void initialize(void){
        if (filename==""){
                printf("\nINVALID FILE NAME\n");
                exit(1);
        }
        inputboxlist();
        temp=1.0; totalvolume=temp*xx*yy*zz; totalboxvol=0.0;
        for (x=1; x<=tbn; x++) totalboxvol=totalboxvol+boxlist[x].vol;
        scrapfirst=malloc(sizeof(struct scrappad));
        if ((*scrapfirst).pos==NULL){
                printf("Insufficient memory available\n");
                exit(1);
        }
        (*scrapfirst).pre=NULL;(*scrapfirst).pos=NULL;
        bestvolume=0.0; packingbest=0; hundredpercent=0; itenum=0; quit=0;
}
```

```
//*************************************************************************
// READS THE PALLET AND BOX SET DATA ENTERED BY THE USER FROM
//                                THE INPUT FILE
//*************************************************************************

void inputboxlist(void){

        short int n;
        char lbl[5], dim1[5], dim2[5], dim3[5], boxn[5], strxx[5], stryy[5], strzz[5];

        strcpy(strtemp, filename);
        strcat(strtemp,".txt");
        if ((ifp=fopen(strtemp,"r"))==NULL) {
                printf("Cannot open file %s", strtemp);
                exit(1);
        }
        tbn=1;
        if (fscanf(ifp,"%s %s %s",strxx, stryy, strzz)==EOF) exit(1);
        xx=atoi(strxx); yy=atoi(stryy); zz=atoi(strzz);
        while (fscanf(ifp,"%s %s %s %s %s",lbl,dim1,dim2,dim3,boxn)!=EOF){
                boxlist[tbn].dim1=atoi(dim1);
                boxlist[tbn].dim2=atoi(dim2);
                boxlist[tbn].dim3=atoi(dim3);
                boxlist[tbn].vol=boxlist[tbn].dim1*boxlist[tbn].dim2*boxlist[tbn].dim3;
                n=atoi(boxn); boxlist[tbn].n=n;
                while (--n) boxlist[tbn+n]=boxlist[tbn];
                tbn=tbn+atoi(boxn);
        }
        --tbn;
        fclose(ifp);
        return;
}
```

```
//*************************************************************************
// ITERATIONS ARE DONE AND PARAMETERS OF THE BEST SOLUTION ARE
//                                        FOUND
//*************************************************************************

void execiterations(void){
        for (variant=1; (variant<=6) & !quit; variant++){
                switch(variant){
                        case 1:
                                px=xx; py=yy; pz=zz;
                                break;
                        case 2:
                                px=zz; py=yy; pz=xx;
                                break;
                        case 3:
                                px=zz; py=xx; pz=yy;
                                break;
                        case 4:
                                px=yy; py=xx; pz=zz;
                                break;
                        case 5:
                                px=xx; py=zz; pz=yy;
                                break;
                        case 6:
                                px=yy; py=zz; pz=xx;
                                break;
                }
                listcanditlayers();
                layers[0].layereval=-1;
                qsort(layers,layerlistlen+1,sizeof(struct layerlist),complayerlist);
                for (layersindex=1; (layersindex<=layerlistlen) & !quit; layersindex++){
                        ++itenum;
                        time(&finish);
                        elapsedtime = difftime( finish, start );
                        printf("VARIANT: %5d; ITERATION (TOTAL): %5d; BEST SO
                                FAR: %.3f%%; TIME: %.0f",
                                variant, itenum, percentageused, elapsedtime);
                        packedvolume=0.0;
                        packedy=0;
                        packing=1;
                        layerthickness=layers[layersindex].layerdim;
                        itelayer=layersindex;
                        remainpy=py; remainpz=pz;
                        packednumbox=0;
                        for (x=1; x<=tbn; x++) boxlist[x].packst=0;
```

```
            do {
                layerinlayer=0;
                layerdone=0;
                if (packlayer()) exit(1);
                packedy=packedy+layerthickness;
                remainpy=py-packedy;
                if (layerinlayer & !quit){
                    prepackedy=packedy;
                    preremainpy=remainpy;
                    remainpy=layerthickness-prelayer;
                    packedy=packedy-layerthickness+prelayer;
                    remainpz=lilz;
                    layerthickness=layerinlayer;
                    layerdone=0;
                    if (packlayer()) exit(1);
                    packedy=prepackedy;
                    remainpy=preremainpy;
                    remainpz=pz;
                }
                findlayer(remainpy);
            }
            while (packing & !quit);
            if ((packedvolume>bestvolume) & !quit) {
                bestvolume=packedvolume;
                bestvariant=variant;
                bestite=itelayer;
                bestpackednum=packednumbox;
            }
            if (hundredpercent) break;
            percentageused=bestvolume*100/totalvolume;
            printf("\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
                \b\b\b\b\b\b\b\b\b\b\b");
            printf("\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
                \b\b\b\b\b\b\b");
        }
        if (hundredpercent) break;
        if ((xx==yy) & (yy==zz)) variant=6;
    }
}
```

```
//*************************************************************************
// LISTS ALL POSSIBLE LAYER HEIGHTS BY GIVING A WEIGHT VALUE TO
//                                   EACH OF THEM.
//*************************************************************************

void listcanditlayers(void){

        char same;
        short int exdim,dimdif,dimen2,dimen3,y,z,k;
        long int layereval;

        layerlistlen=0;
        for (x=1; x<=tbn; x++){
                for (y=1; y<=3; y++){
                        switch(y){
                        case 1:
                                exdim=boxlist[x].dim1;
                                dimen2=boxlist[x].dim2;
                                dimen3=boxlist[x].dim3;
                                break;
                        case 2:
                                exdim=boxlist[x].dim2;
                                dimen2=boxlist[x].dim1;
                                dimen3=boxlist[x].dim3;
                                break;
                        case 3:
                                exdim=boxlist[x].dim3;
                                dimen2=boxlist[x].dim1;
                                dimen3=boxlist[x].dim2;
                                break;
                        }
                        if ((exdim>py) || (((dimen2>px) || (dimen3>pz)) &
                                                ((dimen3>px) || (dimen2>pz))) ) continue;
                        same=0;
                        for (k=1; k<=layerlistlen; k++) if (exdim==layers[k].layerdim){
                                same=1;
                                continue;
                        }
                        if (same) continue;
                        layereval=0;
```

```
                    for (z=1; z<=tbn; z++){
                        if (!(x==z)){
                            dimdif=abs(exdim-boxlist[z].dim1);
                            if (abs(exdim-boxlist[z].dim2)<dimdif)
                                dimdif=abs(exdim-boxlist[z].dim2);
                            if (abs(exdim-boxlist[z].dim3)<dimdif)
                                dimdif=abs(exdim-boxlist[z].dim3);
                            layereval=layereval+dimdif;
                        }
                    }
                    layers[++layerlistlen].layereval=layereval;
                    layers[layerlistlen].layerdim=exdim;
                }
            }
            return;
}


//*****************************************************
// REQUIRED FUNCTION FOR QSORT FUNCTION TO WORK
//*****************************************************

int complayerlist(const void *i, const void *j){
        return *( long int*)i-*( long int*)j;
}


//*************************************************************
// PACKS THE BOXES FOUND AND ARRANGES ALL VARIABLES AND
//                      RECORDS PROPERLY
//*************************************************************

int packlayer( void){

        short int lenx, lenz, lpz;

        if (!layerthickness) {
                packing=0; return 0;
        }
        (*scrapfirst).cumx=px;(*scrapfirst).cumz=0;
        for (;!quit;){
                if (kbhit()) if (toupper(getch())=='Q' ) {
                        quit=1;
                        printf("\n\nWait please...\n");
                }
                findsmallestz();
```

```
//*** SITUATION-1: NO BOXES ON THE RIGHT AND LEFT SIDES ***

        if (!(*smallestz).pre & !(*smallestz).pos){
                lenx=(*smallestz).cumx;
                lpz=remainpz-(*smallestz).cumz;
                findbox(lenx, layerthickness, remainpy, lpz, lpz);
                checkfound();
                if (layerdone) break;
                if (evened) continue;
                boxlist[cboxi].cox=0; boxlist[cboxi].coy=packedy;
                boxlist[cboxi].coz=(*smallestz).cumz;
                if (cboxx==(*smallestz).cumx)
                        (*smallestz).cumz=(*smallestz).cumz+cboxz;
                else {
                        (*smallestz).pos=malloc(sizeof(struct scrappad));
                        if ((*smallestz).pos==NULL){
                                printf("Insufficient memory available\n");
                                return 1;
                        }
                        (*((*smallestz).pos)).pos=NULL;
                        (*((*smallestz).pos)).pre=smallestz;
                        (*((*smallestz).pos)).cumx=(*smallestz).cumx;
                        (*((*smallestz).pos)).cumz=(*smallestz).cumz;
                        (*smallestz).cumx=cboxx;
                        (*smallestz).cumz=(*smallestz).cumz+cboxz;
                }
                volumecheck();
        }

        //*** SITUATION-2: NO BOXES ON THE LEFT SIDE ***

        else if (!(*smallestz).pre){
                lenx=(*smallestz).cumx;
                lenz=(*((*smallestz).pos)).cumz-(*smallestz).cumz;
                lpz=remainpz-(*smallestz).cumz;
                findbox(lenx, layerthickness, remainpy, lenz, lpz);
                checkfound();
                if (layerdone) break;
                if (evened) continue;
                boxlist[cboxi].coy=packedy;
                boxlist[cboxi].coz=(*smallestz).cumz;
                if (cboxx==(*smallestz).cumx){
                        boxlist[cboxi].cox=0;
```

```
                if (*smallestz).cumz+cboxz==(*((*smallestz).pos)).cumz){
                        (*smallestz).cumz=(*((*smallestz).pos)).cumz;
                        (*smallestz).cumx=(*((*smallestz).pos)).cumx;
                        trash=(*smallestz).pos;
                        (*smallestz).pos=(*((*smallestz).pos)).pos;
                        if ((*smallestz).pos)
                                (*((*smallestz).pos)).pre=smallestz;
                        free(trash);
                }
                else (*smallestz).cumz=(*smallestz).cumz+cboxz;
        }
        else {
                boxlist[cboxi].cox=(*smallestz).cumx-cboxx;
                if ((*smallestz).cumz+cboxz==(*((*smallestz).pos)).cumz)
                        (*smallestz).cumx=(*smallestz).cumx-cboxx;
                else {
                        (*((*smallestz).pos)).pre=
                                        malloc(sizeof(struct scrappad));
                        if ((*((*smallestz).pos)).pre==NULL){
                                printf("Insufficient memory available\n");
                                return 1;
                        }
                        (*((*((*smallestz).pos)).pre)).pos=(*smallestz).pos;
                        (*((*((*smallestz).pos)).pre)).pre=smallestz;
                        (*smallestz).pos=(*((*smallestz).pos)).pre;
                        (*((*smallestz).pos)).cumx=(*smallestz).cumx;
                        (*smallestz).cumx=(*smallestz).cumx-cboxx;
                        (*((*smallestz).pos)).cumz=
                                                (*smallestz).cumz+cboxz;
                }
        }
        volumecheck();
}
```

```
//*** SITUATION-3: NO BOXES ON THE RIGHT SIDE ***

else if (!(*smallestz).pos){
        lenx=(*smallestz).cumx-(*((*smallestz).pre)).cumx;
        lenz=(*((*smallestz).pre)).cumz-(*smallestz).cumz;
        lpz=remainpz-(*smallestz).cumz;
        findbox(lenx, layerthickness, remainpy, lenz, lpz);
        checkfound();
        if (layerdone) break;
        if (evened) continue;
        boxlist[cboxi].coy=packedy;
        boxlist[cboxi].coz=(*smallestz).cumz;
        boxlist[cboxi].cox=(*((*smallestz).pre)).cumx;
        if (cboxx==(*smallestz).cumx-(*((*smallestz).pre)).cumx){
                if((*smallestz).cumz+cboxz==(*((*smallestz).pre)).cumz){
                        (*((*smallestz).pre)).cumx=(*smallestz).cumx;
                        (*((*smallestz).pre)).pos=NULL;
                        free(smallestz);
                }
                else (*smallestz).cumz=(*smallestz).cumz+cboxz;
        }
        else {
                if ((*smallestz).cumz+cboxz==(*((*smallestz).pre)).cumz)
                        (*((*smallestz).pre)).cumx=
                                        (*((*smallestz).pre)).cumx+cboxx;
                else {
                        (*((*smallestz).pre)).pos=
                                        malloc(sizeof(struct scrappad));
                        if ((*((*smallestz).pre)).pos==NULL){
                                printf("Insufficient memory available\n");
                                return 1;
                        }
                        (*((*((*smallestz).pre)).pos)).pre=(*smallestz).pre;
                        (*((*((*smallestz).pre)).pos)).pos=smallestz;
                        (*smallestz).pre=(*((*smallestz).pre)).pos;
                        (*((*smallestz).pre)).cumx=
                                (*((*((*smallestz).pre)).pre)).cumx+cboxx;
                        (*((*smallestz).pre)).cumz=
                                                (*smallestz).cumz+cboxz;
                }
        }
        volumecheck();
}
```

```
//*** SITUATION-4: THERE ARE BOXES ON BOTH OF THE SIDES ***

//*** SUBSITUATION-4A: SIDES ARE EQUAL TO EACH OTHER ***

        else if ((*((*smallestz).pre)).cumz==(*((*smallestz).pos)).cumz){
                lenx=(*smallestz).cumx-(*((*smallestz).pre)).cumx;
                lenz=(*((*smallestz).pre)).cumz-(*smallestz).cumz;
                lpz=remainpz-(*smallestz).cumz;
                findbox(lenx, layerthickness, remainpy, lenz, lpz);
                checkfound();
                if (layerdone) break;
                if (evened) continue;
                boxlist[cboxi].coy=packedy;boxlist[cboxi].coz=(*smallestz).cumz;
                if (cboxx==(*smallestz).cumx-(*((*smallestz).pre)).cumx){
                        boxlist[cboxi].cox=(*((*smallestz).pre)).cumx;
                        if((*smallestz).cumz+cboxz==(*((*smallestz).pos)).cumz){
                                (*((*smallestz).pre)).cumx=
                                                (*((*smallestz).pos)).cumx;
                                if ((*((*smallestz).pos)).pos) {
                                        (*((*smallestz).pre)).pos=
                                                (*((*smallestz).pos)).pos;
                                        (*((*((*smallestz).pos)).pos)).pre=
                                                        (*smallestz).pre;
                                        free(smallestz);
                                }
                                else {
                                        (*((*smallestz).pre)).pos=NULL;
                                        free(smallestz);
                                }
                        }
                        else (*smallestz).cumz=(*smallestz).cumz+cboxz;
                }
                else if ((*((*smallestz).pre)).cumx<px-(*smallestz).cumx){
                        if((*smallestz).cumz+cboxz==(*((*smallestz).pre)).cumz){
                                (*smallestz).cumx=(*smallestz).cumx-cboxx;
                                boxlist[cboxi].cox=(*smallestz).cumx-cboxx;
                        }
                        else {
                                boxlist[cboxi].cox=(*((*smallestz).pre)).cumx;
                                (*((*smallestz).pre)).pos=
                                                malloc(sizeof(struct scrappad));
                                if ((*((*smallestz).pre)).pos==NULL){
                                        printf("Insufficient memory available\n");
                                        return 1;
                                }
                                (*((*((*smallestz).pre)).pos)).pre=(*smallestz).pre;
```

```
                        (*((*((*smallestz).pre)).pos)).pos=smallestz;
                        (*smallestz).pre=(*((*smallestz).pre)).pos;
                        (*((*smallestz).pre)).cumx=
                                (*((*((*smallestz).pre)).pre)).cumx+cboxx;
                        (*((*smallestz).pre)).cumz=
                                                (*smallestz).cumz+cboxz;
                }
        }
        else {
                if((*smallestz).cumz+cboxz==(*((*smallestz).pre)).cumz){
                        (*((*smallestz).pre)).cumx=
                                        (*((*smallestz).pre)).cumx+cboxx;
                        boxlist[cboxi].cox=(*((*smallestz).pre)).cumx;
                }
                else {
                        boxlist[cboxi].cox=(*smallestz).cumx-cboxx;
                        (*((*smallestz).pos)).pre=
                                        malloc(sizeof(struct scrappad));
                        if ((*((*smallestz).pos)).pre==NULL){
                                printf("Insufficient memory available\n");
                                return 1;
                        }
                        (*((*((*smallestz).pos)).pre)).pos=(*smallestz).pos;
                        (*((*((*smallestz).pos)).pre)).pre=smallestz;
                        (*smallestz).pos=(*((*smallestz).pos)).pre;
                        (*((*smallestz).pos)).cumx=(*smallestz).cumx;
                        (*((*smallestz).pos)).cumz=
                                                (*smallestz).cumz+cboxz;
                        (*smallestz).cumx=(*smallestz).cumx-cboxx;
                }
        }
        volumecheck();
}


//*** SUBSITUATION-4B: SIDES ARE NOT EQUAL TO EACH OTHER ***

        else {
                lenx=(*smallestz).cumx-(*((*smallestz).pre)).cumx;
                lenz=(*((*smallestz).pre)).cumz-(*smallestz).cumz;
                lpz=remainpz-(*smallestz).cumz;
                findbox(lenx, layerthickness, remainpy, lenz, lpz);
                checkfound();
                if (layerdone) break;
                if (evened) continue;
                boxlist[cboxi].coy=packedy;boxlist[cboxi].coz=(*smallestz).cumz;
```

```
                    boxlist[cboxi].cox=(*((*smallestz).pre)).cumx;
                    if (cboxx==(*smallestz).cumx-(*((*smallestz).pre)).cumx){
                        if((*smallestz).cumz+cboxz==(*((*smallestz).pre)).cumz){
                            (*((*smallestz).pre)).cumx=(*smallestz).cumx;
                            (*((*smallestz).pre)).pos=(*smallestz).pos;
                            (*((*smallestz).pos)).pre=(*smallestz).pre;
                            free(smallestz);
                        }
                        else (*smallestz).cumz=(*smallestz).cumz+cboxz;
                    }
                    else {
                        if ((*smallestz).cumz+cboxz==(*((*smallestz).pre)).cumz)
                            (*((*smallestz).pre)).cumx=
                                        (*((*smallestz).pre)).cumx+cboxx;
                        else if ((*smallestz).cumz+cboxz==
                                                (*((*smallestz).pos)).cumz){
                            boxlist[cboxi].cox=(*smallestz).cumx-cboxx;
                            (*smallestz).cumx=(*smallestz).cumx-cboxx;
                        }
                        else{
                            (*((*smallestz).pre)).pos=
                                        malloc(sizeof(struct scrappad));
                            if ((*((*smallestz).pre)).pos==NULL){
                                printf("Insufficient memory available\n");
                                return 1;
                            }
                            (*((*((*smallestz).pre)).pos)).pre=(*smallestz).pre;
                            (*((*((*smallestz).pre)).pos)).pos=smallestz;
                            (*smallestz).pre=(*((*smallestz).pre)).pos;
                            (*((*smallestz).pre)).cumx=
                                (*((*((*smallestz).pre)).pre)).cumx+cboxx;
                            (*((*smallestz).pre)).cumz=
                                                (*smallestz).cumz+cboxz;
                        }
                    }
                    volumecheck();
                }
            }
            return 0;
}
```

```
//*************************************************************************
// FINDS THE MOST PROPER LAYER HIGHT BY LOOKING AT THE UNPACKED
//              BOXES AND THE REMAINING EMPTY SPACE AVAILABLE
//*************************************************************************

int findlayer( short int thickness){

        short int exdim,dimdif,dimen2,dimen3,y,z;
        long int layereval, eval;

        layerthickness=0;
        eval=1000000;
        for (x=1; x<=tbn; x++){
                if (boxlist[x].packst) continue;
                for (y=1; y<=3; y++){
                        switch(y){
                        case 1:
                                exdim=boxlist[x].dim1;
                                dimen2=boxlist[x].dim2;
                                dimen3=boxlist[x].dim3;
                                break;
                        case 2:
                                exdim=boxlist[x].dim2;
                                dimen2=boxlist[x].dim1;
                                dimen3=boxlist[x].dim3;
                                break;
                        case 3:
                                exdim=boxlist[x].dim3;
                                dimen2=boxlist[x].dim1;
                                dimen3=boxlist[x].dim2;
                                break;
                        }
                        layereval=0;
                        if ((exdim<=thickness) & (((dimen2<=px) &
                                (dimen3<=pz)) || ((dimen3<=px) & (dimen2<=pz))) ) {
                                for (z=1; z<=tbn; z++){
                                        if (!(x==z) & !(boxlist[z].packst)){
                                                dimdif=abs(exdim-boxlist[z].dim1);
                                                if (abs(exdim-boxlist[z].dim2)<dimdif)
                                                        dimdif=abs(exdim-boxlist[z].dim2);
                                                if (abs(exdim-boxlist[z].dim3)<dimdif)
                                                        dimdif=abs(exdim-boxlist[z].dim3);
                                                layereval=layereval+dimdif;
                                        }
                                }
                                if (layereval<eval) {
```

```
                                        eval=layereval;
                                        layerthickness=exdim;
                                }
                        }
                }
        }
        if (layerthickness==0 || layerthickness>remainpy) packing=0;
        return 0;
}


//*****************************************************************
// FINDS THE MOST PROPER BOXES BY LOOKING AT ALL SIX POSSIBLE
//      ORIENTATIONS, EMPTY SPACE GIVEN, ADJACENT BOXES,
//                          AND PALLET LIMITS
//*****************************************************************

void findbox(short int hmx, short int hy, short int hmy, short int hz, short int hmz){
        short int y;
        bfx=32767; bfy=32767; bfz=32767;
        bbfx=32767; bbfy=32767; bbfz=32767;
        boxi=0; bboxi=0;
        for (y=1; y<=tbn; y=y+boxlist[y].n){
                for (x=y; x<x+boxlist[y].n-1; x++) if (!boxlist[x].packst) break;
                if (boxlist[x].packst) continue;
                if (x>tbn) return;
                analyzebox (hmx, hy, hmy, hz, hmz, boxlist[x].dim1, boxlist[x].dim2,
                                                        boxlist[x].dim3);
                if ((boxlist[x].dim1==boxlist[x].dim3) &
                                        (boxlist[x].dim3==boxlist[x].dim2)) continue;
                analyzebox (hmx, hy, hmy, hz, hmz, boxlist[x].dim1, boxlist[x].dim3,
                                                        boxlist[x].dim2);
                analyzebox (hmx, hy, hmy, hz, hmz, boxlist[x].dim2, boxlist[x].dim1,
                                                        boxlist[x].dim3);
                analyzebox (hmx, hy, hmy, hz, hmz, boxlist[x].dim2, boxlist[x].dim3,
                                                        boxlist[x].dim1);
                analyzebox (hmx, hy, hmy, hz, hmz, boxlist[x].dim3, boxlist[x].dim1,
                                                        boxlist[x].dim2);
                analyzebox (hmx, hy, hmy, hz, hmz, boxlist[x].dim3, boxlist[x].dim2,
                                                        boxlist[x].dim1);
        }
}
```

```
//*********************************************************************
// ANALYZES EACH UNPACKED BOX TO FIND THE BEST FITTING ONE TO
//                          THE EMPTY SPACE GIVEN
//*********************************************************************

void analyzebox ( short int hmx,  short int hy,  short int hmy,
                          short int hz,  short int hmz,
                          short int dim1,  short int dim2,  short int dim3){
        if (dim1<=hmx && dim2<=hmy && dim3<=hmz){
                if (dim2<=hy) {
                        if (hy-dim2<bfy) {
                                boxx=dim1; boxy=dim2; boxz=dim3;
                                bfx=hmx-dim1; bfy=hy-dim2; bfz=abs(hz-dim3); boxi=x;
                        }
                        else if (hy-dim2==bfy && hmx-dim1<bfx) {
                                boxx=dim1; boxy=dim2; boxz=dim3;
                                bfx=hmx-dim1; bfy=hy-dim2; bfz=abs(hz-dim3); boxi=x;
                        }
                        else if (hy-dim2==bfy && hmx-dim1==
                                                        bfx && abs(hz-dim3)<bfz) {
                                boxx=dim1; boxy=dim2; boxz=dim3;
                                bfx=hmx-dim1; bfy=hy-dim2; bfz=abs(hz-dim3); boxi=x;
                        }
                }
                else {
                        if (dim2-hy<bbfy) {
                                bboxx=dim1; bboxy=dim2; bboxz=dim3;
                                bbfx=hmx-dim1; bbfy=dim2-hy;
                                bbfz=abs(hz-dim3); bboxi=x;
                        }
                        else if (dim2-hy==bbfy && hmx-dim1<bbfx) {
                                bboxx=dim1; bboxy=dim2; bboxz=dim3;
                                bbfx=hmx-dim1; bbfy=dim2-hy;
                                bbfz=abs(hz-dim3); bboxi=x;
                        }
                        else if (dim2-hy==bbfy &&
                                                hmx-dim1==bbfx && abs(hz-dim3)<bbfz) {
                                bboxx=dim1; bboxy=dim2; bboxz=dim3;
                                bbfx=hmx-dim1; bbfy=dim2-hy;
                                bbfz=abs(hz-dim3); bboxi=x;
                        }
                }
        }
}
```

```
//*********************************************************
// FINDS THE FIRST TO BE PACKED GAP IN THE LAYER EDGE
//*********************************************************

void findsmallestz(void){
        scrapmemb=scrapfirst;
        smallestz=scrapmemb;
        while (!((*scrapmemb).pos==NULL)){
                if ((*((*scrapmemb).pos)).cumz<(*smallestz).cumz)
                                                        smallestz=(*scrapmemb).pos;
                scrapmemb=(*scrapmemb).pos;
        }
        return;
}


//*************************************************************
// AFTER FINDING EACH BOX, THE CANDIDATE BOXES AND THE
//              CONDITION OF THE LAYER ARE EXAMINED
//*************************************************************

void checkfound(void){
        evened=0;
        if (boxi) {
                cboxi=boxi;cboxx=boxx;cboxy=boxy;cboxz=boxz;
        }
        else {
                if ((bboxi>0) & (layerinlayer || (!(*smallestz).pre & !(*smallestz).pos))){
                        if (!layerinlayer) {
                                prelayer=layerthickness;
                                lilz=(*smallestz).cumz;
                        }
                        cboxi=bboxi;cboxx=bboxx;cboxy=bboxy;cboxz=bboxz;
                        layerinlayer=layerinlayer+bboxy-layerthickness;
                        layerthickness=bboxy;
                }
                else {
                        if (!(*smallestz).pre & !(*smallestz).pos) layerdone=1;
                        else {
                                evened=1;
                                if (!(*smallestz).pre){
                                        trash=(*smallestz).pos;
                                        (*smallestz).cumx=(*((*smallestz).pos)).cumx;
                                        (*smallestz).cumz=(*((*smallestz).pos)).cumz;
                                        (*smallestz).pos=(*((*smallestz).pos)).pos;
                                        if ((*smallestz).pos)
```

```
                                        (*((*smallestz).pos)).pre=smallestz;
                        free(trash);
                }
                else if (!(*smallestz).pos){
                        (*((*smallestz).pre)).pos=NULL;
                        (*((*smallestz).pre)).cumx=(*smallestz).cumx;
                        free(smallestz);
                }
                else {
                        if((*((*smallestz).pre)).cumz==
                                        (*((*smallestz).pos)).cumz){
                                (*((*smallestz).pre)).pos=
                                        (*((*smallestz).pos)).pos;
                                if ((*((*smallestz).pos)).pos)
                                        (*((*((*smallestz).pos)).pos)).pre=
                                                (*smallestz).pre;
                                (*((*smallestz).pre)).cumx=
                                        (*((*smallestz).pos)).cumx;
                                free((*smallestz).pos);
                                free(smallestz);
                        }
                        else {
                                (*((*smallestz).pre)).pos=(*smallestz).pos;
                                (*((*smallestz).pos)).pre=(*smallestz).pre;
                                if ((*((*smallestz).pre)).cumz<
                                        (*((*smallestz).pos)).cumz)
                                        (*((*smallestz).pre)).cumx=
                                                (*smallestz).cumx;
                                free(smallestz);
                        }
                }
                        }
                }
        }
        return;
}
```

```
//*********************************************************************
// AFTER PACKING OF EACH BOX, 100% PACKING CONDITION IS CHECKED
//*********************************************************************

void volumecheck (void){
        boxlist[cboxi].packst=1;
        boxlist[cboxi].packx=cboxx;boxlist[cboxi].packy=cboxy;
        boxlist[cboxi].packz=cboxz;
        packedvolume=packedvolume+boxlist[cboxi].vol;
        packednumbox++;
        if (packingbest) {
                graphunpackedout();
                outputboxlist();
        }
        else if (packedvolume==totalvolume || packedvolume==totalboxvol) {
                packing=0;
                hundredpercent=1;
        }
        return;
}


//*************************************************************
// DATA FOR THE VISUALIZATION PROGRAM IS WRITTEN TO THE
//       "VISUDAT" FILE AND THE LIST OF UNPACKED BOXES IS
//              MERGED TO THE END OF THE REPORT FILE
//*************************************************************

void graphunpackedout(void){
        char n[5];
        if (!unpacked){
                itoa( boxlist[cboxi].cox, strcox, 10); itoa( boxlist[cboxi].coy, strcoy, 10);
                itoa( boxlist[cboxi].coz, strcoz, 10);
                itoa( boxlist[cboxi].packx, strpackx, 10);
                itoa( boxlist[cboxi].packy, strpacky, 10);
                itoa( boxlist[cboxi].packz, strpackz, 10);
        }
        else {
                itoa( cboxi, n, 10); itoa( boxlist[cboxi].dim1, strpackx, 10);
                itoa( boxlist[cboxi].dim2, strpacky, 10);
                itoa( boxlist[cboxi].dim3, strpackz, 10);
        }
        if (!unpacked) fprintf(gfp,"%5s%5s%5s%5s%5s%5s\n",
                                strcox,strcoy,strcoz,strpackx,strpacky,strpackz);
        else fprintf(ofp,"%5s%5s%5s%5s%5s\n",n,strpackx,strpacky,strpackz);
}
```

```
//*********************************************************************
// TRANSFORMS THE FOUND COORDINATE SYSTEM TO THE ONE ENTERED
//              BY THE USER AND WRITES THEM TO THE REPORT FILE
//*********************************************************************

void outputboxlist(void){
        char strx[5], strpackst[5], strdim1[5], strdim2[5], strdim3[5], strcox[5], strcoy[5],
strcoz[5], strpackx[5], strpacky[5], strpackz[5];
        short int x, y, z, bx, by, bz;
        switch(bestvariant){
                case 1:
                        x=boxlist[cboxi].cox; y=boxlist[cboxi].coy; z=boxlist[cboxi].coz;
                        bx=boxlist[cboxi].packx; by=boxlist[cboxi].packy;
                        bz=boxlist[cboxi].packz;
                        break;
                case 2:
                        x=boxlist[cboxi].coz; y=boxlist[cboxi].coy; z=boxlist[cboxi].cox;
                        bx=boxlist[cboxi].packz; by=boxlist[cboxi].packy;
                        bz=boxlist[cboxi].packx;
                        break;
                case 3:
                        x=boxlist[cboxi].coy; y=boxlist[cboxi].coz; z=boxlist[cboxi].cox;
                        bx=boxlist[cboxi].packy; by=boxlist[cboxi].packz;
                        bz=boxlist[cboxi].packx;
                        break;
                case 4:
                        x=boxlist[cboxi].coy; y=boxlist[cboxi].cox; z=boxlist[cboxi].coz;
                        bx=boxlist[cboxi].packy; by=boxlist[cboxi].packx;
                        bz=boxlist[cboxi].packz;
                        break;
                case 5:
                        x=boxlist[cboxi].cox; y=boxlist[cboxi].coz; z=boxlist[cboxi].coy;
                        bx=boxlist[cboxi].packx; by=boxlist[cboxi].packz;
                        bz=boxlist[cboxi].packy;
                        break;
                case 6:
                        x=boxlist[cboxi].coz; y=boxlist[cboxi].cox; z=boxlist[cboxi].coy;
                        bx=boxlist[cboxi].packz; by=boxlist[cboxi].packx;
                        bz=boxlist[cboxi].packy;
                        break;
        }
        itoa( cboxi, strx,10);
        itoa( boxlist[cboxi].packst, strpackst, 10);
        itoa( boxlist[cboxi].dim1, strdim1, 10);
        itoa( boxlist[cboxi].dim2, strdim2, 10);
        itoa( boxlist[cboxi].dim3, strdim3, 10);
```

```
        itoa( x, strcox, 10);
        itoa( y, strcoy, 10);
        itoa( z, strcoz, 10);
        itoa( bx, strpackx, 10);
        itoa( by, strpacky, 10);
        itoa( bz, strpackz, 10);
        boxlist[cboxi].cox=x; boxlist[cboxi].coy=y; boxlist[cboxi].coz=z;
        boxlist[cboxi].packx=bx; boxlist[cboxi].packy=by; boxlist[cboxi].packz=bz;
        fprintf(ofp,"%5s%5s%9s%9s%9s%9s%9s%9s%9s%9s%9s\n",strx,strpackst,
                strdim1,strdim2,strdim3,strcox,strcoy,strcoz,strpackx,strpacky,strpackz);
        return;
}


//*********************************************************************
// USING THE PARAMETERS FOUND, PACKS THE BEST SOLUTION FOUND
//              AND REPORS TO THE CONSOLE AND TO A TEXT FILE
//*********************************************************************

void report(void){
        quit=0;
        switch(bestvariant){
                case 1:
                        px=xx; py=yy; pz=zz;
                        break;
                case 2:
                        px=zz; py=yy; pz=xx;
                        break;
                case 3:
                        px=zz; py=xx; pz=yy;
                        break;
                case 4:
                        px=yy; py=xx; pz=zz;
                        break;
                case 5:
                        px=xx; py=zz; pz=yy;
                        break;
                case 6:
                        px=yy; py=zz; pz=xx;
                        break;
        }
        packingbest=1;
        if ((gfp=fopen(graphout,"w"))==NULL) {
                printf("Cannot open file %s", filename);
                exit(1);
        }
```

```
itoa( px, strpx, 10);
itoa( py, strpy, 10);
itoa( pz, strpz, 10);
fprintf(gfp,"%5s%5s%5s\n",strpx,strpy,strpz);
strcat(filename,".out");
if ((ofp=fopen(filename,"w"))==NULL) {
        printf("Cannot open file %s", filename);
        exit(1);
}
percentagepackedbox=bestvolume*100/totalboxvol;
percentageused=bestvolume*100/totalvolume;
elapsedtime = difftime( finish, start );
fprintf(ofp,"                                    *** REPORT ***\n\n");
fprintf(ofp," ELAPSED TIME                                          :
                                    Almost %.0f sec\n", elapsedtime);
fprintf(ofp," TOTAL NUMBER OF ITERATIONS DONE          : %d\n",
                                                        itenum);
fprintf(ofp," BEST SOLUTION FOUND AT                                :
              ITERATION: %d OF VARIANT: %d\n", bestite, bestvariant);
fprintf(ofp," TOTAL NUMBER OF BOXES                    : %d\n", tbn);
fprintf(ofp," PACKED NUMBER OF BOXES                                :
                                    %d\n", bestpackednum);
fprintf(ofp," TOTAL VOLUME OF ALL BOXES                : %.0f\n",
                                                        totalboxvol);
fprintf(ofp," PALLET VOLUME                            : %.0f\n",
                                                         totalvolume);
fprintf(ofp," BEST SOLUTION'S VOLUME UTILIZATION                    :
                          %.0f OUT OF %.0f\n", bestvolume, totalvolume);
fprintf(ofp," PERCENTAGE OF PALLET VOLUME USED                      :
                                    %.6f %%\n", percentageused);
fprintf(ofp," PERCENTAGE OF PACKED BOXES (VOLUME)   : %.6f %%\n",
                                    percentagepackedbox);
fprintf(ofp," WHILE PALLET ORIENTATION                              :
                                  X= %d;  Y= %d;  Z= %d\n", px, py, pz);
fprintf(ofp,"-------------------------------------------------------------------------------
                                                          ----------\n");
fprintf(ofp,"  NO: PACKSTA DIMEN-1  DIMEN-2  DIMEN-3   COOR-X
              COOR-Y   COOR-Z   PACKEDX  PACKEDY  PACKEDZ\n");
fprintf(ofp,"-------------------------------------------------------------------------------
                                                          ----------\n");

listcanditlayers();
layers[0].layereval=-1;
qsort(layers,layerlistlen+1,sizeof(struct layerlist),complayerlist);
packedvolume=0.0;
packedy=0;
packing=1;
```

```
layerthickness=layers[bestite].layerdim;
remainpy=py; remainpz=pz;
for (x=1; x<=tbn; x++) boxlist[x].packst=0;
do {
        layerinlayer=0;
        layerdone=0;
        packlayer();
        packedy=packedy+layerthickness;
        remainpy=py-packedy;
        if (layerinlayer){
                prepackedy=packedy;
                preremainpy=remainpy;
                remainpy=layerthickness-prelayer;
                packedy=packedy-layerthickness+prelayer;
                remainpz=lilz;
                layerthickness=layerinlayer;
                layerdone=0;
                packlayer();
                packedy=prepackedy;
                remainpy=preremainpy;
                remainpz=pz;
        }
        if (!quit) findlayer(remainpy);
}
while (packing & !quit);
fprintf(ofp,"\n\n                *** LIST OF UNPACKED BOXES ***\n");
unpacked=1;
for (cboxi=1; cboxi<=tbn; cboxi++) if (!boxlist[cboxi].packst)
                                                graphunpackedout();
unpacked=0;
fclose(ofp);
fclose(gfp);
printf("\n");
for (n=1; n<=tbn; n++)
        if (boxlist[n].packst) printf("%d %d %d %d %d %d %d %d %d %d\n",n,
                boxlist[n].dim1,boxlist[n].dim2,boxlist[n].dim3,boxlist[n].cox,
                boxlist[n].coy,boxlist[n].coz,boxlist[n].packx,boxlist[n].
                                                packy,boxlist[n].packz);
printf(" ELAPSED TIME                           : Almost %.0f sec\n",
                                                        elapsedtime);
printf(" TOTAL NUMBER OF ITERATIONS DONE        : %d\n", itenum);
printf(" BEST SOLUTION FOUND AT                 :
                ITERATION: %d OF VARIANT: %d\n", bestite, bestvariant);
printf(" TOTAL NUMBER OF BOXES                  : %d\n", tbn);
printf(" PACKED NUMBER OF BOXES                 : %d\n",
                                                bestpackednum);
```

```
        printf(" TOTAL VOLUME OF ALL BOXES                    : %.0f\n",
                                                                totalboxvol);
        printf(" PALLET VOLUME                              : %.0f\n", totalvolume);
        printf(" BEST SOLUTION'S VOLUME UTILIZATION    : %.0f OUT OF
                                                %.0f\n", bestvolume, totalvolume);
        printf(" PERCENTAGE OF PALLET VOLUME USED    : %.6f %%\n",
                                                                percentageused);
        printf(" PERCENTAGE OF PACKED BOXES (VOLUME)        : %.6f %%\n",
                                                        percentagepackedbox);
        printf(" WHILE PALLET ORIENTATION          :
                                        X= %d;   Y= %d;   Z= %d\n\n\n", px, py, pz);
        printf(" TO VISUALIZE THIS SOLUTION, PLEASE RUN  'VISUAL.EXE'\n");
}
```

Appendix C – The C Program Code of the Visualizer

```
//*************************
// INCLUDED HEADER FILES
//*************************

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>


/*******************************/
/*      Function prototypes            */
/*******************************/
void Initialize(void);
void Pack(void);
void PutBox(void);
void SayGoodbye(void);
void Pause(void);
void MainWindow(char *header);
void StatusLine(char *msg);
void DrawBorder(void);
void changetextstyle(int font, int direction, int charsize);
int gprintf(int *xloc, int *yloc, char *fmt, ... );


//*********************************************************
// VARIABLE, CONSTANT AND STRUCTURE DECLARATIONS
//*********************************************************

int   GraphDriver;          /* The Graphics device driver       */
int   GraphMode;            /* The Graphics mode value          */
double AspectRatio;         /* Aspect ratio of a pixel on the screen*/
int   MaxX, MaxY;           /* The maximum resolution of the screen */
int   MaxColors;            /* The maximum # of colors available    */
int   ErrorCode;            /* Reports any graphics errors          */
struct palettetype palette;             /* Used to read palette info  */
struct dataarray {int cx, cy, cz, pax, pay, paz; } data[2000];
int px, py, pz, cox, coy, coz, packx, packy, packz, a, b, index, currenty, q;
double max, sc;
char strpx[8], strpy[8], strpz[8], oldstrcoy[8], strcox[8], strcoy[8], strcoz[8], strpackx[8],
      strpacky[8], strpackz[8];
FILE *igf;
```

```
/*********************************/
/*       BEGIN MAIN FUNCTION     */
/*********************************/

int main()
{
 Initialize();          /* Set system into Graphics mode     */
 Pack();
 SayGoodbye();              /* Give user the closing screen        */
 closegraph();              /* Return the system to text mode      */
 return(0);
}

/*************************************************************/
/*       INITIALIZE: INITIALIZES THE GRAPHICS SYSTEM AND     */
/*                REPORTS ANY ERRORS WHICH OCCURED.          */
/*************************************************************/

void Initialize(void)
{
 int xasp, yasp;                    /* Used to read the aspect ratio*/
 GraphDriver = DETECT;              /* Request auto-detection       */
 initgraph( &GraphDriver, &GraphMode, "" );
 ErrorCode = graphresult();         /* Read result of initialization*/
 if( ErrorCode != grOk ){           /* Error occured during init    */
  printf(" Graphics System Error: %s\n", grapherrormsg( ErrorCode ) );
  exit( 1 );
 }

 getpalette( &palette );            /* Read the palette from board        */
 MaxColors = getmaxcolor() + 1;     /* Read maximum number of colors*/
 MaxX = getmaxx();
 MaxY = getmaxy();                  /* Read size of screen          */
 getaspectratio( &xasp, &yasp );    /* read the hardware aspect     */
 AspectRatio = (double)xasp / (double)yasp; /* Get correction factor        */
}
```

```
/*********************************************************************/
/*      PACK: READS THE DATA FROM "VISUDAT" FILE AND ARRANGES */
/*            BOXES TO PACK FROM FAR END TO THE CLOSE          */
/*********************************************************************/

void Pack(void)
{
 struct viewporttype vp;
 char buffer[10];

 MainWindow( "PACKING OF THE BEST SOLUTION FOUND" );
 if ((igf=fopen("visudat","r"))==NULL){
        outtextxy(0,0,"CANNOT OPEN FILE visudat");
        exit(1);
 }
 fscanf(igf,"%s %s %s",strpx, strpy, strpz);
 px=atoi(strpx); py=atoi(strpy); pz=atoi(strpz);
 max=px;
 if (py>max) max=py;
 if (pz>max) max=pz;
 sc=120/max;
 getviewsettings( &vp );
 settextjustify( CENTER_TEXT, TOP_TEXT );
 changetextstyle( TRIPLEX_FONT, HORIZ_DIR, 4 );
 changetextstyle( DEFAULT_FONT, HORIZ_DIR, 1 );
 setviewport( vp.left+50, vp.top+40, vp.right-50, vp.bottom-10, 1 );
 getviewsettings( &vp );
 settextjustify( CENTER_TEXT, CENTER_TEXT );
 outtextxy( 20, 0, "PRESS 'Q' TO QUIT" );
 setcolor(3);
 outtextxy( 220, 0, "PALLET ORIENTATION (X Y Z) : " );
 outtextxy( 350, 0, strpx);
 outtextxy( 390, 0, strpy);
 outtextxy( 430, 0, strpz);
 setfillstyle( EMPTY_FILL, 1);
 bar3d( 10, 350-sc*2*py, 10+sc*2*px, 350, sc*pz, 1 );
 outtextxy(460, 30, "COORDINATES:");
 outtextxy(460, 40, "CX:   CY:   CZ:");
 outtextxy(460, 70, "DIMENSIONS:");
 outtextxy(460, 80, "DX:   DY:   DZ:");
 index=1;
 fscanf (igf,"%s %s %s %s %s %s", strcox, strcoy, strcoz, strpackx, strpacky, strpackz);
 cox=atoi(strcox); coy=atoi(strcoy); coz=atoi(strcoz);
 packx=atoi(strpackx); packy=atoi(strpacky); packz=atoi(strpackz);
 data[index].cx=cox; data[index].cy=coy; data[index].cz=coz;
 data[index].pax=packx; data[index].pay=packy; data[index].paz=packz;
```

```
 index++; currenty=data[index].cy;
 while (fscanf (igf,"%s %s %s %s %s %s", strcox, strcoy, strcoz, strpackx, strpacky,
                                                                    strpackz)!=EOF){
        cox=atoi(strcox); coy=atoi(strcoy); coz=atoi(strcoz);
        packx=atoi(strpackx); packy=atoi(strpacky); packz=atoi(strpackz);
        data[index].cx=cox; data[index].cy=coy; data[index].cz=coz;
        data[index].pax=packx; data[index].pay=packy; data[index].paz=packz;

        if (data[index].cy!=currenty){
                b=index; index--;
                PutBox();
        }
        index++;
        if (q==1) return;
 }
 index--;
 PutBox();
 fclose(igf);
 Pause();                              /* Pause for user's response    */
}
/********** PUTBOX: DRAW BOXES IN THEIR LOCATIONS **********/
void PutBox(void)
{
  for (a=index; a>0; a--){
        setcolor(0);
        outtextxy(528, 50, "                                                    ");
        outtextxy(528, 90, "                                                    ");
        setcolor(random(15)+1);
        itoa(data[a].pax, strpackx, 10); outtextxy(410, 90, strpackx);
        itoa(data[a].pay, strpacky, 10); outtextxy(460, 90, strpacky);
        itoa(data[a].paz, strpackz, 10); outtextxy(508, 90, strpackz);
        itoa(data[a].cx, strcox, 10); outtextxy(410, 50, strcox);
        itoa(data[a].cy, strcoy, 10); outtextxy(460, 50, strcoy);
        itoa(data[a].cz, strcoz, 10); outtextxy(508, 50, strcoz);
        bar3d( 10+sc*2*data[a].cx+sc*data[a].cz,
                350-sc*2*data[a].cy-sc*.74*data[a].cz-sc*2*data[a].pay,
                10+sc*2*data[a].cx+sc*data[a].cz+sc*2*data[a].pax,
                350-sc*2*data[a].cy-sc*.74*data[a].cz, sc*data[a].paz, 1 );
        if (toupper(getch())=='Q'){ q=1; break;}
 }
 data[1].cx=data[b].cx; data[1].cy=data[b].cy;
 data[1].cz=data[b].cz; data[1].pax=data[b].pax;
 data[1].pay=data[b].pay; data[1].paz=data[b].paz;
 index=1;
 currenty=data[index].cy;
}
```

```
/************************************************/
/*      SAYGOODBYE: GIVE A CLOSING SCREEN      */
/*              TO THE USER BEFORE LEAVING.     */
/************************************************/

void SayGoodbye(void)
{
 struct viewporttype viewinfo;        /* Structure to read viewport */
 int h, w;

 MainWindow( "== The End ==" );
 getviewsettings( &viewinfo );        /* Read viewport settings     */
 changetextstyle( TRIPLEX_FONT, HORIZ_DIR, 4 );
 settextjustify( CENTER_TEXT, CENTER_TEXT );
 h = viewinfo.bottom - viewinfo.top;
 w = viewinfo.right  - viewinfo.left;
 outtextxy( w/2, h/2, "That's all, folks!" );
 StatusLine( "Press any key to EXIT" );
 getch();
 cleardevice();                       /* Clear the graphics screen  */
}


/*****************************************************************/
/*      PAUSE: PAUSE UNTIL THE USER ENTERS A KEYSTROKE.          */
/*****************************************************************/

void Pause(void)
{
 static char msg[] = "Esc aborts or press a key...";
 int c;

 StatusLine( msg );                   /* Put msg at bottom of screen      */
 c = getch();                         /* Read a character from kbd */
 if( 0 == c ){                        /* Did use hit a non-ASCII key? */
       c = getch();                   /* Read scan code for keyboard      */
 }
 cleardevice();                       /* Clear the screen                 */
}
```

```
/*****************************************************/
/*       MAINWINDOW: ESTABLISH THE MAIN WINDOW       */
/*****************************************************/

void MainWindow( char *header )
{
 int height;
 cleardevice();                 /* Clear graphics screen        */
 setcolor( MaxColors - 1 );             /* Set current color to white   */
 setviewport( 0, 0, MaxX, MaxY, 1 );        /* Open port to full screen     */
 height = textheight( "H" );        /* Get basic text height        */
 changetextstyle( DEFAULT_FONT, HORIZ_DIR, 1 );
 settextjustify( CENTER_TEXT, TOP_TEXT );
 outtextxy( MaxX/2, 2, header );
 setviewport( 0, height+4, MaxX, MaxY-(height+4), 1 );
 DrawBorder();
 setviewport( 1, height+5, MaxX-1, MaxY-(height+5), 1 );
}


/*********************************************/
/*       STATUSLINE: DISPLAY A STATUS LINE   */
/*          AT THE BOTTOM OF THE SCREEN.     */
/*********************************************/

void StatusLine( char *msg )
{
 int height;
 setviewport( 0, 0, MaxX, MaxY, 1 );        /* Open port to full screen     */
 setcolor( MaxColors - 1 );             /* Set current color to white   */
 changetextstyle( DEFAULT_FONT, HORIZ_DIR, 1 );
 settextjustify( CENTER_TEXT, TOP_TEXT );
 setlinestyle( SOLID_LINE, 0, NORM_WIDTH );
 setfillstyle( EMPTY_FILL, 0 );
 height = textheight( "H" );        /* Detemine current height      */
 bar( 0, MaxY-(height+4), MaxX, MaxY );
 rectangle( 0, MaxY-(height+4), MaxX, MaxY );
 outtextxy( MaxX/2, MaxY-(height+2), msg );
 setviewport( 1, height+5, MaxX-1, MaxY-(height+5), 1 );
}
```

```
/*************************************************/
/*      DRAWBORDER: DRAW A SOLID SINGLE LINE     */
/*                AROUND THE CURRENT VIEWPORT.   */
/*************************************************/

void DrawBorder(void)
{
 struct viewporttype vp;
 setcolor( MaxColors - 1 );          /* Set current color to white   */
 setlinestyle( SOLID_LINE, 0, NORM_WIDTH );
 getviewsettings( &vp );
 rectangle( 0, 0, vp.right-vp.left, vp.bottom-vp.top );
}


/*********************************************************************/
/*   CHANGETEXTSTYLE: SIMILAR TO SETTEXTSTYLE, BUT CHECKS FOR */
/*      ERRORS THAT MIGHT OCCUR WHIL LOADING THE FONT FILE.        */
/*********************************************************************/

void changetextstyle(int font, int direction, int charsize)
{
 int ErrorCode;
 graphresult();              /* clear error code              */
 settextstyle(font, direction, charsize);
 ErrorCode = graphresult();          /* check result                  */
 if( ErrorCode != grOk ){            /* if error occured              */
  closegraph();
  printf(" Graphics System Error: %s\n", grapherrormsg( ErrorCode ) );
  exit( 1 );
 }
}
```

```
/*************************************************************************/
/*   GPRINTF: USED LIKE PRINTF EXCEPT THE OUTPUT IS SENT TO THE  */
/*       SCREEN IN GRAPHICS MODE AT THE SPECIFIED CO-ORDINATE.   */
/*************************************************************************/

int gprintf( int *xloc, int *yloc, char *fmt, ... )
{
 va_list  argptr;                    /* Argument list pointer        */
 char str[140];              /* Buffer to build sting into   */
 int cnt;                            /* Result of SPRINTF for return */
 va_start( argptr, format );         /* Initialize va_ functions     */
 cnt = vsprintf( str, fmt, argptr ); /* prints string to buffer      */
 outtextxy( *xloc, *yloc, str );     /* Send string in graphics mode */
 *yloc += textheight( "H" ) + 2;     /* Advance to next line         */
 va_end( argptr );                   /* Close va_ functions          */
 return( cnt );              /* Return the conversion count  */
}
```

Appendix D – The Test Problems That We Generated

Randomly Generated Sets:

SET #1	SET #2	SET #3
104, 96, 84	104, 96, 84	104, 96, 84
1. 3, 5, 7, 51	1. 3, 5, 7, 200	1. 3, 5, 7, 200
2. 20, 4, 6, 90	2. 9, 11, 2, 290	2. 9, 11, 2, 29
3. 11, 21, 16, 80	3. 14, 6, 8, 300	3. 14, 6, 8, 30
4. 51, 2, 60, 80	4. 1, 4, 19, 748	4. 1, 4, 19, 51
5. 6, 17, 8, 6	5. 10, 13, 21, 190	5. 10, 13, 21, 12
		6. 27, 23, 34, 5
		7. 12, 9, 13, 10
		8. 24, 15, 19, 50
		9. 5, 16, 9, 100
		10. 10, 20, 5, 100
		11. 9, 18, 15, 50

SET #4	SET #5 (Worst Case)
104, 96, 84	104 96 84
1. 1, 2, 3, 200	1. 1, 2, 3, 1
2. 2, 4, 5, 200	2. 4, 5, 6, 1
3. 6, 7, 1, 200	3. 7, 8, 9, 1
4. 6, 8, 2, 29	4. 10, 11, 12, 1
5. 11, 2, 3, 29	5. 13, 14, 15, 1
6. 9, 4, 2, 29	6. 16, 17, 18, 1
7. 14, 5, 3, 30	7. 19, 20, 21, 1
8. 10, 4, 6, 30	8. 22, 23, 24, 1
9. 11, 8, 3, 30	9. 25, 26, 27, 1
10. 1, 2, 19, 50	10. 28, 29, 30, 1
11. 8, 13, 11, 50	11. 31, 32, 33, 1
12. 1, 3, 21, 10	12. 34, 35, 36, 1
13. 8, 9, 10, 30	13. 37, 38, 39, 1
14. 7, 13, 31, 115	14. 40, 41, 42, 1
15. 12, 66, 3, 30	15. 43, 44, 45, 1
16. 4, 15, 19, 90	16. 46, 47, 48, 1
17. 5, 16, 9, 100	17. 2, 3, 4, 1
18. 10, 2, 5, 100	18. 5, 6, 7, 1
19. 10, 10, 1, 90	19. 8, 9, 10, 1
20. 9, 18, 15, 50	20. 11, 12, 13, 1
21. 6, 9, 14, 1	21. 14, 15, 16, 1
	22. 17, 18, 19, 1
	23. 20, 21, 22, 1
	24. 23, 24, 25, 1
	25. 26 27 28 1
	26. 29 30 31 1
	27. 32 33 34 1
	28. 35 36 37 1
	29. 38 39 40 1
	30. 41 42 43 1
	31. 44 45 46 1

Sets Generated by Dividing Into:

Distributor's Pallet Packing Problem Samples:

SET #6	SET #7	SET #8
104, 96, 84 1. 70, 104, 24, 4 2. 14, 104, 48, 2	104, 96, 84 1. 70, 50, 24, 4 2. 70, 54, 24, 4 3. 14, 104, 48, 2	104, 96, 84 1. 70, 45, 24, 4 2. 70, 59, 24, 4 3. 14, 40, 48, 2 4. 14, 64, 48, 2

SET #9	SET #10	SET #11
104, 96, 84 1. 70, 45, 24, 4 2. 70, 30, 24, 4 3. 70, 29, 24, 4 4. 14, 40, 48, 2 5. 14, 32, 48, 2 6. 14, 32, 48, 2	104, 96, 84 1. 28, 32, 18, 9 2. 24, 21, 35, 16 3. 19, 26, 20, 4 4. 19, 26, 16, 16 5. 16, 26, 20, 4 6. 20, 20, 26, 1 7. 16, 14, 25, 36	104, 96, 84 1. 19, 20, 42, 2 2. 25, 20, 30, 1 3. 25, 20, 25, 1 4. 25, 20, 29, 1 5. 8, 20, 21, 4 6. 36, 46, 84, 1 7. 16, 46, 10, 2 8. 16, 46, 32, 2 9. 20, 30, 15, 1 10. 20, 30, 69, 1 11. 20, 30, 21, 4 12. 12, 30, 7, 12 13. 52, 60, 42, 2 14. 26, 36, 21, 4 15. 26, 36, 84, 1

Manufacturer's Pallet Packing Problem Samples:

SET #12	SET #13	SET #14
104 96 84 1. 14, 13, 8, 576	104 96 84 1. 14, 13, 4, 1152	104 96 84 1. 4, 6, 7, 4992

SET #15	SET #16
104 96 84 1. 14, 13, 2, 576 2. 21, 13, 4, 576	104 96 84 1. 4, 6, 7, 2496 2. 14, 13, 8, 288

Appendix E – Solutions of B/R Test Sets

BOX SET BR #1: 3 DIFFERENT BOX TYPES

SET #	TOTAL NUMBER BOXES	PACKED NUMBER OF BOXES	% PALLET VOLUME UTILIZATION	% PACKED VOLUME OF ALL BOXES	SOLUTION TIME (SECOND)
1	112	104	88.47	89.52	1
2	138	131	87.21	88.21	1
3	127	117	89.27	89.64	0
4	197	186	91.13	91.29	2
5	136	127	90.87	91.34	1
6	147	116	87.01	87.16	1
7	126	84	86.46	86.63	1
8	180	170	92.95	93.18	1
9	101	92	87.32	88.03	0
10	130	121	92.06	92.45	1
11	102	93	89.58	90.66	0
12	104	91	91.37	91.64	1
13	284	276	94.01	94.15	4
14	132	118	90.54	91.77	1
15	119	98	83.62	83.75	0
16	159	143	85.73	85.90	1
17	213	198	91.82	92.35	2
18	82	70	88.93	90.32	0
19	130	102	88.83	88.91	0
20	88	78	87.43	87.62	0
21	79	73	90.44	90.67	0
22	116	101	88.52	89.05	0
23	128	114	95.29	95.60	0
24	114	102	88.14	88.60	1
25	139	128	91.19	91.23	0
26	124	110	88.84	89.24	0
27	103	79	83.73	84.36	1
28	111	100	82.52	83.15	1
29	109	100	88.34	88.59	0
30	405	354	91.51	91.67	10
31	129	116	90.23	90.35	1
32	155	148	95.21	95.35	1
33	282	261	92.16	92.62	3
34	160	146	89.32	90.19	2
35	150	137	90.15	90.42	0
36	172	163	87.71	88.62	1
37	99	87	89.34	89.76	1
38	93	81	87.58	87.77	0

SET #	TOTAL NUMBER BOXES	PACKED NUMBER OF BOXES	% PALLET VOLUME UTILIZATION	% PACKED VOLUME OF ALL BOXES	SOLUTION TIME (SECOND)
39	243	229	91.87	92.16	2
40	107	98	89.93	90.33	0
41	109	91	87.69	87.81	0
42	168	152	90.43	90.47	1
43	141	135	90.94	91.60	1
44	142	121	83.97	84.35	0
45	140	133	90.48	90.71	0
46	116	89	89.16	90.21	1
47	137	126	91.86	92.63	0
48	153	141	90.76	90.86	1
49	86	78	83.96	84.44	1
50	168	161	90.04	90.54	1
51	129	113	91.05	91.25	0
52	187	174	90.06	90.08	1
53	170	152	90.57	90.67	1
54	142	125	84.96	84.96	0
55	114	104	87.46	88.07	0
56	408	389	93.51	93.76	8
57	124	114	91.31	91.76	0
58	92	87	92.85	92.91	0
59	195	179	90.31	90.33	2
60	105	90	90.76	90.90	0
61	101	93	86.52	87.89	0
62	120	106	86.54	86.87	1
63	94	79	78.90	79.08	0
64	139	111	88.43	88.45	1
65	476	437	94.39	94.39	12
66	145	132	89.37	89.82	0
67	221	194	86.98	87.31	3
68	238	216	89.66	89.73	3
69	144	125	82.80	83.57	1
70	163	148	87.26	87.77	1
71	96	84	86.17	86.46	0
72	74	63	86.15	86.51	0
73	127	115	90.12	90.49	0
74	132	110	86.06	86.66	0
75	137	128	92.24	92.46	1
76	269	248	91.49	91.80	4
77	194	178	93.36	93.55	2
78	169	153	87.86	87.92	1
79	200	179	84.90	85.07	1
80	133	121	90.05	90.09	0
81	73	65	88.15	88.92	0

SET #	TOTAL NUMBER BOXES	PACKED NUMBER OF BOXES	% PALLET VOLUME UTILIZATION	% PACKED VOLUME OF ALL BOXES	SOLUTION TIME (SECOND)
82	164	147	90.33	90.80	2
83	120	103	86.19	86.82	0
84	69	60	88.03	88.10	0
85	319	307	94.99	95.09	5
86	156	126	87.28	87.99	0
87	98	91	85.51	85.56	1
88	126	110	85.06	85.12	0
89	96	87	87.98	88.50	1
90	90	83	89.84	90.45	0
91	238	214	90.68	90.94	1
92	88	73	88.63	88.75	0
93	100	90	86.60	86.70	0
94	133	123	86.45	87.32	1
95	139	124	89.89	90.81	0
96	182	165	88.87	89.01	1
97	140	123	87.46	87.80	1
98	122	109	89.98	89.99	1
99	154	144	86.32	86.34	0
100	214	200	91.22	91.61	2

BOX SET BR #2: 5 DIFFERENT BOX TYPES

SET #	TOTAL NUMBER BOXES	PACKED NUMBER OF BOXES	% PALLET VOLUME UTILIZATION	% PACKED VOLUME OF ALL BOXES	SOLUTION TIME (SECOND)
1	81	73	90.70	91.78	0
2	114	106	88.66	88.73	0
3	166	149	87.74	88.42	2
4	201	181	90.71	91.67	3
5	117	100	88.22	88.47	0
6	142	127	87.01	87.10	1
7	166	148	86.88	88.27	2
8	122	109	89.88	89.99	1
9	118	96	87.78	87.80	1
10	174	166	87.51	88.37	2
11	94	82	89.16	89.45	1
12	86	69	87.05	87.27	0
13	228	191	91.89	92.20	4
14	95	82	92.25	92.64	0
15	127	110	87.04	87.26	2
16	163	136	86.49	87.32	2
17	112	106	84.89	85.15	1
18	98	81	89.91	90.27	0

SET #	TOTAL NUMBER BOXES	PACKED NUMBER OF BOXES	% PALLET VOLUME UTILIZATION	% PACKED VOLUME OF ALL BOXES	SOLUTION TIME (SECOND)
19	143	106	87.56	87.99	2
20	120	93	89.52	89.58	1
21	84	73	88.45	88.79	1
22	100	88	86.50	88.35	1
23	120	108	87.38	88.63	1
24	93	76	85.08	86.42	0
25	152	136	90.23	90.53	2
26	169	156	90.79	91.54	1
27	114	89	85.75	86.07	1
28	122	114	87.51	87.80	1
29	156	145	85.95	86.08	2
30	196	191	92.23	92.70	1
31	142	126	88.42	88.47	1
32	142	125	89.49	89.72	2
33	116	110	90.31	92.01	1
34	156	139	89.23	89.47	2
35	147	139	93.98	94.47	2
36	104	95	88.88	88.92	1
37	105	88	89.16	90.24	1
38	116	107	87.70	88.14	1
39	266	246	93.82	93.91	5
40	96	78	87.80	88.81	0
41	98	94	91.71	93.54	1
42	158	133	89.75	89.77	1
43	102	91	89.70	90.22	0
44	129	114	87.22	87.81	1
45	161	139	90.19	90.99	2
46	101	89	91.78	92.39	0
47	160	149	90.41	90.56	1
48	139	117	88.35	88.85	1
49	94	85	88.87	89.88	1
50	155	146	89.70	89.90	2
51	152	139	91.39	92.08	1
52	101	83	88.92	89.00	0
53	141	129	89.67	89.69	1
54	141	120	89.39	89.49	1
55	117	101	85.17	85.72	1
56	181	172	92.70	92.72	2
57	159	136	89.44	89.92	1
58	114	110	91.51	92.38	1
59	201	184	91.85	92.25	3
60	113	99	88.86	89.23	1
61	83	65	84.98	85.64	0

SET #	TOTAL NUMBER BOXES	PACKED NUMBER OF BOXES	% PALLET VOLUME UTILIZATION	% PACKED VOLUME OF ALL BOXES	SOLUTION TIME (SECOND)
62	109	92	88.49	88.58	1
63	120	110	90.01	91.26	1
64	122	112	85.88	86.54	1
65	187	174	88.35	89.49	2
66	133	115	91.15	92.40	1
67	184	160	87.95	88.30	2
68	130	117	87.16	88.15	1
69	146	135	89.75	89.82	1
70	123	108	87.45	87.57	1
71	122	111	87.24	87.79	1
72	88	67	88.03	88.05	0
73	132	117	88.12	88.36	2
74	124	108	86.34	86.41	1
75	163	149	88.78	89.17	1
76	188	173	91.58	91.97	2
77	202	183	92.24	92.62	3
78	191	170	88.25	88.41	3
79	206	187	91.52	91.70	3
80	116	98	87.99	88.32	1
81	86	75	88.45	89.36	1
82	149	132	89.31	89.93	1
83	136	122	88.77	89.41	1
84	85	73	87.22	87.42	0
85	209	201	91.78	92.06	2
86	160	131	90.70	90.95	1
87	102	88	86.81	87.43	1
88	140	119	87.32	87.73	1
89	100	85	84.84	84.95	0
90	82	68	87.33	87.85	0
91	166	158	89.23	89.38	1
92	81	66	87.20	87.66	0
93	85	73	89.41	89.42	1
94	168	138	91.45	91.51	1
95	149	136	90.27	90.68	1
96	202	190	89.98	90.05	2
97	165	147	87.87	88.04	2
98	138	119	92.43	92.89	2
99	174	159	89.49	89.62	2
100	139	131	89.15	89.68	1

BOX SET BR #3: 8 DIFFERENT BOX TYPES

SET #	TOTAL NUMBER BOXES	PACKED NUMBER OF BOXES	% PALLET VOLUME UTILIZATION	% PACKED VOLUME OF ALL BOXES	SOLUTION TIME (SECOND)
1	94	86	88.59	88.88	1
2	115	105	88.00	88.19	1
3	143	117	86.78	86.80	3
4	185	159	86.67	87.02	4
5	113	93	88.61	88.76	1
6	143	125	87.48	87.86	2
7	144	128	86.56	86.88	2
8	104	85	88.68	89.96	1
9	133	118	87.65	87.74	2
10	180	161	87.06	87.97	5
11	119	103	90.61	91.14	2
12	103	90	85.21	85.54	1
13	199	187	88.84	89.25	4
14	104	94	90.78	90.90	1
15	103	85	87.56	87.82	1
16	128	117	89.03	89.70	1
17	86	80	84.51	85.09	0
18	85	78	88.03	88.83	0
19	176	157	89.38	89.67	2
20	135	107	90.69	91.04	2
21	94	80	88.66	88.87	1
22	111	101	88.76	88.88	1
23	130	121	87.39	88.31	1
24	112	96	88.69	89.10	1
25	135	111	88.59	89.16	2
26	161	146	88.81	89.10	2
27	120	105	87.34	87.36	1
28	143	122	86.01	86.34	2
29	194	173	88.18	88.51	3
30	161	144	88.67	88.73	1
31	136	119	87.63	88.15	3
32	141	127	88.44	89.18	2
33	131	121	90.08	90.34	2
34	164	149	89.20	89.91	4
35	149	136	90.22	90.55	2
36	111	101	86.99	87.43	1
37	112	100	89.79	90.38	1
38	103	94	86.67	87.30	1
39	232	217	89.72	90.30	5
40	86	73	86.56	87.31	0
41	124	107	90.25	91.27	1

SET #	TOTAL NUMBER BOXES	PACKED NUMBER OF BOXES	% PALLET VOLUME UTILIZATION	% PACKED VOLUME OF ALL BOXES	SOLUTION TIME (SECOND)
42	131	117	89.80	90.77	2
43	92	82	86.83	88.02	0
44	142	117	85.65	85.98	3
45	147	129	89.50	90.35	2
46	110	97	89.25	89.49	2
47	148	133	86.55	86.96	2
48	103	95	87.96	88.95	1
49	105	96	88.90	89.39	1
50	136	115	88.98	89.09	2
51	165	143	91.06	91.44	3
52	143	128	89.68	90.18	1
53	100	85	87.80	88.21	1
54	178	169	90.70	91.14	3
55	154	143	86.68	86.79	3
56	212	196	89.38	90.29	4
57	118	107	87.80	87.87	1
58	87	80	88.21	88.66	0
59	216	194	87.33	87.43	5
60	114	92	87.23	87.66	1
61	90	76	88.61	89.15	1
62	114	103	85.16	85.34	1
63	124	113	89.15	90.31	2
64	134	123	87.09	87.11	2
65	140	130	89.48	90.37	2
66	97	89	87.66	88.19	1
67	169	148	87.96	88.09	3
68	119	103	89.59	90.86	1
69	162	144	90.42	90.92	3
70	108	100	87.71	87.91	2
71	131	117	86.76	86.92	2
72	138	125	89.54	90.17	2
73	105	94	85.35	85.71	1
74	114	104	88.71	89.15	1
75	139	130	87.46	88.40	2
76	144	127	88.92	89.97	1
77	201	182	91.70	91.74	4
78	155	140	88.81	89.00	3
79	195	179	90.86	90.94	3
80	114	97	88.63	89.30	2
81	80	73	88.92	89.38	1
82	174	160	87.92	88.45	3
83	140	122	90.25	90.41	2
84	101	95	92.14	92.48	1

SET #	TOTAL NUMBER BOXES	PACKED NUMBER OF BOXES	% PALLET VOLUME UTILIZATION	% PACKED VOLUME OF ALL BOXES	SOLUTION TIME (SECOND)
85	146	140	89.55	89.67	1
86	155	125	89.57	90.07	2
87	126	113	88.32	88.88	1
88	167	143	88.11	88.44	3
89	90	80	86.87	87.17	1
90	84	72	86.27	86.38	1
91	183	166	91.01	91.63	4
92	97	82	89.84	91.20	1
93	84	73	85.62	86.35	0
94	154	145	88.21	88.47	2
95	178	160	89.46	89.60	3
96	197	177	91.27	91.35	4
97	166	154	87.66	87.86	2
98	125	104	90.77	90.78	1
99	110	98	86.82	88.09	1
100	137	128	88.74	88.82	2

BOX SET BR #4: 10 DIFFERENT BOX TYPES

SET #	TOTAL NUMBER BOXES	PACKED NUMBER OF BOXES	% PALLET VOLUME UTILIZATION	% PACKED VOLUME OF ALL BOXES	SOLUTION TIME (SECOND)
1	106	94	88.54	90.72	1
2	123	109	87.32	88.26	1
3	135	118	87.42	87.58	3
4	169	145	88.52	88.71	3
5	130	118	89.26	89.74	1
6	132	112	87.04	87.46	3
7	138	116	88.42	88.77	2
8	107	94	89.36	89.50	1
9	149	133	87.93	88.65	2
10	133	114	86.85	87.87	3
11	126	114	90.14	90.77	2
12	114	101	88.13	88.34	1
13	169	150	87.18	87.47	4
14	105	98	89.80	90.40	1
15	100	86	89.64	90.04	1
16	138	116	88.02	88.29	3
17	91	83	88.38	89.67	1
18	75	63	87.18	88.90	1
19	138	128	87.15	88.11	2
20	139	121	89.58	89.68	2
21	114	98	86.72	87.14	1

SET #	TOTAL NUMBER BOXES	PACKED NUMBER OF BOXES	% PALLET VOLUME UTILIZATION	% PACKED VOLUME OF ALL BOXES	SOLUTION TIME (SECOND)
22	111	104	89.10	89.26	2
23	98	72	87.22	89.55	1
24	137	116	85.11	85.15	3
25	133	122	88.07	88.45	1
26	158	143	89.00	89.09	2
27	119	111	87.45	88.51	1
28	175	161	86.72	86.83	3
29	127	113	86.62	86.85	1
30	151	132	88.26	88.40	3
31	145	129	88.62	88.70	3
32	156	139	87.20	87.72	3
33	132	105	89.26	89.46	1
34	136	124	89.80	90.16	3
35	151	131	89.26	89.72	3
36	131	113	89.60	89.73	2
37	136	122	88.94	90.01	2
38	115	98	86.25	86.56	2
39	225	208	90.11	90.17	6
40	102	82	84.38	84.58	1
41	113	89	89.49	89.75	1
42	121	100	87.62	87.78	2
43	90	68	86.23	86.52	1
44	138	114	85.87	86.05	4
45	158	141	88.62	88.63	3
46	98	87	89.88	90.00	1
47	138	122	86.40	86.81	2
48	109	96	88.32	89.19	1
49	106	91	89.00	90.43	1
50	167	152	88.95	89.55	4
51	184	166	91.87	92.23	5
52	143	124	89.23	89.39	2
53	101	78	87.20	87.71	1
54	172	155	90.02	90.41	4
55	143	125	85.69	85.92	3
56	233	213	91.04	92.02	6
57	96	85	87.53	88.23	1
58	106	95	90.05	90.37	1
59	199	168	88.85	88.87	6
60	125	113	87.15	87.36	2
61	121	105	89.14	89.76	1
62	125	118	88.22	88.67	2
63	100	82	90.46	90.56	1
64	118	106	86.36	86.97	1

SET #	TOTAL NUMBER BOXES	PACKED NUMBER OF BOXES	% PALLET VOLUME UTILIZATION	% PACKED VOLUME OF ALL BOXES	SOLUTION TIME (SECOND)
65	156	148	90.52	90.59	3
66	105	92	87.00	87.08	1
67	170	154	87.84	88.42	4
68	103	87	85.00	85.49	2
69	157	146	89.10	89.24	3
70	117	104	86.29	86.65	2
71	138	122	86.05	86.17	3
72	129	114	86.47	86.59	2
73	117	103	86.26	86.69	1
74	91	72	86.70	87.72	1
75	148	125	88.50	88.62	2
76	118	108	89.96	90.72	1
77	179	150	88.58	89.55	4
78	164	149	90.49	90.84	3
79	217	198	90.72	91.05	5
80	132	121	87.43	87.75	2
81	104	96	88.51	88.68	1
82	153	143	88.71	89.89	3
83	128	113	88.84	88.89	2
84	125	104	90.68	91.66	2
85	123	115	86.03	86.34	1
86	152	138	87.40	87.70	3
87	108	94	85.45	87.08	2
88	178	154	87.13	87.38	4
89	93	82	86.33	87.11	1
90	95	88	87.15	88.44	1
91	134	122	90.69	90.82	2
92	93	77	89.02	90.19	1
93	78	73	88.35	89.45	1
94	141	125	87.68	87.82	3
95	181	167	87.24	87.90	4
96	149	131	88.01	88.82	2
97	142	132	87.22	87.78	2
98	116	103	89.69	90.27	2
99	134	123	88.99	89.52	2
100	144	132	90.06	90.88	3

BOX SET BR #5: 12 DIFFERENT BOX TYPES

SET #	TOTAL NUMBER BOXES	PACKED NUMBER OF BOXES	% PALLET VOLUME UTILIZATION	% PACKED VOLUME OF ALL BOXES	SOLUTION TIME (SECOND)
1	98	84	88.24	89.41	1
2	138	131	86.82	88.05	3
3	133	118	87.59	87.65	4
4	142	130	86.52	86.55	3
5	129	118	88.68	89.39	2
6	140	125	87.51	87.99	3
7	132	117	87.83	88.14	3
8	98	91	88.84	89.62	2
9	135	121	86.74	87.13	2
10	133	113	87.19	87.34	3
11	144	131	89.53	89.57	3
12	109	92	86.16	86.43	2
13	199	179	88.15	88.70	7
14	123	115	87.74	88.76	2
15	104	92	84.67	86.17	2
16	118	106	85.31	86.05	2
17	108	100	88.87	89.37	2
18	87	76	88.20	88.48	1
19	142	132	88.26	89.24	2
20	138	124	88.24	89.21	3
21	128	120	89.36	89.54	3
22	116	94	87.69	87.71	2
23	143	124	87.91	89.25	3
24	111	97	86.37	86.86	2
25	143	123	85.88	86.59	3
26	161	147	89.79	89.82	3
27	120	110	86.39	87.09	1
28	180	161	87.44	87.59	4
29	120	104	87.51	89.61	2
30	135	120	88.29	89.42	3
31	147	127	88.07	88.51	3
32	163	150	87.62	87.75	4
33	141	133	88.52	88.79	2
34	115	99	88.19	88.48	2
35	143	130	88.71	89.12	3
36	124	113	87.72	87.95	2
37	122	109	89.90	91.01	2
38	118	108	84.80	85.72	2
39	196	174	88.58	88.86	7
40	106	93	85.87	86.14	2
41	133	114	87.70	88.07	2

SET #	TOTAL NUMBER BOXES	PACKED NUMBER OF BOXES	% PALLET VOLUME UTILIZATION	% PACKED VOLUME OF ALL BOXES	SOLUTION TIME (SECOND)
42	113	94	87.78	88.33	2
43	113	102	86.79	88.26	2
44	136	118	86.25	86.50	3
45	171	153	88.85	89.53	5
46	99	89	88.48	89.18	2
47	146	125	86.38	86.74	3
48	111	99	87.63	88.08	2
49	104	95	86.22	87.13	1
50	172	152	88.22	88.39	5
51	195	170	88.94	89.12	6
52	139	117	88.77	89.07	3
53	122	110	86.03	86.66	2
54	183	158	89.79	89.87	5
55	136	124	86.31	86.53	3
56	218	206	88.14	88.15	8
57	104	78	86.52	87.14	1
58	115	105	88.76	90.05	2
59	154	142	88.70	89.05	3
60	127	115	85.43	85.90	2
61	106	93	88.86	90.31	1
62	131	122	87.01	88.47	3
63	105	98	89.20	89.87	2
64	115	103	89.23	90.44	2
65	160	136	89.24	89.43	4
66	108	99	84.66	85.57	2
67	141	120	86.74	87.12	3
68	103	97	89.19	90.40	1
69	158	140	86.42	86.69	5
70	109	98	87.31	87.79	3
71	141	127	84.26	84.36	4
72	135	122	87.26	87.34	3
73	127	116	85.37	86.06	2
74	101	91	87.28	87.50	2
75	116	108	88.34	89.72	2
76	113	93	89.66	90.05	1
77	171	154	89.26	89.43	5
78	163	148	87.60	88.27	4
79	190	178	88.69	88.88	5
80	143	122	86.03	86.42	3
81	101	86	87.00	87.18	1
82	135	123	87.80	88.20	2
83	127	108	89.18	89.54	2
84	138	129	88.83	89.51	3

SET #	TOTAL NUMBER BOXES	PACKED NUMBER OF BOXES	% PALLET VOLUME UTILIZATION	% PACKED VOLUME OF ALL BOXES	SOLUTION TIME (SECOND)
85	126	112	86.57	86.80	2
86	126	115	89.36	90.52	2
87	113	89	87.06	87.47	2
88	171	153	87.80	87.83	5
89	93	73	87.15	87.52	2
90	99	90	87.08	87.21	2
91	138	126	87.54	87.91	3
92	116	101	87.35	88.97	1
93	84	75	84.01	84.97	2
94	130	124	88.81	89.51	2
95	186	175	86.99	87.22	5
96	155	141	87.27	87.56	2
97	137	119	86.97	87.89	2
98	133	125	88.47	88.86	2
99	135	128	89.93	91.33	3
100	135	122	87.65	87.99	2

BOX SET BR #6: 15 DIFFERENT BOX TYPES

SET #	TOTAL NUMBER BOXES	PACKED NUMBER OF BOXES	% PALLET VOLUME UTILIZATION	% PACKED VOLUME OF ALL BOXES	SOLUTION TIME (SECOND)
1	129	116	87.35	87.58	3
2	149	136	87.90	88.04	4
3	138	119	86.92	87.15	4
4	144	128	87.05	87.27	4
5	145	128	88.51	90.17	3
6	138	123	86.54	86.80	4
7	123	113	89.00	90.03	2
8	104	94	88.14	90.03	2
9	124	110	85.78	86.24	3
10	144	127	85.39	85.98	4
11	131	122	86.68	86.87	2
12	122	105	86.21	86.39	3
13	203	186	88.44	89.32	9
14	143	135	88.64	88.69	4
15	95	82	86.53	87.04	1
16	122	107	86.71	87.57	3
17	121	102	87.14	87.71	3
18	88	75	87.09	87.12	2
19	132	118	87.64	87.86	3
20	140	119	89.83	90.11	4

SET #	TOTAL NUMBER BOXES	PACKED NUMBER OF BOXES	% PALLET VOLUME UTILIZATION	% PACKED VOLUME OF ALL BOXES	SOLUTION TIME (SECOND)
21	129	114	86.99	88.64	3
22	131	123	88.92	89.56	3
23	116	111	88.29	90.17	3
24	105	89	85.60	86.20	2
25	138	121	84.60	84.87	4
26	165	149	87.55	87.64	5
27	128	117	87.40	87.51	3
28	152	134	87.08	87.53	4
29	138	130	87.23	87.46	3
30	135	123	88.93	89.22	4
31	159	142	88.49	88.68	6
32	143	132	87.34	87.37	4
33	141	122	90.52	90.95	4
34	113	94	87.58	89.27	3
35	152	131	89.10	89.81	5
36	113	103	87.89	88.11	2
37	106	94	88.20	89.08	2
38	131	116	86.00	86.34	3
39	183	168	88.92	89.25	7
40	102	89	86.24	86.71	3
41	131	109	88.57	88.66	3
42	125	113	87.09	87.26	3
43	108	98	86.07	86.28	3
44	140	121	87.21	87.55	4
45	157	137	87.41	88.11	5
46	111	98	89.08	89.63	2
47	147	130	85.89	85.96	5
48	110	98	88.82	89.20	2
49	108	102	89.43	91.03	2
50	145	136	88.49	88.51	4
51	175	154	88.47	88.85	6
52	140	130	86.01	86.20	4
53	128	97	87.61	88.29	3
54	165	155	88.05	88.14	5
55	144	126	85.36	85.63	4
56	153	138	87.96	88.36	4
57	111	104	86.64	86.68	3
58	116	109	89.35	89.58	2
59	126	105	85.81	86.65	3
60	132	117	86.91	87.14	3
61	110	96	87.11	87.34	2
62	123	110	86.50	86.79	3

SET #	TOTAL NUMBER BOXES	PACKED NUMBER OF BOXES	% PALLET VOLUME UTILIZATION	% PACKED VOLUME OF ALL BOXES	SOLUTION TIME (SECOND)
63	114	103	88.08	88.13	3
64	125	110	85.78	87.06	2
65	156	132	91.31	91.42	4
66	116	101	86.95	87.44	4
67	150	136	87.82	88.17	5
68	103	91	86.68	88.02	2
69	167	147	85.83	86.65	7
70	106	87	86.78	86.90	2
71	137	124	88.93	89.06	4
72	132	122	87.18	87.40	4
73	114	103	85.90	86.50	2
74	100	88	86.74	88.17	2
75	123	104	87.56	87.83	2
76	122	110	88.46	88.46	3
77	158	138	86.98	87.21	4
78	155	143	87.83	88.07	4
79	168	150	88.53	88.64	4
80	158	142	86.67	87.29	4
81	94	79	85.73	86.23	2
82	139	127	87.19	88.46	3
83	129	112	87.19	87.91	3
84	137	116	87.10	87.38	4
85	128	111	88.17	88.95	3
86	138	114	87.96	88.06	3
87	108	99	86.91	87.48	2
88	171	150	85.73	85.90	7
89	100	90	86.52	87.62	2
90	85	77	86.45	86.68	2
91	123	112	85.88	86.08	3
92	114	102	88.15	88.30	2
93	86	74	84.32	85.05	1
94	138	122	87.32	88.22	3
95	155	143	87.49	87.54	5
96	131	112	90.31	90.35	2
97	141	124	88.49	89.34	4
98	147	129	88.06	88.32	5
99	120	114	89.87	90.26	2
100	139	126	86.02	86.66	4

BOX SET BR #7: 20 DIFFERENT BOX TYPES

SET #	TOTAL NUMBER BOXES	PACKED NUMBER OF BOXES	% PALLET VOLUME UTILIZATION	% PACKED VOLUME OF ALL BOXES	SOLUTION TIME (SECOND)
1	110	97	86.13	88.00	3
2	129	117	86.10	87.42	4
3	126	109	86.89	87.06	5
4	153	138	86.72	86.72	5
5	126	116	86.61	86.93	4
6	156	135	88.36	88.65	7
7	109	94	86.73	86.97	3
8	119	105	87.69	87.81	4
9	129	116	86.25	86.82	4
10	135	119	87.14	87.27	5
11	143	126	87.95	88.21	5
12	146	133	86.73	86.80	6
13	172	158	87.75	88.64	9
14	119	111	88.19	88.41	3
15	117	106	85.50	85.52	4
16	118	106	88.16	88.31	3
17	125	113	85.88	87.09	4
18	103	91	87.04	87.26	3
19	135	121	88.20	89.36	5
20	130	116	88.98	89.32	4
21	133	126	88.33	89.50	5
22	130	124	87.39	87.69	4
23	157	135	86.07	86.63	7
24	105	95	87.27	87.66	2
25	130	113	85.04	85.71	4
26	151	134	86.04	86.13	6
27	121	116	87.88	88.24	3
28	144	134	88.24	89.17	5
29	117	105	85.90	86.24	3
30	143	127	86.88	87.09	6
31	161	144	87.02	87.22	8
32	149	133	88.14	88.53	7
33	142	120	89.04	89.41	5
34	106	98	86.51	87.32	3
35	162	139	88.05	88.10	6
36	133	123	87.09	88.12	5
37	99	88	87.51	88.31	3
38	120	107	86.42	86.73	4
39	167	143	87.35	88.00	8
40	119	109	86.98	87.18	3

SET #	TOTAL NUMBER BOXES	PACKED NUMBER OF BOXES	% PALLET VOLUME UTILIZATION	% PACKED VOLUME OF ALL BOXES	SOLUTION TIME (SECOND)
41	127	108	87.78	88.84	4
42	108	95	87.02	88.00	3
43	96	84	87.70	89.18	2
44	156	139	87.02	87.92	6
45	140	122	85.80	86.79	4
46	116	102	87.18	88.15	3
47	150	135	86.97	87.60	6
48	113	104	88.02	89.18	3
49	131	119	87.56	88.09	4
50	149	135	89.17	89.70	5
51	166	144	88.08	88.25	9
52	138	116	86.63	86.88	5
53	122	105	86.59	86.78	3
54	143	136	87.77	89.25	5
55	142	123	88.11	88.68	5
56	162	150	87.77	88.30	7
57	127	113	86.09	86.48	4
58	116	108	90.16	91.65	3
59	129	112	85.83	86.71	4
60	126	103	86.47	86.56	5
61	100	85	86.55	88.36	3
62	126	112	86.68	87.08	4
63	107	94	87.13	88.18	3
64	108	89	88.02	88.59	3
65	168	160	90.03	90.20	7
66	117	109	84.30	85.05	4
67	162	149	86.90	86.91	6
68	105	95	87.77	87.85	3
69	155	134	85.59	85.78	7
70	111	97	84.91	85.38	4
71	127	111	87.65	87.93	4
72	127	114	85.60	86.20	4
73	121	108	84.87	85.14	4
74	108	100	87.68	88.29	3
75	129	116	88.95	89.12	4
76	105	86	87.37	87.44	3
77	158	137	86.83	87.30	6
78	142	129	85.34	86.22	4
79	166	145	86.20	86.50	6
80	128	109	86.75	86.82	4
81	110	92	85.21	85.65	3
82	131	119	87.87	88.51	4

SET #	TOTAL NUMBER BOXES	PACKED NUMBER OF BOXES	% PALLET VOLUME UTILIZATION	% PACKED VOLUME OF ALL BOXES	SOLUTION TIME (SECOND)
83	139	118	86.73	87.15	5
84	154	136	88.10	88.72	6
85	108	99	86.11	87.93	3
86	134	109	88.50	90.17	4
87	117	105	86.14	86.38	3
88	142	119	86.52	87.81	5
89	105	94	88.66	88.74	3
90	97	86	88.11	88.35	3
91	106	93	85.27	86.05	2
92	118	109	89.09	89.19	4
93	90	75	84.49	84.78	3
94	134	125	87.13	87.52	4
95	146	124	86.52	87.20	6
96	151	135	87.68	87.96	6
97	124	111	86.25	87.10	4
98	144	127	88.50	88.58	6
99	145	132	86.52	87.25	6
100	122	107	86.53	86.79	4

Bibliography

Askin, Ronald G., and Charles R. Standridge. Modeling and Analysis of Manufacturing Systems. New York: John Wiley and Sons, Inc., 1993, (320-321).

Ballew, P. Brian. The Distributor's Three-Dimensional Pallet-Packing Problem: A Mathematical Formulation and Heuristic Solution Approach. MS Thesis, AFIT/GOR/ENS/00M-02. Graduate School of Engineering, Air Force Institute of Technology (AU), Wright Patterson AFB OH, March 2000.

Barr, R. S., Golden, B. L., Kelly, J. P., Resende, M. G. C., and Stewart, W. R., Jr. Journal of Heuristics, 1995, 1:9-32.

Bischoff, E., and Dowsland, W. B., An application of the micro to product design and distribution. Journal of the Operational Research Society, 1982, 33 (3), 271-281.

Bischoff, E. E., and Marriott, M. D., A comparative evaluation of heuristics for Container loading. European Journal of Operational Research, 1990, 44 (2), 267-276.

Bischoff, E. E., and Ratcliff, M. S. W. Issues in the development of approaches to container loading. OMEGA, 1995, 23(4):377-390.

Bischoff, E. E., F. Janetz, and M. S. W. Ratcliff. "Loading pallets with non-identical Items," European Journal of Operational Research 84: 681-692 (1995).

Bortfeld, A., and Gehring, H. Ein tabu search-Verfahren mit schwach heterogenem Kistenvorrat. FB Wirtschaftswissenschaft, Fern Universität Hagen, 1997, Technical Report 240.

Chocolaad, Christopher A. Solving Geometric Knapsack Problems Using Tabu Search Heuristics. MS Thesis, AFIT/GOR/ENS/98M-05. Graduate School of Engineering, Air Force Institute of Technology (AU), Wright Patterson AFB OH, March 1998.

CAPE Systems, Inc., CAPE PACK'99 and Truckfill packaging design and pallet loading softwares. http://www.capesystems.com. 24 November 2000.

Chen, C.S., Lee, S. M., Shen, Q. S., An analytical model for container loading problem. European Journal of Operational Research 80, 1995, 68-76.

Coffman, E. G. JR., and Shor, P. W. Average-case analysis of cutting and packing in two-dimensions. European Journal of Operational Research, 1990, 44 (2), 134-145.

Computer Sciences Corporation. Unit Type Code Development, tailoring, and Optimization (UTC-DTO) Phase 2 Final Report. Contract DCA 100-94-D-00144. Falls Church VA: Defense Enterprise Integration Services, December 1997.

Faina, L., A global optimization algorithm for the three-dimensional packing problem. European Journal of Operational Research 126, 2000, 340-354.

Gehring, H., Menschner, K., and Meyer, M., A computer-based heuristic for packing pooled shipment containers. European Journal of Operational Research, 1990, 44 (2), 277-289.

Gehring, H., and Bortfeld, A. Ein genetischer Algorithmus für das Containerbe-Ladungsproblem. FB Wirtschaftswissenschaft, Fern Universität Hagen, 1996, Technical Report 227.

George, J. A., and Robinson, D. F., A heuristic for packing boxes into a container. Computers and Operational Research, 1980, 7, 147-156.

Glover, F., Future Paths for Integer Programming and Links to Artificial Intelligence, Computer and Operations Research, 1986, Vol. 13, 533-549 (9-21)

Glover, F., Laguna, M., Tabu Search. USA: Kluwer Academic Publishers, 1997.

Haessler, R. W., and Talbot, F. B., Load planning for shipments of low density products. European Journal of Operational Research, 1990, 44 (2), 289-299.

Han, C. P., Knott, K., and Egbelu, P. J., A heuristic approach to the three-dimensional cargo-loading problem. International Journal of Production Research, 1989, 27 (5), 757-774.

Imperial College Management School. 2001, http://mscmga.ms.ic.ac.uk/jeb/orlib/thpackinfo.html. 13 January 2001

Liu, N. C., and Chen, L. C., A new algorithm for container loading. Compsac 81-5th International Computer Software and Applications Conference Papers (Chicago: IEEE), 1981, 292-299.

Loh, H. T., and Nee, A. Y. C. A packing algorithm for hexahedral boxes. In Proceedings of the Industrial Automation '92 Conference, Singapore, 1992, pages 115-126.

Manship, Wesley E., and Jennifer L. Tilley. A Three-Dimensional 364L Pallet Packing Model and Algorithm. MS Thesis, AFIT/GIM/LAL/98S-3. School of Systems and Logistics, Air Force Institute of Technology (AU), Wright Patterson AFB OH, September 1998.

Magic Logic Optimization, Inc., Cube-IQ Load Optimization Software. http://www.magiclogic.com/cube-iq.html 6 June 2000.

Martello Silvano, Pisinger, David, and Vigo, Daniele. The Three-Dimensional Bin Packing Problem. Operations Research, 2000 Informs. Vol. 48, No. 2, March-April 2000, pp. 256-267.

Mohanty, B. B., Mathur, K., Ivancic, N. J., Value considerations in three-dimensional packing – A heuristic procedure using the fractional knapsack problem. European Journal of Operational Research 74, 1994, 143-151.

Ngoi, B. K. A., Tay, M. L., and Chua, E. S. Applying spatial representation techniques to the container packing problem. Int. J. Prod. Res, 1994, 32:111-123.

Ravindran, A., Phillips, Don T., Solberg, James J., Operations Research Principles and Practice, Second Edition. U.S.A.: John Wiley & Sons, Inc., 1986.

Reeves, Colin R. Modern Heuristic Techniques for Combinatorial Problems. London: Mc Graw-Hill Book Company, 1995, 151-188.

Romaine, Jonathan M. Solving the Multidimensional Multiple Knapsack Problem with Packing Constraints Using Tabu Search. MS thesis, AFIT/GOR/ENS/99M-15. Graduate School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1999.

Steudel, H. J., Generating pallet loading patterns: a special case of the two-dimensional Cutting stock problem. Management Science, 1979, 25 (10), 997-1004.

TASC, Inc. (8 May 1998) Interim Project Report on the Feasibility of Finding Optimal Solutions to Three-Dimensional Packing Problems. Contract number F41624-97-D-5002, Air Force Research Labs/HESR, Wright-Patterson AFB OH.

Taylor, Gregory S. A Pallet Packing Postprocessor for the Logistics Composite Model. MS thesis, AFIT/GST/ENS/94M-11. Graduate School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1994.

Terno, J., Scheithauer, G., Sommerweiß, U., and Riehme, J. An Efficient Approach for the Multi-Pallet Loading Problem. Institute for Numerical Mathematics, Technical University Dresden Mommsenstr. 13, D-01062 Dresden, Germany 1997.

Tinarelli, G. U., and Addonizio, M., Un problema di caricamento di containers, Proc. AIRO, Rome, Italy 1978.

Vita

1Lt. Erhan BALTACIOĞLU was born in Ankara, Turkey. He grew up in Istanbul and eventually graduated from Kuleli Military High School. He then attended the Turkish Air Force Academy in Istanbul. On 30, August 1994 he received his commission as well as his Bachelor's of Science degree in Computer Science. On 25, May 1996 he graduated from Turkish Flight School in Izmir. He was assigned to be a C-130 pilot in Kayseri. In August of 1999, he was assigned by the Turkish Air Force to the Air Force Institude of Technology in pursuit of a Master's degree in Operations Research.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
20-03-2001	Master's Thesis	March 2001 - March 2001

4. TITLE AND SUBTITLE
THE DISTRIBUTER'S THREE-DIMENSINAL PALLET-PACKING PROBLEM: A HUMAN INTELLIGENGE-BASED HEURISTIC APPROACH

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Erhan BALTACIOGLU, First Lieutenant, TUAF

5d. PROJECT NUMBER
HE-AFIT-99-10

5e. TASK NUMBER
ZA

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 P Street, Building 640
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GOR/ENS/01M-02

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Dayton Area Graduate Studies Institute
Attn: Dr. Frank Moore
3155 Research Blvd, Suite 205
Kettering, OH 45420 (937) 781-4000

10. SPONSOR/MONITOR'S ACRONYM(S)
DAGSI

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The Distributor's Pallet Packing Problem is to load a set of distinct boxes with given dimensions on pallets or in containers to maximize volume utilization. This problem is still in its early stages of research, but there is a high level of interest in developing effective models to solve this NP-hard problem to reduce the time, energy and other resources spent in packing pallets.

In its search to improve operations, the Air Force is also making an effort to solve this problem. Building an analytical model and developing a genetic algorithm approach have been tried, but the problem needs more research and there is a need to produce realistic solutions in a reasonable amount of time.

We develop a special heuristic algorithm and code it in the C programming language. In our model, we used powerful heuristic tools and dynamic data structure to mimic human behavior, providing a new solution approach to pallet packing. We created another program to visualize packing results. Tests on hundreds of problems show that our model makes the most of volume utilization in minimal time making it a leader among presented and published works.

15. SUBJECT TERMS
Three-dimensional pallet-packing, packing problem, bin-packing problem, multi-dimensional knapsack problem, pallet-loading problems, distributor's pallet packing problem, meta-heuristic algorithms

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			James T. Moore, Lt Col, USAF
U	U	U	UU	135	19b. TELEPHONE NUMBER *(Include area code)* (937) 255-6565, ext 4337 (James.Moore@afit.edu)

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18