Analysis on the Less Flexibility First (LFF) Algorithm and Its Application to the Container Loading Problem

WU Yuen-Ting

A Thesis Submitted in Partial Fulfilment

of the Requirements for the Degree of

Master of Philosophy

In

Computer Science and Engineering

Abstract of thesis entitled:

Analysis on the Less Flexibility First (LFF) Algorithm and Its Application to the Container Loading Problem

Submitted by WU Yuen-Ting

for the degree of Master of Philosophy

at The Chinese University of Hong Kong in August 2005

This thesis is a deep investigation in the Less Flexibility First (LFF) algorithm for solving packing problems. LFF is an effective deterministic heuristic originally designed for solving 2D packing problems. It generated up to 99% packing densities. Yet the algorithm may generate unsatisfactory packing densities when the length/width ratio of the rectangles is very large. To improve its performance, the algorithm is analyzed by applying it to those extreme cases. The inexistence of LFF error bound is proven, followed by a discussion on the causes of the unsatisfactory results. This research result can help improving original LFF in future.

As the average packing density of LFF in 2D packing problems is very high comparing to other 2D-packing algorithms, it is also extended to the container loading problem with the objective to maximize volume utilization. The principle is to pack objects according to their flexibilities. Less flexible objects are packed to less flexible positions of the container. Pseudo-packing procedures enable improvements on volume utilization. Encouraging packing results with up to 93% volume utilization (average volume utilization is about 87.93%) are obtained in experiments running on benchmark cases from other authors. The result of this research was published in the paper "A Less Flexibility First Based Algorithm for the Container Loading Problem" in the Operations Research Proceedings 2004.

The disadvantage of the LFF algorithm is its large CPU time. To cut up the CPU time, we try to modify the algorithm by applying tightness calculation. This algorithm is called "Less Flexibility First with Tightness Measure" (LFFT). There is a substantial reduction in CPU time. The volume utilization is also reduced as the trade-off. Detail comparison of the two algorithms is done by a series of experiments and reported in this thesis.

摘要

論文題目: 「低靈活性優先處理法」分析及應用於三維包裝的研究
作者: 胡婉婷
修讀學位: 哲學碩士
香港中文大學計算機科學及工程學部
日期: 二零零五年六月

本論文就如何將「低靈活性優先處理法」應用於三維包裝問題作出深入研究。「低靈活性優先處理法」最早見於二零零二年的一本《歐洲運籌學期刊》一篇名爲「模擬人類思維法以解決平面方形的包裝問題」的文章內，該篇文章透過標準測試案例證實運用「低靈活性優先處理法」包裝平面方形，可達致 99 個百分點的密度。可惜這種有效的「模擬人類思維法」仍有其不足之處。在某些極端情況下，「低靈活性優先處理法」可能產生極不理想的包裝密度。本文嘗試找出這些影響「低靈活性優先處理法」表現的極端情況，並加以研究，計算錯誤邊界及分析其成因，冀能改善「低靈活性優先處理法」的整體表現。

由於「低靈活性優先處理法」在平面包裝的問題上表現勝於其他同類程式，是次研究的另一重點便放在如何擴大「低靈活性優先處理法」的應用，將這種方法運用在三維包裝問題上，以求取得更好空間佔用率的一種包裝方法。「低靈活性優先處理法」的原則是根據物件的靈活性決定其入箱的先後次序。由於大型的物件較欠缺靈活性，故會優先處理。至於箱內的空間也列入靈活性的考慮因素之內。體積較小的空間靈活性較低，因此會被優先檢查是否適合存放正被處理的物件。虛擬包裝步驟有助改善空間佔用率。「三維低靈活性優先處理法」曾被應用在標準包裝案例上，並產生出高達 93 個百分點的空間佔用率(平均空間佔用率約爲 88 至 89 個百分點)，實在是令人鼓舞的結果。

「低靈活性優先處理法」的缺點在於較長的程式作業時間。爲了縮短其作業時間，作者在程式中加入緊密度的計算。結果作業時間被大大降低，而程式所計算出來的空間佔用率亦相對減低了。本論文將對兩種不同的設計作出討論，並比較其實驗結果。

Acknowledgement

I would like to take this opportunity to express my gratitude to my supervisor Prof. David Y.L. Wu, for his generous guidance and patience given to me in the past three years. His numerous support and encouragement, as well as his inspiring advice are extremely essential and valuable and valuable in my research paper (conference paper published in Operations research 2004 Proceedings) and my thesis.

I am also grateful for the time and valuable suggestion that Prof. Y.S. Moon and Prof. Evan Young have given in marking my term paper. Without their effort, I will not be able to strengthen and improve my research project and papers.

I would also like to show my gratitude to the Department of Computer Science and Engineering, CUHK, for the provision of the best equipment and pleasant office environment required for high quality research.

Special thanks should be given to Miss Chan Pik Wah and Miss Chan Pik Hin who have given me valuable suggestions, encouragement and supports. And I would like to give my thanks to my fellow colleagues. They have given me support, and a joyful and wonderful university life.

Finally, I am deeply indebted to my family for their unconditional love and support over the years.

This work is dedicated to my family for the support and patience

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recently, packing problems have received increasing attention due to its importance in the area of production and distribution of goods. An effective three-dimensional packing algorithm can increase the packing density of a transportation device such as containers and palettes. The higher the packing density, the better is the utilization of transportation capacities. As the volume of waste space is reduced, the reduction on resources used on transportation can in turn be achieved. Therefore, research on three-dimensional packing problem is of great economic values, especially for the transportation and logistics industry.

1.1 Background

In Three-Dimensional Packing Problems, a set of rectangular-shaped boxes is to be packed into single or multiple container(s) with fixed dimensions orthogonally. All packed boxes must be completely stowed inside the container with all edges parallel to the container edges. Overlapping of boxes is not allowed. All boxes should be supported by the container base, by any other boxes or by added supporting materials underneath to an extent that it is stable. Orientation constraints are usually considered in real-life applications.

The problem was firstly studied by Gilmore and Gomory [9] in early sixties. Thereafter many researchers have published papers discussing several variants of this problem. Classification of 3D packing problems can be done in two ways: by types of boxes and by objective functions. The former classified the problem into three types. In homogeneous cases, a single type of boxes (all with the same dimensions) is to be packed. A weakly heterogeneous box set refers to a few number of box types with a lot of individual box in each type. A strongly heterogeneous box set refers to a large number of box types with only a few individual boxes in each type [25]. For the latter classification, three objectives are usually considered: [8, 12, 13]

(i) *Strip Packing.* All boxes should be packed into a container with fixed width and height but infinite depth. The depth is continuously increased during the packing process until all boxes are loaded. The objective here is to minimize the depth of the container. Many heuristics and algorithms are designed for this application. The first heuristic for this problem was presented by George and Robinson in [18]. Approximation algorithms are widely used for solving container loading problems. In [7], a parametric online algorithm for packing boxes is illustrated. [5] describes another approximation algorithm. A comparison of several algorithms focusing on strip packing can be found in [21].

(ii) *Bin Packing.* All boxes should be packed into containers (or bins) with fixed dimensions. Multiple containers are usually necessary. The objective is to minimize the number of containers used. [3]

presents an incorporation of approximation algorithm with exact algorithm to solve 3D bin packing problem. It first defines an exact algorithm ONEBIN for finding the best filling of a single bin by branch-and-bound, then incorporates approximation algorithm to perform bin packing. Greedy algorithms are applied to variable sized bin packing in [4], where Iterative first-fit decreasing (IFFD) and Iterative best-fit decreasing (IBFD) are illustrated.

(iii) *Knapsack Loading.* In this variant of the problem, some boxes can be left unpacked. A subset of the given boxes is chosen to be packed into a fixed dimension container to maximize a pre-defined profit. Heuristics for knapsack loading are presented in [17] and [19]. When the profit is set to be the utilized volume, the objective will be the minimization of wasted space. In industries, knapsack loading is applied to the loading of cargoes into a container in a way that can minimize the wasted space and in turn reduce the transportation cost. This is called Container Loading.

Our research is focused on the container loading problem, aiming at maximizing the volume utilization of the container. This is proven to be NP-complete [15]. No existing algorithms are able to give optimal solutions in polynomial time. Heuristics are the mostly adopted approaches for this kind of problems.

Several heuristics using layering approach is introduced in [8, 25]. The advantage is mainly on the balancing of load inside the container. An unbalanced condition can be solved by simply exchanging the order of layers. As the width and height of each layer coincide with the

corresponding dimensions of the container, a key factor affecting packing density is the determination of layer depth. [8] presents several ranking rules for selection of the most promising layer depths. [25] introduces a hybrid genetic algorithm, also based on layering. It uses basic heuristics developed by [27] to generate and complete stowage plan layers while genetic algorithm is used in later phase to perform layer transfer and layer extension.

Genetic algorithms are common on non-layering methods. [16, 20] are some examples. Other methods are also developed. An integer programming model is presented in [6]. [15] presents a greedy heuristic improved by tree search. [26] runs tabu search in several computers in parallel to generate local optimum solutions. The diversity is increased by exchanging solutions between different computers.

1.2 Research Objective

In this thesis, we present a Less Flexibility First (LFF) based algorithm for solving the container loading problem. The objective is to maximize volume utilization, i.e. the % of the container volume occupied by the packed items.

The LFF algorithm was originally applied to two-dimensional packing problems and was in fact proposed in [1]. It is a *quasi-human* based heuristic. The term "*quasi-human*" means cogitative, which refers to the accumulated experience of human beings in solving similar problems in everyday life [10, 11]. The idea is inspired by an old strategy known by Chinese ancient professionals for packing polygon-shape stone plates. In

2D packing cases demonstrated in [1], it consistently produces results with around 99% packing densities in a large number of randomly generated rectangle packing instances with at least 45 rectangles.

According to some past research [24], algorithms with good performance in two-dimensional packing can probably generate satisfactory results when similar techniques are extended to three dimensional cases. The performance of 2D LFF shows its potential to produce promising volume utilization if it is extended to 3D.

In our research, the first step is to evaluate the 2D LFF algorithm by analyzing its worst case performance. Although there is even no error bound for this algorithm in the worst cases, its average performance is very satisfactory. The algorithm is determined to be possible to have good performance in 3D packing problem. The next step is to identify the possible problems which may be encountered. Such problems should be taken into consideration during implementation. Overcoming these difficulties is important in our research.

The major objective is to implement the 3D LFF algorithm which can do packing with high volume utilization. Its performance was evaluated by running it on the benchmark test cases and compared its results with other container loading algorithms through which the pros and cons of 3D LFF will be disclosed and evaluated.

1.3 Contribution

Our research has the following contributions:

- The 2D LFF algorithm is evaluated. Worst Case Analysis is performed on the algorithm and the inexistence of Error Bound of this

algorithm is proven. Such analysis facilitates further research on improvement of this algorithm.

- The LFF algorithm is firstly applied on container loading problem. Its performance is satisfactory when being run on benchmark cases. In Bischoff and Ratcliff benchmark cases, the average volume utilization achieved by our algorithm is 87.93%.

- In 3D packing research area, genetic algorithms and layering approach are the two most common solutions under research. The 3D LFF is an innovative approach which can provide a new direction for further research.

1.4 Structure of this thesis

There are six sections in this thesis. Section 1 is the Introduction which gives reader a brief description about what the research topic is, the objective and its contributions. Section 2 is Literature Review which compares different algorithms proposed by other researchers on the same problem. Section 3 introduces the Principle of the Less Flexibility First (LFF) and discusses how it was applied to 2D packing problems. Section 4 contains a worst case analysis as well as a proof on the error bound of LFF algorithm. Section 5 explains how the LFF algorithm is extended to 3D container loading problem and demonstrates some experimental results on benchmark test cases. Section 6 is the Conclusion discussing the superiorities and also issues of the LFF algorithm.

Chapter 2

Literature Review

Two traditional approaches for solving Knapsack Loading (Container Loading) problems are Genetic algorithms (GA) [16, 20, 24, 25] and Layering approach [19, 25, 29, 31]. Some algorithms use one of them while combination of the two is also possible. Other approaches like Tabu Search [26], Mix Integer Programming [6] are also proposed. As the problem is NP-complete, exact algorithm is normally not employed.

2.1 Genetic Algorithms

Genetic algorithm was introduced by HOLLAND (1975). It can be regarded as a type of meta-heuristics. In GA, the genetic operators *Crossover* and *Mutation* can explore the solution space and generate different solutions for evaluation. The *Selection* procedures ensure that only the individuals considered as the best are allowed to stay in the population and act as parents to generate offsprings. As the individuals with low fitness values are eliminated, GA can gradually improve the fitness of the whole population and finally achieve satisfactory results. Therefore it is involved in many researches on NP-complete problems [16, 20, 24, 25, 31].

H.Gehring and A.Bortfeldt had published twice on using genetic algorithms to solve container loading problem. Their first proposal [24] is to arrange boxes into tower sets and make use of an already developed 2D

Packing GA in [17] to obtain a tower packing result. This is called stack-building. Their second proposal [25] is similar. This is a combination of GA and layer approach. Boxes are packed into layers and the packing of layers is done by GA. C. Pimpawat and N. Chaiyaratana divided the individuals into sub-populations (or species) and performed GA on each sub-population by which the solution space is explored [20]. D.Y. He and J.Z. Cha represented packing patterns by permutations and applied GA on the permutations [16].

In this section, the operations and performance of the above four GAs are compared and analyzed.

2.1.1 Pre-processing step

Before generating the initial populations, the GA will do some preprocessing steps to encode the problem into a suitable data structure to facilitate further genetic operations. Such preprocessing steps are always very important for forming a suitable initial condition for GA to be run.

In [24], boxes to be packed are firstly built into a set of disjunctive towers. In a pair of disjunctive towers, none of the boxes is shared by both towers. Generation of a tower set comprises the subdivision of the given set of boxes into disjunctive subsets as well as the fitting together of the boxes of each subset into a tower.

The generation of towers takes volume utilization into consideration. The first step is to try using each free box (boxes which are not packed) as the base box of a new tower. By repeatly dividing the residual spaces into three regions, i.e., the space in front of, besides and above the packed box, and packing the remaining boxes to the residual spaces, a tower is built.

As the step repeats for each free box and different orientations are also considered, many towers are formed and the one with the minimal spare space is chosen. This ensures that the formed towers are those with better volume utilization. The sets of towers are finally sorted in descending order according to the area of the tower bases. This ends the preprocessing step of this algorithm.

The preprocessing step in [25] is very similar to that in [24], but with some improvements. This time the tower sets are regarded as "layers" while both are referring to sets of boxes grouped together by certain criteria.

The major difference is that in each step of packing, one or two boxes can be selected to be placed. The order of space-filling is determined by a new rule. As mentioned before, after packing a box to a tower, three regions of residual spaces are formed. These spaces, called daughter spaces, are filled in arbitrary orders in [24] but are filled in ascending order of volume this time. Smaller daughter spaces have higher priority to be filled next. Rules are also defined to determine rotation variants and allocation of boxes to the current and residual daughter-space.

Another difference is the ability to extend residual sister-spaces. In daughter-space formation, there can always be two ways to define the "in-front" space and "beside" space. Before choosing either of these division methods, the algorithm will first determine whether any of them will lead to unfillable daughter-space. If one method leads to waste of daughter-space while another does not, the latter one will be selected. This improvement can reduce the volume of unfillable spaces as much as possible.

Except these two differences, other steps of layer filling steps are similar to the tower set formation in [24]. The preprocessing step ends

with all the layers outputted for generating initial populations.

The CCGA algorithm [20] aims at exploring the solution space by means of utilizing a number of species or sub-population where each individual in a species represents a component of a complete solution. The major preprocessing of this algorithm is to initialize M sub-populations where each sub-population has N individuals. These sub-populations are all regarded as a part of the original problem which will be optimized separately.

No preprocessing is performed in [16].

2.1.2 Generation of initial population

In [24], after the tower sets are generated, the remaining problem is to arrange tower bases on the container floor. To encode the initial population as chromosomes, the base box of each tower is related to an index indicating its sequence of placement and a rotation variant specifying its orientation. As the unit under consideration is a tower, the number of feasible orientations can only be two.

Two approaches of GAs are introduced. The process of initial population generation of the first one runs on a random basis while the second one hybridizes this process by running serial testing before inserting a tower base into the placement vector of the chromosome.

In [25], with the layers generated as described in previous section, stowage plans are formed as the initial populations. These stowage plans are generated by the *start* procedure in which an empty container is to be filled by a list of feasible layer definitions. Feasible layers should have its base box free (unpacked before) and can be placed in the remaining

container area with its own rotation variant. These layer definitions are added one by one to the stowage plan under the rule that the feasible layer with the highest volume layer utilization should always be added. The total packed volume is calculated as the objective function value (fitness value) of that stowage plan. *Start* can be operated on different layer definition lists to generate different stowage plans for further processing by GA.

In CCGA [20], during generation of initial population, all possible combinations between all individuals from different species are explored. This means that an individual will participate in more than one solution which results in a number of fitness values for each individual. Finally the highest fitness value for each individual is assigned to it and is used in other genetic operations.

The initial population of permutations is randomly initialized in [16].

2.1.3 Crossover

The *crossover* can be considered as a standard operator for all genetic algorithms. It is always done by selecting two parents with high fitness value and cross them to generate two descendents. The difference in *crossover* for different algorithms is not on this operation itself, but on the chromosomes. The result of *crossover* for different chromosomes will certainly be different.

However, the *crossover* operator in [25] involves two phases and is quite different from the traditional *crossover*. The first phase *layer transfer* of layers from parents is similar to generic crossover, yet the transfer is traced by constraints which, when violated, will stop the transfer

of a layer from parent to offspring. Therefore some layers may not be transferred to the descendant. The second phase is *layer extension* which generates new layers to extend the descendant into a complete stowage plan.

2.1.4 Mutation

The *mutation* operator varies more in different algorithms. In [24], two types of *mutation*, namely *scramble sublist mutation* and *mutation by inversion* are adopted. The mutation in CCGA [20] is a standard one, using the reciprocal change approach. In [16], mutation is carried out by inversion. Same as the case in crossover, the author of [25] applies two types of special mutation operators involving *layer transfer* and *layer extension.* One type is *standard mutation* in which 1% to 50% (determined randomly) of the layers with maximum utilization are transferred from parent to descendant which is followed by extending the incomplete mutant to complete stowage plan by inserting newly generated layers. For *merger mutation,* all layers are transferred from parent layer to offspring except two layers. Layer extension is then performed in the same way as in crossover but only a single additional layer is generated which causes the stowage plan to contain one layer fewer than the parent stowage plan. This *mutation* does not follow the generic approaches but is tailored for the problem under investigation.

2.1.5 Selection

The standard *selection* for all algorithms is by evaluating the fitness value

according to the objective function and selecting those individuals with higher fitness value. Different algorithms have different objective functions due to different constraints and objective of the particular algorithm.

2.1.6 Results of GA on Container Loading Algorithm

Among the four papers discussed, [16] and [20] do not present packing results for benchmark cases. [24] and [25] achieved 87.5% and 88.6%. Comparison with other algorithms will be done in later sections.

2.2 Layering Approach

Layering approach means that packing is achieved by building vertical layers first and the layers are then packed one by one into the container. This is a traditional approach for solving container loading problem [17, 2, 21, 29, 32]. In fact, one of the GAs [25] discussed in last section employs the layering approach

In [19], a wall-building layering approach is proposed. In this algorithm, layers are formed by firstly selecting box to determine the depth of a layer. The ranking of boxes are based on their length of smallest dimension as well as the frequency of the dimensions. The longer the length of smallest dimension, the higher is the rank for that box as such box may be difficult to accommodate in later stage. With the depth of a layer determined, the wall is packed in a greedy way as a number of horizontal strips. Each of these strips is packed by inserting boxes with largest

ranking to the strips. This heuristic is a greedy one. The determination of layer depths and strip widths are very important decisions.

This algorithm uses a tree-search heuristic for determining the above two factors. Ranking rules are set for choosing depth based on the dimensions of the remaining boxes. The second step is to fill the strips by the free boxes.

In that paper, different ranking rules are discussed. According to the author, the volume utilization achieved by the algorithm can reach 95% for "large sized instances". The results in the paper shows that the % volume utilization can achieve about 88-90% for weakly heterogeneous and strong heterogeneous problems while its performance on homogeneous problems are not that satisfactory. The % of volume utilization drops to 75-83%. The algorithm is proven to be unstable for different packing instances.

2.3 Mixed Integer Programming

An analytical model is presented for the container loading problem to capture the mathematical essence of the problem [6]. Container loading process is formulated as a zero-one mixed integer programming.

This approach involves many mathematical concepts and the problem is modeled by a set of mathematical expressions representing the problem itself and some constraints to be evaluated during packing.

Details of mixed integer programming will not be presented here as it is not the focus of this research.

2.4 Tabu Search Algorithm

A. Bordfeldt, H. Gehring and D. Mack have proposed a parallel tabu search algorithm (TSA) for solving the container loading problem [26]. This tabu search algorithm is emphasized on weakly heterogeneous packing instances.

The basic heuristic of this algorithm is to firstly generate all 1-local arrangements (involving one type of boxes only) and 2-local arrangements (two types of boxes are packed in different arrangements). The local arrangements are evaluated to determine which should be inserted to the stowing list. New residual spaces are generated after the packing step. The generation of local arrangements is based on greedy heuristics.

In order to enhance the chances of loading small packing spaces, the packing space with the smallest volume is always process first due to its low flexibility.

The generated local arrangements are used in sequential TSA for encoding feasible solutions. With the encoded solutions, two neighbourhoods, one large and one small, are defined for starting tabu search. All packing sequences of the feasible solutions are embraced by the large neighbourhood. Tabu search then starts.

A drawback of the tabu search problem may be the risk of stucking at the local neighbourhood. This violates the aim of tabu search algorithm, which is searching a solution near the global optimum. In this algorithm, the greedy heuristic applied on the basic heuristic fills all packing spaces in a local-optimizing way. To solve this problem, the parallel TSA approach is developed.

In parallel TSA, a distributed environment is used to run tabu search.

This is based on the concept of multi-search threads [33]. For each thread, the normal sequential TSA is run, but with different parameter configuration. The difference in parameter configuration results in diversification of solutions for different threads.

The difference between the sequential and parallel approaches is that the solutions generated by the threads in parallel approach are diversified due to different parameter configuration. At the end of neighbourhood search of each thread, the threads will communicate with one another to exchange the foreign solutions generated by other threads. This foreign solution becomes the starting point of next round of search in each thread. In this way, the diversity of solution in each thread can gradually be increased.

When sequential TSA and parallel TSA are applied to Bischoff and Ratcliff test cases (BR cases), their volume utilization is 92.0% and 92.7% respectively. The BR cases are divided into 7 sets, ranging from homogeneous to heterogeneous packing instances. The result of most homogeneous test cases is about 93% for both while for most heterogeneous packing result is about 90%. Although there are differences in results between the two types of packing, the worst case of TSA is still better than the average performance of some other algorithms. This proves that TSA is not only good at solving homogeneous packing problems, but also can generate a satisfactory result for heterogeneous ones.

2.5 Other approaches

Some researchers choose not to use traditional packing approaches like

layering and GA, but design new heuristics for solving the container loading problem.

2.5.1 Block arrangement

The most innovative part of the algorithm in [15] is the formation of blocks. In this algorithm, the first step is to group identical items in blocks. One of the advantages is that homogeneous blocks are easy to arrange and therefore enable a quicker loading time.

Greedy heuristic is also pursued. The items are sorted by volume with larger items being chosen first. All possible positions for stowing more items in the container are examined.

The steps following the block formation is to stow the item at the lower back left corner of the empty space and then residual spaces are generated. When stowing each item, all empty space should be considered. The process repeats itself for stowing other items until no empty space or no item is left in the list.

The experimental result shows that this algorithm favors the homogeneous packing instances while its performance in heterogeneous packing instances is the worst.

2.5.2 Multi-Directional Building Growing algorithm

This approach introduces the idea of packing the objects by using wall of containers as starting surface of packing [30]. User can chooses the walls of container for acting as ground. (the base from which the boxes are

packed). Then the packing steps continue based on the best matching between base area of boxes and empty spaces.

Experimental results of this algorithm do not come from benchmark cases and will not be used to evaluate and compare performance with other algorithms.

2.6 Comparisons of different container loading algorithms[1]

Among all container loading algorithms which have benchmarking results, the Parallel Tabu Search algorithm achieves the most satisfactory result (92.7% volume utilization). This is due to its increasing diversity in solution space being searched.

The two GAs employing stack-building and layering approach have lower volume utilization, 87.5% and 90.1% respectively, because in both stack building and layering approach, the packing unit becomes a large set of boxes as a whole instead of individual boxes. This will in fact reduce the flexibility of packing units due to their large size after grouping together. As the boxes in a layer may be of different dimensions, wasted space formed between layers will usually larger than gaps formed between individual boxes. Though the merger mutator is already introduced to minimize such wastage of space, the problem cannot be completely eliminated. It is very difficult avoid wasting such space if layering approach is used.

The problem of the block arrangement approach is quite similar because its lower volume utilization (88.75%) is due to the decrease in flexibility of objects as blocks are formed from them. The large size of a

[1] Data for comparisons in this section come from the reports from other authors

block prevents the block to be placed in narrow gaps, but if only one of the items in the block is packed individually, it may be able to fill the gap. This is the major cause of lower volume utilization of these algorithms comparing with parallel TSA.

Chapter 3

Principle of LFF Algorithm

The "Less Flexibility First" (LFF) algorithm was firstly introduced in [1]. It is a simple but effective heuristic inspired by a strategy used by Chinese ancient masons. Having been applied on packing problems for more than 1000 years, its effectiveness and value had already been proven. Such heuristics developed based on accumulated experience of human beings are called quasi-human based heuristics. Before going into the details of LFF algorithm, some concepts, rules and the basic principle will firstly be discussed in this section.

3.1 Definition of Flexibility

The core idea of LFF algorithm is the concept of flexibility. The order of packing is determined by the flexibility of objects, in ascending order (i.e. less flexible objects are packed before more flexible ones). The space of a container also has an order of flexibility. Flexibility measure is based on two rules:

- Flexibility of empty space follows this order: flexibility of a corner is less than flexibility of a side, while flexibility of a side is less than flexibility of a central void area. This order is determined according

to the freedom of move. In Fig. 3.1a), an object at a corner is bound by two sides and can no longer be moved. If it is moved in any of the three directions as shown, it will leave the corner position, resulting in an invalid location, i.e., no longer a corner. Therefore, no movement is possible if the object is required to stick to a corner. In Fig. 3.1b), objects on a side can be moved along the side as shown. However, only the two horizontal moves towards left or right can keep it touching the side. The flexibility is one-dimensional. In Fig.3.1c), those objects at centre void area can move freely to any direction as shown in Fig. 3.1c) without violating the rule of keeping the object at void area [1]. In this case, flexibility is multi-dimensional, i.e., the most flexible.

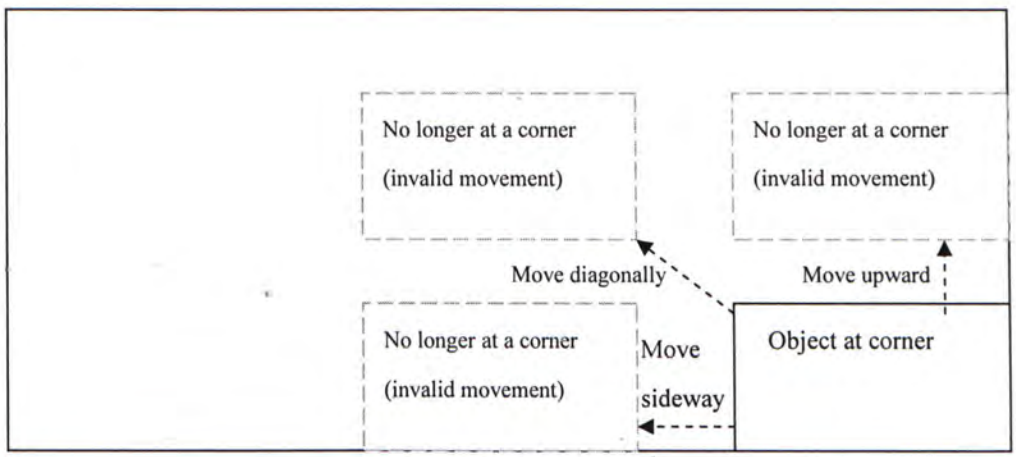


Fig. 3.1a) flexibility of a corner (move along dashed arrows place the object in non-corner positions)

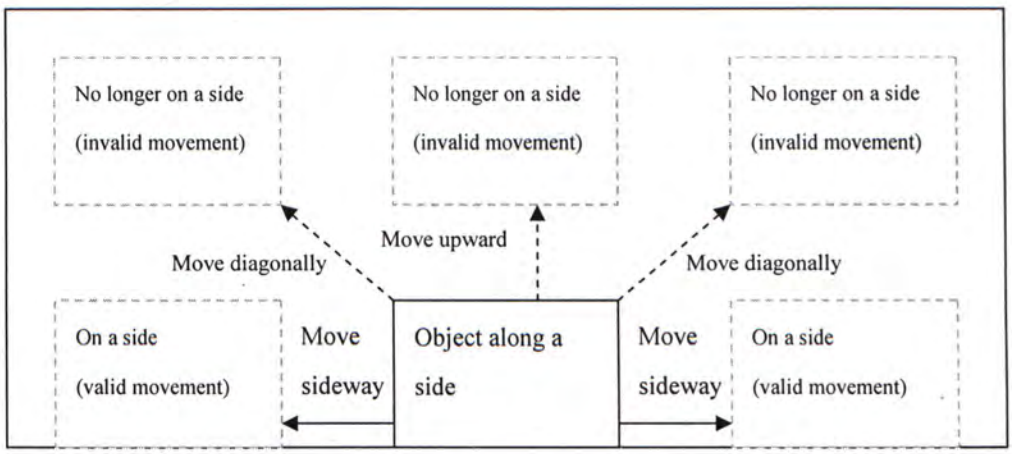


Fig. 3.1b) flexibility of a side (move along dashed arrows place the object in positions not along a side)

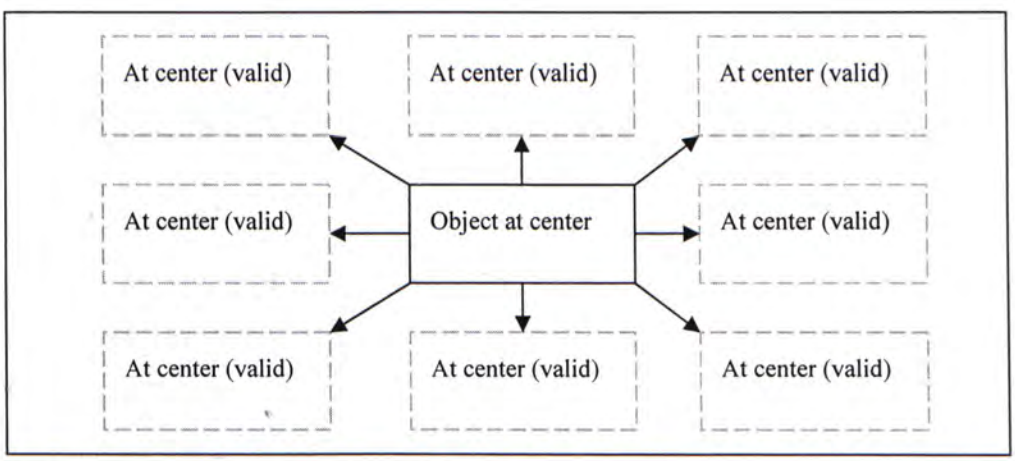


Fig. 3.1c) flexibility of a central void area (can move in any direction)

- Flexibility of objects to be packed cannot be evaluated by an exact formula, but can be roughly determined based on the size and shape of an object. In general, there are fewer positions that can accommodate a larger object, while a space may be found for smaller object more easily. Fig. 3.2 shows the condition. The smaller object *Y* can be packed into space I, II or III while the bigger object *X* can only be

packed into space II. This implies that flexibility of larger objects (with longer side) is less than that of smaller objects.

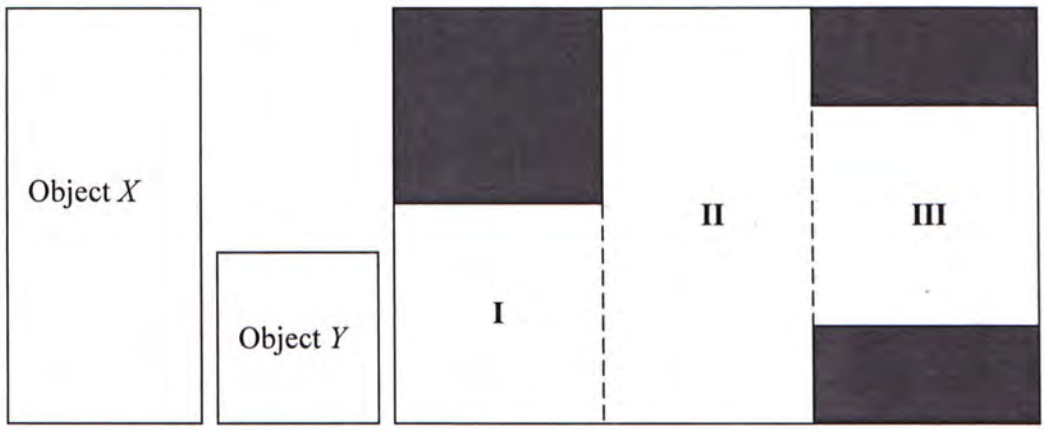


Fig.3.2 Flexibility of objects

3.2 The Less Flexibility First Principle (LFFP)

LFFP is inspired by the wisdom of ancient Chinese masons who perform packing task using the rule "Golden are the corners; silvery are the sides, and strawy are the voids" [1]. From this rule, the selection criterion of empty space is clear; empty corners should be filled first, which is followed by boundary sides and void areas come last.

We derived our LFFP from this ancient packing strategy. Corners are defined as the least flexible space for packing while larger objects are less flexible objects. For detailed explanation of flexibility, please refer to Section 3.1 of this thesis. LFFP packs the least flexible objects to the least flexible empty space. In other words, large objects are packed first to the empty corners. This ensures that a box has at least two adjacent boxes (or sides of container walls) touching two of its lateral surface and its base is supported by another box (or container walls) at the bottom (or touches

another box on the top, if gravity is not taken into consideration). Packing in a void space should be avoided as this creates empty space surrounding the packed box b_p. Such space may not be large enough for accommodating other boxes in later stages. The box b_p becomes an obstacle in this case.

Fig.3.3 illustrates an example (Top view). Dashed arrows points to the directions with empty space left while other arrows points to directions bounded by container walls or adjacent boxes. Consider three candidate packing positions I, II and III for a box. Position I has empty space on its right only while position II has empty spaces on two sides. Position III is surrounded by empty spaces in all directions. I is the best choice among the three (two corners are occupied in I). II is the medium choice (one corner is occupied). III is the worst (object placed in central void area). It is obvious that the more the corners occupied by a box, the lower is the probability of having empty spaces surrounding this box.

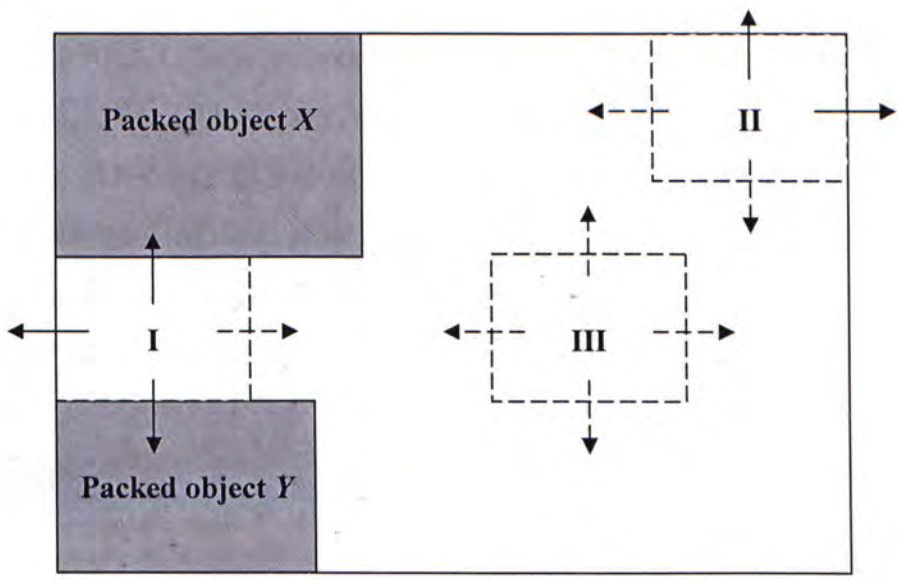


Fig.3.3 Candidate packing position for an object

There are always corners in any packing configuration unless 100% packing density is achieved. In LFF, the only candidates of space for packing an object are the corners. A packed object must occupy at least one corner in the current packing configuration.

To select a corner for packing the current object among all the available corners, a mechanism should be developed to evaluate the suitability for the object under investigation to be put to the evaluated corner. The following section describes the 2D LFF algorithm introduced in [1] which achieves around 99% packing density in most randomly generated large-sized examples.

3.3 The 2D LFF Algorithm

As the 3D LFF algorithm is developed based on the 2D LFF algorithm,its predecessor is introduced here to give readers a better understanding before proceeding to the later sections. But to avoid repetition, the 2D LFF is described briefly. More detailed concepts and examples will be discussed in Section 5 in the 3D LFF part, which is the focus of this research.

In the 2D Packing problem, a large empty rectangle (acting as a container for packing) and a set of smaller rectangles with arbitrary sizes are given. The aim is to find out whether it is possible for the container to accommodate all the smaller rectangles, provided that the placement is:

- orthogonal
- no overlapping and
- with all rectangles packed within the container, i.e. the boundary of the container is not exceeded.

If a complete packing solution does not exist, the algorithm will obtain a partial solution with minimized unpacked space, i.e. the highest packing density. The following briefly describes the steps of the 2D LFF algorithm.

3.3.1 Generation of Corner-Occupying Packing Move (COPM)

As mentioned before, there may be more than one corner for an object to be packed to. In order to represent the relationship between objects and the corners occupied by them, a quinary-tuple is introduced:

<longer side, shorter side, orientation, x_l, y_l>

where longer side and shorter side are the length and width, respectively, of the rectangle being packed. Orientation indicates whether the rectangle's length is placed horizontally or vertically. x_l and y_l are the coordinates of the lower-left corner of the rectangle.

Based on the current configuration, every unpacked rectangle can be used to generate a list of COPMs representing the possible corners and orientations for packing them into the container. The invalid COPMs causing overlapping of rectangles or exceeding the boundary will not be included in the COPM list.

This COPM list is firstly sorted in lexicographical order to indicate their priority. The longer side will be the first key for sorting while the shorter side will be considered when resolving a tie. The COPMs are sorted in descending order, which follows the concept of less flexibility first mentioned in Section 3.2 as the longer rectangles are less flexible and

therefore, being assigned with a higher priority in the packing process. Fig. 3.4 shows some candidate COPMs for an object.

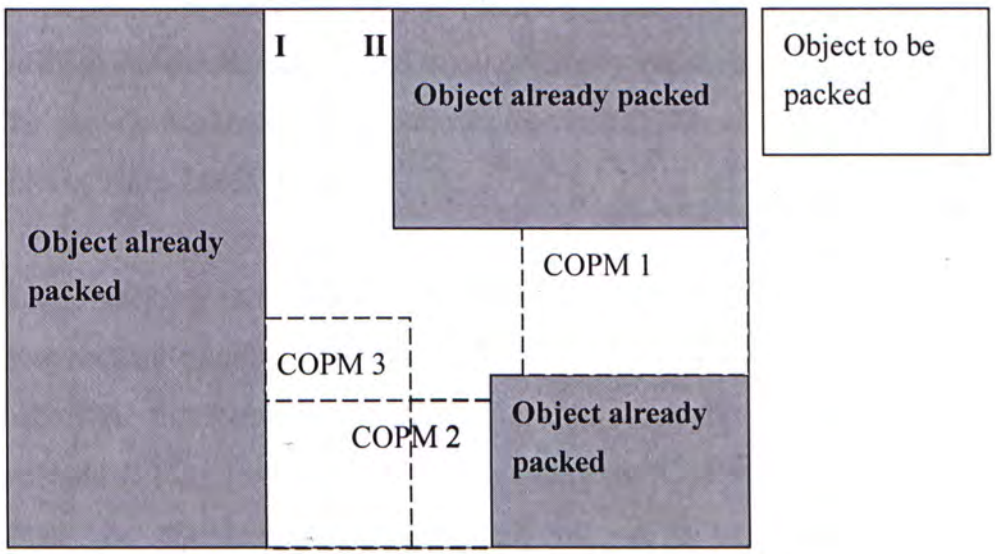


Fig.3.4 Candidate COPM in a certain packing configuration

Corners marked with I and II are invalid for packing the object because such arrangements will result in overlapping. The next step is to evaluate the three COPMs to see which corner is the best to accommodate the object.

The evaluation of a COPM is based on a fitness cost function (FFV). Each COPM is associated with an FFV which indicates the resulting total area of packed space if this COPM is applied. To measure the FFV of COPMs, pseudo-packing and greedy approach should be carried out.

3.3.2 Pseudo-packing and the Greedy Approach

With all the COPMs generated, every rectangle will be pseudo-packed

according to its own COPM list. Pseudo-packing means to temporarily pack a rectangle to a corner indicated in the COPM, do greedy-packing on other rectangles left, measure the fitness cost function (FFV) and remove rectangles from the corner. The pseudo-packing process then repeats by packing the same rectangle to another corner mentioned in next COPM. The pseudo-packing priority follows that in COPM, with the less flexible objects being handled first.

To simplify the problem while giving readers a whole picture of pseudo-packing and greedy approach, we consider a packing scenario with three rectangles only. The dimension of the container is "9m × 6m" and that of the three objects for packing are object A (6m × 4m), B (5m × 4m) and C (5m × 1m). When evaluating the COPM list for the largest object, A, pseudo-packing is carried out. A is firstly pseudo-packed according to one of its COPM:

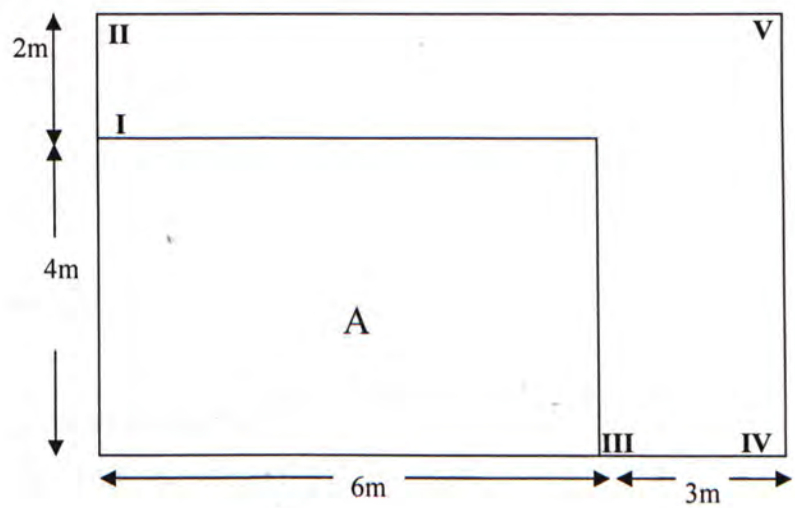


Fig. 3.5 Object A pseudo-pack to its COPM

To calculate the FFV for this COPM of A, greedy packing of the remaining objects is carried out. It means that the objects are packed to the first corner which can accommodate them without violating the three rules

mentioned in the beginning of this section. It is obvious that object B cannot be packed to the container because its width is 4m, which is larger than the widest gap left in the current packing configuration. Object C can in fact be packed to corner I, II, III, IV and V. However, in greedy approach, no selection is required. C will just be packed to the first available corner, i.e. I, without considering other candidates. The result is shown in Fig. 3.6

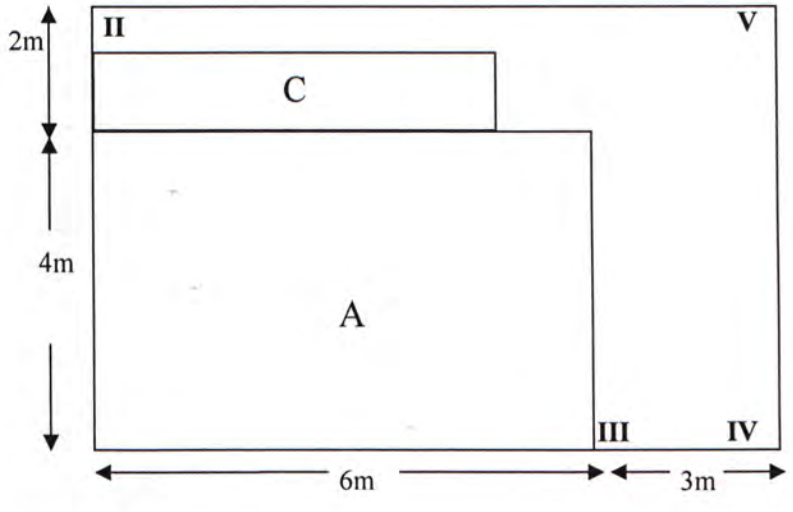


Fig.3.6 Object C pseudo-pack to its COPM

$$FFV(COPM1) = \text{Area of A} + \text{Area of C} = 6m \times 4m + 5m \times 1m = 29\ m^2$$

After the FFV of COPM1 is calculated, the pseudo-packed objects are all removed from the container and the original packing configuration (in this case, an empty container) is restored. The pseudo-packing continues.

The second COPM under consideration is to pack A to the same corner but with another orientation, B and C are then greedily packed to the first available corner. The result is shown in Fig. 3.7

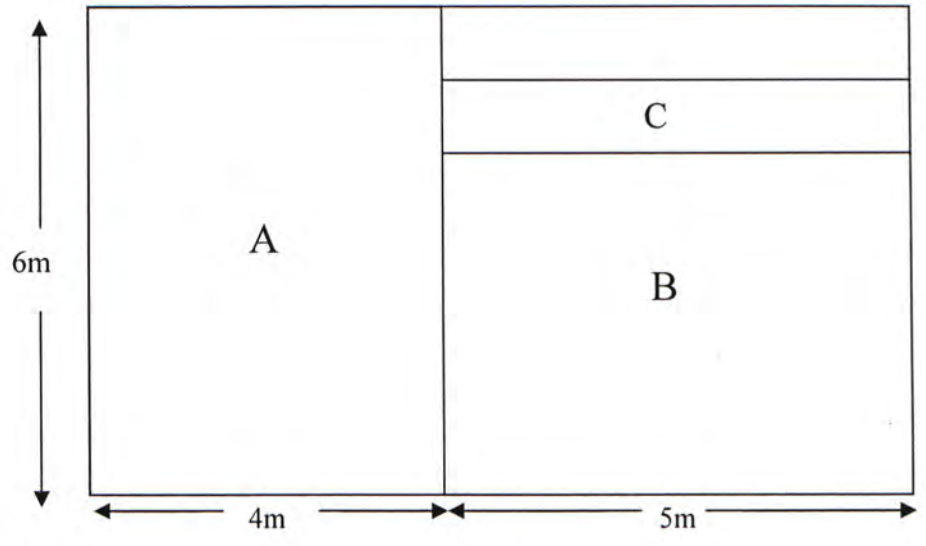


Fig.3.7 Object A, B and C can all be pseudo-packed into the container

$$FFV(COPM2) = 6m \times 4m + 5m \times 4m + 5m \times 1m = 49\ m^2$$

Therefore the FFV of COPM2 is higher than that of COPM1, which implies that COPM2 is a better candidate. In this case, as complete packing is achieved, further pseudo-packing to other COPMs are skipped and the pseudo-packing step stops here. However, in real situation, there are always many objects to be packed. Complete packing is difficult to be achieved. Normally all COPMs for an object will be evaluated by the above steps and the best one is then chosen.

3.3.3 Real-packing

The COPM with the highest FFV will be selected and applied in the real packing step and the COPM list of the next unpacked object will be

processed by repeating steps described in this section.

3.4 Achievement of 2D LFF

According to the experimental results shown in [1], when 2D LFF is run on randomly generated test problems, it found existing optimum solutions for 40% of test cases and the average packing density is around 99%. When packing benchmarks are used to run the experiments, the unpack area ranges from 2% to 0% with the average unpack ratio of 0.92%. The performance is quite consistent, while packing density increases with the number of rectangles.

Due to the effectiveness of 2D LFF proven in the experiments on benchmarks, its potential in producing good results in 3D packing problem is believed to be large. However, LFF may have extremely low performance in particular types of problem instances. The next section focuses on analyzing the worst case and the error bound of LFF algorithm.

Chapter 4

Error Bound Analysis on 2D LFF

Error bound is a kind of performance bound. It can be considered as a measure on the worst case performance of an approximate algorithm. As mentioned in previous sections, LFF is an approximate algorithm, aiming at finding a near-optimal solution, for the packing problem which is NP-complete.

4.1 Definition of Error Bound

When applying an approximation algorithm to an optimization problem, the result is always reflected by a cost, which is calculated in different ways for different problems. For a maximization problem, the optimal solution is the one with the highest cost while the optimal for a minimization problem is the one with the lowest cost.

The solution given by an approximate algorithm is usually near-optimal. Let the cost of optimal solution be C^* and that of near-optimal be C. To assess how good the near-optimal result is, the relative error is calculated.

$$\text{Relative error} = \frac{|C - C^*|}{C^*} \tag{4.1}$$

The larger the relative error, the worst the result is. To evaluate the performance of an algorithm, the worst case analysis is important. Error bound is the relative error when the program processes the worst case. For any input size n, an approximation algorithm has a relative error bound of $\varepsilon(n)$ if

$$\frac{|C - C^*|}{C^*} \leq \varepsilon(n) \tag{4.2}$$

where $\varepsilon(n) < 1$ and it shows dependency on n. Error bound independent of n uses the notation ε.

For the 2D (or 3D) packing problems, the objective is to maximize the packing density (or volume utilization) and they are thus maximization problems. The cost is the occupied area (or volume) in the bounding rectangle (or container). Consider a 2D packing problem. Let A* be the optimal area packed with object while A is the near-optimal area given by LFF. The relative error of this packing algorithm is calculated by:

$$\text{Relative Error} = \frac{A^* - A}{A^*} \tag{4.3}$$

In Section 4.2, some packing scenarios for which LFF generates unsatisfactory results are demonstrated. Section 4.3 is a mathematical proof to show that the solution given by LFF does not have an error bound.

4.2 Cause and Analysis on Unsatisfactory Results by LFF

The principle of LFF algorithm is to pack the less flexible objects to the less flexible space. Logically this works since the less the flexibility of an

object, the higher is the difficulty for it to be packed. Packing them at early stage can increase the chance for them to find a valid location. Yet in some special cases, LFF may give very low packing density. Problem arises when the evaluation of "flexibility" does not reflect the real situation or when a "less flexible object" is packed in such a way that blocks many other objects from being packed. The existing version of LFF decides flexibility of an object by the length of its longest side. The object with its longer side being longest among all other objects is regarded as the least flexible and will be packed first, without taking other factors into account. Consider the following incomplete packing scenario:

Dimensions of Bounding Rectangle = 150 cm×40 cm

Dimensions of Rectangles to be packed:

Rectangles 1 to 3 = 50 cm×40 cm

Rectangle 4 = 111cm×5 cm

The optimal solution is obvious. The packing density is 100%, if Rectangle 4 is left unpacked. Fig. 4.1 shows the optimal packing result with Rectangles 1 to 3 all packed.

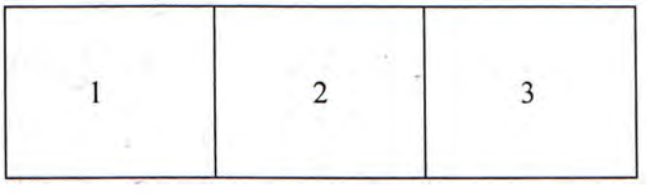
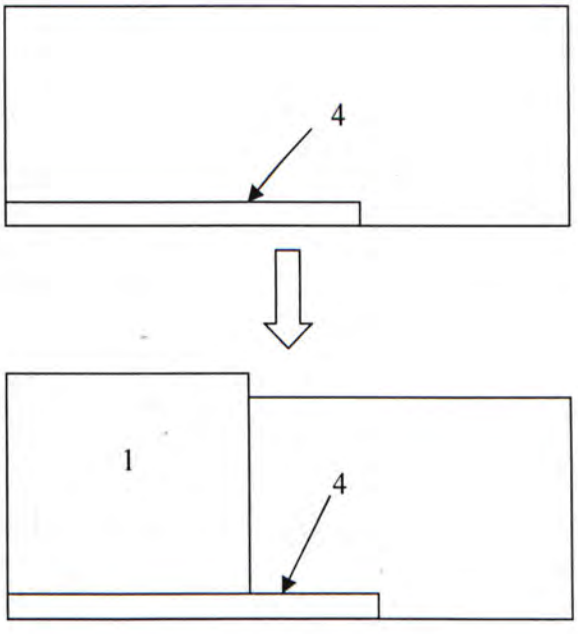


Fig. 4.1 the optimal packing result with Rectangles 1 to 3 all packed

The packed area = $3 \times 50\ cm \times 40\ cm = 6000 cm^2$

According to the LFF algorithm, Rectangle 4 is the least flexible because its longer side is 60 cm while that of others is 50 cm. Pseudo-packing starts with Rectangle 4. Some of the pseudo-packing process is shown in Fig. 4.2



a) Pseudo-pack Rectangle 4 at the lower left corner. It has the longest longer side and is considered as the least flexible

b) Greedily pack Rectangle 1 is illegal because part of Rectangle 1 is outside the bounding rectangle. Rotating Rectangle 1 by 90° results in the same situation

Fig. 4.2 Example of pseudo-packing process

As Rectangles 1 to 3 are of the same dimensions, none of them can be packed after Rectangle 4 is placed at the lower left corner. The next pseudo-packing step will be placing Rectangle 4 at another orientation by rotating it 90° but this again exceeds the boundary of the bounding rectangle. Pseudo-packing Rectangle 4 at the other three corners generates the same result. Therefore, it is packed to the first COPM, i.e., the lower left corner. None of the other three rectangles can be packed. The packed area is the area of Rectangle 4, i.e., 111 cm x 5cm = 555 cm^2

$$\text{Relative error} = \frac{(6000 - 555)}{6000} = 0.9075$$

In this case, the rectangle that should be left unpacked is chosen to be processed at the first place. The packing order according to "flexibility" becomes the cause of the large relative error. In fact, the area occupied by Rectangle 1 is much larger than Rectangle 4. If area is regarded as the criteria for assessing flexibility, the packing density here will become 100%. Therefore, assessing "flexibility" carefully by considering different factors instead of only length of the longer side may lead to better results.

Low packing density may also occur in complete packing. In complete packing, all objects can be packed inside the container when optimal solution is achieved. However, LFF may generate a solution far from optimal by packing an object at a place which blocks other objects from being packed. Consider this example:

Dimensions of Bounding Rectangle = 25 cm × 20 cm

Dimensions of Rectangles to be packed:

Rectangles 1 = 19 cm × 19 cm

Rectangles 2 to 4 = 20 cm × 2 cm

The optimal solution is shown in Fig. 4.3.

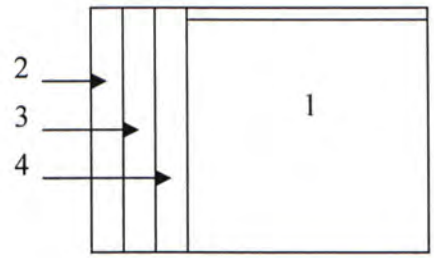
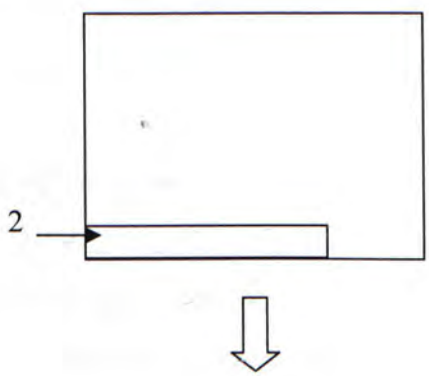


Fig. 4.3 The optimal solution

$$\text{Packed area} = 20\ \text{cm} \times 2\ \text{cm} \times 3 + 19\text{cm} \times 19\ \text{cm} = 481\ \text{cm}^2$$

$$\text{Packing Density} = = \frac{481}{500} \times 100\% = 96.2\%$$

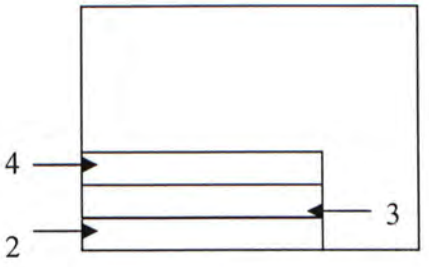
When this packing scenario is processed by LFF, Rectangles 2 to 4 will be packed first as they are "less flexible". In pseudo-packing and greedy packing of LFF, the horizontal placement[2] will be tried first. For greedy packing, if horizontal placement is legal, vertical placement[3] will not be considered. Some of the steps of LFF are shown in Fig. 4.4



a) Pseudo-pack Rectangle 2 to lower-left corner

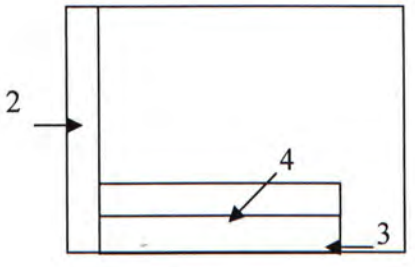
[2] Horizontal Placement means that the longer side of the rectangle is placed horizontally while the shorter side is placed vertically

[3] Vertical placement means that the shorter side of the rectangle is placed horizontally while the longer side is placed vertically



b) Greedily pack other rectangles until no more can be packed. Now Rectangles 2 to 4 are pseudo-packed. The packed area is 20 cm x 2cm x 3 = 120 cm^2

Some steps are omitted



c) Rectangle 2 is then packed to lower-left corner by rotating 90°. Rectangles 3 and 4 are greedily packed. Rectangle 1 cannot be packed. The packed area is still 120 cm^2

Fig. 4.4 Sample steps of LFF

The pseudo-packing continues by placing Rectangle 2 to the other three corners with these two orientations. The total packed area after greedy packing is also 120 cm^2 in each case. The first COPM is chosen for real-packing.

After packing Rectangle 2 at the first COPM, the algorithm continues to perform pseudo-packing until no more rectangles can be packed. In fact, it is obvious that Rectangle 1 cannot be packed into any corners in the bounding rectangle. Since there is not enough space for accommodating Rectangle 1, the maximum packed area will be 120 cm^2. Packing density is $\frac{120}{500} \times 100\% = 24\%$, which is very low.

$$\text{Relative error} = \frac{481-120}{481} = 0.751, \text{ to 3 sig. fig.} \qquad (4.4)$$

If the second COPM of Rectangle 2 is chosen for real-packing, it can be possible for optimal solution to be achieved. The reason for discarding this COPM is because it generates the same packed area as the first COPM after greedy packing of other rectangles. According to LFF, the choice sticks to the first. Yet the second COPM can in fact lead to the optimal solution. The problem here is on greedy packing, which proceeds to another object immediately after the current one being packed find a place. If greedy packing packs the current object to more than one COPM, say, instead of greedily pack Rectangle 3 as shown in Fig. 4.4 and proceed to pack Rectangle 4, rotate Rectangle 3 by 90° as the second step of greedy pack and assess the two greedy pack options by tightness measure [2], it is possible to obtain the optimal solution as the second COPM will be selected for Rectangle 2. However, in the original LFF, this situation is not handled and may lead to large relative errors.

From the above two scenarios, it is discovered that when there are long but narrow rectangles, i.e., large length/width ratio, the risk of getting large error bound increases.

4.3 Formal Proof on Error Bound

As mentioned in Section 4.1, relative error bound is the maximum relative error of the result produced by an algorithm in the worst case. From the definition of relative error, we can deduce the situation in which LFF generates the largest relative error.

In most cases, packing algorithm deals with complete packing scenarios rather than incomplete packing scenarios. We will try to calculate the error bound for complete packing.

In the worst case, the relative error is at maximum. By equation (4.4), to simulate the worst case, the relative error should be the largest, i.e., when A^* is set to be as large as possible while A is set to be as small as possible. In other words, the cases with the largest wasted space should be considered.

To simplify the case, assume there are two types of rectangles, R_1 and R_2, only. Let length and width of container be L and W respectively, length and width of R_1 be l_{R1} and w_{R1} respectively, length and width of R_2 be l_{R2} and w_{R2} respectively, where $L > W$, $l_{R1} > w_{R1}$ and $l_{R2} > w_{R2}$

By investigating the properties of the second example shown in Section 4.2, we have come up with a series of deductions shown below which can gradually lead to a simulation of the general worst case for LFF. Note that the following steps always have two goals: to maximize the optimal packed area (A^*) and minimize the packed area (A) achieved by LFF.

Step 1: Deduce the dimensions for the rectangles being packed:

a. There should be rectangles with very large length/width ratio. These rectangles will be referred to as R_1. The large length ensures that such rectangles are pseudo-packed before other rectangles. This increases their chances to block other rectangles from being packed and reduces A. l_{R1} should be maximized. The short width ensures that their area is small so that A is further reduced. w_{R1} should be minimized.

b. With reference to the complete packing scenario in Section 4.2,

R_1 should not be packed with length placed vertically but should be placed in another orientation so that they prevent a large area from being filled with other rectangle. By Step 1a), l_{R1} must be as large as possible, provided that they can be placed with their longer side along the width of the container to generate optimal solution:

$$l_{R1} = W \tag{4.5}$$

c. There should be rectangles with very large area but with their length slightly shorter than R_1 to ensure that they will be processed after all R_1 are packed. These rectangles will be referred to as R_2. They should have their length and width as large as possible to ensure that they will be blocked by R_1. In fact, the larger the length and width, the larger is the area. If these rectangles cannot be packed, its large area will result in small A. To maximize l_{R2} and w_{R2}:

$$l_{R2} = w_{R2} = l_{R1} - 1 \tag{4.6}$$

Step 2: Deduce the number of rectangles in each type

It is better to have one R_2 only. If multiple rectangles are present, each of these rectangles will be smaller in area and their flexibility will increase. This means that it is easier for each of them to be packed and the wasted area will be reduced. The relative error will become smaller.

$$\text{No. of } R_1 = n - 1$$
$$\text{No. of } R_2 = 1 \qquad (4.7)$$

Step 3: Some constraint to minimize packed area by LFF

a. To ensure that the area "above" or "below" R_1 cannot be filled by the R_2, the following must be satisfied:

$$w_{R1} + l_{R2} \geq w_{R1} + w_{R2} > W \qquad (4.8)$$

b. Step 1a) states that w_{R1} should be minimized. According to (4.5) and (4.7), the minimum value of w_{R1} should be 2.

$$w_{R2} = 2 \qquad (4.9)$$

c. To ensure horizontal orientation of R_1 disables R_2 to be packed on their "left" or "right":

$$l_{R1} + l_{R2} \geq l_{R1} + w_{R2} > L \qquad (4.10)$$

Step 4: Deduce the dimensions of the container to maximize optimal packed area

The packed area in optimal solution should be maximized by occupying as large area in bounding rectangle as possible. L and W should be set to values that can just accommodate all rectangles.

According to the above deductions, the maximized optimal solution with input size n is shown in Fig. 4.5:

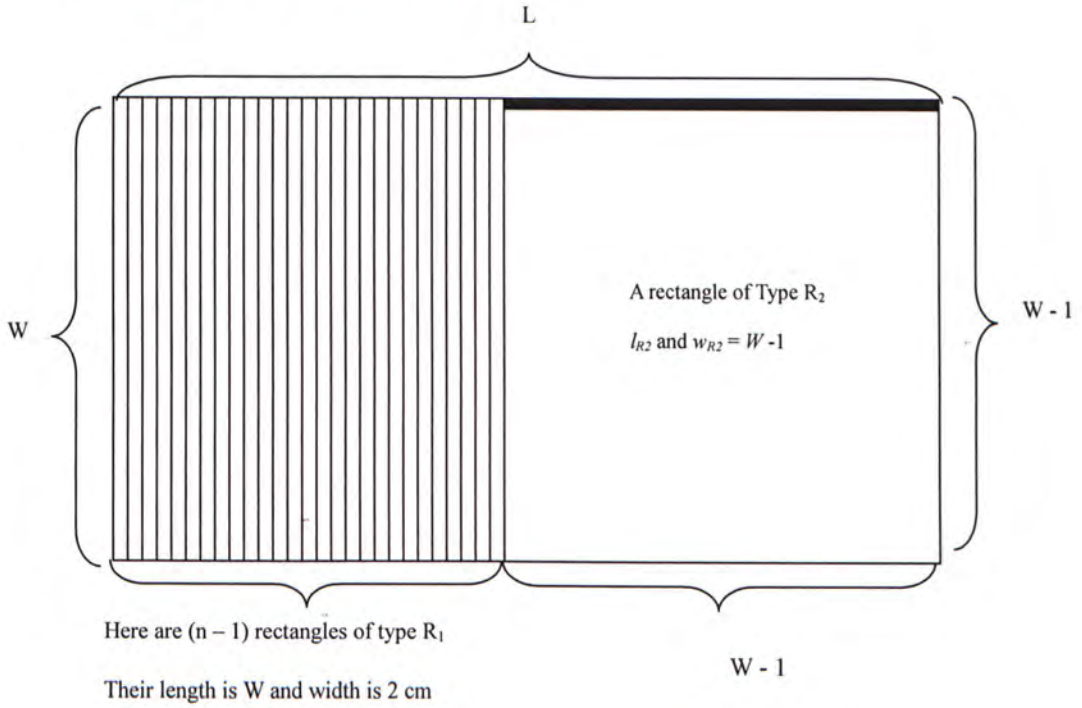


Fig. 4.5 the maximized optimal solution with input size n

The area filled with color is left empty.

As shown in Fig. 4.5,

$$L = 2(n-1) + (W-1) \tag{4.11}$$

Step 5: Deduce the maximized optimal packed area

a. By (4.7), to maximize relative error for input size n [4], the number of R_1 should be $(n-1)$ and the number of R_2 should be 1.

[4] Input size n means there are n rectangles to be packed. Therefore, $n > 0$

$$A^* = l_{R1} \times w_{R1} \times (n\text{-}1) + l_{R2} \times w_{R2} \times 1 \tag{4.12}$$

b. By (4.5), $l_{R1} = W$ and by (4.9), $w_{R1} = 2$, therefore,

$$A^* = W \times 2 \times (n\text{-}1) + l_{R2} \times w_{R2} \tag{4.13}$$

By (4.6), $l_{R2} = w_{R2} = l_{R1} - 1$, with (4.5), $l_{R2} = w_{R2} = W\text{-}1$

$$A^* = W \times 2 \times (n\text{-}1) + (W-1)^2 \tag{4.14}$$

Step 6: Deduce the minimized packed area by LFF

Consider the packed area A generated by LFF. Since horizontal placement is chosen for the first rectangle, by (4.8) and (4.10), it is certain that the R_2 rectangle cannot be packed. For large W, all R_1 can be packed, therefore,

$$\begin{aligned} A &= l_{R1} \times w_{R1} \times (n\text{-}1) \\ &= W \times 2 \times (n\text{-}1) \end{aligned} \tag{4.15}$$

By (4.3), relative error e(n) in this worst case:

$$\begin{aligned} \varepsilon(n) &= \frac{A^* - A}{A^*} \\ &= \frac{W \times 2 \times (n-1) + (W-1)^2 - W \times 2 \times (n-1)}{W \times 2 \times (n-1) + (W-1)^2} \\ &= \frac{(W-1)^2}{W \times 2 \times (n-1) + (W-1)^2} \end{aligned} \tag{4.16}$$

Step 7: Proof for the inexistence of error bound

For any given value of n, which must be positive, $e(n)$ increases with W. By (4.11), L increases with W and $L > W$. Thus the R_1 can always be packed in the worst case at horizontal position and equation (4.15) and (4.16) always hold.

By the definition of Error Bound of an approximate algorithm, for any input size n, the error bound $\varepsilon(n)$ is always larger than or equal to the relative error $e(n)$ produced by the algorithm, i.e.,

$$\varepsilon(n) = \max. e(n) \qquad (4.17)$$

As $e(n)$ increases with W, maximum $e(n)$ can be achieved when W is maximum. To prove whether any error bound exists, we attempt to find the limit of the error bound deduced in (4.16) on next page.

$$
\begin{aligned}
\varepsilon(n) &= \lim_{W\to\infty} \frac{(W-1)^2}{[W\times 2\times(n-1)+(W-1)^2]} \\
&= \lim_{W\to\infty} \frac{1}{\frac{2W(n-1)+(W-1)^2}{(W-1)^2}} \\
&= \lim_{W\to\infty} \frac{W^2-2W+1}{2W(n-1)+W^2-2W+1} \\
&= \lim_{W\to\infty} \frac{\frac{W^2-2W+1}{W^2}}{\frac{2W(n-1)+W^2-2W+1}{W^2}} \\
&= \lim_{W\to\infty} \frac{1-\frac{2}{W}+\frac{1}{W^2}}{\frac{2(n-1)}{W}+1-\frac{2}{W}+\frac{1}{W^2}} \\
&= \frac{1+0+0}{0+1+0} \\
&= 1
\end{aligned}
\qquad (4.18)
$$

It is impossible for relative error to be 1 because there will surely be at least one rectangle packed, no matter how worst the case is. A cannot be 0 and relative error cannot be 1. However, when W increases and approaches infinity, the relative error increases and approaches 1. Therefore in LFF, error bound does not exist for any input size n. This can be regarded as a theorem which will be a useful reference for future research.

Theorem:

No error bound exists when applying LFF to 2D packing problems of any input size n

Chapter 5

LFF for Container Loading Problem

Although the performance of LFF in worst case can be very unsatisfactory, on average it works very well. It consistently produces about 99% packing densities on most randomly generated large examples. As its application on 2D packing gives encouraging results, its usage is extended to 3D packing problems.

As mentioned in Section 1, based on "objective function classification", there are three types of 3D packing cases. Our research is focused on the container loading problem, aiming at maximizing the volume utilization of the container. This is proven to be NP-complete [15]. No existing algorithms are able to give optimal solutions in polynomial time. Heuristics are the mostly adopted approaches for this kind of problems.

According to the Literature Review in Section 2, it is not difficult to discover that some researchers [24] develop their 3D packing algorithms based on some 2D packing algorithms with satisfactory results. The algorithm we are going to introduce is also an extension from 2D to 3D.

LFF is a heuristic for solving packing problems by the principle "packing the less flexible object to the less flexible space". Its success in 2D is the reason for implementing it as a container loading algorithm. It is believed that its extension to 3D packing can produce promising volume utilization. Although 2D and 3D packing problems are similar in nature,

3D cases are more complicated due to the larger number of possibilities. The number of COPMs for each object is much larger than that in 2D since there are much more corners in a 3D space and for each corner, six possible orientations can be considered. The container loading problem is formulated in Section 5.1.

5.1 Problem Formulation and Term Definitions

The problem being studied involves the packing of a subset of boxes into a single container with fixed dimensions. The objective is to maximize the volume utilization of the container. Given a set of n rectangular-shaped boxes $\{b_1, b_2, b_3, \ldots, b_n\}$, with known dimensions $l_i \times w_i \times h_i$, where $l_i \geq w_i \geq h_i$, for the i^{th} box, and a single rectangular-shaped container B with fixed dimensions L x W x H, a subset of the boxes should be chosen and packed orthogonally and entirely into the container. Orientation constraints and stability constraints should be taken into consideration.

Orientation defines horizontal or vertical placement of the box's surfaces. It states which sides of the box are placed along x-dimension, y-dimension and z-dimension. Orientation constraints states that some sides of the box cannot be placed vertically. This restricts the rotation of the box and reduces the number of possible packing position.

Stability constraints require every packed box to be supported by the container base or by another box underneath. The supporting material must ensure that the box on the surface is supported in a stable and balanced manner, i.e., will not fall off. This constraint is normally handled in two ways. The first way is to ensure that the ratio of supported surface area to the total surface area of the upper box exceeds a predefined

value, i.e., the area supported by underneath layer / the total base area ≥ a predefined value ($\frac{A_s}{A_t} \geq R$), usually 0.5 is taken as a default value R if no value is specified by users. Another way is to fill all empty space with foam rubber to ensure a proper support of boxes on top. The latter way is a simpler one as the computation time for area ratio calculation can be saved. In our research, the space-filling approach is used for handling stability constraints.

Volume utilization is defined as the percentage of occupied volume in the container. It can be calculated by: Volume of occupied space / total volume of the container ($\frac{V_o}{V_t}$).

The core idea of LFF is "flexibility". The flexibility of objects and flexibility of space are taken into account. The definitions of these two "flexibility" are described in Section 3.1.

A corner C in a 3D space is defined in Fig. 5.1a) to Fig. 5.1d). It shows the top view (on x - y plane) of the packing configurations. Consider the points pointed by arrows. Draw a cross on the point to form four regions I, II, II and IV as shown. The shaded regions are occupied by packed objects while the white region is empty space. A point is a corner when three criteria are satisfied:

(i) Any three out of the four regions are occupied by other objects (or the boundary of the container). Four types of corners are shown.

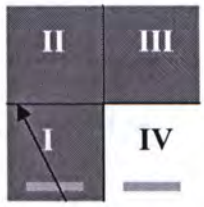


Fig. 5.1a) An upper left corner

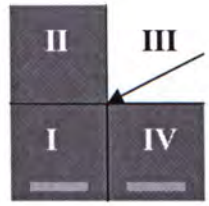


Fig. 5.1b) A lower left corner

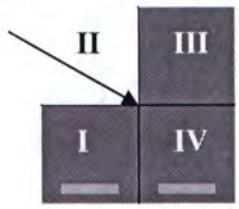


Fig. 5.1c) A lower right corner

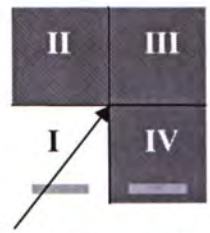


Fig. 5.1d) An upper right corner

(ii) The corner points must lie on the container base or on the upper surface of another object. These corner points are for bottom-up packing. In this research, as foam rubber is used for supporting purpose, corner points lying below the container roof or below lower surface of another object are also considered. This direction of packing is top-down packing.

(iii) In the height dimension, the upper surface of the three surrounded objects must be higher than the corner point, unless the corner is bounded by container walls when considering bottom-up packing. Fig.5.2a) shows a corner of this type. For top-down packing, the lower surface of the three surrounded objects must be lower than the corner point, unless the corner is bounded by container walls. An example is shown in Fig 5.2b).

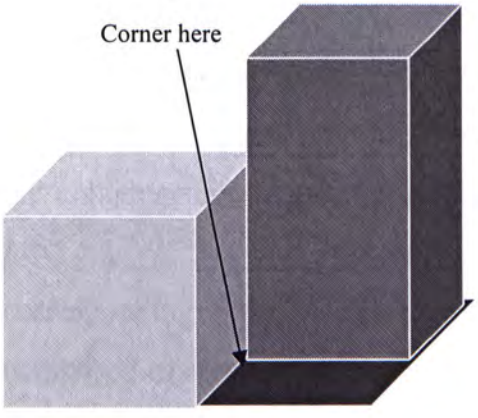


Fig 5.2a) a corner for bottom-up packing (side view)

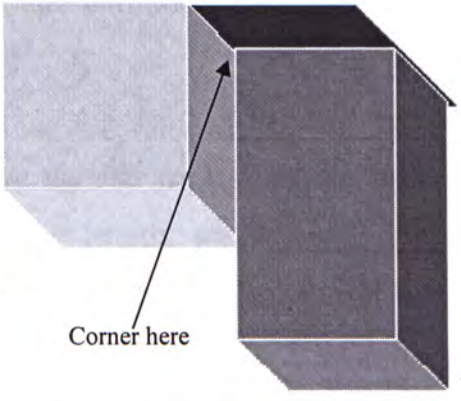


Fig 5.2b) a corner for top-down packing (side view)

The representation of an object in a container in the 3D space is by six keys in a 3D coordinate system. The coordinates of its lower-left point nearer to the origin of the coordinate system forms the first three keys (x_1, y_1, z_1) while the coordinates of its upper-right point farther from the origin forms the last three keys (x_2, y_2, z_2). The exact region occupied by the object can now be determined by these keys. A box with keys (0, 0, 0, 3, 2, 5) in the coordinate system is illustrated in Fig. 5.3.

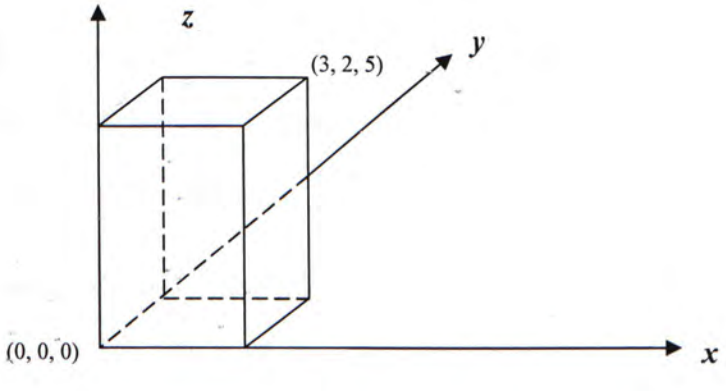


Fig. 5.3 The representation of an object in a 3D coordinate system

Following the object representation and definitions, the problem can also be formulated as follows.

Given a triple of coordinates $\{x_{B1}, y_{B1}, z_{B1}, x_{B2}, y_{B2}, z_{B2}\}$ representing the container and a set of n positive number triples $<l_1, w_1, h_1>$, $<l_2, w_2, h_2>$, ..., $<l_n, w_n, h_n>$ representing the length, width and height of the corresponding boxes to be packed, the objective is to find a solution composed of n sets of tuples $\{x_{11}, y_{11}, z_{11}, x_{12}, y_{12}, z_{12}\}$, ..., $\{x_{n1}, y_{n1}, z_{n1}, x_{n2}, y_{n2}, z_{n2}\}$ where $x_{i1} < x_{i2}$ and $y_{i1} < y_{i2}$ and $z_{i1} < z_{i2}$ for all $1 \leq i \leq n$ while the following conditions should be satisfied:

(i) According to the representation of the object, there are six possible orientations:

- $[(x_{i2} - x_{i1}) = l_i$ and $(y_{i2} - y_{i1}) = w_i$ and $(z_{i2} - z_{i1}) = h_i]$
- $[(x_{i2} - x_{i1}) = w_i$ and $(y_{i2} - y_{i1}) = l_i$ and $(z_{i2} - z_{i1}) = h_i]$
- $[(x_{i2} - x_{i1}) = l_i$ and $(y_{i2} - y_{i1}) = h_i$ and $(z_{i2} - z_{i1}) = w_i]$
- $[(x_{i2} - x_{i1}) = h_i$ and $(y_{i2} - y_{i1}) = l_i$ and $(z_{i2} - z_{i1}) = w_i]$
- $[(x_{i2} - x_{i1}) = w_i$ and $(y_{i2} - y_{i1}) = h_i$ and $(z_{i2} - z_{i1}) = l_i]$
- $[(x_{i2} - x_{i1}) = h_i$ and $(y_{i2} - y_{i1}) = w_i$ and $(z_{i2} - z_{i1}) = l_i]$

(ii) To ensure no overlapping of objects occur, for all $1 \leq i, j \leq n$, at least one of the following six conditions must be met: $x_{j1} \geq x_{i2}$, $x_{j2} \geq x_{i1}$, $y_{j1} \geq y_{i2}$, $y_{i1} \geq y_{j2}$, $z_{j1} \geq z_{i2}$, $z_{i1} \geq z_{j2}$,

(iii) To ensure all the boxes are completely inside the container, for all $1 \leq i \leq n$, $x_{B1} \leq x_{i1}$, $x_{i2} \leq x_{B2}$, $y_{B1} \leq y_{i1}$, $y_{i2} \leq y_{B2}$, $z_{B1} \leq z_{i1}$, $z_{i2} \leq z_{B2}$,

If such a solution cannot be obtained, a subset of the given boxes is chosen and packed in a way that minimizes the volume of wasted space, i.e. maximizes the volume utilization.

The above definitions and notations are used throughout this section unless specified.

5.2 Possible Problems to be solved

In our research, the first step is to analyse the 2D LFF algorithm to see if it is suitable for being extended to 3D. Possible difficulties include:

- Region query: As the scenarios of overlapping of 2D items are much fewer than 3D items, this research should involve an analysis on how to detect overlapping of objects in 3D. The region queries are performed by using the data structure K-D tree [35] which can effectively represent, update and query for the current packing configuration.
- Corner list maintenance: For 3D packing problem, the number of corners generated after each packing step will be much more than that in 2D. If corner formation is by testing for conditions of each possible scenario, the corner list maintenance will take a very long time to process the current packing configuration. The corner detection method must be efficient and no corners should be missing during corner update procedures. The method for detection of corner is developed based on the corner definition shown in Section 5.1. For each potential corner point, we test for all rules of a corner to see

if any of the rules cannot be satisfied. A corner is inserted to corner list only when all the conditions of a corner are satisfied.

- Substantial increase in the number of COPMs during pseudo-packing procedure should be handled with care to prevent errors when the program is running.
- The long running time due to the great increase in pseudo-packing steps should be cut down, if possible. This problem is handled in the development of Less Flexibility First with Tightness Measure (LFFT) algorithm.

5.3 Implementation in Container Loading

The LFF for the container loading problem involves 4 major steps:

(i) COPM generation
(ii) Pseudo-packing and greedy packing
(iii) Real packing

The details of the steps will be discussed in this section. To illustrate the complete flow of the LFF algorithm, we first visualize the flow in Fig 5.4.

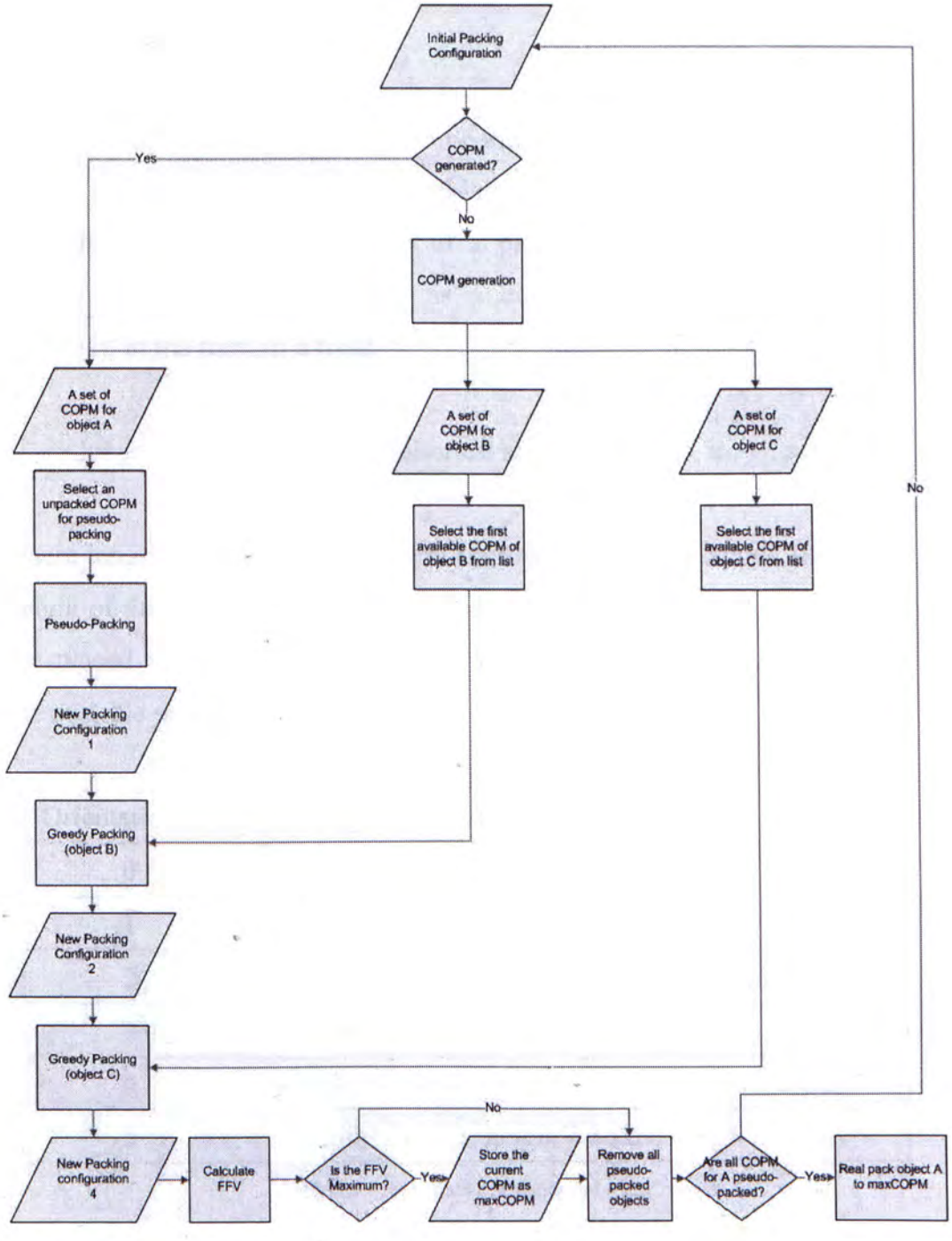


Fig 5.4 A flowchart showing the flow of LFF algorithm

5.3.1 The Basic Algorithm

For each unpacked object, there can be more than one candidate corners to which it can be packed, without violating the three conditions stated in Section 5.1. The representation of a packing relationship between an object and a corner is defined as a corner-occupying packing move (COPM), in the form of a tuple

<longest side, medium side, shortest side, orientation, x_1, y_1, z_1>

where longest side, medium side and shortest side are the length, width and height of the box being packed. Orientation states which sides of the box are placed along x-dimension, y-dimension and z-dimension. There are six possible orientations:

Orientation ID	Actual Placement
0	<width // x-axis, length // y-axis, height // z-axis>
1	<height // x-axis, length // y-axis, width // z-axis>
2	<length // x-axis, width // y-axis, height // z-axis>
3	<height // x-axis, width // y-axis, length // z-axis>
4	<length // x-axis, height // y-axis, width // z-axis>
5	<width // x-axis, height// y-axis, length // z-axis>

Table 5.1 Six possible orientations of a box

Fig. 5.4 shows the six possible orientations of a box. S1 here represents the "width-height" surface; S2 represents the "length-height" surface while S3 represents the "length-width" surface. Fig.5.4a) shows the case with

orientation ID = 0; Fig.5.4b) shows the case with orientation ID = 1; Fig.5.4c) shows the case with orientation ID = 2 and so on. Fig.5.5 is shown on next page.

Note that x_1, y_1, z_1 are the coordinates of the object's lower-left corner nearer to origin, which are the first three keys representing this candidate location. With the given dimensions, the last three keys can easily be calculated.

Given a current packing configuration in which all boxes are packed at fixed locations, the first step of LFF algorithm is to generate a list of COPMs representing all valid candidate packing positions for all unpacked boxes. Consider packing an object which does not have any orientation constraints, i.e. six orientations are possible. For a packing configuration with 6 corners, a maximum of $6 \times 6 = 36$ COPMs can be generated, assume all of them obey the conditions stated in Section 3. Section 5.4 will present a detailed example of COPM generation process.

The generated COPM list, containing all valid COPMs for all unpacked objects, is sorted in descending order according to the first three members of the tuple (the sorting consider longest side first and use medium side and shortest side to solve a tie). This order is significant as it places the less flexible objects (longer and larger) in higher ranks of the list and this list is processed from top to bottom in order. This ensures that less flexible objects are packed first, which is the principle of LFF.

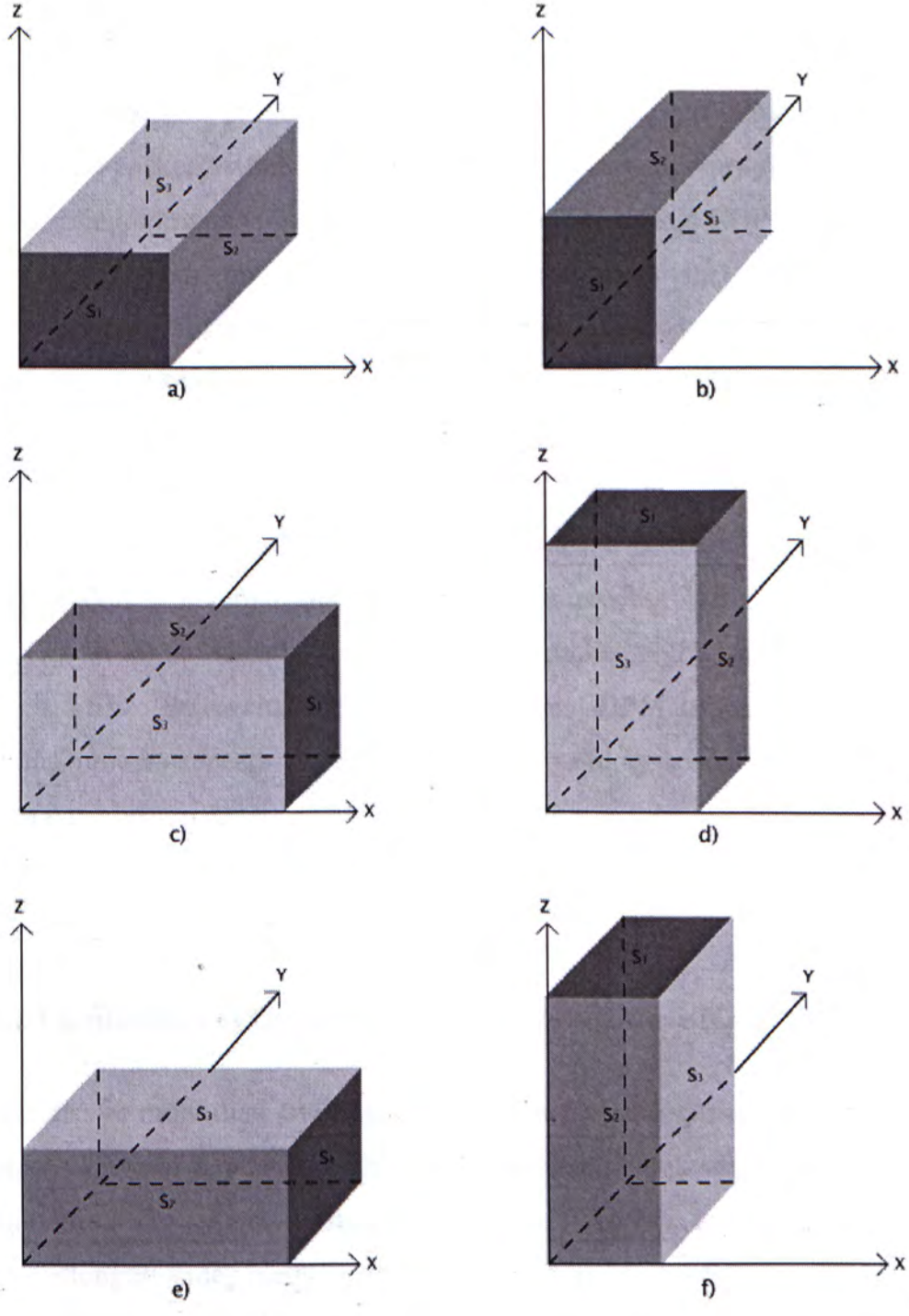


Fig.5.5 The six possible orientations of a box

A fitness cost function (FFV) is calculated and associated to each COPM in the list to determine which candidate is the best one. This FFV is calculated through a procedure called "Pseudo-Pack". A COPM is said to be "pseudo-packed" when the corresponding box is temporarily packed to the specified corner for the assessment of the FFV of this COPM. After FFV is obtained, the box can be removed from that corner and pseudo-packing of the next COPM is performed. In each pseudo-packing step, the left-over boxes (excluding the pseudo-packed one) are pseudo-packed to the first available corner greedily, following the COPM list order until no further packing is possible. The FFV for this COPM candidate is the total volume of all pseudo-packed boxes. The packing configuration is reverted and the pseudo-pack process continues for the next COPM for the same box, which is finally packed to the corner with the highest FFV. By evaluating the FFV of the COPM of each box, the packing process repeats until no large enough empty space is left or all boxes have been packed. The volume utilization is then calculated by the V_o / V_t.

5.3.1.1 Generation of Corner Occupying Packing Move (COPM)

There can be more than one candidate corners for an unpacked box. The representation of a packing relationship between a box and a corner is defined as a corner occupying packing move (COPM), in the form of a tuple: <longest side, medium side, shortest side, orientation, x_l, y_l, z_l> where longest side, medium side and shortest side are the dimensions of the box being packed.

Orientation states which sides of the box are placed along x, y and

z-dimensions. It can be set to 0-5. Please refer to the earlier part of this Section for the placement of each orientation. x_l, y_l, z_l are the coordinates of the box's lower-front-left corner in a 3D coordinate system representing the container.

Given a current packing configuration in which all boxes are packed at fixed locations, the first step of LFF algorithm is to generate a COPM list representing all valid candidate packing positions for all unpacked boxes. When packing a box to a corner, without orientation constraints, six orientations are possible. For a packing configuration with six corners, a maximum of $6 \times 6 = 36$ COPMs can be generated. This COPM list is sorted in ascending order of flexibility in the way discussed in Section 3. As this list is processed from top to bottom, less flexible boxes are packed first, which obeys the principle of LFF. Note that as the longest side is the first parameter for sorting, the object with the "longest longest side" will be considered as the least flexible.

5.3.1.2 Pseudo-Packing and the Greedy Approach

Each COPM for an unpacked box is then evaluated to choose the best as the final packing position. COPMs that cause overlapping of adjacent boxes or exceeding container boundary are deleted before evaluation takes place.

"Pseudo-Packing" means placing a box temporarily to a location specified by a COPM. In each pseudo-packing process, the least flexible unpacked box b_i is pseudo-packed to one COPM in the list. A Fitness Cost Function Value (FFV) is associated with every COPM of b_i for assessing the suitability of that COPM. The boxes left unpacked are

"pseudo-packed" to the first available corner in the current packing configuration greedily, without violating the criteria stated in Section 5.1, until no valid corners can be found for any of the unpacked boxes. The volume utilization ($\frac{V_o}{V_t}$) is the FFV of that COPM of b_i. Then all "pseudo-packed" boxes, including b_i, are removed from the container and pseudo-packing continues to evaluate other COPMs of b_i.

5.3.1.3 Update of Corner List

Coordinates of corners are stored in a list for generation of COPMs. After "pseudo-packing" a box, at least one existing corner is occupied and some new corners are produced. The corner list must be updated before pseudo-packing the remaining boxes. Occupied corners are deleted while new corners are inserted.

5.3.1.4 Real-Packing

After all COPMs of a box are evaluated, the box will be "real-pack" to the COPM with the highest FFV. The corner list is updated and the COPMs of the next unpacked box will undergo pseudo-packing. The process continues until no boxes can be packed to any corners. The LFF algorithm is shown in Fig. 5.6.

LFF algorithm for container loading:

Starting with a empty container

1. Based on the current packing configuration, find all possible COPMs for each unpacked box; represent each COPM by a tuple <longest side, medium side, shortest side, orientation, x_l, y_l, z_l>.
2. Sort all of these tuples according to their flexibility.
3. For each candidate COPM, do 3.1 to 3.3 to find its fitness function value (FFV).
   3.1 Pseudo-pack this COPM.
   3.2 Pseudo-pack all the remaining boxes based on the current COPM list and with a greedy approach, until no more COPM can be packed.
   3,3 Calculate FFV of this candidate COPM as the occupied volume

   .

   Note: Before the pseudo-packing for the next candidate COPM is evaluated, the previously pseudo-packed COPMs must be removed.

4. Pick the candidate COPM with the highest FFV and really pack the corresponding box according to the COPM. The corner list is updated for later packing procedures.

Fig 5.6 The LFF Algorithm

5.4 A Sample Packing Scenario

This example illustrates part of the packing process of a container loading problem. For the sake of simplicity, only three boxes and a small portion of steps are considered. Orientation constraints are not applicable.

The dimensions of the bounding box (the container) B is <20m × 15m × 10m>. The three boxes to be packed are with dimensions b_1 <8m × 14m × 6m>, b_2<12m × 6m × 4m> and b_3 <9m × 7m × 9m>.

The packing process is assumed to be started with the empty container. Before generating the COPM list, we should find out all of the available corners. Fig. 5.7 shows the empty container and the corners. The four corners on the base of the container are valid because boxes packing at these corners can satisfy the four conditions stated in Section 5.1. The four corners at the top of the container will be excluded from consideration when dealing with the effect of gravity. However, as our algorithm makes use of rubber foam to fill up all spaces, we need not exclude these corners. In this case, all corners are valid.

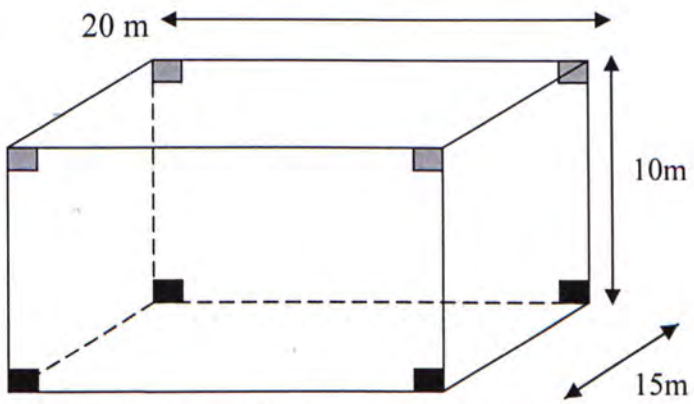


Fig. 5.7 Corners in the container (Black ones are valid, grey ones are invalid)

5.4.1 Generation of COPM list

The six keys representing the container are (0, 0, 0, 20, 15, 10). Initial corners are I(0, 0, 0), II(20, 0, 0), III(20, 15, 0), IV(0, 15, 0), V(0, 0, 10), VI(20, 0, 10), VII(20, 15, 10) and VIII(0, 15, 10). The dimensions of the three boxes to be packed should be rearranged in descending order, i.e. <14m × 8m × 6m>, <12m × 6m × 4m> and <9m × 9m × 7m>. The COPM list is as follows (the base dimensions are written in the form

<x × y>):

(14, 8, 6, 0, 0, 0, 0) – box b_1 packed at corner I with surface <14m × 8m> as the base

(14, 8, 6, 1, 0, 0, 0) – box b_1 packed at corner I with surface <8m × 14m> as the base

(14, 8, 6, 2, 0, 0, 0) – box b_1 packed at corner I with surface <14m × 6m> as the base

(14, 8, 6, 3, 0, 0, 0) – box b_1 packed at corner I with surface <6m × 14m> as the base

As the side 14m cannot be placed vertically (otherwise it exceeds the container boundary), only four orientations are possible here

(14, 8, 6, 0, 6, 0, 0) – box b_1 packed at corner II with surface <14m × 8m> as the base

(14, 8, 6, 1, 12, 0, 0) – box b_1 packed at corner II with surface <8m × 14m> as the base

(14, 8, 6, 2, 6, 0, 0) – box b_1 packed at corner II with surface <14m × 6m> as the base

(14, 8, 6, 3, 14, 0, 0) – box b_1 packed at corner II with surface <6m × 14m> as the base

(14, 8, 6, 0, 6, 7, 0) – box b_1 packed at corner III with surface <14m × 8m> as the base

(14, 8, 6, 1, 12, 1, 0) – box b_1 packed at corner III with surface <8m × 14m> as the base

… omitted COPM for box b_1

(12, 6, 4, 0, 0, 0, 0) – box b_2 packed at corner I with surface <12m × 6m> as the base

(12, 6, 4, 1, 0, 0, 0) – box b_2 packed at corner I with surface <6m × 12m> as the base

… omitted COPM for box b_2 at corner I

(12, 6, 4, 0, 8, 0, 0) – box b_2 packed at corner II with surface <12m × 6m> as the base

(12, 6, 4, 1, 14, 0, 0) – box b_2 packed at corner II with surface <6m × 12m> as the base

…

(12, 6, 4, 0, 8, 9, 0) – box b_2 packed at corner III with surface <12m × 6m> as the base

(12, 6, 4, 1, 14, 3, 0) – box b_2 packed at corner III with surface <6m × 12m> as the base

…

(12, 6, 4, 0, 0, 9, 0) – box b_2 packed at corner IV with surface <12m × 6m> as the base

…

(9, 9, 7, 0, 0, 0, 0) – box b_3 packed at corner I with surface <9m × 9m> as the base

No orientation 1 since the length and width are both 9m

(9, 9, 7, 2, 0, 0, 0) – box b_3 packed at corner I with surface <9m × 7m> as the base

(9, 9, 7, 3, 0, 0, 0) – box b_3 packed at corner I with surface <7m × 9m> as the base

(9, 9, 7, 0, 11, 0, 0) – box b_3 packed at corner II with surface <9m × 9m> as the base

(9, 9, 7, 2, 11, 0, 0) – box b_3 packed at corner II with surface <9m × 7m> as the base

...

(9, 9, 7, 0, 11, 6, 0) – box b_3 packed at corner III with surface <9m × 9m> as the base

(9, 9, 7, 2, 11, 8, 0) – box b_3 packed at corner III with surface <9m × 7m> as the base

... omitted

This COPM list has been sorted in decreasing order which is the processing order in pseudo-packing process.

5.4.2 Pseudo-packing and the greedy approach

According to the list, the first COPM processed is (14, 8, 6, 0, 0, 0, 0). b_1 is pseudo-packed as shown in Fig. 5.8. The corner occupied is (0, 0, 0).

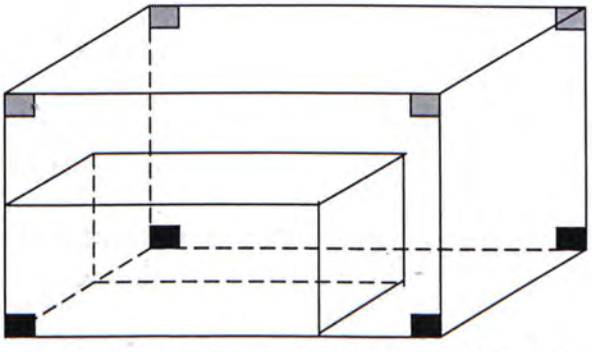


Fig. 5.8 The pseudo-pack of COPM (14, 8, 6, 0, 0, 0, 0)

The COPMs at the corner (0, 0, 0) can be deleted as no pseudo-packing at this corner is possible. The COPMs of b_1 can also be skipped after b_1 is pseudo-packed. The following COPM list is the shortened version:

(12, 6, 4, 0, 8, 0, 0) – box b_2 packed at corner II with surface <12m × 6m> as the base

(12, 6, 4, 1, 14, 0, 0) – box b_2 packed at corner II with surface <6m × 12m> as the base

…

(12, 6, 4, 0, 8, 9, 0) – box b_2 packed at corner III with surface <12m × 6m> as the base

(12, 6, 4, 1, 14, 3, 0) – box b_2 packed at corner III with surface <6m × 12m> as the base

…

(12, 6, 4, 0, 0, 9, 0) – box b_2 packed at corner IV with surface <12m × 6m> as the base

…

(9, 9, 7, 0, 11, 0, 0) – box b_3 packed at corner II with surface <9m × 9m> as the base

(9, 9, 7, 2, 11, 0, 0) – box b_3 packed at corner II with surface <9m × 7m> as the base

…

(9, 9, 7, 0, 11, 6, 0) – box b_3 packed at corner III with surface <9m × 9m> as the base

(9, 9, 7, 2, 11, 8, 0) – box b_3 packed at corner III with surface <9m × 7m> as the base

… omitted

This shortened COPM list can save the time for the subsequent steps of pseudo-packing. The list is scanned from the top to bottom. When a COPM is found to be valid (no overlapping of boxes and no cross of boundaries), the box is immediately packed according that COPM and

shorten the list again. Other COPMs for that box will not be considered in this pseudo-packing step. The next box is again packed to the first valid COPM directly without scanning through the whole list.

The first tuple of the COPM list results in overlapping of boxes b_1 and b_2 and is thus invalid. The second tuple (12, 6, 4, 1, 14, 0, 0) is checked to be valid and the pseudo-packing result is shown in Fig. 5.9.

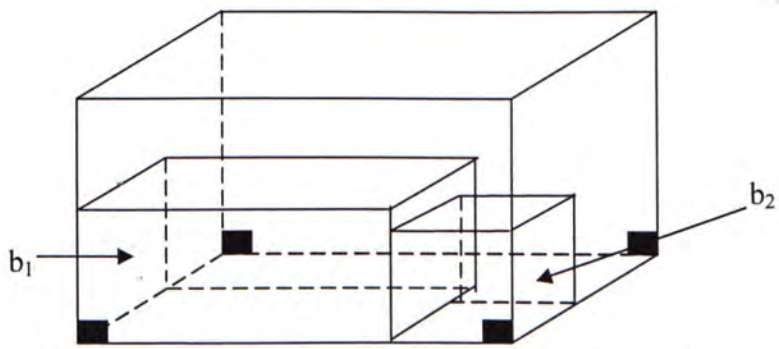


Fig. 5.9 The second step of pseudo-packing. b_2 is pseudo-packed by greedy approach

The COPM is shortened to contain only one box b_3 associating with corner III and IV.

(9, 9, 7, 0, 11, 6, 0) – box b_3 packed at corner III with surface <9m × 9m> as the base

(9, 9, 7, 2, 11, 8, 0) – box b_3 packed at corner III with surface <9m × 7m> as the base

… omitted

The first tuple is invalid while the second is valid. b_3 is pseudo-packed at corner III.

The three boxes are all pseudo-packed into the boxes. The total occupied

volume is:

$14 \times 8 \times 6 + 12 \times 6 \times 4 + 9 \times 9 \times 7 = 1527$

The FFV for the COPM (14, 8, 6, 0, 0, 0, 0) is 1527. The FFV of other COPMs are calculated in the same way.

5.4.3 Update of corner list

Three boxes b_0, 14m x 8m x 6m; b_1, 12m x 6m x 5m and b_2, 9m x 9m x 4m are being packed into a container 20m x 15m x 10m. Some COPMs of b_0 is listed in Section 5.4.1. The FFV of COPM <14, 8, 6, 0, 0, 0, 0> is calculated by pseudo-packing b_0 at (0, 0, 0) with 14m side along x-dimension, 8m along y-dimension and 6m along z-dimension[5]. Corner at (0, 0, 0) is deleted from corner list. Corners at (14, 0, 0), (0, 8, 0) and (0, 0, 6) are inserted. b_1 can neither be packed at (0, 0, 6) nor (0, 0, 10). The first available corner for pseudo-packing b_1 greedily without overlapping is (0, 8, 0). New corners are (0, 8, 5), (0, 15, 5), (12, 8, 0) and (12, 15, 0). Fig. 5.10 shows the packing configuration.

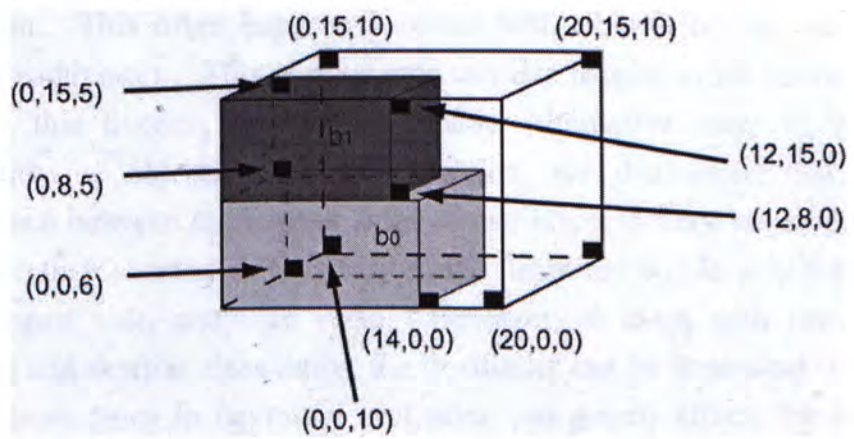


Fig. 5.10 Packing configuration after b_0 and b_1 are pseudo-packed (all corners are shown)

[5] x-dimension refers to the length of the container, y-dimension refers to the height of the container, z-dimension refers to the depth of the container

If there is no orientation constraints for b_2, it can be placed at (0, 0, 6) with 4m side along z-dimension. FFV is total volume of b_0, b_1 and b_2. All boxes are then removed. b_0 is pseudo-packed to another COPM till all COPMs are evaluated.

5.4.4 Real-Packing

After all COPMs of a box are evaluated, the box will be "real-pack" to the COPM with the highest FFV. The corner list is updated and the COPMs of the next unpacked box will undergo pseudo-packing. The process continues until no boxes can be packed to any corners. The LFF algorithm is shown in Fig. 5.5.

5.5 Ratio Approach: A Modification to LFF

As discussed in Section 4, the LFF will obtain extremely low volume utilization when the assessment of "flexibility" does not reflect the real situation. This often happens in cases with objects having very large length/width ratio. This phenomenon can also happen in 3D problems.

In this Section, we try to consider alternative ways to measure flexibility of objects. After investigation, we discovered that if the difference between the longest sides of two boxes is very large while that between their shortest sides is very small, flexibility will largely depend on the longest side, and vice versa. However, in cases with comparable longest and shortest sides ratios, the flexilibilty can be dependent on either. Such dependency in flexibility evaluation can greatly affects the result of our LFF algorithm. To illustrate the situation, some examples are shown first.

Consider the flexibility of the following two boxes. In fig 5.11a), the length of the box, named b_1 is 40cm, its width is 30cm and its height is 1

cm. In fig 5.11b), the length of the box, named b_2 is 39cm, its width is 38cm and its height is 37cm.

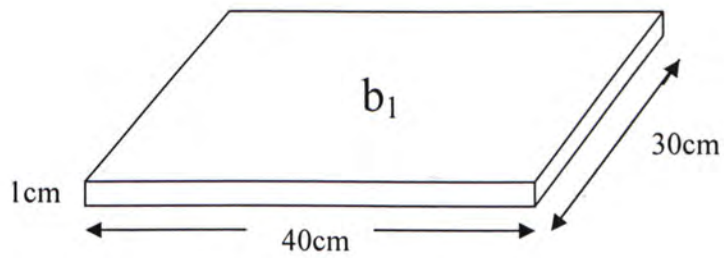


Fig 11a) Dimension of b_1

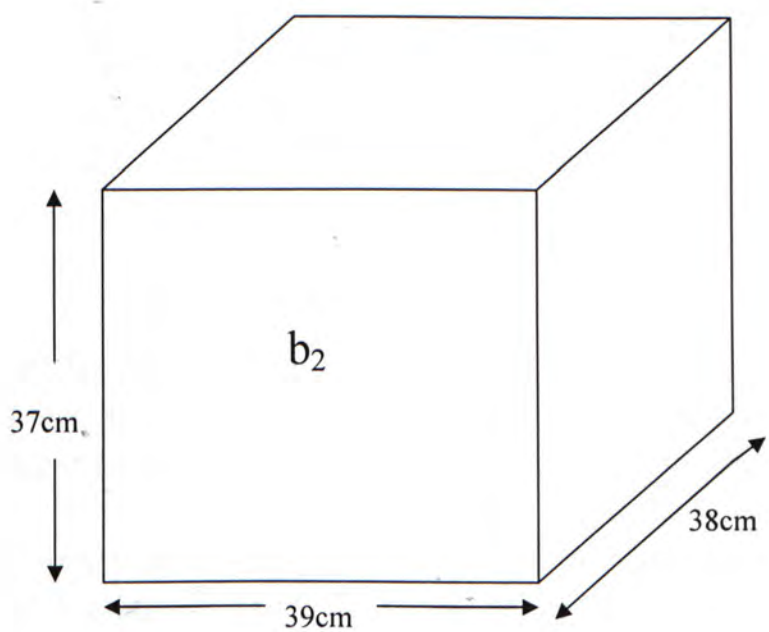


Fig. 5.11b) Dimension of b_2

It is obvious that b_1 can fit into narrow gaps with base area large enough while b_2 requires a very large space to accommodate it. However, based on the principle of the original LFF, b_1 is considered to be less flexible as its longest side is 40cm, i.e., 1cm longer than the longest side of b_2. In

this case, the flexibility determined by LFF does not reflect the real flexibility of the two boxes. Such wrong determination of packing order may lead to very low volume utilization, which is similar to the 2D cases discussed in Section 4.

If we focus on the above example, we may draw the conclusion that the box with larger volume has lower flexibility. However, this is not always a fact. Consider two boxes: The dimension of b_3 is 30cm × 20cm × 5cm while the dimension of b_4 is 80cm × 5cm × 2cm. If the dimension of the container is 80cm × 30cm × 5cm, it is obvious that the flexibility of b_4 is lower because it only can be placed with one orientation while b_3 can rotate to fit in spaces with different dimensions if necessary. However, volume of b_3 is larger. Therefore, it can be concluded that volume is not always a good measure of an object's flexibility.

If an object is said to be more flexible than another object, the basic definition is that there are more possibilities of spaces to accommodate this object. If we consider two objects b_5 and b_6 with dimensions 20cm × 16cm × 10cm and 22 cm × 20cm × 4cm respectively, the flexibility of b_6 is higher because the difference between their lengths is not very large. Comparison between their heights becomes an important factor for assessment on flexibility. In fact an object with a longer shortest side can always lead to higher difficulty for itself to be placed to a space. For objects with shorter shortest sides, despite orientation constraint, they can rotate to fit themselves to narrow gaps. Generally speaking, determining flexibility by the lengths of the shortest side is reasonable because for objects with "longest longest side" but rather short shortest side, they may still have higher possibility to fit into narrow residual spaces while those objects with long shortest side can in no way rotate itself to be packed to narrow residual spaces.

We have tried to implement the LFF by changing the way to assess flexibility. In original LFF, the objects with longest longest side are regarded as the least flexible but in our new version, objects with longest shortest side are regarded as the least flexible. It is found that the average volume utilization has a slight increase. For some cases the improvements

are very obvious while reduction in volume utilization can be found in some other cases.

The reduction occurs in packing instances with objects having large "longest side / shortest side" ratio. When a pair of objects have their shortest sides differ by only 1cm while the difference between their longest sides is very large, we should switch back to compare their longest side. In fact, under different situations, the assessment of flexibility should be different. It is very difficult to apply one standard to all cases otherwise there must be undesirable results for some cases.

Therefore we have two rules for determining object flexibility:

(i) by comparing the longest sides
(ii) by comparing the shortest sides

The problem is how to choose which rule should be used.

Two approaches have been tested for rule selection. During the flexibility assessment, the rule selection is based on each pair of objects under comparison. This means that the rule for each pair of objects can be different.

Assume there are two objects b_7 and b_8 with dimensions 20cm × 10cm × 4cm and 12cm × 10cm × 8cm respectively. The first approach is to compare their longest side ratio and their shortest side ratio. Note that when calculating these ratios, we pick the larger number to be numerator and the smaller number to be denominator, i.e., the ratio is ensured to be larger than 1. For example, if the ratio of the longest side (length) is calculated by l_7 / l_8, it is not necessary to have another ratio (height) calculated by h_7/ h_8. If $h_8 > h_7$, we will calculate the ratio by h_8/ h_7. The pseudo-code is as shown in Fig 5.12:

```
1. R1 = ratio of the objects' longest sides
2. R2 = ratio of the objects' shortest side
3. If (R1 > R2)
       Use Rule (i)
   Else
       Use Rule (ii)
```

Fig. 5.12 Rule selection approach 1for determining flexibility

In this case, the ratio of their longest sides is 20/12 = 1.67 (to 3 sig. fig.) while the ratio of their shortest sides is 8/4 = 2. Therefore, rule (ii) will be used.

The second approach is to set rule (ii) as default and change it to rule (i) under certain condition. The pseudo-code is as shown in fig 5.13:

```
1. Use Rule (ii) as the default rule
2. R1= ratio of the objects' longest side
3. If R1 > a preset constant
       Use Rule (i)
```

Fig. 5.13 Rule selection approach 2 for determining flexibility

The preset value here is defined by running experiments to see which value is most likely to obtain a better result.

After running experiments on benchmark test cases, it is discovered that the second approach leads to better results. The difference is about 0.5%-0.8%.

The second approach is therefore selected to generate experimental results shown in Section 5.7. It is named as LFF in ratio approach (LFFR).

5.6 LFF with Tightness Measure: CPU time Cut-down

The tightness measure concept was firstly introduced to the 2D LFF in [2]. The original objective of the tightness measure is to improve the packing density. With the tightness measure being introduced to the algorithm, when a box is being pseudo-packed greedily, it is no longer placed to the first available corner and skipping other candidate COPM. Instead the whole list of COPMs is evaluated to find out the "tightness" value of packing the box there. Fig. 5.14b) shows how this tightness value is obtained. The 12 circles indicate the points for tightness measure. If a point with circle is touching with another box or the container boundary, the tightness value will be incremented by 1.

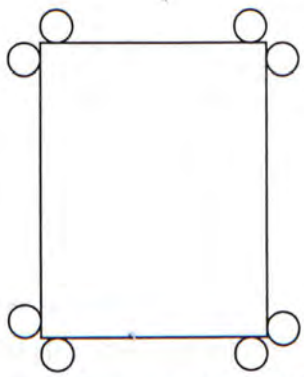


Fig 5.14a) Tightness measure points in 2D case

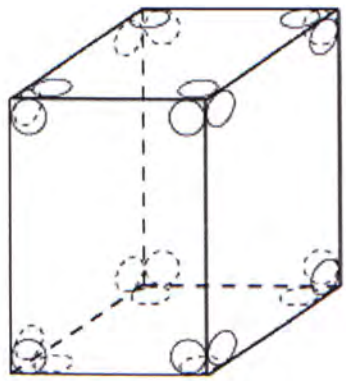


Fig 5.14b) Tightness measure points in 3D case

Note that for each corner, it is surrounded by 3 surfaces for measuring tightness value. The higher the tightness value, the better the COPM is. Please refer to Section 3.1, Fig 3.3 for an explanation based on a 2D problem. In 2D, for each corner, we have two points for evaluating the tightness, as shown in Fig.5.14a). In Fig 3.3, position I is considered to be the best because it is bound by three sides. The tightness value for position I is $2 \times 2 + 1 + 1 = 6$. The tightness for position II is $1 \times 2 + 1 + 1 = 4$. The position with higher tightness value is obviously having less

flexibility and considered to be a better position for packing.

We do not apply this idea to greedy packing step because our LFF algorithm for container loading problem already takes quite long CPU time to complete. If the tightness measure is integrated to the algorithm, the CPU time will further increase.

As tightness measure is also a reflection of the suitability for an object to be packed to a certain position, in other words, it can be used to evaluate the fitness of a COPM.

In our LFF described earlier in this Section, the fitness of a COPM is evaluated by pseudo-pack that COPM, greedily pack remaining objects to valid COPMs to calculate the packed volume which is then assigned to be the fitness. The long CPU time for the algorithm is due to the greedy packing steps. If we avoid doing greedy packing and evaluate the fitness by measuring tightness of each COPM, the CPU time is expected to be sharply cut down. The algorithm of LFF with tightness measure is presented in fig 5.15:

LFF algorithm with tightness measure for container loading:

Starting with a empty container

1. Based on the current packing configuration, find all possible COPMs for each unpacked box; represent each COPM by a tuple <longest side, medium side, shortest side, orientation, x_l, y_l, z_l>.
2. Sort all of these tuples according to their flexibility.
3. For each candidate COPM, do 3.1 to 3.3 to find its fitness function value (FFV).
   - 3.1 Pseudo-pack this COPM.
   - 3.2 Calculate the tightness after this COPM is pseudo-packed
   - 3.3 Assign the tightness value to be the FFV of this candidate COPM

   Note: Before the pseudo-packing for the next candidate COPM is evaluated, the previously pseudo-packed COPM must be removed.
4. Pick the candidate COPM with the highest FFV and really pack the corresponding box according to the COPM. The corner list is updated for later packing procedures.

Fig. 5.15 Algorithm of LFF with tightness measure

Experimental results generated by this algorithm are shown in Section 5.7. When comparing with the original approach, a very large reduction on running time is observed. However, there is trade-off in its volume utilization, which is also reduced. Please refer to the experiments in Section 5.7 for further details.

5.7 Experimental Results

To evaluate the performance of the proposed algorithms, container loading experiments are run on the original LFF and its two variants: the LFF in ratio approach (LFFR) and the LFF with Tightness Measure (LFFT). The test cases for generating experimental results are benchmarks from Bischoff and Ratcliff [8] and Loh and Nee [11]. The performance of the proposed algorithms is compared with the results from other researchers [22, 23, 24, 25, 26].

All experiments are run on the Sun Blade 1000 machines with 2GB RAM. The Operating System is Solaris 8. The machines are accessed remotely and it is possible for many connections to access the same machine simultaneously. Therefore the CPU utilization by our program may vary throughout the experiments. This did not affect the results on volume utilization of the container loading problem. The area that may be affected is the CPU time of the experiments.

5.7.1 Comparison between LFF and LFFR

The LFF in Ratio Approach (LFFR) is introduced in Section 5.5. It is a variant of LFF with an alternative method to evaluate objects' flexibility. For details, please refer to Section 5.5. To compare the performance of the LFF and LFFR, we run experiments on both of them and obtain the results in Table 5.2:

Test case (no. of box types, mean no. of boxes per type)	Average volume utilization (%) by LFF	Best volume utilization (%) by LFF	Average volume utilization (%) by LFFR	Best volume utilization (%) by LFFR
BR1 (3, 50.1)	85.41	93.42	87.19	93.42
BR2 (5, 27.3)	86.47	93.54	87.97	93.92
BR3 (8, 16.8)	87.15	93.4	88.37	93.84
BR4 (10, 13.3)	87.18	91.23	88.07	92.45
BR5 (12, 11.1)	87.09	91.83	88.10	92.09
BR6 (15, 8.8)	87.09	90.33	88.15	92.43
BR7 (20, 6.5)	86.89	90.86	87.65	90.90
Average	86.75	92.09	87.93	92.72

Table 5.2 Numerical results obtained by LFF and LFFR on 700 problems of Bischoff and Ratcliff

In Bischoff and Ratcliff test cases, there are seven sets with 100 individual cases in each set. The heterogeneity increases from BR1 to BR7.

The average volume utilization of the seven sets obtained by LFF is 86.75% while that obtained by LFFR is 87.93%. The best volume utilization in every set is over 90% for both LFF and LFFR. When focusing on the results in each of the seven sets, it can be noticed that the LFFR can always obtain volume utilization higher than LFF by 0.8% - 1.2%. Hence from this experiment, the better performance of LFFR has been proven. The experimental results shown in later sections are obtained by LFFR.

5.7.2 Comparison between LFFR, LFFT and other algorithms

Table 5.3 shows the volume utilizations obtained by LFFR, LFFT and 5 other algorithms.

Test case (no. of box types, mean number of boxes per type)	Bischoff et al. [23] Volume utilization (%)	Bischoff et al. [22] Volume utilization (%)	Gehring and Bortfeldt [24] Volume utilization (%)	Gehring and Bortfeldt [25] Volume utilization (%)	A. Bortfeldt et al. [26] Volume utilization (%)	LFFR volume utilization (%)	LFFT Volume utilization (%)
BR1 (3, 50.1)	81.76	83.79	85.80	87.81	93.52	87.19	82.73
BR2 (5, 27.3)	81.70	84.44	87.26	89.40	93.77	87.97	84.08
BR3 (8, 16.8)	82.98	83.94	88.10	90.48	93.58	88.37	84.21
BR4 (10, 13.3)	82.60	83.71	88.04	90.63	93.05	88.07	84.48
BR5 (12, 11.1)	82.76	83.80	87.86	90.73	92.34	88.10	84.84
BR6 (15, 8.8)	81.50	82.44	87.85	90.72	91.72	88.15	84.84
BR7 (20, 6.5)	80.51	82.01	87.68	90.65	90.55	87.65	84.29
All test cases	82.0	83.5	87.5	90.1	92.7	87.9	84.2

Table 5.3 Numerical results for the 700 problems from Bischoff and Ratcliff [22] by LFFR, LFFT and 5 other algorithms

From Table 5.2, we can see that although the volume utilization obtained by LFFR is not the best among the seven algorithms, its advantage is on its stability when being run on cases with different heterogeneity. The result obtained by the G.A in [24] shows that the algorithm achieves its best performance in heterogeneous problems and the performance degrades when the algorithm is being applied to homogeneous problems. The situation for the hybrid G.A in [25] is the opposite. Its design favors its performance in homogeneous cases while in heterogeneous cases, it obtains lower volume utilization. When using LFFR to solve container loading problem, the average volume utilization for weakly heterogeneous set BR1 is only slightly lower than that of strongly heterogeneous ones but the difference is very small. This shows that the performance of LFFR is stable for all kinds of cases.

The volume utilization obtained by LFFT is quite low when compared with other algorithms. However, its computational time is much faster than other algorithms. Please refer to Section 5.7.3 for details.

Test cases	Ngoi et.al. [28]	Bischoff et al. [23]	Bischoff and Ratcliff [22]	Gehring and Bordfeldt [25]	Bortfeldt and Gehring [26]	LFFR	LFFT
Mean Vol. Util.(%)	69.0	69.5	68.6	70.0	70.1	70.1	69.4

Table 5.4 Numerical results obtained by six algorithms on 15 problems of Loh and Nee

Table 5.4 shows results obtained by six algorithms on 15 Loh and Nee problems.

Using LFFR, no boxes are left unstowed in 13 out of 15 Loh and Nee

problems. The average volume utilization is 70.1%, which is better than four other methods. For LFFT, it achieves higher volume utilization than LFFR in 2 out of the 15 cases while there are no box left unstowed in 10 out of 15 Loh and Nee problems.

5.7.3 Computational Time for different algorithms

To see how LFFT outperforms LFFR in terms of computational time, we firstly compare their computational time for solving the Bischoff problems. To make analysis easier, the grouping of cases is not by heterogeneity. Instead we group the problems by their sizes, i.e. the number of objects to be packed.

Problem size (No. of objects being packed)	Average computational time of LFFR (s)	Average computational time of LFFT (s)
<=100	228.25	1.03
101 – 120	339.00	1.77
121 – 140	537.39	2.60
141 – 160	782.95	3.22
161 – 180	965.11	4.00
181 – 200	1361.00	4.30
201 – 300	3375.00	5.05
>300	10542.75	9.33

Table 5.5 Average computational time of LFFR and LFFT on different problem size

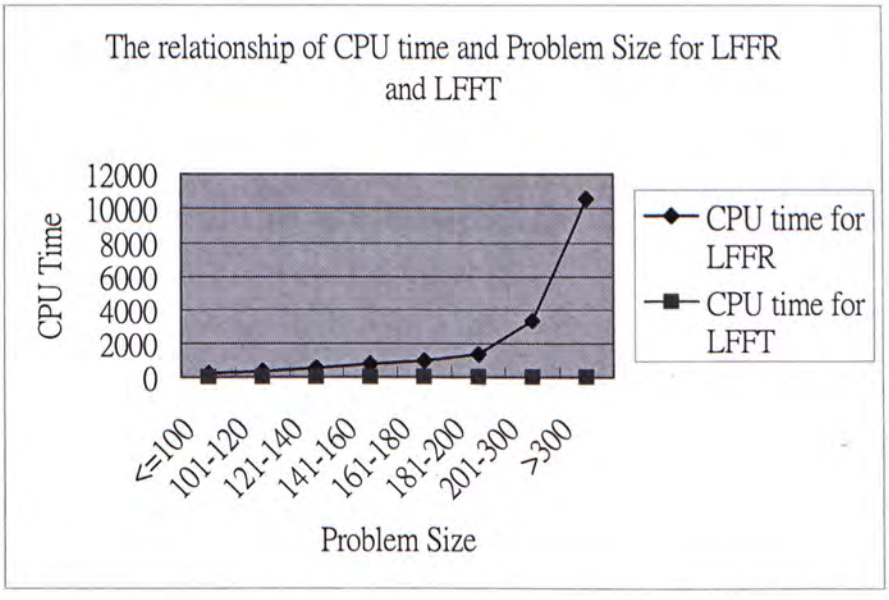


Fig 5.16 A chart showing the relatoinship of CPU time and Problem Size for LFFR and LFFT

Fig. 5.16 is a graph showing the relationship of CPU time and Problem Size for the two algorithms being discussed. Obviously even when there is a substantial increase in problem size, LFFT can still handle the cases in very short CPU time, i.e., a little increase in CPU time while great increase in problem size. For LFFR, the increase in CPU time is very large when the problem size increases. The last step in the graph is very large because it includes all cases >300. Only several cases are in this category but it ranges from 319 to 476. When the problem size becomes 476, LFFR takes 6 hours, i.e., about 21600 seconds to obtain the result, leading to such a sharp increase in the chart. For LFFT, the CPU time handling the case with 476 boxes is 12.11 seconds, which is 1/1783 of the LFFR CPU time.

Algorithm	Gehring and Bortfeldt [24]	Gehring and Bortfeldt [25]	A. Bortfeldt and Gehring [26]	LFFR	LFFT
Average CPU time(s)	11.7	316.0	121.0	812.4	2.7

Table 5.6 Average computation time for 5 different algorithms

5.7.4 Conclusion of the experimental results

Although the volume utilization obtained by LFFR is lower than that given by [24] and [25], it still has its advantage. The lower volume utilization in LFFR may be further improved in future if:

- the measure of flexibility can be further modified to reflect flexibilities of objects in a better way. In fact, the orientation constraint is also a factor affecting flexibility. One possible improvement can be setting weights to each factor affecting flexibility so that all aspects are considered. This is one of the extensions of this research.

- the greedy-packing procedure is modified in such a way that objects are not just greedily packed to the first available corner. Instead, they will be greedily packed to the COPM that can achieve a better result.

Following these two directions, the performance of LFFR can be further improved and its stability will remain high. Therefore the stable performance of LFFR in different heterogeneity is of very important value.

It is obvious that LFFT obtains lower volume utilization when being compared with four other algorithms in Table 5.2. The value of LFFT is on its very short computation time. In Table 5.5 and 5.6, we can see that the computational time of LFFT is 1/301 of LFFR, 10/43 of the G.A in [24], 1/117 of the G.A. in [25] and 1/45 of the parallel T.S.A. in [26]. The computational time of the other two algorithms in Table 5.2 is not

reported by their authors.

The lower volume utilization in LFFT is the trade-off of its short computational time. In fact, among all container loading algorithms with their computational time reported, the LFFT is the fastest algorithm for solving this type of problem.

Chapter 6

Conclusion

This thesis presents the LFF algorithm that achieves a satisfactory volume utilization in container loading problems. By comparing with other algorithm, the underlying principle of LFF, the Less Flexibility First concept, is in fact shared by other researchers. [24] packs large objects first and [25] gives smaller spaces higher priority to be filled. LFF has two superiorities:

- Comparing with the layering or block management approach, which regards layers or blocks as packing unit, the LFF pack boxes one by one and the flexibility of each packing unit is much higher due to their smaller sizes. The higher the flexibility of items, the higher is the chance for the item to be packed to a container because smaller space can only be filled by smaller items.

- The available packing positions for items are "spaces" in most of the algorithms by other researchers and items can only be packed to lower, left and rear corner of the space. Instead of "spaces", the possible packing positions in LFF are "corners". In each space, there are 8 possible corners. Although some corners are invalid, the number of valid corners must be more than 1. Therefore the solution space searched by LFF is more.

Yet there is still room for improvements.

- The evaluation of object flexibility greatly affects the result. We believe that , as mentioned in Section 5 , more can be done to sort the flexibility in a way leading to higher volume utilization.

- Another direction of improvement is the choice of packing position of boxes based on the FFV of their COPMs. There are often many COPMs having the same FFV. The first one is always selected without further assessment. If a mechanism is introduced to solve this tie, a further raise in volume utilization is possible.

- The running time of LFF is a great concern. As the problem size increases, the number of COPMs will increase at a very fast rate, leading to long running time for packing instances with more than 200 objects. Reduction in running time can make the LFF algorithm more practical.

Future research can focus on these three issues.

To conclude our work, we contribute on the followings:

- We have perform the Worst Case Analysis to evaluate the 2D LFF algorithm and prove the inexistence of the Error of this algorith. This can facilitate further research on improvement of this algorithm.

- We have firstly aplied the LFF algorithm to the container loading problem. Its performance is satisfactory when being run on benchmark cases.

- We have proposed another version of LFF with tightness measure to substantially cut down the CPU time of the original LFF.

However, there is trade-off in the volume utilization.

- The proposed 3D LFF is an innovative approach providing a new direction for further research.

Bibiography

[1] Y. L. Wu, W. Q. Huang, S. C. Lau, C. K. Wong and G. H. Young, "An effective quasi-human heuristic for solving the rectangle packing problem", *European Journal of Operational Research* 141 (2002), pp. 341 – 358

[2] Y. L. Wu, C. K. Chan, S. Dong, X. Hong, "A simple but powerful Least-Flexibility-First packing and cutting algorithm"

[3] S. Martello, D. Pisinger, D. Vigo, "The three-dimensional bin packing problem", *Operations Research* 48 (2000), pp. 256 – 267

[4] J. Kang, S. Park, "Algorithms for the variable sized bin packing problem", *European Journal of Operational Research* 147 (2003), pp. 365 – 372

[5] F. K. Miyazawa, Y. Wakabayashi, "An algorithm for the three-dimensional packing problem with asymptotic performance analysis", *Algorithmica* 18 (1997), pp. 122 – 144

[6] C. S. Chen, S. M. Lee, Q. S. Shen, An analytical model for the container loading problem", *European Journal of Operational Research* 80 (1995), pp. 68 – 76

[7] F. K. Miyazawa, Y. Wakabayashi, "Parametric on-line algorithms for packing rectangles and boxes", *European Journal of Operational Research*, to appear.

[8] D. Pisinger, "Heuristic for the container loading problem", *European Journal of Operational Research* 141 (2002), pp. 982 – 392

[9] P. C. Gilmore, R. E. Gomory, "Multistage cutting stock problems of two and more dimensions", *Operations Research* 13 (1965), pp. 94 – 120

[10] W. Huang, "Quasi-physical and quasi-social methods for tackling NP-hard problems", *Proc. International Workshop on Discrete Mathematics and Algorithms*, Jinan University Press, Guang Zhou, China, 1994.

[11] W. Huang, R. Jin, "The quasi-physical and quasi-sociological algorithm solar for solving SAT problem", *Science in China* 27 (2) (1997), pp. 179–186.

[12] H. Dyckhoff, "A typology of cutting and packing problems", *European Journal of Operational Research* 44 (1990), pp. 145 – 159

[13] H. Dyckhoff, G. Scheithauer, J. Terno, "Cutting and Packing" *M Dell'Amico,*

F. Maffioli, S. Martello (Eds.), Annotated Bibliographies in Combinatorial Optimization, Wiley, Chichester, 1997

[14] T. H. Loh, A. Y. C. Nee, "A packing algorithm for hexahedral boxes", *Proceedings of the Conference of Industrial Automation,* Singapore (1992) pp. 115 - 126

[15] Michael Eley, "Solving container loading problems by block arrangement", *European Journal of Operational Research* 141 (2002) pp. 393 – 409

[16] D. Y. He, J. A. Cha, "Research on solution to complex container loading problem based on genetic algorithm", *Proceedings of the First International Conference on Machine Learning and Cybernetics, Beijing* (2002)

[17] M. Gehring, K. Menscher, M. Meyer, "A computer-based heuristic for packing pooled shipment containers", *European Journal of Operational Research* 44 (1990) pp. 277 – 288.

[18] J. A. George, D. F. Robinson, "A heuristic for packing boxes into a container", *Computers and Operations Research* 7 (1980), pp. 147 – 156.

[19] D. Pisinger, "The container loading problem", *Proceedings NOAS'97*, 1997.

[20] C. Pimpawat, N. Chaiyaratana, "Using a co-operative co-evolutionary genetic algorithm to solve a three-dimensional container loading problem", *Evolutionary Computation, 2001, Proceedings of the 2001 Congress*, pp. 1197 -1204 Vol. 2

[21] E. E. Bischoff, M. D. Marriott, "A comparative evaluation of heuristics for container loading", *European Journal of Operational Research* 44 (1990), pp. 267 – 276

[22] E. E. Bischoff, B. S. W. Ratcliff, "Issues in the development of approaches to container loading", *Omega* 23 (1995), pp. 377 – 390

[23] E. E. Bischoff, F. Janetz, M. S. W. Ratcliff, "Loading pallets with non-identical items", *European Journal of Operational Research* 84 (1995), pp. 681 – 692

[24] H. Gehring, A. Bortfeldt, "A genetic algorithm for solving the container loading problem", *International Transactions in Operational Research* 4 (1997), pp. 401 - 418

[25] A. Bortfeldt, H. Gehring, "A hybrid genetic algorithm for the container loading problem", *European Journal of Operational Research* 131 (2001),

pp. 143 – 161

[26] A. Bortfeldt, H. Gehring, D. Mack, "A parallel tabu search algorithmfor solving the container loading problem", *Parallel Computing* 29 (2003), pp. 641 – 662

[27] H. Gehring, K. Menschner, M. Meyer, "A computer-based heuristic for packing pooled shipment containers", *European Journal of Operational Research* 44 (1990), pp. 277 - 288

[28] B. K. A. Ngoi, M. L, Tay, E. S. Chua, "Applying spatial representation techniques to the container packing problem", *International Journal of Production Research* 1 (1994), pp. 59 - 73

[29] J. Hemminki, "A heuristic for container loading, Report 141, Univerisity of Turku, Department of applied mathematics, 1993

[30] A. Lim and Y. Wang, "A new method for the three dimensional container packing problem"

[31] J. Hemminki, "Container loading with variable strategies in each layer", presented in *ESI-X, EURO Summer Institue, Jouy-en-Josas*, France, 2-15 July 1994

[32] George, G.A., Robinson, D.F., "A heuristic for packing boxes into a container, Computers and Operations Research 9 (1980), 147-156
presented in *ESI-X, EURO Summer Institue, Jouy-en-Josas*, France, 2-15 July 1994

[33] Toulouse et al., "Issues in designing parallel and distributed search algorithms for discrete optimization problems", *Publication CRT-96-36*, Centre de recherché sur les transports, Universitede, Montreal, Canada, 1996

[34] Y. T. Wu, Y. L. Wu, "A less flexibility first based algorithm for the container loading problem", *Operations Research Proceedings* 2004, pp. 368 – 376

[35] J.B.Rosenberg, Geographical data structures compared: A study of data structures supporting region queries, IEEE Transaction on Computer-aided Design CAD-4 (1) (1985) 5, pp. 3 – 57.