Two and Three – Dimensional Bin Packing Problems: An efficient implementation of evolutionary algorithms

Andile Ntanjana

Submitted in fulfilment of the academic requirements for the degree of

Master of Engineering
in Mechanical Engineering

in the
Department of Mechanical Engineering
Durban University of Technology

Durban
2017

Declaration

The research work described in this thesis was carried out under the Department of Mechanical Engineering, Durban Univesity of Technology, under the supervision of Professor Pavel, Y Tabakov and Professor Sibusiso Moyo.

This thesis presents original work by the author and has not been submitted in any form for any other degree or examination. Where use has been made of the work of others it is duly acknowledged in the text.

April, 2018.

Student: ______________________	______________________
Andile Ntanjana	Date
Supervisor: ______________________	______________________
Prof.P,Y. Tabakov	Date
Co-supervisor: ______________________	______________________
Prof. S. Moyo	Date

Acknowledgements

My sincere thanks go to my two supervisors, Professor Pavel, Y Tabakov, and Professor Sibusiso Moyo, for the encouragement and guidance they have given to me over the last two years. Through their support and trust, I have had the opportunity to direct my part in an interesting and very challenging project, and to enjoy myself along the way.

I also thank those who let me describe the technical details of my research to them on numerous occasions. When I do not communicate my thoughts, they often become jumbled in my head and never clearly develop. Having people who would sit and listen as I explained my ideas to them was thus essential in the development of this work. For this I thank Donnah Biyela, Zolisa Dolwana, Happy Christian Tshimbiluni, Aaron Lecheko and Euclid Nkuna. There is a long list of names missing from this list of those with whom I only spoke once briefly about my work, and I would like to thank those people as well.

My thanks also go to the Durban University of Technology and, particularly, the Directorate of the Research and Postgraduate Support for the invaluable assistance and support that made this research work possible.

Finally, I would like to extend my sincere thanks to my dear God, who led me through the most difficult time of my thesis writing with His profound and unconditional love for me.

Abstract

The present research work deals with the implementation of heuristics and genetic algorithms to solve various bin packing problems (BPP). Bin packing problems are a class of optimization problems that have numerous applications in the industrial world, ranging from efficient cutting of material to packing various items in a larger container.

Bin packing problems are known to be non-deterministic polynomial-time hard (NP-hard), and hence it is impossible to solve them exactly in polynomial time. Thus heuristics are very important to design practical algorithms for such problems. In this research we avoid the use of linear programming because we consider it to be a very cumbersome approach for analysing these types of problems and instead we proposed a simple and very efficient algorithm which is a combination of the first fit heuristic algorithm in combination with the genetic algorithm, to solve the two and three – dimensional bin packing problems. The packing was carried out in two phases, wherein the first phase the bins are packed by means of the first fit heuristic algorithm with the help of other auxiliary techniques, and in the second phase the genetic algorithm is implemented. The purpose of the second phase is to improve the initial arrangements by performing combinatorial optimization for either a limited number of bins or the whole set at one time without destroying the original pattern (elitist strategy).

The programming code developed can be used to write high-speed and capable software, which can be used in real-time applications. To conclude, the developed optimization approach significantly helps to handle the bin packing problem. Numerical results obtained by optimizing existing industrial problems demonstrated that in many cases it was possible to achieve the optimum solution within only a few seconds, whereas for large-scale complex problems the result was near optimum efficiency over 90% within the same period of time.

Dedication

I dedicate this thesis to my daughter, Lonwabo Ntanjana. You provided the inspiration necessary for me to complete this process and also sacrificed immensely along the way. I love you!

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

The Bin packing problem is a major industrial issue which has a number of practical applications, which include cutting stock, warehouse management, container ship loading, plane cargo management and pallet loading. In a variant of maximizing, the problem boils down to efficiently filling the empty space inside a two or three-dimension rectangular item (called *bin*). With proper management of the available space it is possible to reduce the costs associated with the storage of goods and their transportation. The problem belongs to the difficult optimization problems ranked amongst the class of NP-hard problems.

Many algorithmic approaches are used to find approximate solutions to problems of one –, two – and three – dimensional packing. However, in connection to the fact that these methods are general and do not use detailed information about the problem, many researchers supplement them with additional heuristic approaches, more suitable for the specific nature of the issue and accepted limitations. Hence, packing and cutting problems have been studied extensively, and since Gilmore and Gomery published their integer linear programming (LP) based approach [33], most methods have been based on this idea. The method is based on solving an integer linear programming problem with a reduced set of columns each corresponding to some packing patterns of a single bin, and generating additional columns as necessary. This method does not perform well on bin packing problems where the bin size might vary, as by increasing the number of bin sizes, the number of columns increases and must be added to the basis set before an optimal solution of the linear programming relaxation is reached. An overview of the literature on the variable size bin packing problem (VSBPP) can be found in [18, 86], and most employed methods are based on Gilmore and Gomerys work. It noted that the common approaches suffer from multiple potential drawbacks when considering real problems coming from industry due to the following reasons:

- Some industries cannot guarantee a single bin size, but require the size to vary from bin to bin, thus increasing the number of packing patterns dramatically. For example, this situation can arise due to deformation of the material during or after a casting process.
- Some industries require that the code be run on embedded systems with very limited memory and sometimes even without online memory allocation, thus excluding the use of linear programming solvers.
- Some industries gain more knowledge about the problem data online, and require this information to be taken into account immediately. They might also require a solution within milliseconds while allowing a possible negative impact on its quality.
- Some industries are capable of reusing unused capacity in the last bin packed. This situation often arises when modelling a cutting problem, where cutting can be resumed on the last used item, given that it is large enough to warrant storage.

Motivated by these difficulties, the aim of this research work is to develop and implement a constructive heuristic and evolutionary approach method, which intends to produce high quality solutions for two–dimensional (2D) and three–dimensional (3D) bin packing problems at a considerable time rate, when a large number of bin sizes are available and at the same time the algorithm should adapt faster to online changes in data.

1.2 Structure of Thesis

The thesis is structured as follows. Chapter 2 presents a review of the relevant literature for bin packing problems. Chapter 3 states the 2D bin packing problem formulation and presents the design of the heuristic algorithms for solving this class of packing problems. In Chapter 4, a 3D packing approach is developed using similar heuristic approaches presented in Chapter 3. Chapter 5 presents the design of the evolutionary algorithm for optimizing the results provided by the heuristic algorithms presented in Chapter 3 and 4 with the aim of finding more optimum results. Only the 2D packing problem is covered in this chapter and the developed algorithm is easily modified later to solve the 3D problem and the results are included in chapter 6. Chapter 6 presents the computational experiment for both 2D and 3D packing problems and the results obtained when solving the 3D packing problem are compared with the results obtained when using known algorithms. Chapter 7 provides the conclusion to the research and recommendations for future work.

Chapter 2

Literature review

2.1 Introduction

This chapter provides a review of the relevant literature for the bin packing problem. Optimization of the bin packing problem is not a new area. The solution techniques have been proposed and analysed since the 1960s [7]. All optimization problems are about minimizing the waste of resources (such as raw material in manufacturing industries, etc.) and maximizing the output of processes being tackled. Inputs and outputs of processes have different limitations which lead researchers to propose ideas about finding the best solution to these different processes. Many heuristic approaches have been developed and proposed including advantages and disadvantages for solving the bin packing problem. The bin packing problem can be categorized by dimensional characteristics and the problem definition is based on the relationship between items and bins.

2.2 Problem Dimensions

The bin packing problem can be categorised in one of n-dimensions. The most usual values for n are 1, 2 and 3 and these types of problems are discussed below.

The one–dimensional (1D) problem can be seen as packing pieces from a given length of material. It is often referred to as strip packing. In this problem we are only concerned with the length of the pieces that result from the packing process. The width of the resultant pieces is either immaterial or (more usually) fixed. Typical applications for the 1D problem involve cutting lengths of steel from steel bars. In the two–dimensional (2D) packing problem we are concerned with packing two dimensional shapes from two dimensional objects (bins). The shapes can either be rectangular or irregular shapes. Typical applications include glass cutting, sheet metal cutting and cardboard cutting. In the three–dimensional (3D) problem we are normally concerned with packing three dimensional shapes into a given area. Typical applications for the 3D problem include pallet loading and loading of cargo containers inside the ship.

2.3 Classification

If the feasible area where variables of the problem are placed has a convex shape, in which any two variables can be connected directly without the line between them exceeding the field, it is called a convex set and searching for the optimum in the convex set is called convex optimization

If one of the variable pairs cannot be connected directly as defined above, it is called a non-convex set and the name for searching for the optimum in this set is called non-convex optimization. The shapes of the convex and non-convex set are shown as in Fig.2.1 and Fig.2.2. In addition to the problem of subdivision due to their convexity characteristics, there are two kinds of problems due to whether they tackle decimal or integer variables which are discrete and continuous problems [78].

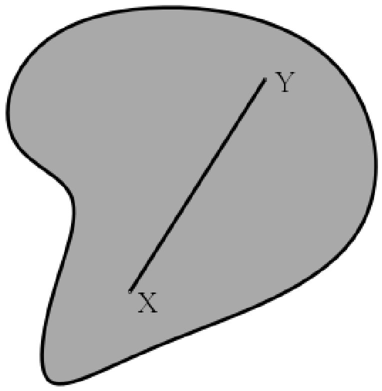


Figure 2.1: Convex Set.

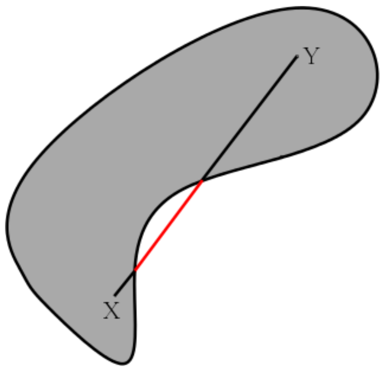


Figure 2.2: Non-Convex Set.

While tackling continuous problems, the solution searching (feasible) area for the optima consists of real numbers. In this circumstance, any solution can be found in the whole searching area. This means that, obtaining a better solution has a high possibility rate with all facets of the searching area. Mathematical optimization problems are a good example of continuous problems [78].

When dealing with tackling discrete problems, the feasible area consists of integer numbers. Hence, the solution is also an integer. In this case there is a restriction in the search in feasible areas and may give us a lower possibility of obtaining a better solution which can be between two integer solutions and would be unacceptable for discrete problems. For example, combinatorial optimization problems are a branch of discrete problems [78].

There is another and final subdivision for optimization due to linearity characteristics of objective functions. If none of the variables of the objective function are quadratic, polynomial or high degree, this objective function is called a linear function. Otherwise, it is called a non-linear function. Fig.2.3 shows the subdivision of optimization problems by characteristics [53].

Figure 2.3: Branches of Optimization Problems [53].

In this case, we have the following definitions: LP–Linear Programming, NLP – Non-linear Programming, IP – Integer Linear Programming, MILP – Mixed Integer Linear Programming and MINLP – Mixed Integer Non-Linear Programming [78].

2.4 Deterministic Optimization

Mathematical programming or deterministic optimization is the classical branch of optimization algorithms in mathematics. The algorithms for obtaining the best possible solutions which are heavily related to linear algebra comprise the deterministic optimization. These algorithms are based on the computation of gradient, and in some problems also of the Hessian [65]. Deterministic optimization has some remarkable advantages alongside its drawbacks. One of the aforementioned advantages is the convergence to a solution which would be much faster than stochastic and heuristic optimization algorithms in comparison because applying the deterministic algorithms does not require the amount of iterations or evaluations as stochastic and heuristic algorithms do to reach the solution. When deciding which algorithm or function would be tackled by researchers, the results of necessary convergence time for the optimum solution should be measured for comparison between them. In some cases an algorithm may obtain better results than others; however, time requirements for convergence can be too much so researchers look for a more efficient algorithm.

Deterministic algorithms are commonly based on complex mathematical formulations which means that the results are never obtained by randomisation and they obtain the same solution in each run. This fact could be true also for stochastic and heuristic algorithms, however, these algorithms are based on randomisation and they may obtain different solutions in each run [9].

Most researchers find heuristic approaches more flexible and efficient than deterministic methods which commonly take advantage of analytical properties of the problem; however, many optimization problems consist of complicated elements beside their analytical aspects. The quality of the obtained solutions may not always be sufficient. One of the stochastic and heuristic approach drawbacks is reduction in the probability of finding a global optimum while searching in big size problems. Conversely, deterministic approaches can provide general tools for solving optimization problems to obtain a global or approximately global optimum which is a solution that a function reaches highest or lowest value, depending on problem, by using it [53].

Many different fields in optimization problems would apply to deterministic optimization. Skiborowski et al. [75] developed a hybrid optimization approach based on one of the necessary attributes of applied algorithms which is combinational flexibility in process design. They also applied the traditional deterministic optimization algorithm as a second phase using the results data of the first phase, which is processed by an evolutionary algorithm in a case study. Trindade and Ambrosio [83] proposed one of the deterministic optimization methods that can provide some qualified estimates for segment characteristics in the latent segment model when only aggregate store level data is available. In [83], the storage system works dynamically, so they test the Sequential Quadratic Programming (SQP) method which is a branch of deterministic optimization and accept it as a model fit for estimation in the case study. Sulaiman et al. [79] used a deterministic method approach to improve the power density of the existing motor used in hybrid electric vehicles (HEV) and they implemented the proposed deterministic method on hybrid excitation flux switching motor (HEFSM) as a candidate.

2.5 Calculation Difficulties in the Newton Search Method

Newtons Method which is a commonly applied deterministic method and its improved forms such as Quasi-Newton and BFGS (Broyden, Fletcher, Goldfarb, and Shanno) Formula are based on the idea of approximation of objective function $H(f)$ around the current solution by a *quadratic* function and then location of the quadratic function minimum. All these methods work with the Hessian Matrix in which an objective function is twice differentiable, which is typically the case for the smooth functions occurring in most applications. The second partial derivatives can tell us more about the behaviour of the function in the neighbourhood of a local solution and formulation of Hessian would be complicated in multi-dimensional problems [28].

The Hessian matrix formula $H(f)$ for an n-variable objective function is given by:

$$H(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 x_n} \\ \frac{\partial^2 f}{\partial x_2 x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n x_1} & \frac{\partial^2 f}{\partial x_n x_2} & \cdots & \frac{\partial^2 f}{\partial x_n x_n} \end{bmatrix}. \tag{2.1}$$

Even searching via deterministic methods has some advantages such as convergence speed, starting with a closer point to optimum and constant search results; there are also some backward aspects to applying them. Basically, there are some types of problems such as Non-deterministic Polynomial-Time (NP) not eligible for applying to deterministic methods based on Newtons Method Search and its improved forms. Even though deterministic methods are very useful especially for analytical problems, basic calculations for deterministic methods may have a high cost because of time wastage and calculation cost. In addition, there is a big risk of falling with the local optimum and hill climbing while searching for a global optimum. This means that searches for the global optimum may be misled and so, researchers may not benefit from the global optimum which also means extra cost.

2.6 Heuristic Optimization

In recent decades, tackled optimization problems have been getting more difficult or impossible to solve by applying the usual optimization algorithms such as deterministic methods, which have been replaced by several heuristic methods and tools [51].

Heuristic algorithms are criteria and computational methods which decide on an efficient way among alternative methods to reach any goal or solution. These algorithm approximations to the global optimum are not provable. Heuristic algorithms have approximation attributes but do not guarantee the exact solution unlike deterministic algorithms that guarantee a solution near to the global optimum. Basically, heuristic algorithms are inspired by nature's behaviour such as bee and ant colonies because creatures' mechanisms are too complicated and cannot be explained by mathematical formulations. Hence this condition fits problems which are not eligible to be solved by mathematical based methods. There are many reasons for applying heuristic algorithms.

Firstly, there may not be an available method among traditional algorithms to find an exact solution for the tackled optimization problem such as NP-hard. In this case, researchers need to find a new suitable method for this kind of problem.

Secondly, heuristic algorithms can be more helpful because they are not based on intricate formulae unlike deterministic algorithms. Heuristic algorithms are suitable to apply as a part of different algorithms to improve their efficiency for searching for an exact solution. However, this advantage is not easy to implement on deterministic algorithms.

In addition, mathematical formulation-based deterministic algorithms usually disregard the most difficult aspects of real world problems. If the tackled deterministic algorithm uses incorrect parameters, then it may find a solution much worse than a heuristic approach. Heuristic algorithms can tolerate incorrect data because they are not based on precise mathematical formulations. Therefore heuristic methods are more flexible than deterministic methods. Despite all these advantages of heuristic algorithms, there are some criteria to determine them as an applied method instead of a deterministic method because optimization problem characteristics determine which method is useful or not. Criteria about evaluation of a heuristic method are aligned as follows:

- A solution quality and convergence time are really important criteria for making decisions about the efficiency of any heuristic algorithm. Therefore, a qualified algorithm should have changeable parameter sets which are able to be compared easily according to their calculation cost and obtained solution quality attributes. In other words, the connection between the obtained solution quality and convergence time should be considered.

- The applied algorithm should be commonly applicable and its components should be clear to understand the system easily. In this condition, even though there is little information about problem characteristics, the attributes discussed earlier allow the algorithm to be applied easily in new areas.

- Heuristic algorithms involve flexible characteristics because they have to provide instant changes for the objective function and boundaries. Thus, heuristic algorithms can accommodate all conditions and problems.

- Heuristic algorithms should be capable of re-generating acceptable high quality solutions, in other words robust, which are not dependent on the initial solution swarm. In this way, they can tolerate inaccurate initial data or swarm.

- It should be easier to analyse the applied heuristic algorithm due to its flexibility and obtained solution quality compared with complicated algorithms which may not be eligible for considering their mentioned attributes.

Heuristic algorithms are based on greedy neighbourhood search methods to improve initial solutions (swarm) step by step to reach the global optimum or approximate it [44]. Some heuristic-neighbourhood (local) search algorithms generate the local optimum based on initial solutions and this situation is common for iterative improvable algorithms. In some cases, the local optima could be far from the global optima and cannot approximate them. In the most discrete optimization problems, necessary information may not be available to determine the useful initial solution heap. In order to eliminate some disadvantages of heuristic-local search methods, some options are discussed, thus, providing clearness and generality as follows:

- The heuristic-local search could be started with multiple initial solutions and apply the same number of iterations for them. However, randomly distributed different solutions may have high cost and this technique does not guarantee optimum solutions because of possible different aims of the initial solutions.
- If researchers focus on defining a complex neighbourhood function rather than multiple initial solutions, they can probably obtain better solutions regenerated from previous ones.
- Researchers can use complex learning strategies to get information about running algorithms and then this information can be used to define the penalized solutions or districts in the feasible area. In this way, they can avoid processing the same solution repeatedly.
- It is acceptable that, if a movement from a solution to another takes the algorithm out from local optima and avoids falling hill climbing, this movement should be applied even if it is not beneficial because the primary purpose of local search methods is approximating the global optima as much as possible [45].

According to all information about advantages and disadvantages of heuristic algorithms with their insufficient aspects, researchers need to choose or improve methods and tools that should be flexible to tolerate various conditions.

In recent years, researchers have been trying to combine algorithms among these new tools with their knowledge elements to improve search quality and apply the hybrid algorithms to these challenging problems. In this dissertation, we will also use the same way to obtain better solutions. Hybrid algorithms are able to give two advantages to researchers. Developed hybrid algorithms may decrease the run time by developing calculation techniques or decreasing the number of iterations. On the other hand, the hybridization process can make an algorithm more robust and flexible. In this way, developed algorithms can tolerate more noise, missing data and changing conditions [51].

In recent years, most researchers working on optimization problems are trying to develop new algorithms and apply them in case studies and real world optimization problems. Kovacevic et al [60] developed a new meta-heuristic method and applied it to determine optimal values of machining parameters traditionally performed by algorithms based on mathematical models and exhaustive iterative search. The main reason for the meta-heuristic model in [60] is that, meta-heuristic methods are capable of handling various optimization problems and obtaining high quality solutions by running less iterations, in other words less computational time. Their motivation for the development of the software prototype was obtaining sufficient solutions from the meta-heuristic algorithm. Li et al. [52] applied a particle swarm optimization approach -a branch of heuristics- and developed an adjustment strategy for weighted circle packing problems which is a kind of important combination optimization problem and has an NP-hard characteristic. The purpose of this study is to obtain a better layout scheme for larger circles and show that the proposed method is superior to the existing algorithms in performance. Liu and Xing [56] constructed a special heuristic method for the double layer optimization problem proposed to answer rapid developing logistic systems via advanced information technologies by reforming the current system. In [15], experimental results correctly proposed methods and heuristic feasibility. Kaveh and Khyatazadat [48] developed a new branch of heuristic methods called ray optimization. The main idea of this new optimization method is based on Snell's law about light travel and its refraction, in other words its direction changes. The purpose of this [48] development is to increase the effectiveness of optimization elements in search phase and convergence phase. Min et al. [63] searched for efficient algorithms to find the best orientation between two randomly deployed antenna capable of determining the best orientation to receive the strongest wireless signal,under the effect of various wireless interferences. In [63], four different heuristic optimization methods with some modification are presented as candidates and are also described with test cases and real world experiments to determine which algorithm is suitable to apply.

On the other hand, heuristic algorithms involve a few disadvantages besides their advantages. Firstly, heuristic methods usually take longer convergence time in comparison with deterministic methods. In the case where we consider mathematical based problems, deterministic methods such as the Quasi-Newton method, golden section search, and quadratic fit search would be more useful due to shorter convergence time than heuristic methods. Deterministic methods are able to reach exact solutions for deterministic problems. However, if heuristic algorithms are applied on deterministic methods, they may not reach exact solutions. In this study, one heuristic method is tackled and modified with another one to obtain a better solution for a specific NP-hard real world problem. In the subsequent section we briefly describe branches of heuristic optimization before implementation of these algorithms.

2.6.1 Evolutionary Computation

In the last decades, many researchers have been interested in evolutionary based algorithms and the number of studies about this field has increased significantly and all these studies are considered sub-fields of evolutionary computation [45]. This algorithm simulates the hypothetical population based natural evolution optimization process on a computer resulting in stochastic optimization methods. This way, we can obtain better solutions than traditional methods in difficult real world optimization problems [51].

Evolutionary computation mainly consists of evolutionary programming, evolution strategies, genetic algorithms, and differential evolution. Additionally, many hybrid algorithms replicate some evolutionary algorithm attributes. Generally, all these algorithms require five major elements; genetic representation of solutions, generator methods for initial solutions population, fitness evaluation function, and operators differentiating the genetic structure and values of control parameters. Evolutionary algorithms improve all the solution populations instead of one single solution [45]. In this way, these algorithms are able to increase their effectiveness and convergence speed while searching for the global optimum or optima.

2.6.1.1 Genetic Algorithms

Genetic algorithms are based on generating a new set of solution populations from the current one inspired by the conjecture of natural selection and genetics to improve the fitness of solutions. This process continues until the stopping criteria are satisfied or the number of iterations reaches its limit. Genetic algorithms involve three elements to obtain sufficient results from a search: reproduction, crossover, and mutation [3].

The reproduction operator works with the elimination of candidate solutions according to their quality inspired by natural selection. The crossover operator provides generating hybrid structures to increase the number of outputs. The mutation operator is a basic structure of a genetic algorithm inspired by natural genetic mutation by checking all bits of solutions and reversing them according to their mutation rate. These operators generate more candidate solutions to increase the number of options [45] . Genetic algorithms are different from other search methods due to their specialized attributes: multipath search to reduce the possibility of being trapped in local optimum area, coding parameters themselves to make genetic operators more efficient by minimizing step computation, evaluating only the objective function (in other words, fitness) to guide the search, no requirement for computation of derivatives or other auxiliary functions to make this algorithm easy to apply, and searching space with high probability of obtaining improved solutions [51]. These attributes make the genetic algorithm applicable to all kinds of problems such as discrete, continuous, and non-differentiable problems [3].

Genetic algorithms can be applied to bin packing problems. Goncalves and Resende [34] developed a kind of genetic algorithm named novel biased random key (BRKGA) for 2D and 3D bin packing problems, which can be more complex than 1D bin packing problems according to their shape complexity. They improved the solution quality significantly with a modification on the placement algorithm. In this way, they proved that an applied placement algorithm can outperform all other ones.

2.6.1.2 Evolution Strategies and Evolutionary Programming

Evolution strategies rely on the mutation processes as a search operator and evolving the strings from a population size of one. Evolution strategies are similar to the genetic algorithm in a few aspects. The major similarity between them is using selection process to obtain the best individuals from potential solutions. On the other hand, three main differences between genetic algorithms and evolution strategies can be expressed as: type of operated structures, basic operators they rely on and the link between an individual and its offspring they maintain [51].

Evolutionary programming (EP) is a stochastic optimization method developed by Lawrence J, Fogel and his co-workers in the 1960s. Evolutionary programming has also many similarities with the genetic algorithm like evolution strategies. However, this technique does not imitate nature like the genetic algorithm emulates a specific genetic operator; EP emphasizes the behavioural link between individuals and their off-spring like evolutionary strategies do. In this way, EP does not involve mutation or crossover operations together [22]. In fact, evolutionary programming and evolutionary strategies are similar but developed by applying different approaches [51]. Evolutionary programming was thought of as an applicable method just for evolving finite states for prediction tasks for a long time. After a while, it appeared that this method is also eligible to optimize the continuous function with real valued vector representation [22].

2.6.1.3 Differential Evolution

Differential evolution is the most powerful population based optimization method with its simplicity [82]. Mutation operation is also processed by the DE algorithm like other sophisticated evolutionary algorithms. However, the main difference between them is the type of applied method for the mutation process. The method the DE algorithm applies is using differences of random pairs of objective vectors [51]. Random candidates (pairs) are generated by different techniques and if new generated candidates beat existing ones, new pairs replace them and in this way, the DE algorithm acquires a more efficient search technique to find the global optimum [82].

The differential evolution algorithm is a direct search method based on a stochastic process and is considered as accurate, reasonably fast and robust. It is easy to apply for minimization processes in real world problems and multi-modal objective functions. Mutation operation process in the DE algorithm uses arithmetical combinations of individuals whereas the genetic algorithm uses the method of perturbing genes in individuals with small probability. Besides, DE algorithms do not use binary representation unlike genetic algorithms for searching. Their search works with floating point representation. All these main characteristics make this algorithm a competitive alternative and attractive especially for engineering optimizers [51].

2.6.2 Swarm Intelligence

Researchers have recently been interested in swarm intelligence by imitating animals foraging or path searching behaviours like in Particle Swarm Optimization, Ant Colony Optimization and Bee Colony Optimization.

2.6.2.1 Particle Swarm Optimization

Particle swarm optimization (PSO) is based on simulating a swarm of birds foraging, initially introduced by Kennedy and Eberhart in 1995 [73]. The PSO algorithm is initialized with random solutions (swarm) like other sophisticated evolutionary computation algorithms [51]. Swarm and particles correspond to the population and individuals in other evolutionary computation techniques respectively. Particles of a swarm move in multi dimensional space and each particle's position in this space is adjusted according to their and their neighbours performance [45].

The PSO algorithm is similar to other population based algorithms in many aspects. PSO is also useful for optimising a variety of difficult problems. Besides, it is possible that the PSO algorithm can show higher convergence performance on some problems. The PSO algorithm requires a few parameters to adjust and this attribute makes the algorithm more compatible with implementations. On the other hand, PSO can fall into local minima easily [45].

Liu et al [56] improved on a formulation for 2-D bin packing problems with multiple constraints abbreviated as (MOBPP-2D). They proposed and applied a multiobjective evolutionary particle swarm optimization algorithm (MOEPSO) to solve MOBBP-2D and performed the formulation with MOEPSO on various examples to make a comparison between the developed formulation embedded particle swarm algorithm and other optimization methods introduced in the paper. This comparison illustrates the effectiveness and efficiency of MOEPSO in solving multi-objective bin packing problems.

2.6.2.2 Ant Colony Search Algorithm

Basically, an ant colony optimization algorithm ACO imitates the foraging activities of an ant colony based on a random stochastic population. The ACO algorithm is a considerable method for optimising combinatorial optimization problems and real world applications. Ant colonies have an interesting behaviour in that they can find the shortest path to reach a food source from their nest while foraging. In order to find or change the path the colony uses, each ant deposits on the ground a chemical substance called a pheromone. Marked paths by strong pheromone concentration allow ants to choose their path to find the nest or food sources in quasi random fashion with probability. Basically, the ACO algorithm applies two procedures: specifying the way of solution construction performed by ants for the problems and updating the pheromone trail [17].

Fuellerer et al [29] considered a 2-D vehicle loading as part of the routing problem. Probable solutions to this problem involving routing success depend on loading success and should satisfy the demand of the customers. They applied different heuristic methods to obtain an optimum solution for loading part and applied ant colony algorithm (ACO) for overall optimization. Determinative factors of this problem are the size of the items and their rotation frequency. Gathering information about these factors allow us to make low cost decisions in the transportation sector. Fuellerer et al investigation about the combination of different heuristic methods with ACO was insufficient; however, in [29] their approach obtained sufficient results for this combination.

2.6.2.3 Artificial Bee Colony ABC Algorithm

The bee colony algorithm is one of the most recent population based metaheuristic search algorithms first systematically developed by Karaboga [44] and Pham et al [69] in 2006. Independently, however, these two are slightly different. Basically, they use the same techniques to generate initial swarm, but they use different elimination techniques for bees.

As mentioned in previous sections, scientists are inspired by natural occurrences and they investigate the behaviours of creatures to adapt these behaviours to the artificial systems. The main reason for simulating nature rather than applying mathematical calculations is that, creatures can overcome the problems they face every moment with guidance of their instincts. In this section, a bee swarm behaviour based algorithm is introduced step by step.

Basically, the Artificial Bee Colony (ABC) algorithm applies two main search methods as a combination, neighbourhood search and random search, to obtain optimum solutions for various problems. The ABC algorithm is inspired by food searching techniques and shares information about sources between bees in a honey bee swarm. Moreover, the ABC algorithm includes a hierarchical system and in this way, the bee swarm assigns bees and divides them according to their mission in the swarm. Definitions for bees missions can change throughout the food search. Even though the ABC algorithm is a recently developed algorithm, it is considered very promising by many researchers with sufficient results from experiments in a variety of optimization problems. The ABC algorithm is also eligible for modification or as part of a combination with other algorithms as the main characteristic of heuristics. There are many studies about the ABC algorithm.

Karaboga, the inventor of the ABC algorithm, has also published papers about this algorithm and improved his idea over time. Karaboga and Ozturk [45] applied the ABC algorithm for data clustering, used in many disciplines and applications as an important tool for identifying objects based on the values of their attributes, on benchmark problems and compared with the Particle Swarm Optimization (PSO) algorithm and nine other classification techniques [45]. On average, the ABC algorithm achieved better optimum solutions as a cluster than other algorithms for a variety of tackled datasets in [45]. Akay and Karaboga [1] applied a modified ABC algorithm for real parameter optimization to increase the efficiency of the algorithm and the results of this modified algorithm and three other algorithms were compared. In [1], the standard ABC algorithm and modified ABC algorithm present different characteristics with different kinds of problems. In this study, ABC algorithm techniques developed by Karaboga are applied on the problem we tackle as a base because ABC algorithm involves a swarm based search which is based on starting to search from multiple points in feasible area which means that ABC algorithm is capable of reducing convergence (to global optimum) time. The studies about the ABC algorithm [1, 45, 69] express the other valuable aspects of this algorithm. Pham et al [69] developed the bees algorithm as mentioned above and they demonstrated the efficiency of the newly developed algorithm on different functions. The results of the bees algorithm are sufficient and very promising for further studies. There is another study that shows the efficiency of the bees algorithm by applying it on a real world problem, also published by D.T. Pham as a co-author in [91]. Xu et al [91] applied a binary bees algorithm (BBA), which is different from a standard bees algorithm, to solve a multi-objective multi-constraint combinatorial optimization problem. They demonstrated the treatment of this algorithm by explaining the change of parameters.

Many other researchers have also benefited from the bees algorithm. Dereli and Das [23] improved a hybrid bees algorithm for solving container loading problems. The aim in this problem is loading a set of boxes into containers which are discrete variables to be worked on. They explain the necessity of applying a hybridized algorithm for the container loading problem and comparing the mentioned hybrid algorithm with other algorithms known in literature. The study demonstrates that, according to the results, the hybridized bees algorithm is very stimulating with its techniques for further studies in optimization. Kang et al [43] also proposed a hybridized algorithm for searching for numerical global optimum and this algorithm was applied on a comprehensive set of benchmark functions. In [43], the results show that a new improved algorithm is reliable in most cases and is strongly competitive. Ozbakir et al [66], developed the standard bees algorithm (BA) to apply a generalized assignment problem, which is an NP-hard problem, and compared this algorithm with several algorithms from the literature. In order to investigate the performance of the proposed algorithm further, they applied it on a complex integer optimization problem. According to the results and comparisons between several algorithms, they found the bees algorithm and proposed algorithm promising and these algorithms can be improved further. Gao and Liu [30] also improved the standard artificial bee colony (ABC) algorithm by using a new parameter with two improved solution search equations and then applied the new algorithm on benchmark functions and compared it with different algorithms according to their convergence performance for numerical global optimum. The results of the improved ABC algorithm show sufficient performance in all criteria. They also strongly recommend other researchers to carry out further studies about the bees colony algorithm. Xiang and An [89] improved the standard artificial bee colony to accelerate the convergence speed. They also made a change in the scout bee phase to avoid being trapped in local minima. In [89], several benchmark functions are tested by applying the improved algorithm and two other ABC based algorithms for a comparison between the these algorithms. According to test results, the improved algorithm can be considered as a very efficient and robust optimization algorithm. Szeto et al [81] proposed an enhanced version of the standard ABC algorithm to improve the solution quality. They applied the standard and enhanced ABC algorithm for one of the real world problems, which is the capacitated vehicle routing problem by using two sets of standard benchmark instances. In comparison, the enhanced algorithm outperformed the standard algorithm.

All these papers about the bees colony or artificial bee colony algorithm show that there is a wide area for possible and promising improvements for the standard ABC algorithm which is also eligible for modification by using different equations, formulations or parameters. It is also possible that other heuristic or deterministic algorithms with an entire structure or a minor part can be merged with the ABC algorithm easily. The main objective of improvements or hybridizations on the ABC algorithm is obtaining better solutions and making the algorithm more robust. There are not only advantages to the ABC algorithm, it also has some disadvantages. Even though convergence speed of the ABC algorithm can be sufficient in most studies, the robustness of the ABC algorithm needs to be improved because a robust algorithm is not affected by parameter changes or misleading starting points and it can obtain approximately the same results in this circumstance.

All creatures use different techniques for foraging repeatedly in a sequence with guidance of their instinct. The ABC algorithm mimics the bee behaviours in swarm and nature as mentioned above and uses the same collective intelligence of the forager honey bee. There are three main components of the ABC algorithm: food sources, employed foragers and unemployed foragers. In addition, forager bees use two main behaviours, recruitment for a nectar source and abandonment of a nectar source [44].

Food Sources: The source eligible or not for foraging visited by honey bees can be chosen due to its overall quality. There are many factors to measure the food source quality such as the distance between nest and source, food level, and nectar quality [44]. In the ABC algorithm, every single food source represents a possible solution in a feasible area and the overall food quality represents the fitness of possible solutions as well [46].

Employed Foragers: Forager bees are sent to food sources explored and determined by scout bees to exploit the nectar until they reach their capacity. Forager bees are not only carrying the nectar,they also transfer information about food sources to share with other unemployed bees. The information about food sources is shared with certain probability. In this way, scout bees can determine the best food source in the environment they investigate and onlooker bees can find the food source easily.

Unemployed Foragers: Unemployed bees consist of two types, onlooker bees and scout bees. Scout bees search the environment surrounding the nest for new sources or update information about existing sources. Onlooker bees wait in the nest to gather information about food sources from employed bees or scout bees. In a hive, the percentage of scout bees is between 5-10 [44].

Collective intelligence of bee swarms is based on the information exchange between honey bees. Every single bee shares the information they have on a kind of stage with others. In this way, they can do their job with minimal failure. The most important occurrence while they exchange information is a kind of dancing on a stage in the hive. This dance is called a *waggle dance*.

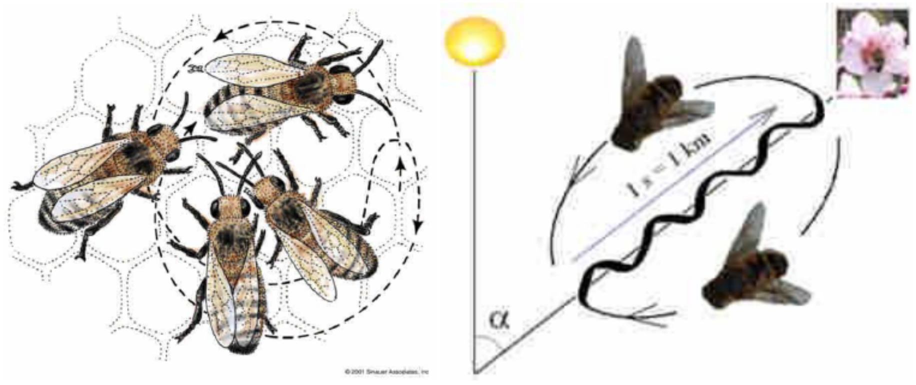


Figure 2.4: The Waggle Dance Performed by Honey Bees [38].

Austrian ethologist Karl von Frisch was one of the first people to translate the meaning of the waggle dance. If the food source is less than 50 metres away from the hive, the bee performs a round dance otherwise it performs a waggle dance. In addition, honey bees perform a waggle dance with different movements and speed. In this way, they can communicate among the swarm and share information about food sources.

The waggle dance consists of five components [38]:

- Dance tempo (slower or faster).
- The duration of the dance (longer duration for better food source.)
- Angle between vertical and waggle run means angle between sun and food source.
- Short stop in which the dancer shares the sample of food source with their audience.
- Other bees follow the dancer (audience) to gather information from the dancer.

The audience of the waggle dance determines the most profitable food source according to dance figures and then they divide the food source environment. There is a foraging cycle in the collective intelligence. This cycle involves four types of characteristics to provide the necessary information to other bees:

- Positive Feedback: If the highest nectar amount of the food sources in an environment increases, the number of visitors (onlooker bees) increases as well.
- Negative Feedback: If the nectar amount of a food source is exploited completely or has poor quality, it is not found sufficient and onlooker bees stop to visit this source.
- Fluctuation: Random search process for exploring the promising food sources is carried out by scout bees.
- Multiple interactions: As mentioned above, scout bees and forager bees share their information about food sources with onlooker bees on the dance area [44].

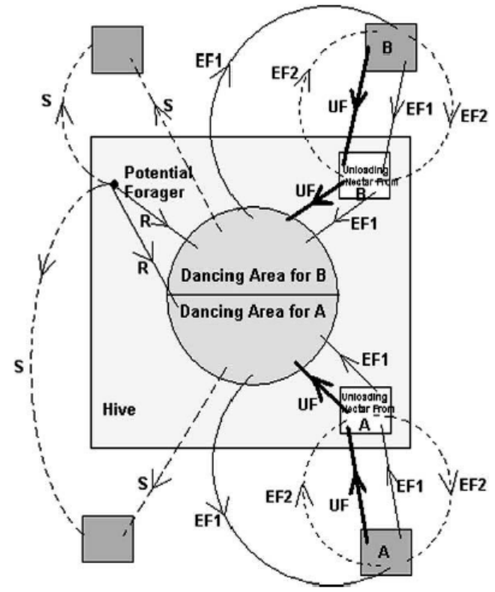


Figure 2.5: The Behaviour Of Honey Bee Foraging For Nectar [44].

Abbreviations in Fig 2.5 represent the following: Scout bees (S), newly recruited (onlooker) bees (R), uncommitted follower bees after abandoning the food source (UF), returning forager bees after performing a waggle dance for an audience (EF1), and returning forager bees without performing a waggle dance (EF2) [46].

2.6.2.4 Formulation of ABC Algorithm

In the ABC algorithm, the number of onlooker bees or employed bees equals the number of solutions in the populations. In fact, there is no difference between onlooker bees or employed bees in the ABC algorithm because the initial population cannot be changed. However, in nature it can be changed due to the needs of the swarm.

As the first step, the initial population is generated randomly for solutions in a feasible area. In the ABC algorithm, initial solutions represent the food source positions in an environment. Each solution $x_i(i = 1, 2, SN)$ is a K-dimensional vector where K is the number of characteristics, in other words the parameters of the investigated solution in optimization, and SN is the size of the initial population. Parameters for each solution can be represented as $k(k = 1, 2, .., n)$ and the following definition might be used for initialization to generate the solution vector:

$$x_{ik} = \ l_k + \ rand(0, 1) \cdot (u_k - l_k). \tag{2.2}$$

In the employed bees phase, all bee swarms separate to the source field homogeneously and every single food source represents one honey bee. The employed bees search for new possible food sources (v_i) and this process is a kind of random neighbourhood search; so, they move from the memorized source (x_i) to the neighbour source to investigate the food quality and food level. Neighbour solutions around the initial solutions in a cycle have different fitness value and the honey bees evaluate them in this phase. The following equation defines the neighbourhood function to generate the neighbour solutions,

$$v_{ik} = \ x_{ik} + \phi_{ik} \ (x_{ik} - x_{mk}), \tag{2.3}$$

where x_m is randomly selected food sources, i is a randomly chosen parameter index and ϕ_{ik} is a random number positive or negative in range. After generating the neighbourhood food sources, the bees evaluate the sources and they make a greedy selection according to fitness values of neighbour food sources. The fitness value of the solution, $fit_i(x_i)$, can be calculated with the following equation,

$$fit_i(x_i) = \begin{cases} \dfrac{1}{1 + f_i(x_i)} & \text{if} \quad f_i(x_i) \geq 0 \\ 1 + \text{abs}(f_i(x_i)), & \text{if} \quad f_i(x_i) < 0 \end{cases} \tag{2.4}$$

where $f_i x_i$ is the objective function value of solution x_i.

After greedy selection between memorized food source and neighbour food sources, employed bees determine the best one in this neighbourhood. As a result of greedy selection, if there is a better valued food source than a memorized one, the bees memorize the new one and forget the existing food source and then they transfer this information to the hive as feedback.

In the onlooker bee phase, employed bees and onlooker bees as a subgroup of unemployed bees meet on the dancing area and employed bees share the information by performing a waggle dance to communicate to other bees. While employed bees perform the waggle dance, onlooker bees evaluate the dance figures and determine the food source. In the ABC algorithm, the food source selection according to their fitness value by onlooker bees can be formulated as in equation (2.5) given below:

$$p_i = \frac{fit_i(x_i)}{\sum_{i=1}^{SN} fit_i(x_i)}, \tag{2.5}$$

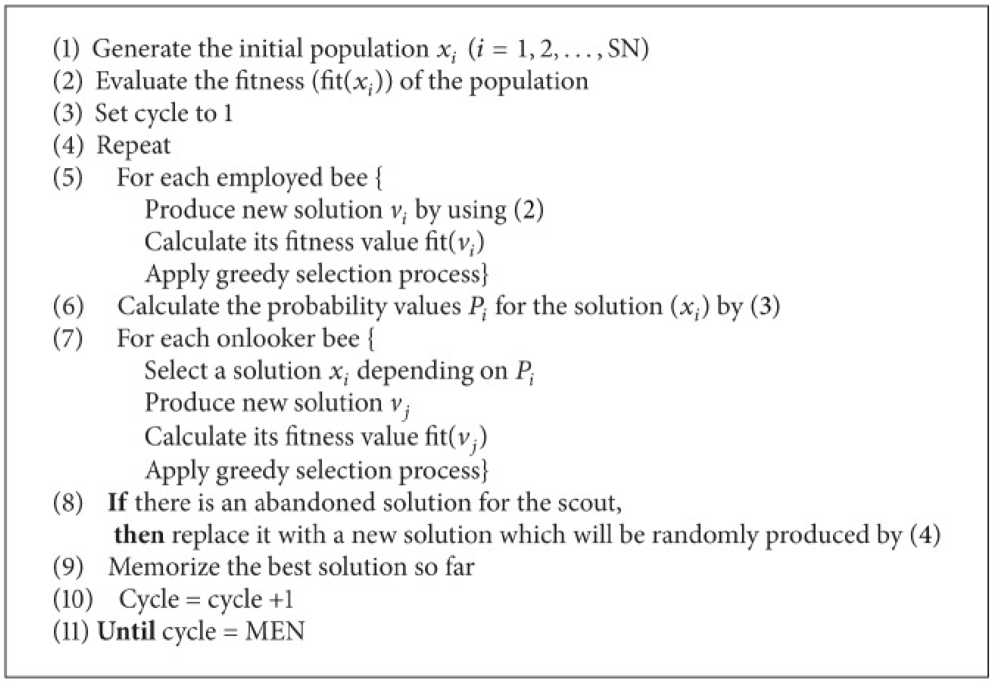
The probability value p_i with x_i can be calculated by using this expression. In order to select food sources according to their p_i value, a special technique can be used in the ABC algorithm such as the roulette wheel selection method [44].

The roulette wheel selection method works by calculating the cumulative sum of all fitness values. After this calculation in (2.5), random numbers which correspond to the range of the fitness of a solution in the cumulative sum are generated and the bee is assigned to the food source position according to these random values.

Scout bees which are the other type of unemployed bees, search for food sources randomly. If an employed bee cannot find better food sources around the predetermined one after several neighbourhood greedy searches, they abandon that area and inform scout bees about it. Scouts search for a new food source randomly and inform onlooker bees about the position of the source. In other words, in the ABC algorithm, if a solution cannot be improved for predetermined times, called the search limit, it is erased from memory and another solution is re-generated randomly instead of the abandoned one and memorized. In this study, erasing the information about the abandoned solution is investigated [46].

The ABC algorithm can be explained briefly as in Fig 2.6.

In addition, an ABC algorithm flowchart is demonstrated in Fig.6. The ABC algorithm has some disadvantages besides its advantages. For instance, many researchers have proposed several models to enhance the ABC algorithm because of its weak robustness (explained in section 2.6.2.3) characteristics especially.



```
(1)  Generate the initial population $x_i$ $(i = 1, 2, \ldots, \mathrm{SN})$
(2)  Evaluate the fitness ($\mathrm{fit}(x_i)$) of the population
(3)  Set cycle to 1
(4)  Repeat
(5)     For each employed bee {
           Produce new solution $v_i$ by using (2)
           Calculate its fitness value $\mathrm{fit}(v_i)$
           Apply greedy selection process}
(6)     Calculate the probability values $P_i$ for the solution $(x_i)$ by (3)
(7)     For each onlooker bee {
           Select a solution $x_i$ depending on $P_i$
           Produce new solution $v_j$
           Calculate its fitness value $\mathrm{fit}(v_j)$
           Apply greedy selection process}
(8)    **If** there is an abandoned solution for the scout,
        **then** replace it with a new solution which will be randomly produced by (4)
(9)    Memorize the best solution so far
(10)    Cycle = cycle +1
(11) **Until** cycle = MEN
```

Figure 2.6: Pseudocode of main body of ABC algorithm [11].

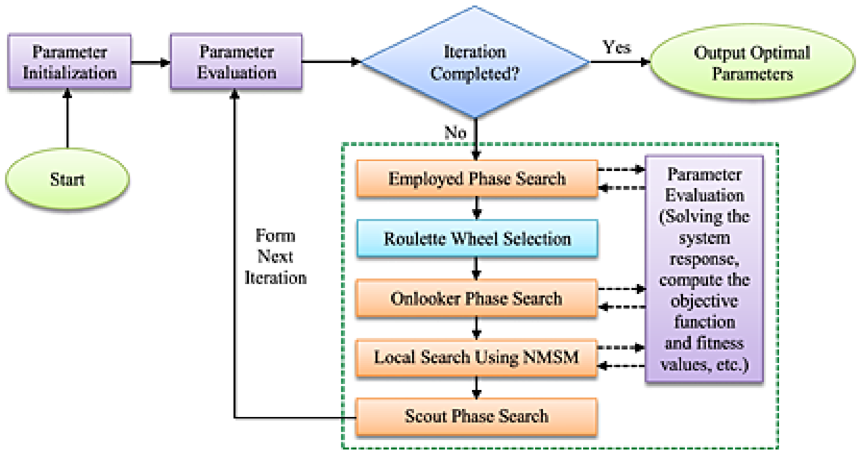


Figure 2.7: Flow chart of the hybrid artificial bee colony algorithm for a single independent run [80].

2.7 One Dimensional (1D) Bin Packing Problems

One dimensional bin packing (1DBP) problems can be considered as the easiest problem set in bin packing problems due to their simplicity. In the 1DBP, objects have a single dimension (cost, time, size, weight, or any number of other measures).

1DBP problem can be diversified such as stock cutting problems and weight annealing by the agency of their dimensional simplicity characteristic used by many researchers for their studies. Berberler et al [6] considered a 1D cutting stock problem and proposed a new heuristic algorithm with a new dynamic programming operator to solve it. They compared the obtained results of the proposed algorithm with other 1D cutting stock problems to illustrate its efficiency.

Xavier and Miyazawa [88] presented the hybrid approximation algorithms based on First Fit (Decreasing) and Best Fit (Decreasing) algorithms for a class constrained bin packing problems (CCSBP) in which items must be separated into non-null shelf divisions. The purpose of the paper in [88] was to illustrate performance characteristics of First Fit, Best Fit, First Fit Decreasing and Best Fit Decreasing algorithms according to implementation results of the proposed approximation.

In order to solve the Bin Packing (BP) problems, many heuristic methods have been developed in the literature. Some of the most popular algorithms are described in the following subsequent. In order to make clear the descriptions of the following heuristic algorithms, an example set of items is given so that the sequence is S= 4, 8, 5, 1, 7, 6, 1, 4, 2, 2 and the capacity of bins is 10.

- Next Fit (NF) Algorithm: Place the items in the order in which they arrive. Place the next item into the current bin if it fits. If it does not, close that bin and start a new bin [4]. The result of using this algorithm is shown in Fig: 2.8. The result of the (NF) algorithm is six bins and this result is clearly wasteful. However, it is acceptable if the information about free space in previous bins is not available or accessible.

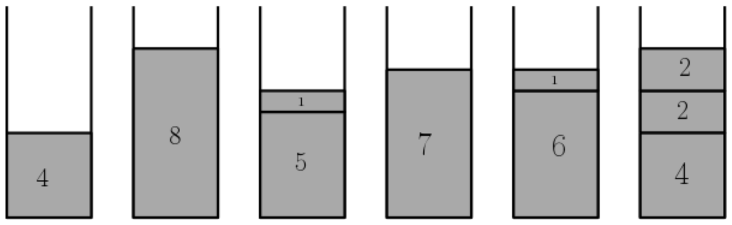


Figure 2.8: Bin Packing Under Next Fit Algorithm [4].

- First Fit (FF) Algorithm: Place the items in the order in which they arrive. Place the next item into the lowest numbered bin in which it fits. If it does not fit into any open bin, start a new bin [4]. The result of using this algorithm is shown in Fig: 2.9. The result of the (FF) algorithm is five bins and this algorithm requires a memory of previous bins.

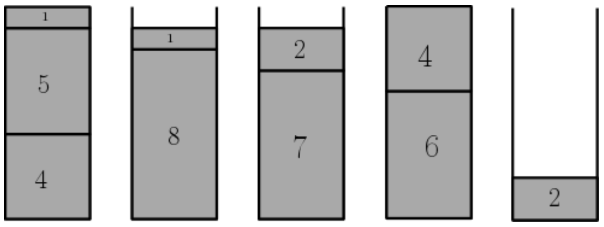


Figure 2.9: Bin Packing Under First Fit Algorithm [4].

.

- Best Fit (BF) Algorithm: Place the items in the order in which they arrive. Place the next item into that bin which will leave the least room left over after the item is placed in the bin. If it does not fit in any bin, start a new bin [4]. The result of using this algorithm is shown in Fig: 2.10. The result of the (BF) algorithm is five bins as well and this algorithm also requires a memory of previous bins. The BF algorithm generally obtains the best solution in online algorithms.

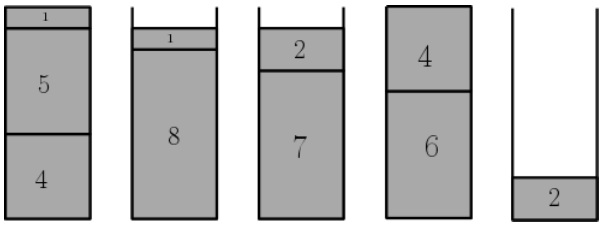


Figure 2.10: Bin Packing Under First Best Algorithm [4].

.

- Worst Fit (WF) Algorithm: Place the items in the order in which they arrive. Place the next item into that bin which will leave the most room left over after the item is placed in the bin. If it does not fit in any bin, start a new bin [4]. The result of using this algorithm is shown in Fig: 2.11 and the result of the (WF) algorithm is five bins. This algorithm is useful if all bins are desired to be the same weight approximately. However it may not be useful for the upcoming items.

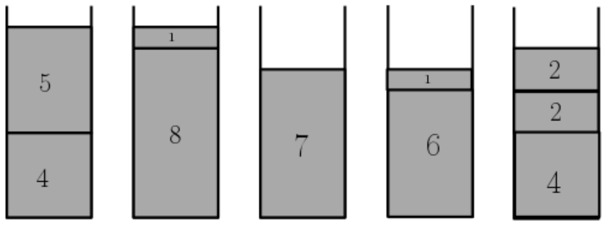


Figure 2.11: Bin Packing Under Worst Fit Algorithm [4].

.

- First Fit Decreasing and Best Fit Decreasing (FFD-BFD) Algorithm: Sort the items in decreasing order [4]. In this case, the FF or BF algorithm can be applied because they obtain the same results. The result of using these algorithms is shown in Fig: 2.12 and the result of the (FFD-BFD) algorithm is four bins as the best result by the agency of working with the offline supply system, because the information about items is available altogether.

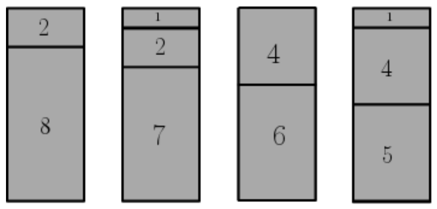


Figure 2.12: First Fit Decreasing and BestFit Decreasing [4].

Even searching via deterministic methods has some advantages such as convergence speed, starting with a closer point to optimum and constant search results; there are also some backward aspects to applying them. Basically, there are some types of problems such as Non-deterministic Polynomial-time (NP) not eligible for applying to deterministic methods based on Newtons Method Search and its improved forms. Even though deterministic methods are very useful especially for analytical problems, basic calculations for deterministic methods may have a high cost because of time wastage and calculation cost. Besides, there is a big risk of falling with the local optimum and hill climbing while searching for a global optimum. This means that searches for the global optimum may be misled and so, researchers may not benefit from the global optimum which also means extra cost.

2.8 Two Dimensional (2D) Bin Packing Problems

The 2-D bin packing approach is a generalization of the one dimensional problem. It is used to optimize any two parameters, the length and breath, breath and height or height and length and the third parameter is assumed to remain the same for all bins. This problem has many industrial applications, especially in optimization cutting (wood, cloth, metal, glass, etc.) and packing in transportations and warehouses. Csirik et al [20] used a renewal theory to solve the dual bin packing problems. Renewal theory is a branch of probability theory that generalizes Poisson processes for arbitrary holding times. They assigned bins of given size to all the largest possible number of containers, subject to constraint that the total size of the items assigned to any container is at least equal to 100%. They also carried out a probabilistic analysis to reveal the connections between the classical bin packing approaches and renewal theory.

Henrik Esbensen [27] implemented the concept of macro cell placement using the GA (Genetic Algorithm). Macro cell placement denotes the placement of cells in VLSI (Very Large Scale Integration), the layout which is the special case of 2-D bin packing problem. In this case, cells have to be placed in the larger rectangular layout either in horizontal or vertical position without any overlapping. Roy and Rajat [67] developed a parallel stochastic optimization algorithm for packing squares and rectangles into a larger rectangle of infinite length, such that the packed lengths should be at minimum as possible. Shim-Miin et al [40] packed the rectangles without overlapping into a larger rectangle using GA (Genetic Algorithms) to minimize the packing area, height of packing and number of containers required. A slicing tree was generated with branches as horizontal and vertical packing with the sub-branches of bins in it. The special hybrid crossover operator was used to divide the parents.

Jong et al [49] used Fuzzy logic techniques to pack non-rigid rectangles into an open rectangular prismatic bin. Fuzzy logic is a mathematical technique for imprecise data and problems that have many solutions rather than one. Lodi et al [57] reviewed the allocation of smaller rectangles into a set of standardized larger rectangles by minimizing the waste empty space. They compared the results of mathematical model, classical approximation algorithm, heuristic method and enumerative approach for minimizing the number of larger rectangles. Manhmood Amintoosi et al [2] used pattern matching methods for a special type of packing problem called tilling problems using hybrid GA and simulated annealing. The rectilinear of the standard unit size were taken to place them in a board without gab and overlapping. The chain rule was used to identify the boundary of the object by reducing the 2-D problem to a 1-D problem. This helped in the rotation of objects inside the board and also to identify the suitable position within the board.

Wenqi and Daugbing [39] compared GA, SA and heuristic algorithm for packing rectangles inside a rectangular container. Assumptions, namely, edges of the rectangles should be parallel to the boundary and should not overlap with each other, rectangles should be packed-horizontally or vertically were considered. They also insisted that first rectangles should be placed at a container corners. The concepts namely corner occupying action, cave occupying action and edge degree configuration were applied to pack the bins in the corners, between bins by touching them and without overlapping respectively. Zhang et al [92] introduced a recursive based heuristic GA for the rectangular pieces to form trips, strips were arranged in an ascending order and then packed in sequences, thereby the-waste space between rectangles were avoided. The remaining available empty space was - found using the recursive algorithm and the empty space was filled with the help of GA and it was found that the methodology performs better for longer benchmark problems. Crainic et al [19] developed a lower bound algorithm which dominates continuous Lagrangian model for packing bins into minimum number of containers without exceeding the volume of the container. Liu et al [55] solved the multi-objective bin packing problem using particles swarm optimization technique. This technique does not only reduces the waste empty space, but also satisfies the packing constraints namely, weight constraints, centre of gravity of bins, irregularly shaped bins and bin packing based on priority.

2.9 Three Dimensional (3D) Bin Packing Problems

The 2D and 3D bin packing problems could be more complicated than the 1D bin packing problems because of the loading complexity of items and bins. One of the main difficulties of the multi-dimensional packing problems is that, unusable spaces may not be rearranged as well as the waste in 1D bin packing problems [93]. Many heuristic approaches for multi dimensional bin packing problems are developed to solve the aforementioned complexities.

Three dimensional bin packing approaches - consider all three dimensions namely, length, breadth and height of the bin packing into a container. George [32] investigated the case of multiple containers loading for pipe packing. They followed three strategies for packing the pipes, the first strategy was sequential packing of pipes into a container the second was pre-allocation approach, in which rules were formulated for packing the pipes and the third was simultaneous packing of the pipes into multi-containers. Lauro et al [54] used a simple recursive uniform algorithm called n trial graph approach to pack n-dimensional bins into the minimum number of containers. Input bins were categorised based on bin symmetry and packed accordingly in separate containers or sub-containers to make homogeneity in containers.

Marcel and Christian [64] optimized the loading of bins into aircraft containers with the multi-objective of the maximizing number of packed bins, minimizing the fuel consumption and satisfying the stability criteria using the integer linear programming method. Air containers were divided into two along longitudinal axis of aircraft and bins have to be loaded on both containers simultaneously to maintain centre of gravity along the axis. Marco [8] developed an algorithm for packing the rectangular containers without any overlapping. The bin rotations at 90 degrees were allowed such that the edges should be orthogonal. Nihat and Anurag [47] applied an Augmented Neural-Network approach for solving classical bin packing problems which combines a priority rule heuristic with iterative learning approach of neural networks, to generate better solutions in less computation time. Artificial Neural Network (ANN) is an information processing system that is inspired by the way of biological nervous systems, namely the neurons and process information. Even though ANN resembles human neurons and process information, it requires data to train the neural network. This was the first time the approach has been applied to the bin packing problems. Four hundred and fifty two problems were solved to optimality. The average gab between the solution obtained and upper bound for all the problems were under 0.38% with computation time averaged below two seconds per problem.

2.10 Summary

The literature survey reveals various strategies and methodologies applied by the researchers in the field of bin packing problems. Generally, the items used for the bin packing problems are rectangular in shape. Literature proved that simple mathematical methods cannot be more appropriate for multi-objective and multi-constrained problems. The Researchers have also succeeded in using genetic approaches for problems involved with discrete and continuous variables. None of the literature in this study concentrated on, placement constraints, boundary crossing constraints, orientation constraints all together. In the literature researchers compared the developed algorithms with the lower bound values and have not considered much on the output format that is the format which practitioners can easily understand and implement the solution. Hence in this work, an attempt has been made to develop a conceptual framework for solving the 2D and 3D problems in the field of bin packing. The framework aids in developing an optimal solution for the multi-objective problem by satisfying the major packing constraints and industrial needs, while keeping the implementation procedure simple.

In the next chapter, the problem formulation for 2D bin packing problems is presented and the design of the first phase of our proposed algorithm is constructed based on the heuristic algorithms to provide efficient packing solutions to this class of problems. A couple of examples are given to demonstrate the success of the first phase packing solutions.

Chapter 3

Design Optimization of 2–Dimensional Bin Packing Problems

To design the packing strategy for 2–dimensional (2D) bin packing problems, we consider the cutting concept of this problem. The main goal in the basic form of the 2D bin packing problem is to find a best 2D cutting pattern for cutting a set of rectangular items from larger rectangular objects (bins) so that the total area of the items cut-out is maximized, and thus, minimizing the wasted area of the larger objects.

In the previous chapter, we presented a review of the relevant literature regarding known algorithms and linear programming strategies for solving bin packing problems. In this chapter, in section 3.1, we present the relevant mathematical models of the 2D bin packing problem. In section 3.2 and 3.3, we describe the placement strategies and in section 3.3 we summarise the chapter.

3.1 Problem Descriptions

Optimization can be loosely described as a process of evaluation of current options, with the intention of finding the best option. In other words it is the minimisation or maximisation of tasks. For example, when given a set of bins $S_1, S_2...$ with the same volume V and a list of n items with sizes $a_1, ..., a_n$ to pack, the partition involves finding an integer of bin B and a $B-partition\ US_1...US_B$ of the set $1, ..., n$ such that

$$\sum_{i \in S_k} a_i \leq V. \tag{3.1}$$

During packing the items may not overlap, however, they can be rotated by 90° if rotation is allowed. Rotation of the items might not be allowed if the items to be cut off are decorated with specific patterns (for example furniture manufacturing) or corrugated, whereas it is allowable in the case of plain material (e.g glass or steel). For simplicity, the thickness of the blade is ignored or guillotine and assume that items can touch each other. This assumption does not influence the algorithm in anyway, since the blade width can be simply incorporated into the dimensions of the items. The location of each item is represented by a pair of the coordination (x_i, y_i) $i\epsilon$ of its lower left corner. Each set of the coordinates is unique and called placement of I. The problem can be formulated for all $k = 1, ..., B$. A solution *optimal* if it has a minimal B. A possible formulation of the problem is:

Minimize:

$$B = \sum_{i=1}^{n} y_i \tag{3.2}$$

subject to

$$B \geq 1, \tag{3.3}$$

$$\sum_{j=1}^{n} a_j x_{ij} \leq V y_i, \qquad i \in \mathrm{N} = \{1, ..., n\}, \tag{3.4}$$

$$\sum_{i=1}^{n} x_{ij}, \qquad j \in \mathrm{N} = \{1, ..., n\}, \tag{3.5}$$

$$y_i, \in \{0, 1\}, \qquad i \in \mathrm{N} = \{1, ..., n\}, \tag{3.6}$$

$$x_{ij}, \in \{0, 1\}, \qquad i \in \mathrm{N} = \{1, ..., n\}, \tag{3.7}$$

where

$$y_i = \begin{cases} 1, & \text{if bin } i \text{ is used} \\ 0, & \text{otherwise} \end{cases}$$

$$x_{ij} = \begin{cases} 1, & \text{if bin } i \text{ is used} \\ 0, & \text{otherwise.} \end{cases}$$

This is based on the fact that the weights a_j are positive integers . Hence, without loss of generality, it is assumed that V is a positive integer,

$$a_j \leq V \ \forall j \ \in \mathrm{N}. \tag{3.8}$$

If assumption above is violated , V can be replaced by $|V|$. If an item violates assumption (3.8), then the instance is trivially infeasible. There is no easy way instead, of transforming an instance so as to handle negative weights. For the sake of simplicity it is also assumed that, in any feasible solution, the lowest indexed bins are used, for example $y_i \geq y_{i+1}$ for $i = 1, ..., n-1$. A thorough survey and analysis of the bin packing problem formulations can be found in [16].

In summary, the bin packing problem has been formulated and in the sections below we present the known heuristic algorithms. These heuristic algorithms are then applied to basic examples to understand their packing differences. This also helps us to identify the most efficient packing heuristic that we can evolve to make its solution better.

3.2 A new typology for packing problems

Researchers have covered a broad-spectrum of cutting and packing problems in general terms [25, 85]. However, in this section a novel classification or subtypology is specially developed for packing problems. This classification comprises six fields, denoted in array format,

α	β	χ	γ	λ	τ

,

Table 3.1: A new typology for packing problems

which is used throughout the remainder of this research. The most important characteristic in packing problems is dimensionality. It describes the geometry of the items to be packed. Therefore the entry in the first field of table 3.1, $\alpha \in \{1\text{D},2\text{D},3\text{D},\text{HoD}\}$, denotes the dimension in which the packing takes place. Here $\alpha = n\text{D}$ indicates that the problem is an n dimensional packing problem, for $n = 1; 2; 3$. Furthermore, $\alpha = \text{HoD}$ denotes a higher dimensional packing problem.

In one–dimensional packing problems, the widths of the items to be packed are equal to the width of the bin but the items have varying heights; hence only one dimension (namely height) is of importance. In two–dimensional packing problems, the items to be packed have varying widths and heights, while in three–dimensional packing problems, the items have varying widths, heights and depths. Higher dimensional packing problems involve packings in dimensions higher than three, where dimensions are not necessarily spatial the fourth dimension may represent time (for example, three dimensional items may be required to be packed for a fixed period of time).

The second field characterises the shapes of the objects to be packed and this is closely related to dimensionality. A packing of either regularly or irregularly shaped items may be required; hence $\beta \in$ {I,R}. Here β = R indicates that regular shaped items are packed, while β = I denotes the fact that irregular shaped items are packed. The notion of regular or irregular shapes is as defined by Hopper [37].

In packing problems, smaller items are packed into a well defined regions commonly called a bin or strip (depending on the application). The third field of the proposed classification notation in table 3.1 defines these regions as $\chi \in$ {MFB,MVB,SB,SP}. Here χ = MFB denotes a multiple fixed sized bin packing, χ = MVB denotes a multiple variable sized bin packing, χ = SB corresponds to a single bin packing and lastly χ = SP represents a strip packing problem.

In most applications, the items to be packed are drawn from a finite list. The fourth field in 3.1 differentiates the level of available information (specialisations of the list of items to be packed) as $\gamma \in =$ {Off,Aon,On}. Here γ = Off represents an off-line packing problem, γ = Aon indicates an almost on-line packing problem, and γ = On denotes an on-line packing problem.

The critical issue in packing problems is to make efficient use of time and space [76]. The objective in a packing problem is either to minimise or maximise a particular quantity and is addressed in the fifth field of the notation in table. In particular, $\lambda \in$ {MaI,MiA,MiB,MiC,MiS}. Here λ = MaI denotes maximising the number of items to be packed, λ = MiA denotes minimising the area of packing, λ = MiB denotes minimising the number of bins, λ = MiC denotes minimising the cost of the packing, while λ = MiS denotes minimising the strip height. Similar to other problems, packings are typically also subjected to certain constraints. The more basic and common constraints encountered in packing problems are dealt with in binary fashion in the sixth field of 3.1, in which $\tau = \{\tau_o, \tau_p, \tau_m, \tau_g\}$ is a binary 4–vector. The parameter $\tau \in$ {0,1} indicates whether the orientation of the items to be packed is fixed or not. Some applications allow for items within a bin or strip to be rotated while others do not. Here, $\tau = 0$ represents a fixed orientation, while $\tau = 1$ means that rotation is allowed.

The parameter $\tau_p \in$ {0,1} indicates that constraints on the placement of items are present. An example of such a constraint application may be that boxes carrying fragile material may not be placed at the bottom of a packing. In particular, $\tau_p = 0$ means that no restriction on the placement of items is enforced, while $\tau_p = 1$ indicates that restrictions on the placement of items are present.

The parameter $\tau_m \in$ {0,1} indicates whether the shape of the items to be packed may be modified. Shape modification may arise in the scheduling of tasks on a computer, where the length and width of an item may represent the time and resource required for completing a particular task. Shape modification usually takes place by either lengthening the item (thereby using less resource and more time) or widening the item (hence using more resource, but taking less time to complete a task). Here, $\tau_m = 0$ indicates that the shapes of items may not be modified, while $\tau_m = 1$ means that modification of shapes is allowed.

The parameter $\tau_g \in \{0,1\}$ represents the constraint of guillotine cuts. Some applications may disallow a certain packing pattern unless it may be disentangled by performing edge to edge cuts parallel to the edge of the bin or strip (i.e. unless it is a guillotine packing). Here, $\tau_g = 0$ means that there is no restriction on guillotinability, while $\tau_g = 1$ means that a guillotine packing is required.

Finally, the convention is adopted that an asterisk in any field of table 3.1 denotes the fact that the contents of that field are not specified, resulting in a class of packing problems rather than in a single packing problem. Due to the considerable diversity of real-world packing problems, this new classification scheme does not in any way cover all possible properties of packing problems. The characteristics covered are considered to be basic, but representative for packing problems. The classification has, however, been constructed to be flexible and may easily be adapted to suit any problem by adding additional properties to the fields in 3.1. Some fields, such as the constraints field, may be more detailed and the list of possibilities may increase when considering certain special cases or variants of packing problems. The classification notation introduced above is illustrated by means of an example and a summary is provided in tabular form in Appendix B.

3.3 Known heuristics

Heuristics are strategies for solving optimization problems approximately by constructing good, but not necessarily optimal solutions at a reasonable computational cost. Since the bin packing problem is NP-hard and its real world applications normally lead to large scale problems, practitioners usually resort to heuristic methods rather than to exact methods. The first class of heuristics, referred to as *level algorithms*, is described in 3.3.1 and for this research work we only focus on these algorithms.

3.3.1 Level algorithms

The workings of all heuristic algorithms considered in this section are illustrated by means of a single example and formalised by means of pseudocode listing. The use of the pseudocode is preferred instead of a mere description or specic programming language listing, because of its precision, structure and universality [35].

	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8	L_9	L_{10}
width, $w(L_i)$	14	2	9	10	4	4	2	8	3	5
Hight, $h(L_i)$	4	4	3	5	10	1	6	2	3	1

Table 3.2: Rectangle dimensions used as an example throughout this dissertation to illustrate the steps of the various algorithms.

The general notation used throughout this chapter and the section of heuristics is that, for a given list L of rectangles, where $h(Li)$ and $w(Li)$ denote respectively the height and width of rectangle Li, $\forall\ i = 1, ..., n$ (here n denotes the number of rectangles to be packed). The dimensions of the rectangles to be packed into a strip of width 16 spatial units in the example used throughout the heuristic development section may be found above in Table 3.1.

Each item is packed in the order given on any one of a collection of horizontal levels drawn across the strip in a level algorithm. These algorithms are primarily used when dealing with offline packing problems and the first level of the strip is the bottom of the strip. The height of each level is determined by the height of the tallest rectangle placed on the previous level (a horizontal line is drawn through the top of the tallest rectangle placed on the previous level).

3.3.1.1 The next fit decreasing height algorithm

The so-called *next fit decreasing height* (NFDH) algorithm [13] was developed in 1980 to solve problems of the form:

2D	R	SP	Off	MiS	0,0,0,1

Table 3.3: Typology for packing problems (explained in section 3.2).

The list of rectangles to be packed is pre-ordered according to non-increasing height. This allows for the first rectangle placed on a level to determine the height of the next level. A rectangle is placed on the current level, left justified, if it fits there. However, if it does not fit, then a new level is created above the current level (which becomes the new current level) and the rectangle is placed there. The packing progresses from left to right per level and from the bottom of the strip upwards level-wise. Levels lower than the current level are never revisited. Rectangles of equal height retain their original order in the packing list relative to each other. When analysing the performance of the NFDH algorithm, Coman et al. [13] found the asymptotic performance bound.

$$NFDH(L) \leq 2\ OPT(L) + 1 \tag{3.9}$$

in the limit as n $\rightarrow \infty$, where $NFDH(L)$ denotes the packing height achieved for the list of rectangles (L) by the NFDH heuristic, and where OPT(L) is the optimal packing height for the list of rectangles L. In (3.9) the multiplicative constant 2 is the smallest possible. The steps of the NFDH algorithm are given in pseudocode as Algorithm 3.1.

Algorithm 3.1: The next fit decreasing height (NFDH) algorithm
Description: Packing a list of rectangles into a strip of fixed width and infinite height. The list of rectangles is fully specified in advance, before packing commences.
Input: The number of rectangles to be packed n, the dimensions of the rectangles $\langle w(L_i), h(L_i)\rangle$ and the strip width W.
Output: The height of a packing obtained in the strip.

```
1: level ← 0; $h$(level) ← 0; $w$(level) ← 0; $i$ ← 1
2: Renumber the rectangles in order of non-increasing height such that $h(L_1) \geq h(L_2) ... \geq h(L_n)$
3: Pack rectangle $L_i$ left-justified at the bottom of the strip
4: $h$(level) ← $h(Li)$; $w$(level) ← $w(L_i)$
5: for $i = 2, ..., n$ do
6:    if $W - w$(level) $\geq$ $w(L_{i+1})$ then
7:        pack rectangle $L_{i+1}$ to the right of rectangle $L_i$
8:        $w$(level) ← $w$(level) + $w(L_{i+1})$
9:    else [$W - w$(level) $<$ $w(L_{i+1})$]
10:       create a new level above the previous one and pack rectangle $L_{i+1}$ on the new level
11:       level ←level + 1; $w$(level) ← $w(L_{i+1})$; $h$(level) ← $h$(level - 1) + $h(L_{i+1})$
12:   end if
```

```
13: end for
14: print $H = h(\text{level})$.
```

Example 3.1 *The rectangles in Table 3.2, which are required to be packed into a strip of width 16 units, are first arranged in non-increasing order by height as* $\{L_5, L_7, L_4, L_2, L_1, L_9, L_3, L_8, L_{10}, L_6\}$ *and then packed, as shown in Figure 3.2. Rectangles* $\{L_5, L_7, L_4\}$ *are packed on the first level. This level is closed off and a new level is created where rectangles* $\{L_2, L_1\}$ *are packed. Packing progresses in this manner until all rectangles are packed and a total packing height of 20 units is obtained.*

Since lines 6 – 12 of Algorithm 3.1 have constant time complexity, the for-loop spanning lines 5 – 13 has a time complexity of $O(n)$, where n is the number of rectangles in the list L. However, the complexity of the algorithm is dominated by line 2 which has a worst-case time complexity of $O(n\ log\ n)$, since line 1, lines 3 – 4 and line 14 also have constant time complexity. This worst-case time complexity may be obtained when using an efficient sorting procedure such as the merge-sort algorithm which uses a divide-and conquer technique [41]. Therefore the worst-case time complexity of the NFDH algorithm is $O(n\ log\ n)$.

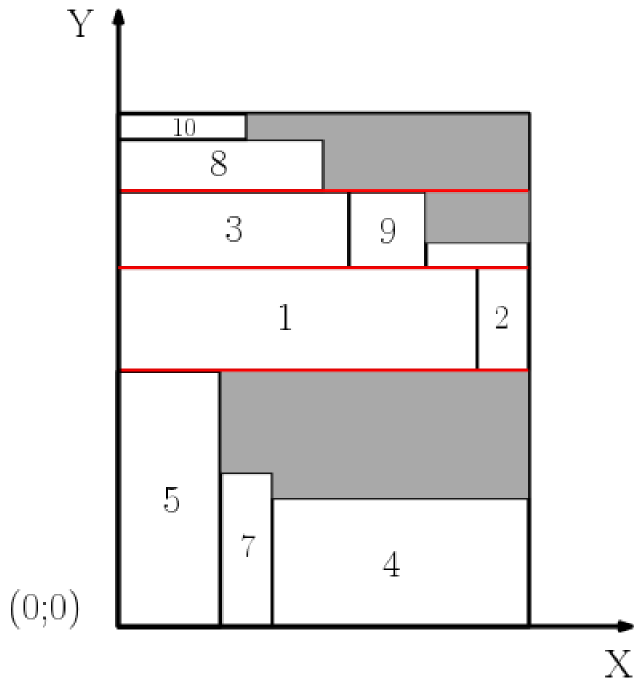


Figure 3.1: The NFDH algorithm packing.

3.3.1.2 The first fit decreasing height algorithm

In the so-called *first fit decreasing height* (FFDH) algorithm [13], the list of rectangles is also pre-ordered according to non-increasing height. The algorithm also dates from 1980 and may again be applied to problems of the form:

2D	R	SP	Off	MiS	0,0,0,1

.

Rectangles of equal height retain their original order relative to each other in the packing list. A rectangle is placed left justified on the lowest level with sufficient space. The strip is searched level-wise from the bottom upwards for sufficient packing space and if the current rectangle does not fit into any of the existing levels, a new level is created above the current top level (which becomes the new top level) and the rectangle is placed there. The difference between the NFDH and FFDH algorithms is therefore that in the latter, previously packed levels are always searched for sufficient space to pack a rectangle whereas in the former, previously packed levels may not be revisited. When analysing the performance of the FFDH algorithm, Coffman et al. [13] found the asymptotic performance bound such that

$$FFDH(L) \leq 1.7\, OPT(L) + 1 \tag{3.10}$$

in the limit as $n \to \infty$, where $FFDH(L)$ denotes the packing height achieved for the list of rectangles L by the FFDH heuristic and $OPT(L)$ is the optimal packing height for the list of rectangles L. The multiplicative constant 1.7 in (3.10) is smaller than that of the NFDH algorithm hence the FFDH algorithm generally performs better than the NFDH algorithm for large packing lists. In fact for all packing lists $FFDH(L) \leq NFDH(L)$ [13]. A pseudocode listing for the FFDH algorithm is given as Algorithm 3.2.

Example 3.2 *The rectangles are ordered in the same manner as in the NFDH algorithm, according to non-increasing height as $L_5, L_7, L_4, L_2, L_1, L_9, L_3, L_8, L_{10}, L_6$. The packing produced is similar to the NFDH algorithm packing, except that rectangle L_6 is packed on the lowest level with sufficient space, which is the third level. A total packing height of 19 units is obtained for the example instance in Table 3.2, as shown in Figure 3.2.*

Since lines 6 – 14 of Algorithm 3.2 have constant time complexity and the while-loop on line 7 does not depend on n, the for-loop spanning lines 5 – 18 has a time complexity of $O(n)$. Line 1, lines 3 – 4 and line 19 also have constant time complexity. Line 2 therefore dominates the time complexity of the algorithm and has a worst-case time complexity of $O(n\ log\ n)$ when using an efficient sorting procedure such as the merge-sort algorithm [41]. Consequently, the overall worst-case time complexity of the FFDH algorithm is also $O(n\ log\ n)$.

Algorithm 3.2: The first fit decreasing height (FFDH) algorithm
Description: Packing a list of rectangles into a strip of fixed width and infinite height. The list of rectangles is fully specified in advance, before packing commences.
Input: The number of rectangles to be packed n, the dimensions of the rectangles $\langle w(L_i), h(L_i)\rangle$ and the strip width W.
Output: The height of a packing obtained in the strip.

```
1:  level ← 0; $h$(level) ← 0; $i$ ← 1;LevelNum ← 1
2:  Renumber the rectangles in order of non-increasing height such that $h(L_1) \geq h(L_2) ... \geq h(L_n)$
3:  Pack rectangle $L_i$ left-justified at the bottom of the strip; $h$(level ← 1) ← $h(L_i)$
4:  for $i = 2, ..., n$ do
5:    search all levels (starting with the bottom) for the lowest with sufficient space
6:        if such a level exists then
7:        pack rectangle $L_i$ left justified on that level
8:     else [there is insufficient space in all existing levels]
9:        levelNum ←levelNum + 1; level←levelNum; $h$(level) ← $h$(level - 1) + $h(L_i)$
10:       pack rectangle on new level
11:    end if
12: end for
13: print $H = h$(level).
```

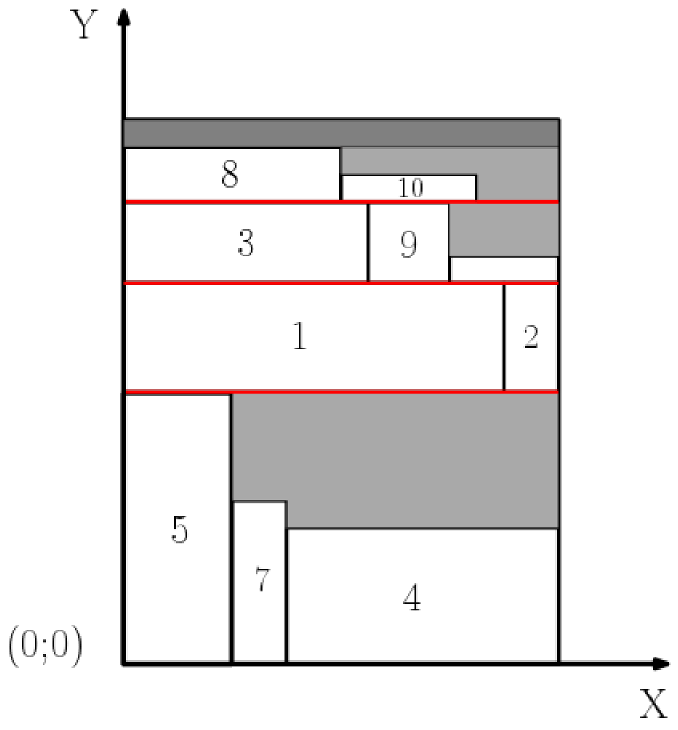


Figure 3.2: The FFDH algorithm packing.

3.3.1.3 The best fit decreasing height algorithm

The *best fit decreasing height* (BFDH) algorithm [15] dates from 1990 and may also be applied to problems of the form

2D	R	SP	Off	MiS	0,0,0,1

.

It is analogous to the FFDH algorithm, except that in this algorithm rectangles are placed left justified towards the right of the last rectangle packed on the level with *minimum residual horizontal space*. This means that, to pack the next rectangle, all existing levels are searched for sufficient space and the area of the horizontal space towards the right of the level packing that would remain utilised if the rectangle were to be placed in any of the levels is computed. The rectangle is placed on the level that leaves the smallest horizontal space. Dogrusoz [24] found the asymptotic performance bound of the BFDH algorithm also to be

$$BFDH(L) \leq 1.7\ OPT(L) + 1 \tag{3.11}$$

in the limit as $n \to \infty$, where BFDH(L) denotes the packing height achieved for the list of rectangles L by the BFDH heuristic and OPT(L) is the optimal packing height for the list of rectangles L. Coffman [14] performed an asymptotic average-case performance analysis to establish the expected value of the height of a packing produced by the BFDH algorithm as

$$E[BFDH(L)] = \frac{n}{4} + \Theta(\sqrt{n}\ \log^{\frac{3}{4}} n) \tag{3.12}$$

as $n \leftarrow \infty$, where n is the number of rectangles in the list L, where BFDH(L) denotes packing height achieved for the list of rectangles L by the BFDH heuristic, where$E[BFDH(L)]$ denotes the expected value operator and where Θ denotes the asymptotically tight order notation. Coffman [12] assumed that all $2n$ variables $w(L_1)$,...,$w(L_n)$,$h(L_1)$,...,$h(L_n)$ are independent and uniformly random samples from the interval [0,1] and that the strip width is one. The average-case analysis in (3.12) seeks to find a function $f(n)$ for expressing the expected packing height obtained by an algorithm such that

$$E[A(L)] = \sum_{i=1}^{n} w(L_i) + \Theta(f(n)), \tag{3.13}$$

where $A(L)$ is the packing height produced by an algorithm A for the list L and $\Theta(f(n))$ represents the wasted area of the packing [12]. The first term in (3.12) is $n/4$ because the n rectangles are assumed to be independent. Hence the packing produced on average by any algorithm must occupy at least $n/4$ units of space. This is because both the average width and average height of the rectangles in the uniform model is 1/2 therefore the average area of a rectangle is 1/4. For proof of (3.12), the reader is referred to [14]. The algorithm appears in pseudocode form as Algorithm 3.3.

Example 3.3 *When the BFDH algorithm is applied to the example instance in Table 3.1, a total packing height of 19 units is also obtained. The packing produced is the same as that produced by the FFDH algorithm, as shown in Figure 3.3. Rectangle L_6 is again packed on the third level, which is the level with minimum residual horizontal space.*

The overall worst-case time complexity of the BFDH algorithm is also $O(n\ log)$. This is due to the dominant worst-case complexity in line 2 of $O(n\ log\ n)$ when using an efficient sorting procedure such as the merge-sort algorithm [41], since lines 1, 3 – 4 and 6 – 17 have constant time complexity. The for-loop spanning lines 5 – 19 has worst-case time complexity $O(n)$.

Algorithm 3.3: The best fit decreasing height (BFDH) algorithm
Description: Packing a list of rectangles into a strip of fixed width and infinite height. The list of rectangles is fully specified in advance, before packing commences.
Input: The number of rectangles to be packed n, the dimensions of the rectangles $\langle w(L_i), h(L_i)\rangle$ and the strip width W.
Output: The height of a packing obtained in the strip.

```
1: level $\leftarrow 0; h$(level) $\leftarrow 0; i \leftarrow 1$;LevelNum $\leftarrow 1$
2: Renumber the rectangles in order of non-increasing height such that $h(L_1) \geq h(L_2) ... \geq h(L_n)$
3: Pack rectangle $L_i$ left-justified at the bottom of the strip; $h$(level $\leftarrow 1) \leftarrow h(L_i)$
4: for $i = 2, ..., n$ do
5:    search all levels (starting with the bottom) for the lowest with sufficient space
6:        if such a level exists then
7:        pack rectangle $L_i$ left justified on that level
8:     else [there is insufficient space in all existing levels]
9:        levelNum $\leftarrow$levelNum + 1; level$\leftarrow$levelNum; $h$(level) $\leftarrow h$(level - 1) $+ h(L_i)$
10:        pack rectangle on new level
11:     end if
12: end for
13: print $H = h$(level).
```

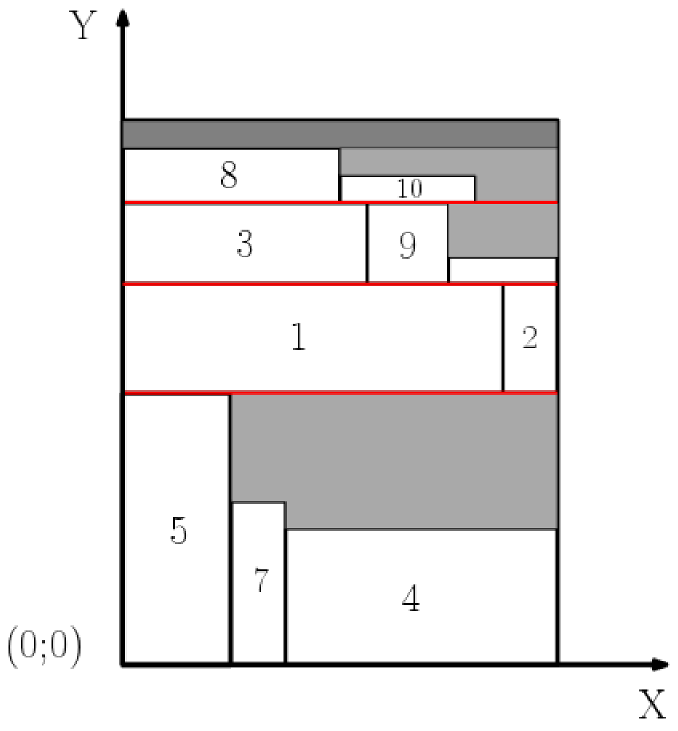


Figure 3.3: The FFDH algorithm packing.

3.4 Summary

Previous sections of this chapter have explained how heuristic algorithms are applied to problem instances of 2D packing in order to obtain efficiency packing solutions, and what a heuristic consists of. We draw our inspiration from the $First - Fit\ Decreasing\ Hight$ $(FFDH)$ and this approach is used as the first phase during problem solving of the practical problems considered in this research work. In order to make more wide application of these heuristics (in chapter 6), the constraints are set by following the industrial requirements as described in chapter 1.

Chapter 4

Design Optimization of the 3-Dimensional Bin Packing Problem

The main goal in the basic form of the three-dimensional (3D) bin packing problem is to find a best 3D packing pattern for loading a set of rectangular boxes into a container so that the total volume of the boxes loaded is maximized, and the boxes do not overlap. The bin packing problem initially appears fairly simple. However, scholars have found its behaviour rather complex.

In practical applications, there are several issues in the production and the transportation planning directly modelled by the 3D bin packing problem, which includes warehouse management, container ship loading, plane cargo management and pallet loading. For these enterprises, possessing an efficient utilization of materials and the transportation capacity is a significant competitive advantage. Therefore, the requirement to improve the efficiency of maximizing the utilization is strongly necessary.

Section 4.1 below introduces the overall detail of the 3D bin packing problem. In section 4.2, we present the relevant mathematical models. In the section 4.3, we describe the placement strategy.

4.1 Problem Definition

The problem of loading boxes into containers can be classified into four variants [70]: the Strip Packing Problem (SPP), the Bin packing Problem (BPP), the Multi-Container Loading Problem (MCLP), and the Knapsack Loading Problem (KLP or CLP). The SPP considers a container of which two dimensions are fixed (e.g., width and height), and the third dimension (e.g., length) is a variable. The problem is to decide how to pack all boxes of different sizes inside the container, so that the variable dimension (length) is minimized (e.g., [58, 84, 90]). For the BBP, the aim is to find a minimal number of container (Bin) to load all boxes of different sizes (e.g., [36, 50, 59, 77]). Unlike the BPP, the containers in the MCLP do not necessarily have the same sizes and costs, and the problem is to decide how to load all the boxes so that the total cost of the chosen subset of containers to be loaded is minimized (e.g., [26, 74]).

This chapter mainly focuses on the Container Loading Problem (CLP). The number of the container in this problem is only one with fixed size. There are more than one group of packing items, each item in the same group is assigned the same size. The objective of this problem is to maximize the space utility of the container.

4.1.1 Practical Requirements

There are twelve practical considerations in [72], which can be used to model more realistic container loading problems. It is perhaps necessary to emphasize that no claim is made that the factors described are of importance in every case. What is claimed, however, is that there are many cases in which some of the factors listed below play an important role.

- **Orientation Constraints:**

  The instruction is usually seen on cardboard boxes. It is a simple example of this kind of restriction. However, it may not only be the vertical orientation which is fixed, but also the horizontal orientation is restricted. For instance, a two-way entry pallet is loaded by forklift truck.

- **Load Bearing Strength of Items:**

  Stack no more than x items high is another instruction seen on many boxes in many situations. This constraint can be considered as a straightforward figure for the maximum weight per unit of area on which a box can support dependent on its construction and also its contents. Usually the side walls of a cardboard box provides the bearing strength, so that it might be acceptable to stack an identical box directly on top, whereas placing an item of half the size and weight in the centre of the top face causes damage. The load bearing ability of an item may, of course, also depend on its vertical orientation.

- **Handling Constraints:**

The items of positioning within a container is usually determined by its size, or weight and the loading equipment. For instance, the large items should be placed on the container floor, or to fix its position below a certain height. It may also be desirable from the viewpoint of easy/safe materials handling to place certain items near the door of the container.

- **Load Stability:**

  If the cargo is easily damaged, to ensure that the load cannot move significantly during transport is necessary. Also, during loading and (especially) unloading operations, an unstable load can have important safety implications. For handling this constraint, some devices will be used to restrict or prevent cargo movement, such as Straps and Air-bags. However, the cost can be considerable, especially in terms of time and effort spent.

- **Grouping of Items:**

  A load might be easy to operate when the items belong to the same group. For instance, several items that are defined by a common recipient or the item type are positioned in close proximity. It may also have advantages in terms of the efficiency of loading operations.

- **Multi-drop Situations:**

  In order to avoid unloading and reloading a large part of the cargo several times when the container is to send consignments for a number of different destinations, the items have to be loaded within the same consignment closely and to order the consignments within the container.

- **Separation of Items within a Container:**

  To ensure that cargo which may adversely affect some of the other goods separate to load is necessary. For instance, if they include both food stuffs and perfumery articles, or different chemicals, this constraint has to be taken into account.

- **Shipment Priorities:**

  In the real world, the shipment of some items will be given more priorities to delivery when these items are more important than the others, for instance, delivery deadlines or the shelf life of the product concerned. More specifically, the item might have a priority rating. Depending on the practical context, this rating may represent an absolute priority. In the sense if no item in a lower priority class should be shipped this causes items with higher rating to be left behind. This may have a relative character, reflecting the value placed on inclusion in the shipment without debarring trade-offs between priority classes merely.

- **Weight Distribution within a Container:**

  From the viewpoint of transporting and handling the loaded container- such as lifting it onto a ship, it is desirable that its centre of gravity be close to the geometrical mid-point of the container floor. If the weight is distributed very unevenly, certain handling operations may be impossible to carry out. In cases where a container is transported by road at some stage of its journey, the implications of its internal weight distribution for the axle loading of a vehicle can be an important consideration. The same, of course, applies if the 'container' is a truck or trailer.

- **Container Weight Limit:**

  If the cargo to be loaded is fairly heavy, the weight limit of a container may represent a more stringent constraint than the loading space available.

Although the aforementioned studies consider the practical issues but they rarely present the mathematical formulation. Some papers, such as [5, 61], present formulations for two-dimensional cutting and packing problems that can be easily extended to the three dimensional container loading problem.

4.2 Mathematical Formulations

The definition of the mathematical model of the three dimensional bin packing problems in this research work is referred to the literature [42]. In this definition, the item can be presented as different types of boxes with given length l_i, width w_i, height h_i, value v_i, and a maximum quantity b_i, $i = 1, ..., m$ which can be loaded inside the object (container, truck, rail-road car or pallet) with given length L, width W, and height H (when considering a pallet, H is the maximum allowed height of the cargo loading). The dimensions of the boxes are integer, and they can only be placed orthogonally into the container.

This last assumption can be easily relaxed in the models presented and here it is considered only to simplify the presentation of the formulations.

The back-bottom-left corner of the container can bee seen as the origin of the Cartesian coordinate system, and the possible coordinate where the back-bottom-left corner of a box can be placed is represented by (x; y; z). These possible positions along axes L, W, and H of the container belong to the sets: $X = \{0, 1, 2, . . ., L - \min_i(\mathrm{l}_i)\}$, $Y = \{0, 1, 2,. . . , W - \min_i(\mathrm{w}_i)\}$, $Z = \{0, 1, 2,. . . ,H - \min_i(\mathrm{h}_i)\}$, respectively. As the view of [5, 87], for a given packing pattern, each packed box could be moved back or down and/or the left, until its back, bottom and left-hand face are adjacent to other boxes or the container. Due to each type box can rarely be packed completely, the number of practical packed items can be represented by ε_i,$\varepsilon_i \leq\ b_i$. Thus, without loss of generality, the sets X, Y and Z can be expressed as below:

$$X\ = \{x|x = \sum_{i=1}^{m} \varepsilon_i L_i,\ 0 \leq x \leq L{-}\min_i(L_i),\ 0 \leq \varepsilon_i \leq b_i, i = 1, ..., m\} \tag{4.2}$$

$$Y\ = \{y|y = \sum_{i=1}^{m} \varepsilon_i W_i,\ 0 \leq x \leq W{-}\min_i(W_i),\ 0 \leq \varepsilon_i \leq b_i, i = 1, ..., m\} \tag{4.4}$$

$$Z\ = \{z|z = \sum_{i=1}^{m} \varepsilon_i H_i,\ 0 \leq x \leq H{-}\min_i(H_i),\ 0 \leq \varepsilon_i \leq b_i, i = 1, ..., m\} \tag{4.6}$$

A possible placement of a box of type i inside the container is depicted by figure 4.1. To describe the constraints that avoid overlapping of boxes inside the container we define $c_{ixyzx'y'z'}$, $(i = 1, ..., m), (x, x' \in X), (y, y' \in Y), (z, z' \in Y)$ as

$$c_{ixyzx'y'z'} = \begin{cases} 1, & \text{if a box of type } i \text{ placed with its back-bottom-left corner at } (x, y, z,); \\ 0, & \text{otherwise} \end{cases}$$

The mapping $c_{ixyzx'y'z'}$ is not a decision variable and it is computed a priori as below:

$$c_{ixyzx'y'z'} = \begin{cases} 1, & \text{if } 0 \leq x \leq x' \leq x + l_i - 1 \leq L - 1; \\ & 0 \leq y \leq y' \leq y + w_i - 1 \leq W - 1; \\ & 0 \leq z \leq z' \leq z + h_i - 1 \leq H - 1; \\ 0, & \text{otherwise} \end{cases}$$

Before packing the item in the position (x,y,z), the algorithm can check the feasibility of this placement based on the above formula. Here, the (x',y',z') represents the position of packed item. a feasible packing solution should hold all $c_{ixyzx'y'z'}$ to be zero.

Let $X_i = \{x \in X | 0 \leq x \leq L - l_i\}, Y_i = \{y \in Y | 0 \leq y \leq W - w_i and \{z \in Z | 0 \leq z \leq H - h_i\}, i = 1, ..., m$. The decision variables $a_{ixyz}, i = 1, ...m, x \in X_i, y \in Y_i, z \in Z_i$, of the model are defined as

$$a_{ixyzx'y'z'} = \begin{cases} 1, & \text{if a box of type } i \text{ is placed;} \\ & \text{with its back-bottom-left corner at the postion}(x, y, z); \\ & \text{so that } 0 \leq x \leq L - l_i, 0 \leq y \leq W \leq w_i and 0 \leq z \leq H - hi; \\ 0, & \text{otherwise} \end{cases}$$

The single container loading problem that is without additional considerations

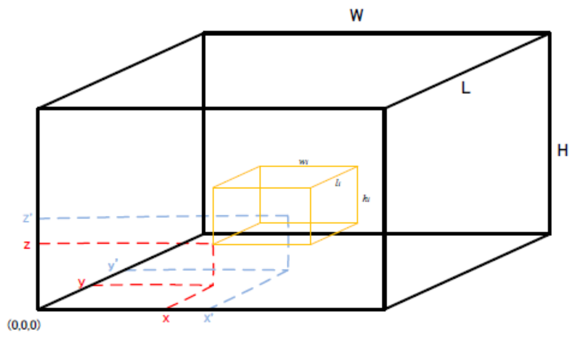


Figure 4.1: Example of placement of a box of type i inside a container.

can be written as a direct extension of a $0-1$ integer linear programming model proposed in [5] for the two dimensional non-guillotine cutting problems:

$$max \sum_{i=1}^{m} \sum_{x \in X_i} \sum_{y \in Y_i} \sum_{z \in Z_i} v_i.\ a_{ixyz} \tag{4.7}$$

$$100\% \frac{\sum_{k=1}^{K} v_k NP_k}{L.W.H} \rightarrow max \tag{4.8}$$

$$\sum_{i=1}^{m} \sum_{x \in X_i} \sum_{y \in Y_i} \sum_{z \in Z_i} c_{xyzx'y'z'}.\ a_{ixyz} \leq 1,\ \ x' \in X,\ \ y \in Y,\ \ z' \in Z. \tag{4.9}$$

$$max \sum_{x \in X_i} \sum_{y \in Y_i} \sum_{z \in Z_i} a_{ixyz} \leq b_i,\ \ i = 1, ..., m. \tag{4.10}$$

$$a_{ixyz} \in \{0,1\},\ i = 1, ..., m. \quad x \in X_i,\ y \in Y_i,\ z \in Z_i. \tag{4.11}$$

For formulations (4.7-4.11), the objective function (4.7) aims to maximize the total value of the boxes packed inside the container (if $v_i = (l_i.w_i.hi)$, (4.7) maximizes the total volume of the boxes). The objective function (4.8.) is extended from (4.7) in order to calculate the maximal container utility rate, in which NP_k is the number of the type k box packed in a solution, v_k is the volume of a box of the type k and the denominator represents the volume of the container. Constraints (4.9) avoid the overlapping of the boxes packed, constraints (4.10) limit the maximum number of boxes packed, and constraints (4.11) define the domain of the decision variables.

4.3 Placement Strategy

In order to calculate the fitness, the space utility rate, of a solution in bin packing problems, the evaluation component in the relevant algorithm has to simulate the packing process that packs the item into the container by following particular rule when the packing pattern is given. In this project, the deep bottom left method is selected, which is widely used in many literatures. This placement strategy is divided into three main components: the Maximal-Empty-Spaces Selection, the Deep Bottom Left Packing Procedure, and the Empty-Spaces Update.

4.3.1 Maximal-Empty-Space Selection

There is a list S, which saves all empty maximal-spaces (EMSs) that are largest empty parallelepiped spaces available for filling with boxes. The EMSs are represented by their vertices with the minimum and the maximum coordinates (x_i, y_i, z_i and X_i, Y_i, Z_i). When an EMS has been searching in the list S to pack a box, only the EMS with minimum vertices coordinate (x_i, yi, zi) is considered. The initial stage of list S only has one element, the size which is equal to the size of the container. After each time packing a box, this list is updated and reordered by the Difference Process (DP), shown in the section 4.3.2, and the Deep-Bottom Left Procedure (DBLP), described in the section 4.3.3.

4.3.2 Empty-Spaces Update

In order to keep track of the EMSs, the Difference Process (DP) is introduced, which is developed by Lai and Chan [10]. To demonstrate this method we are assisted by an example of the application of the DP process in a 2D packing instance as shown in Fig.4.2.

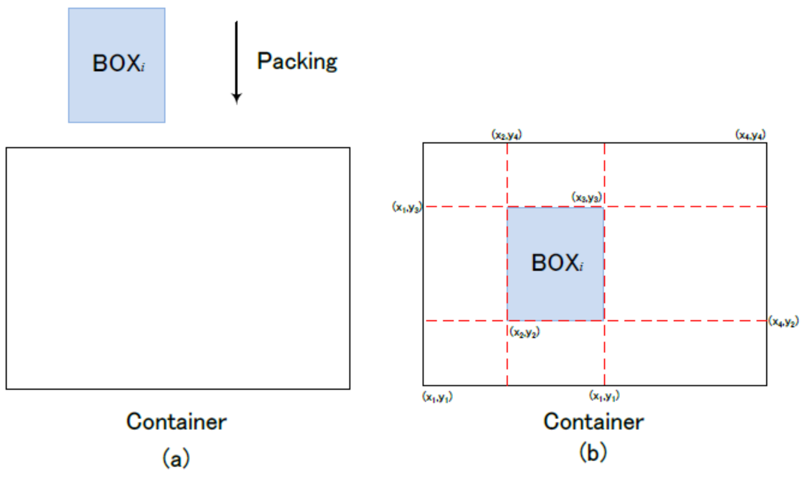


Figure 4.2: The Example of Using The Difference Process.

- **New Empty Space Generation**

  As shown in Fig.4.2(a), the Box$_i$ has to be packed into the container. Suppose that the bottom left corner of the Box$_i$ can be located in the position of the container as shown in Fig.4.2(b), the bottom-left and the top right corners of the container and the Box$_i$ are represented by four points, $\{(x_1, y_1), (x_4, y_4)\}, \{(x_2, y_2), (x_3, y_3)\}$. After packing Box$_i$, the current empty space that was used to load the Box$_i$ is divided into at most four new EMSs. Each of these EMSs is represented by two points that are calculated by the following rules:

  Difference: $[(x_1, y_1), (x_4, y_4)] - [(x_2, y_2), (x_3, y_3)]$, where

$$\text{NewEMS}_1 = [(x_1, y_1), (x_2, y_4)] \tag{4.12}$$

$$\text{NewEMS}_2 = [(x_1, y_1), (x_4, y_2)] \tag{4.13}$$

$$\text{NewEMS}_3 = [(x_1, y_3), (x_4, y_4)] \tag{4.14}$$

$$\text{NewEMS}_4 = [(x_3, y_1), (x_4, y_4)]. \tag{4.15}$$

In the 2D problem, the maximal empty spaces are the area that is between two sides which are parallel and belong to the container and Box$_i$, respectively. Thus, to extend this rule to the 3D problem a maximal empty space can be seen as the space between two surfaces which are parallel and belong to the container and the Box$_i$ as depicted by the Fig.4.3.

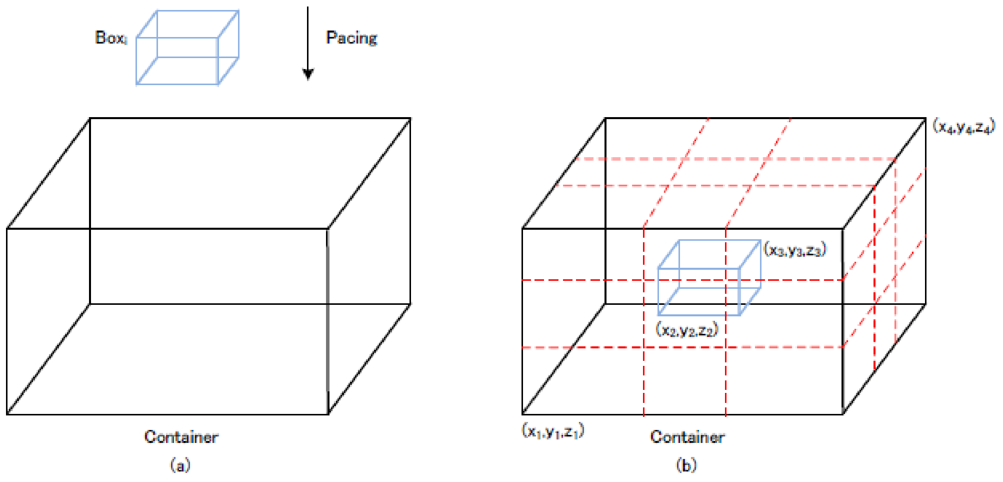


Figure 4.3: The Example of Using The 3-Dimensional Difference Process.

In the 3D case, after packing a box, there are at most six EMSs generated. To implement the 3D DP, equation (4.3.2) can be rewritten as the following model:

Difference: $[(x_1, y_1, z_1), (x_4, y_4, z_4)] - [(x_2, y_2, y_4), (x_3, y_3, y_3)]$, where

$$\text{NewEMS}_1 = [(x_1, y_1, z_1), (x_2, y_4, z_4)] \tag{4.16}$$

$$\text{NewEMS}_2 = [(x_1, y_1, z_1), (x_4, y_2, z_4)] \tag{4.17}$$

$$\text{NewEMS}_3 = [(x_1, y_3, z_1), (x_4, y_4, z_4)] \tag{4.18}$$

$$\text{NewEMS}_4 = [(x_3, y_1, z_3), (x_4, y_4, z_4)] \tag{4.19}$$

$$\text{NewEMS}_5 = [(x_1, y_3, z_1), (x_4, y_4, z_4)] \tag{4.20}$$

$$\text{NewEMS}_6 = [(x_3, y_1, z_1), (x_4, y_4, z_4)]. \tag{4.21}$$

- **Elimination Process**

  After generating the new empty spaces, or called intervals, some of them will be eliminated because they are unavailable to pack any items, and to eliminate them can save the computer memory storage space. The intervals with an infinite thinness or those are totally inscribed by the other intervals will be removed from the list. To implement this process the new empty spaces have to undergo two types checking:

  - **Cross-Checking**

  Involves comparing each interval with each other in order to check whether it is totally inscribed by another interval. Suppose that we have two intervals: $(x_1, y_1, z_1), (x_4, y_4, z_4)$, $(x_2, y_2, z_2), (x_3, y_3, z_3)$. Eliminate $(x2, y2, z2), (x3, y3, z3)$ from $(x_1, y_1, z_1), (x_4, y_4, z_4)$, if $x_2 \geq x_1$ and $y_2 \geq y_1$ and $z_2 \geq z_1$ and $x_3 \leq x_4$ and $y_3 \leq y_4$ and $z_3 \leq z_4$

  - **Self-Elimination**

  Here we check whether the interval is infinitely thin or not. Suppose we have an interval $[(x2, y2, z2), (x3, y3, z3)]$. Eliminate $[(x2, y2, z2), (x3, y3, z3)]$ from the maximal empty spaces list S: if $x_2 = x_3$ or $y_2 = y_3$ or $z_2 = z_3$.

  The above checking processes can easily be adapted to the two-dimensional problem by setting $z_1 = z_2 = z_3 = 0 : 0$.

  There is a 2-Dimensional instance, depicted by Fig.4.4, used to introduce this process. For convenience purposes, the coordinates in Fig 4.4 are represented by capital letters. By following the bottom left rule, we put the Box$_i$ into the bottom-left corner of the container as shown in Fig 4.4(a). By using the two dimensional DP, the four empty spaces are generated, which can be presented as follows:

$$[\text{A, I}] - [\text{A, E}] = \{[\text{A, G}], [\text{A, C}], [\text{D, I}], [\text{B, I}]\}.$$

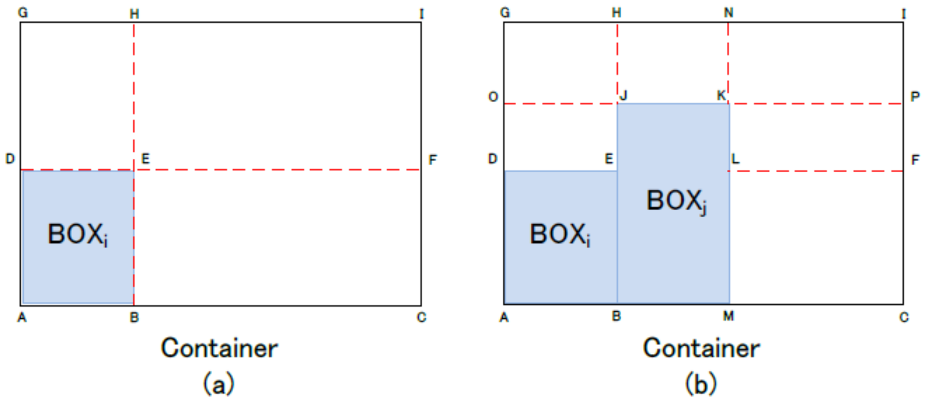


Figure 4.4: Example of Elimination Process.

Where [A,G] and [A,C] will be eliminated by following the Self-Elimination process. Thus [D,I] and [B,I] will be saved into the list S. After that, we assume that the Box$_j$ is the next packing item, which is packed in the space [B,I] by following the Deep-Bottom-Left rule as illustrated by the Fig.4.4(b). The space [B,I] is divided into four subspaces as well by following same rules:

$$[B, I] - [B, K] = \{[B, H], [B, C], [J, I], [M, I]\}.$$

However, the difference from packing the Box$_i$ is that when the Box$_j$ has been packed into space [B, I], the space [D, I] was no longer available because the space [D, I] was intersected by the Box$_j$. To detect which space will be intersected by the Box$_j$, we use the following criteria: suppose there is an empty space s, which can be represented by their bottom left point and top right point:$(x_1, y_1), (x_2, y_2)$. Also, a box can be represented by (x_3, y_3) and (x_4, y_4).

Judgement: the space s does not intersect with the current box,if and only if $x_1 \geq x_4$ or $x_2 \leq x_3$ or $y_1 \geq y_4$ or $y_2 \leq y_3$ Otherwise the space will intersect with the box.

To extend this criteria to the 3 dimensional problem we only need to add coordinate z by the same way. After finding the intersected empty spaces, it has to be updated by using the DP as well. Thus, there are four more empty spaces generated,

$$[D, I] - [B, K] = \{[D, H], [D, F], [O, I], [L, I]\}.$$

To summarize current empty spaces, there are eight successors waiting to be saved into the list S. However, before saving them, their availability has to be checked by using the Elimination Process. In this case, [B, H], [B, C], and [D, F] will be eliminated by implementing the Self-Elimination process. The interval [J, I] inscribes within [O, I], so it was eliminated by using the Cross-Checking process. After elimination, four empty spaces, [M, I], [D, H], [O, I], and [L, I], were saved into the list S.

To implement the Empty-Spaces updating all above steps are integrated into one function. During the whole packing procedure, this function will be called at each packing operation. The algorithm 4.1 shows the pseudo-code of the Empty-Spaces update method.

Algorithm 4.1: Empty Spaces Update(Box$_j$, S) .

Initialization:
Define a empty list TS and NS. Let (x_b, y_b, z_b) and (x'_b, y'_b, z'_b) represent the deepest-bottom-left corner and the front-topright corner of the Box$_j$.
Iteration:

```
 1: for each element $i$ in the list $S$ do
 2: if $x_b \geq x_i$ or $x'_b \leq x_i$ or $y_b \geq y'_i$ or $y'_b \leq y_i$ or $z_b \geq z'_i$ or $z'_b \leq z_i$ or then
 3:         //Separate the spaces that are not intersected
 4:         //by element $i$ from the old list;
 5:         Save the element $i$ into list $TS$
 6:    else
 7:         Implement 3-Dimensional DP process to element $i$.
 8:         Save the new empty spaces into list $NS$.
 9:  end if
10: end for
11: Implement Elimination Process to list $NS$ and $TS$.
12: Clear all element in $S$.
13: $S = NS + TS$. // Update the list $S$;
```

Output: The new list S.

4.3.3 The Deep-Bottom-Left Procedure

Recall from the section 4.3.1 that the fist step in the placement strategy is to select an EMS with the minimum coordinates from the list S, all the elements in the list S will be arranged into the lexicographical order. Thus, the Deep-Bottom- Left procedure is actually a sorting algorithm, which is used to sort the EMS$_s$ into the lexicographical order: EMS$_i$ $\leq$ EMS$_j$ if $y_i < y_j$; or if $y_i = y_j$ and $z_i < zj$; or if $y_i = y_j$; $z_i = z_j$ and $x_i < x_j$: After each time of the packing, the list S will be sorted again by using this method. The algorithm 4.2 shows the pseudo-code of the Deep Bottom Left (DBL) procedure.

Algorithm 4.2: Deepest-Bottom-Left Sorting (S) **Initialization:**
Set sw = **false**;
Let Let $rindex$ be the maximal index of list S
Iteration:

```
 1: for $i = 0$ to $rindex - 1$ do
 2:        for $j = rindex$ ; $j > i$ ; $j++$ do
 3:         if $y_j < y_{j-1}$ then
 4:         swap $(S_j, S_{j-1})$;  //First check the length order
 5:         $sw$ = true;
 6:    else if $y_j = y_{j-1}$ then
 7:         if $z_j < z_{j-1}$ then
 8:         swap $(S_j, S_{j-1})$;  //Second check the height order
 9:  $sw$ = true;
10: else if $z_j = z_{j-1}$ then
11:         swap $(S_j, S_{j-1})$;  //Third check the width;
12: $x_j = x_{j-1}$ then
13:  $sw$ = true;
14:      end if
15:    end if
16:  end if
17: end for
18: if not $sw$ then
19:   return
20:  end if
21: end for.
```

The available loading space will be divided into several subspaces after packing a layer. The algorithm 2 will be called during each packing procedure, in which the subspaces will be rearranged into a lexicographical order and saved into the list S.

4.3.4 Practical Implementation

According to the above sections, the placement strategy can be considered as an important part in the fitness function. In the 3-dimensional packing problem, there does not exist a normal object function as in the standard linear problem. Thus, the whole packing process has to be simulated in order to calculate the space utility rate. The functions that were mentioned in the above sections are integrated together, called the Placement Strategy as shown in the algorithm 3, which is used to simulate the whole packing process. The layer selection and packing part in the algorithm 3 are used to group the boxes with the same size together to produce a new block. In such way, the problem search space will be reduced.

4.3.5 Summary

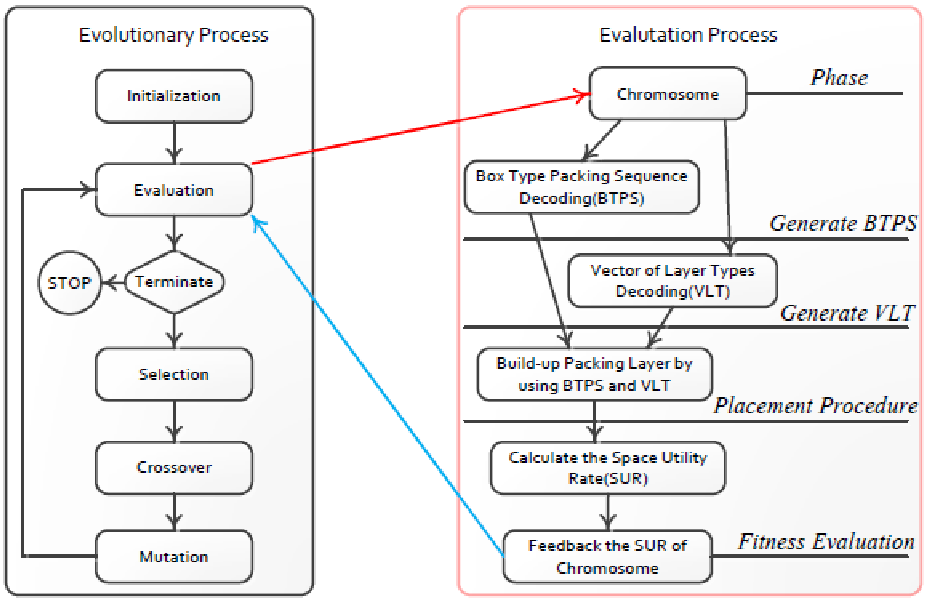
In this chapter, all relevant knowledge of the container loading problem that are use in this project are presented. In order to make more wide comparison, the constraints in this problem are set by following the most relevant literatures.

Chapter 5

Evolutionary Algorithm for Two and Three-Dimensional Bin Packing Problem

5.1 Evolution Optimization

In this research work, the optimizer follows the evolutionary process of the standard heuristic algorithms using the GA, so there are five components involved. The first one is the initialization part in which the population is initialized with the random-key vector. The elements of this vector are random real numbers uniformly sampled for the interval [0; 1]. The second one is the fitness component in which each individual in the population will be evaluated and assigned a merit value. After that the population will be put into the selection part where a fixed number of individuals are selected based on their fitness value. In this project, we use the *elitist strategy* [21], so that a fixed number of best individuals will be copied into the next generation directly without change. The fourth part is the crossover section in which the selected individuals will be used to produce new offspring. The final part of this optimizer,is called a mutation operator, in which the several random selected genes of the offspring in the new population will be mutated by a small probability. However, the mutation operator (Mutant) in the evolutionary algorithm (EA) strategy is quite different from one in the standard GA, which generates a fixed number individuals completely at random by using the same way as in the initialization component. The purpose of using this mutation operator is also to prevent premature convergence. The Fig.5.1 illustrates the architecture of this algorithm and the evolutionary process.



Evolutionary Process
Initialization
Evaluation
STOP
Terminate
Selection
Crossover
Mutation
Evalutation Process
Chromosome
Phase
Box Type Packing Sequence Decoding(BTPS)
Generate BTPS
Vector of Layer Types Decoding(VLT)
Generate VLT
Build-up Packing Layer by using BTPS and VLT
Placement Procedure
Calculate the Space Utility Rate(SUR)
Feedback the SUR of Chromosome
Fitness Evaluation

Figure 5.1: The Architecture of The Evolutionary Algorithm.

5.2 Bin packing with Genetic Algorithm

Genetic Algorithms (GAs for short) are mathematical procedures based on analogies to the natural evolutionary process. However, the evolutionary process simulated by GAs is extremely simplified. Even though recent work reported on GAs focuses on GAs as an optimization procedure, it is cautioned that GAs are not function optimizers but merely procedures that simulate the evolutionary process. GAs belong to a class of probabilistic algorithms, yet they are different from random algorithms and they combine elements of directed and stochastic search. Algorithm 4.1 illustrates pseudo code of a simple GA.

Algorithm 5.1 Simple GA

```
1: Begin
2:   t ← 0
3:       initialise $P(t)$
```

```
 4:     evaluate $P(t)$
 5: **While** (!(termination-condition)) **do**
 6:     **begin**
 7:         t ← t + 1
 8:         select $P(t)$ from $P(t-1)$
 9: alter $P(t)$
10:         evaluate $P(t)$
11:     **end**
12: **end**.
```

A genetic algorithm is a probabilistic algorithm which maintains a population of individuals, $P(t) = x_1^t, ..., x_n^t$, that are created and selected in an iterative process. Each individual x_i^t consists of a *genome*, a *fitness* and possibly some auxiliary variables such as age and sex. The genome consists of a number of *genes* that altogether *encode* a solution to some optimization problem. The *encoding* is the internal representation of the problem this the data structure holding the genes. Every member of the population x_i^t is evaluated to measure its fitness. A new population at iteration $t + 1$ is formed by selecting those individuals which have more fitness. Some members of the population undergo transformations (alter step in the pseudo code), this is achieved by means of some variation operators (these are sometimes referred to as genetic operators) to form new solutions. The transformations fall into two categories which are unary transformations u_i that create new individuals by a change in a single individual ($m_i : S \leftarrow S$), and higher order transformations c_j that create new individuals by combining parts from several (two or more) individuals ($c_j : Sx...S \rightarrow S$). These two transformations are popularly known as mutations and crossovers respectively. The algorithm executes until some predefined halting condition is reached, The condition might be the solution quality, number of generations or simply running out of time. During the run of the algorithm the fitness of the best individual (hopefully) improves over time. Ideally at the halting time the best individual found so far should coincide with the discovery of the global optimum. However, it is possible for the best individual to converge at a local optimum which is usually the undesirable result. Since GAs are population based search algorithms, this means that at anytime during the search, the fitness function has to evaluate the entire population. This is a serious drawback of GAs as this results in long computational times. For in depth discussions on GAs see [21].

5.2.1 Encode

Encoding implies representing solutions in a format that will make search operators or genetic operators maintain a functional link between parents and their offspring. The encoding should make it possible for there to be a useful relationship between parents and offspring. As to which encoding to use differs from problem to problem. It is fair to say no one encoding technique is best for all problems. Popular examples of encodings are:

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

5.2.2 Fitness Evaluation

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

5.2.3 Selection

The basic idea behind selection is that it should be related to the fitness of each individual. The original scheme for its implementation is commonly known as roulette wheel selection because a common method of accomplishing this procedure can be thought of as a roulette wheel being spinned once for each available slot in the next population. Where each solution has a slice of the roulette allocated in proportion to their fitness score (see Figure 5.2 for an example). In this scheme it is possible to choose the best individual more than once, and chances are that the worse individual has a very slim chance of being selected.

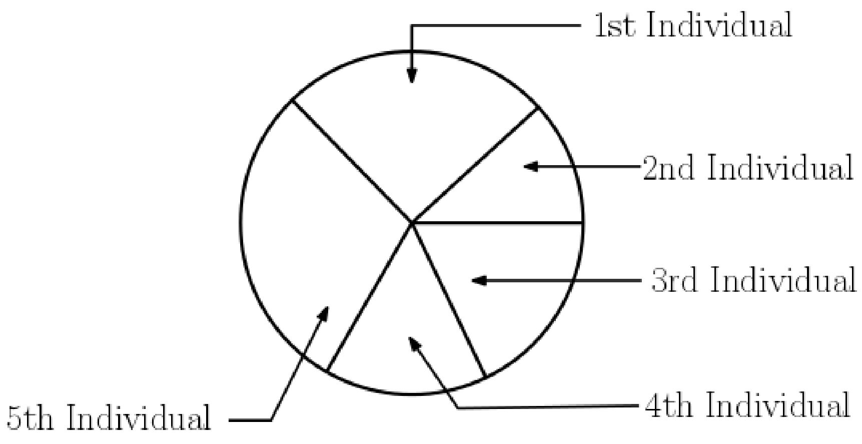


Figure 5.2: A roulette wheel with 5 slices.

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

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

$$SP = \frac{Prob.[selecting\ fittest\ individual]}{Prob.[selecting\ average\ string]}$$

The fitness value for the individual is calculated as:

$$Fitness(Pos) = 2\ -\ SP + 2 \cdot\ (SP - 1) \cdot \frac{(Pos - 1)}{(Nind - 1)}, \qquad SP\ \in [1, 2]$$

For further discussions concerning the strict fitness - proportion are given in [21].

5.2.4 Variation Operators

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

- Crossover Operator
- Mutation operator.

5.2.4.1 Crossover Operator

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

$$(\mathrm{a}_1, a_2, a_3, a_4, a_5, a_6) \; and \; (b_1, b_2, b_3, b_4, b_5, b_6),$$

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

$(\mathrm{a}_1, a_2, b_3, b_4, b_5, b_6) \; and \; (b_1, b_2, a_3, a_4, a_5, a_6)$.

If we were to choose two cross points randomly between numbers $1, ..., 5$ for example if the points were 2 and 4 the offspring solutions would be $(a_1, a_2, b_3, b_4, b_5, a_6)$ and $(b_1, b_2, a_3, a_4, a_5, b_6)$. In the example presented above we did not use binary strings for the solutions to emphasise that binary representation is not a critical aspect of GAs. Another aspect to crossover operator is that the operation can involve more than two parents.

5.2.4.2 Mutation

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

5.3 Solution Representation

In section 5.2 a survey was presented on how GAs can be used in order to provide a solution for bin packing problems. In this section a generic solution representation is presented, which serves as a template solution for all the 2-D and 3-D bin packing problems. The general solution representation consists of two parts:

- **Problem code**
- **Problem specific encoding**.

The problem code part is the 4-tuple code and consists of the following fields (Problem type, Dimension, Orientation Constraint, Cutting Technology Constraint). This code augments the problem specific encoding for every problem. The interpretation of this code was fully explained, the advantage to using this code is the ability of uniquely identifying a problem with associated constraints.

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

$(x_1, i_1, \phi_1), (x_2, i_2, \phi_2), ..., (x_n, i_n, \phi_n)$.

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

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

5.4 Interpretation of the solution for two-dimensional problems

The representation for 2D problems follows from the general representation introduced in section 5.3, which is

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

For two dimensional problems the solution representation is evaluated by means of a placement algorithm which is fully explained in section 5.5. What this implies is the problem specific part of the encoding can be considered ordered based representation, i.e. items are considered one item at a time for the placement. Let (x_i, y_i) be the bottom left corner of the rectangular item chosen as the reference for the rectangular item to be placed and be the reference vertex $\nu(x_i, y_i)$ if the item to be placed is the polygon. These items are to be placed in rectangular regions. Let the bottom left corner of the containment region (Bin/strip) be the origin $(0, 0)$ with it's four sides parallel to the $X-$ and $Y-$ axis respectively. The meaning for the problem specific part variables is provided below:

- x_k = The x-coordinate value of the reference vertex for the k_{th} item
- i_k = The index of the k_{th} item
- ϕ_k = is the orientation of the k_{th} item.

Now that the meaning of the variables for the encoding has been explained, below the coding for each of the two dimensional bin problems is presented.

5.4.1 Two dimensional Bin packing problems

For the two-dimensional bin packing problems the solution encoding is given below. The items packed may be rotated by 90° and no guillotine cutting required:

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

$\phi_i \in \{0°, 90°\}$.

Items may be rotated by 90° and guillotine cutting is required. Hence

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

$\phi_i \in \{0°, 90°\}$.

Items may not be rotated and no guillotine cutting required. We than have

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

$\phi = 0°$.

Items may not be rotated and guillotine cutting required. This gives

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

$\phi = 0°$.

5.4.2 Representation for 2D problems

For the two-dimensional strip packing problems the solution encoding is given below:

Items may be rotated by 90° and no guillotine cutting required. In this case

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

$\phi_i \in \{0°, 90°\}$.

Items may be rotated by 90° and guillotine cutting is required. Hence

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

$\phi_i \in \{0°, 90°\}$.

Items may not be rotated and no guillotine cutting required. Therefore

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

$\phi = 0°$.

Items may not be rotated and guillotine cutting required. This gives

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

$\phi = 0°$.

5.5 Initial Population Generation

In section 5.2, we have mentioned that at any time during the search a GA maintains a population of solutions. In the pseudo code presented in algorithm 4.1 the initial step is the initialisation of the population. In this section the population initialisation process for the problems described above is explained.

5.5.1 Initial Population generation for two dimensional problems

To generate the initial population it is ensured that every individual belonging to the initial population of solutions is feasible. In section 5.4 a general problem representation was introduced, which enables us to both uniquely identify problems and encode all the problems considered in this work in a standard format. Every item in the two dimensional problems, was represented by a 3-tuple (x_k, i_k, ϕ_k). Each variable in this 3-tuple has been described. Where x_k was defined as the x-coordinate of the reference vertex for each item (the bottom left corner for rectangular items). For rectangular items the only feasible values for x_k are in the interval $P_x i = [0, W - w_k]$ where W is the width of the container and w_k is the width of the rectangular item r_k see figure 4.1 for illustration. The same goes with the third element of the 3-tuple, ϕ_k is always a member of a feasible set of orientation constraints, for example, in problems where rectangular items can only be rotated by 90°, the set of feasible constraints O_c consists of only two elements, i.e. $O_c = \{0°, 90°\}$.

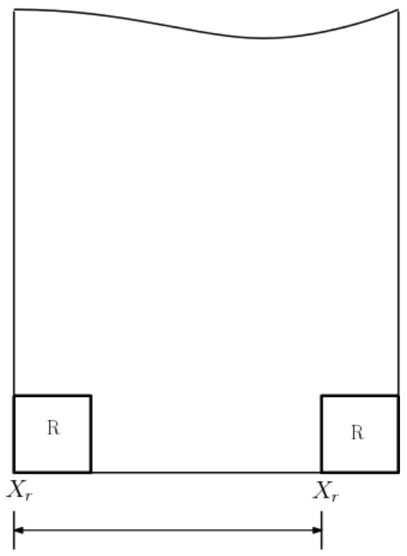


Figure 5.3: A set of Feasible x co-ordinates for rectangular items.

The same argument holds for polygonal items, this is there is a set of feasible x-coordinate positions for the reference vertex of each polygon item for the containment constraint and a set of feasible orientation constraints for each item.

To generate the initial population the following procedure is followed:

- Randomly order items.

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

5.5.2 Variation Operators in the general GA

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

5.5.3 The variation operators for 2D Problems

In subsection 5.5.1 it is stated that the fitness evaluation (which is explained in section 5.6) of all solutions is done through a placement heuristic. This implies that the solution representation for 2D problems is of ordered nature, and the horizontal position (x-coordinate) of each item is also part of the encoding. The variation operators for the 2D problems take all of these into consideration.

5.5.3.1 Crossover Operator

As a crossover operator for 2D problems two crossover variants are used,*cross_var1* and *cross_var2*. The choice always has to be made as to which one to use, i.e. a coin has to be flipped to decide which of these two variants will be operational. A partially mapped crossover (see, [62]) is slightly modified and applied for that purpose.

Cross_var1

In subsection 4.4.1 the solution representation for 2D problems was represented with item characteristics that were part of the encoding, i.e. the x-coordinate of the reference vertex and the orientation of the item. The *cross_var1* allows for the orientation, x-coordinate of the reference vertex components of the solution to be directly inherited from one parent. The ordering of the items is then achieved through breeding between both parents. Let $\overrightarrow{X_{p1}}$ and $\overrightarrow{X_{p2}}$ be two parent solutions representing n items and let O_1 be the offspring that results out of the breeding of the parents. For example consider the following situation where

$\overrightarrow{X_{p1}} = [(SPP, 2, 1, F), (0, 6, 0°), (9, 3, 90°), (5, 2, 90°), (2, 4, 0°), (10, 5, 90°), (3, 1, 90°)]$
and
$\overrightarrow{X_{p2}} = [(SPP, 2, 1, F), (1, 1, 90°), (5, 2, 90°), (4, 3, 0°), (6, 5, 90°), (2, 4, 0°), (8, 6, 0°)]$.

We need to arrange efficiently a layout of 6 items.

The *Cross_var1* works as follows:

1. Copy the x co-ordinate of the reference vertex and orientation of every item from solution $\overrightarrow{X_{p2}}$ to the offspring O_1. In this example, at this stage the offspring becomes:
   $O_1 = [(SPP, 2, 2, F), (1, X, 90°), (5, X, 90°), (4, X, 0°), (6, X, 90°), (2, X, 0°), (8, X, 0°)]$.
   The symbol 'X' can be interpreted as "at present unknown".

2. Generate two random positions p_1 and p_2, such that $1 \leq p_1 \leq p_2 \leq n$:
   For example say p_1 is generated to be 4 and p_2 to be 5.

3. Create a one to one mapping between item indices in positions $p_1 - p_2$ from both parents. The series of mappings for this example is:
   $5 \longleftrightarrow 4, 4 \longleftrightarrow 5$.

4. Copy every item index between positions $p_1$and p_2 from $\overrightarrow{X_{p2}}$ to O_1 to corresponding positions. After the copying the offspring becomes:
   $O_1 = [(SPP, 2, 2, F), (1, X, 90°), (5, X, 90°), (4, X, 0\circ), (6, 5, 90°), (2, 4, 0°), (8, X, 0°)]$

5. Every item indexed from $\overrightarrow{X_{p2}}$ not in O_1 is copied with its position in the order to O_1 starting from the left most index to the right most excluding $p_1 - p_2$ positions, conflict is avoided by making use of the mapping in stage 3. The final solution becomes:
   $O_1 = [(SPP, 2, 2, F), (1, 6, 90°), (5, 3, 90°), (4, 2, 0°), (6, 5, 90°), (2, 4, 0°), (8, 1, 0°)]$.

   From the description of this variant of crossover it should be obvious that, there is a possibility that infeasible solutions might be introduced into the population. To counteract this a penalty function is used to degrade the quality of infeasible solutions (more about this is given in section 4.6).

Cross_var2

The major difference between these variants of crossover is that *cross_var1* allows a situation where breeding involves both item characteristics in the solution representation and the ordering of the items for the packing. *Cross_var2* on the other hand is mainly concerned with the ordering of the items without separating the item characteristics and the ordering, i.e. when items change positions in the ordering, the item moves with the characteristics that define it. In other words the whole 3-tuple (x_k, i_k, ϕ_k) moves. The parents used to demonstrate *cross_var1* will again be used to demonstrate *cross_var2*. *Cross_var2* breeds offspring as follows:

1. Generate two random positions p_1 and p_2, such that $1 \leq p_1 \leq p_2 \leq n$:
   For example say $p_1 = 3$ and $p_2 = 5$.

2. Create a one-to-one mapping of indexes from both solutions in positions $p_1 - p_2$. For this example that would be:
   $2 \longleftrightarrow 3, 4 \longleftrightarrow 5$ and 4 $5 \longleftrightarrow 4$.

3. Copy from parent $\overrightarrow{X_{p1}}$ items in position $p_1 - p_2$ with their related characteristics. This results in a partial offspring solution, which is:
   $O_2 = [(SPP, 2, 2, F), (X, X, X), (X, X, X), (5, 2, 90°), (2, 4, 0°), (10, 5, 90°), (X, X, X)]$.
   The symbol X can be interpreted as at present unknown.

4. We complete the solution by copying items from parent $\overrightarrow{X_{p2}}$ starting from left to right excluding those items in positions $p_1 - p_2$ and try and avoid conflict by using the mapping in stage 2:
   $O_2 = [(SPP, 2, 2, F), (1, 1, 90°), (9, 3, 90°), (5, 2, 90°), (2, 4, 0°), (10, 5, 90°), (8, 6, 0°)]$.

5.5.3.2 2D Mutation Operator

The 2-swap mutation operator which is usually made use of in sequencing problems (see [62] for the TSP example) has been adapted and modified as the mutation operator for 2D problems. The operator works as follows:

1. Randomly choose two items $item1$ and $item2$.

2. Randomly generate a number $num \in \{0, 1\}$ to decide if the orientation of the chosen items will be randomly perturbated.

3. If $num = 1$, then change the orientation of both items randomly. This applies if more than one orientation is allowed.

4. Exchange the position of $item1$ with that of $item2$.

5.6 Fitness Function

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

$$max\ f(x) = \begin{cases} f_1(X) \text{ if } \text{ problemcode} = \begin{cases} (BPP, 1, *, *) \\ (CSP, 1, *, *) \end{cases} \\ \\ f_2(X) \text{ if } \text{ problemcode} = \begin{cases} (SPP, 2, 2, F) \\ (SPP, 2, 1, F) \end{cases} \\ \\ f_3(X) \text{ if } \text{ problemcode} = \begin{cases} (SPP, 2, 2, G) \\ (SPP, 2, 1, G) \end{cases} \\ \\ f_4(X) \text{ if } \text{ problemcode} = \begin{cases} (BPP, 2, 2, F) \\ (BPP, 2, 1, F) \end{cases} \\ \\ f_5(X) \text{ if } \text{ problemcode} = \begin{cases} (BPP, 2, 2, G) \\ (BPP, 2, 1, G) \end{cases} \\ \\ f_6(X) \text{ if } \text{ problemcode} = \begin{cases} (ISPP, 2, 1, *) \\ (ISPP, 2, 2, *) \\ (ISPP, 2, 4, *). \end{cases} \end{cases} \tag{5.1}$$

5.6.1 Evaluation of nonguillotine-able 2D Strip packing problems

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

1. Item i_k is placed at the topmost position at horizontal position x_k, with the orientation of item k being that reflected by ϕ_k.
2. Item i_k is slid as far down as possible, until it collides with either the bottom edge of the strip or another item.

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

To demonstrate the above heuristic consider the solution:
$\overrightarrow{X} = [(SPP, 2, 2, F), (0, 6, 0°), (9, 3, 90°), (5, 2, 90°), (2, 4, 0°), (10, 5, 90°), (15, 1, 90°)]$
consisting of 6 rectangular items to be packed on a strip whose width is 20 units. The item dimensions are given in table 4.1. Figure 4.3 shows how solution $\overrightarrow{X}$ would be decoded using the placement heuristic discussed above.

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

Table 5.1: Item dimensions example.

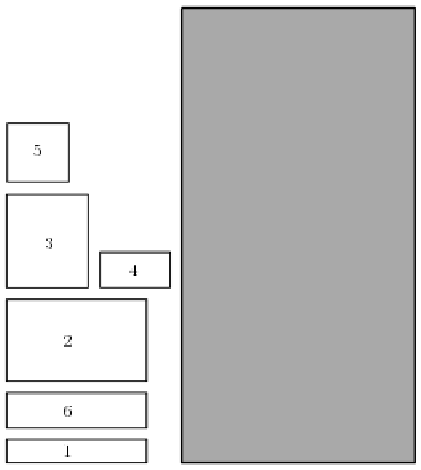


Figure 5.4: Items to be placed in the bin.

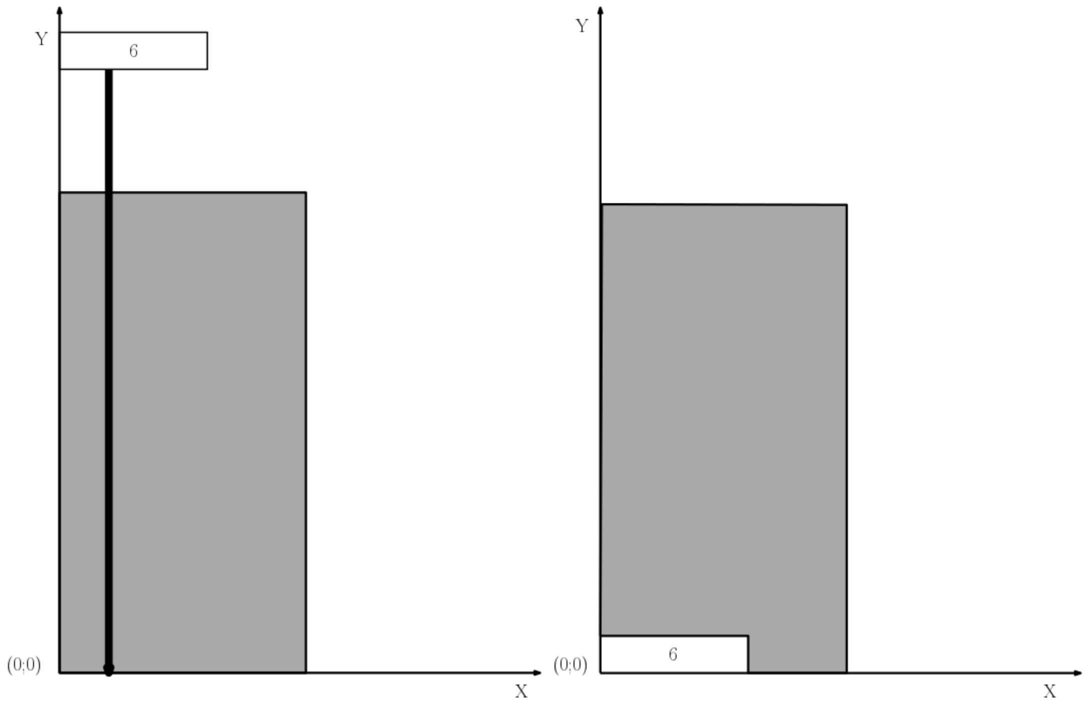


Figure 5.5: First Item to be placed in the bin.

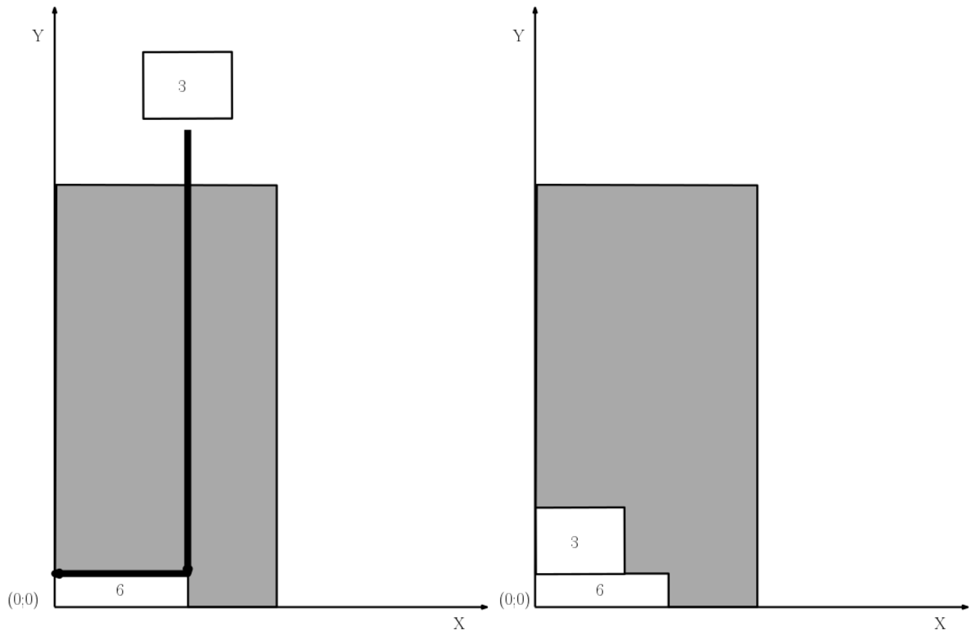


Figure 5.6: Second Item to be placed in the bin.

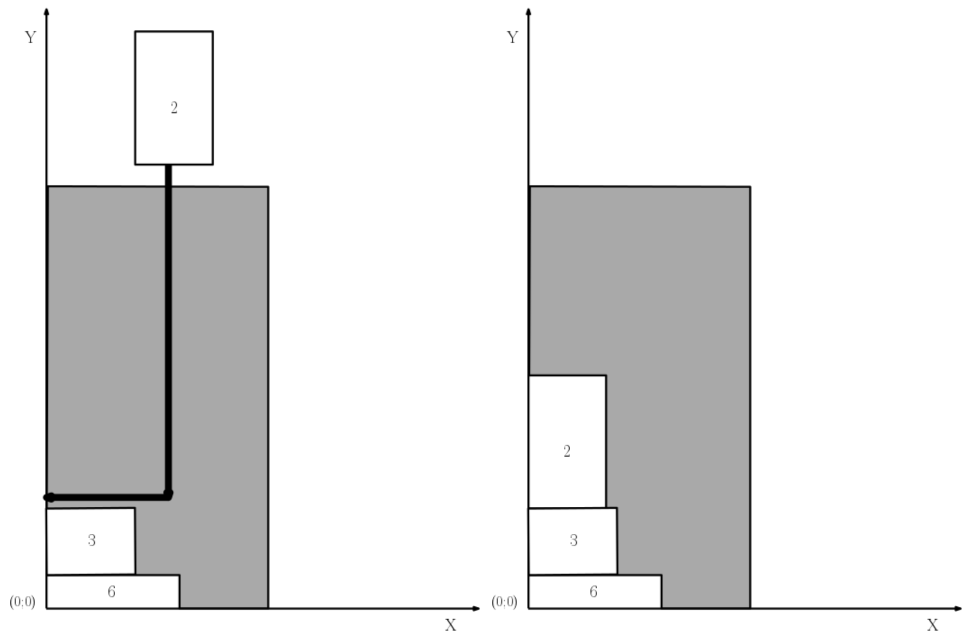


Figure 5.7: Third Item to be placed in the bin.

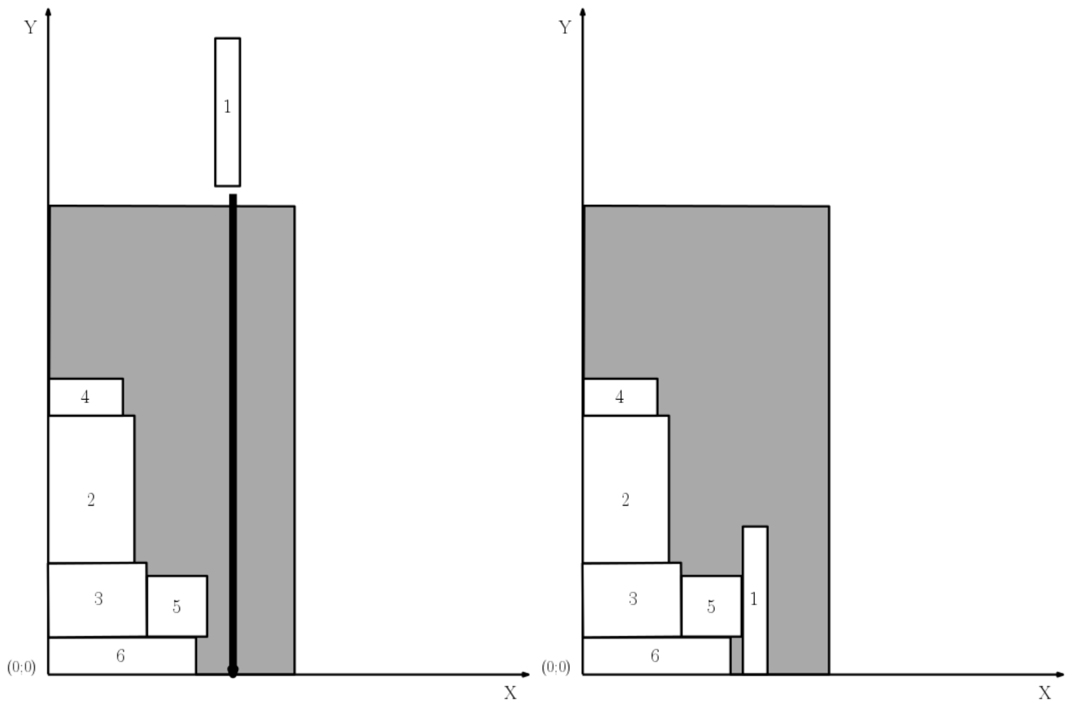


Figure 5.8: Last Item to be placed in the bin.

In the discussion on cross-over operators for 2D problems in section 5.5.3.1 we have mentioned that *cross_var1* can introduce infeasible solutions into the population. There are two possible violations of constraints that can occur in 2D problems, e.g. overlap constraint, etc. It is the violation of the latter constraint that *cross_var1* is guilty of, i.e placement of items outside the borders of the strip. In the design of the fitness function for 2D problems this has to be taken into account. Taking this into account the fitness function for 2D strip packing problems is given by:

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

where, $P_2(X)$ is a penalty function used as a constraint handling mechanism. Any solution in violation of the above mentioned constraint is killed, i.e. the solution is made undesirable. Eff(X) measures the efficiency of the packing. The total area of items to be packed is given below where h_p is the packing height (see figure 5.9 for an example of packing height).

$$A = \sum_{i=1}^{n} w_i h_i. \tag{5.2}$$

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

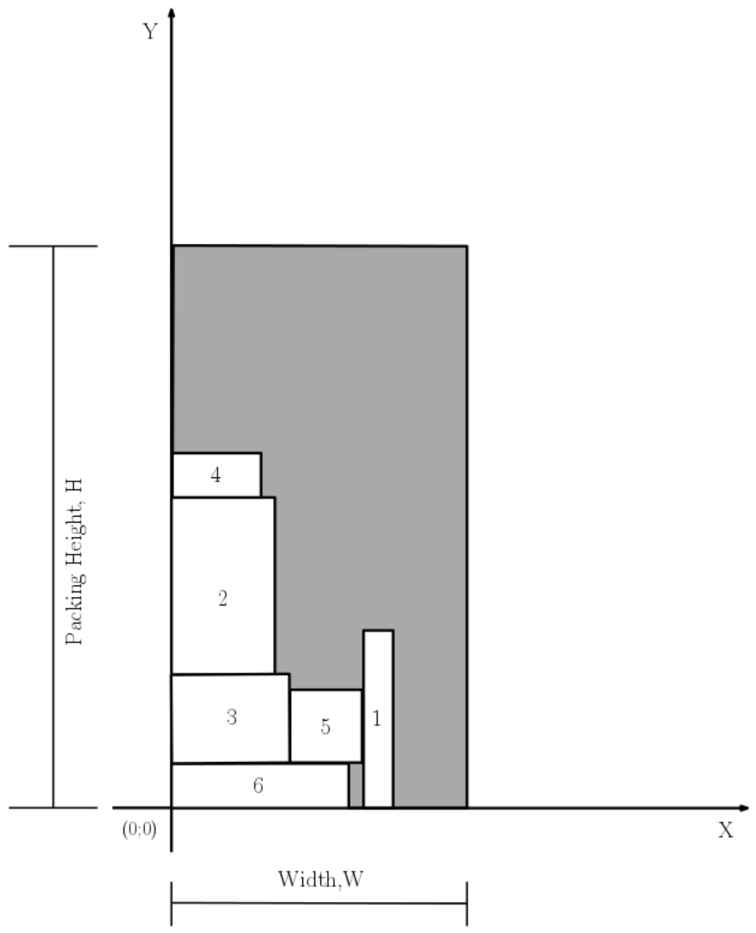


Figure 5.9: Example of packing height.

$$A_p = h_p W \tag{5.3}$$

for efficiency calculation equation 5.4 is used

$$Eff(X) = \frac{A}{A_p} \tag{5.4}$$

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

$$h_L = \frac{A}{W}. \tag{5.5}$$

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

5.6.2 Evaluation of the guillotine-able 2D Strip Packing Problems

The function f_3 is the fitness function for strip packing problems with guillotine constraint. This function is also evaluated by means of a placement heuristic, the only difference is the guillotine cutting constraint should be taken into account when placing items. Items are placed such that the guillotine constraint is never violated. An observation that is of great help when placing items on the strip with guillotine cutting required, is that guillotine cutting subdivides the strip into blocks whose top edge and bottom edge is parallel to the bottom edge of the strip, see figure 5.8 for illustration. Blocks consist of rectangular items and wasted space.

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

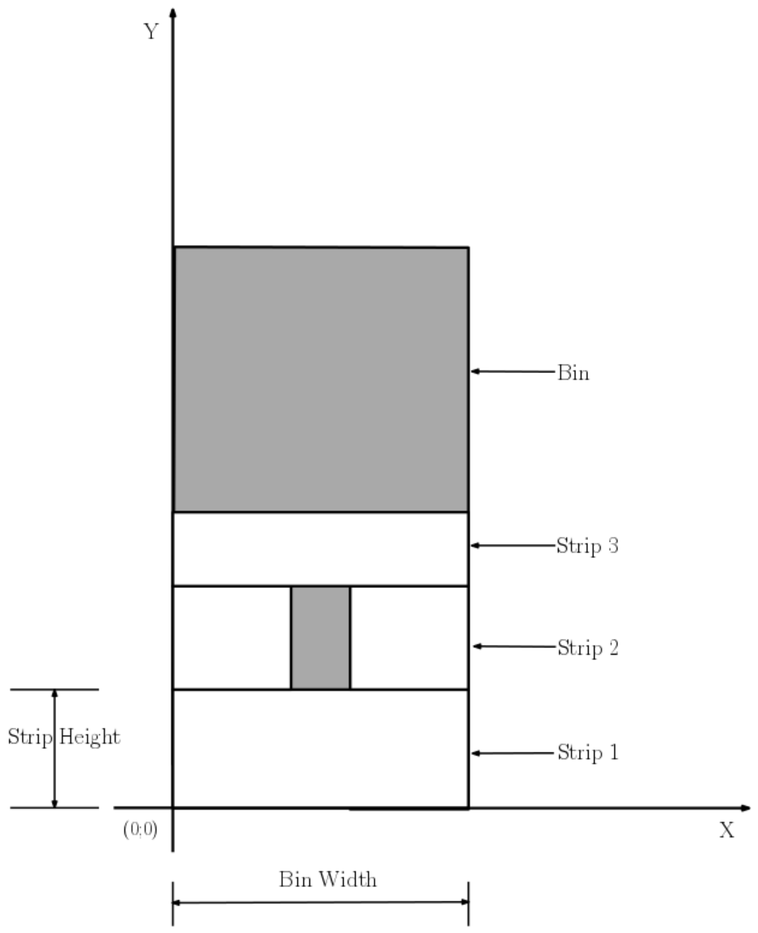


Figure 5.10: An example of the Guillotine Strip Packing.

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

1. Item i_k is placed at the topmost position at horizontal position x_k, with the orientation of item k being that reflected by ϕ_k.
2. Item i_k is slid as far down as possible, until the item either collides with the bottom horizontal edge of the strip or collides with another item.
3. Item i_k is checked if it is in violation of the guillotine constraint. If it is item i_k's vertical position it is corrected to satisfy the guillotine constraint.
4. Item i_k is shifted as far left as possible until it collides with another item or the vertical left edge of the strip.

The fitness function f_3 is also given by

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

where $P_3(X)$ is a penalty function used to kill infeasible solutions and is worked out as $P_2(X)$ x $F_3(X)$ is a function whose purpose is to value efficiently packed blocks and efficiently packed overall layout. Another way of representing the fitness function f_3 is:

$$F_3(X) = Eff(X) + qB(X),\ 0 < q \leq 1 \tag{5.6}$$

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

$Eff(X)$ is as described in equation (5.4). Let Eff_{Bi} be an efficiency of block B_i, and H_{Bi} be the height of block B_i. Let A_i be the area of rectangle r_i. Let N_B be the total number of blocks in a layout, then

$$Eff_{Bk} = \frac{\sum_{rk} \in B_k\ A_k}{WH_{Bk}}, \tag{5.7}$$

B(X) is then given by

$$B(X) = \frac{\sum_{i=1}^{N_B} Eff_{Bi}}{N_{Blocks}} \tag{5.8}$$

5.6.3 Fitness function for 2D Bin packing problems

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

5.6.3.1 Evaluation of nonguillotine-able 2D Bin Packing Problems

For the evaluation of these problems we take the strip packing approach presented in section 5.6.1. The placement heuristic which gives the fitness function f_4 is as described below.

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

1. Item i_k is placed at the topmost position at horizontal position x_k, with the orientation of item i_k being reflected by ϕ_k.
2. Item i_k is slid as far down as possible until it collides with the bottom edge of the strip or collides with another item. If item i_k is the first item then the first bin is opened and stays open until all items are placed.
3. If item i_k is not the first item, then item i_k has collided with an item in some bin k already opened, item i_k is checked if it can be wholly contained in bin k. If item i_k can not be wholly contained by bin k item i_k is placed in bin $k+1$ which is immediately on top of bin k. If no such bin exists a new bin, $k+1$, is opened.
4. After the final vertical position of item i_k is decided upon in some bin, item i_k is shifted as far to the left as possible until it collides with either the vertical left edge of the strip or some other item in the same bin. This becomes the final position for item i_k.

To illustrate this heuristic consider the following example. Suppose we have bins of dimensions 100 x 100 with items of the following sizes in table 5.2. Consider the following individual to be decoded by the above placement heuristic explained above,

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

Figure 5.10 to 5.14 illustrate how $\overrightarrow{\mathbf{X}}$ is decoded by the placement heuristic for the 2D bin packing problem.

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

Table 5.2: Items dimensions for 2D bin packing problem

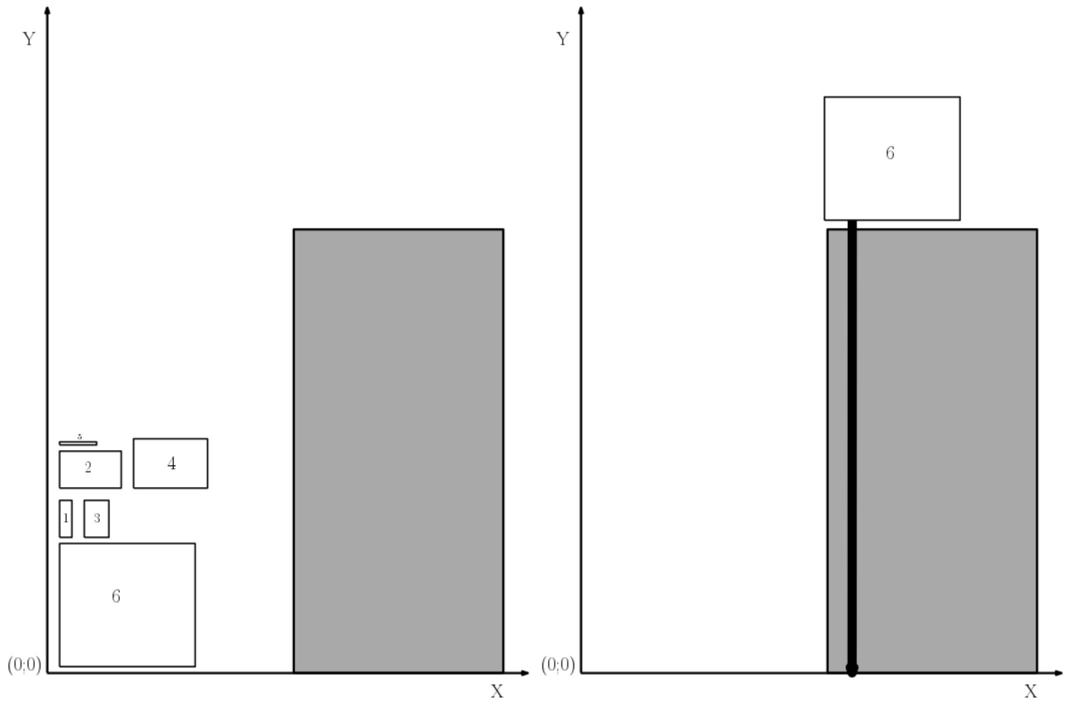


Figure 5.11: Placement of the first item.

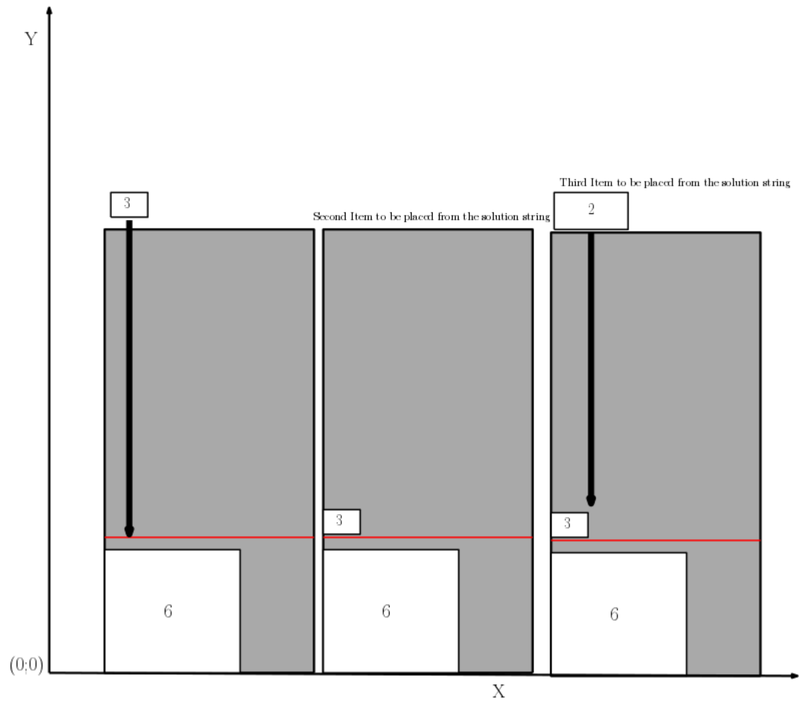

Y
X
(0;0)
Third Item to be placed from the solution string
Second Item to be placed from the solution string
3
2
3
3
6
6
6

Figure 5.12: Placement of the second and third items.

The fitness function f_4 is also given by:

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

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

$$Eff_{Bi} = \frac{\sum_{A_i} \in B_i \ A_i}{WH}. \tag{5.9}$$

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

$E = \{Eff_{B_1}, Eff_{B_2}, ..., Eff_{B_n}\}$ which can be expressed as
$E = \{\frac{A_1}{HW}, \frac{A_2}{HW}, ..., \frac{A_n}{HW}\}$.

The penalty function P_4 is given by

$$P_4(X) = minE \tag{5.10}$$

This is in an attempt to degrade infeasible solutions, which are all assigned the worst possible efficiency for problem instance i. Let N_B be the number of bins used in the layout. Let $BE(X)$ be the average bin efficiency for the entire layout, i.e.

$$BE(X) = \frac{\sum_{i=1}^{N_B} Eff_B}{N_B}. \tag{5.11}$$

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

$$F_4(X) = BE(X) + qEff(X) \tag{5.12}$$

where $Eff(X)$ is as described in equation 5.4, $0 < q < 1$.

5.6.3.2 Evaluation of the guillotine-able 2D Bin Packing Problems

Evaluation of the two dimensional bin packing problem with a guillotine constraint is evaluated by means of the same heuristic that has been presented in this work. To be precise in section 4.6.1 the evaluation placement heuristic for 2D guillotine-able strip packing problems is presented. The same heuristic is used with one modification, i.e. a limit is put on the size of the strip, i.e the bin height becomes the height. The fitness function, $F_5(X)$ for the guillotine-able bin packing problem is similar to the ones presented previously and is given by:

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

Here $F_5(X)$ is a penalty function, whose purpose is similar to other penalty functions presented previously. $P_5(X)$ is computed in the same way as $P_5(X)$ in equation (4.11) above. Let Eff_k be the efficiency of the bin with items arranged with a guillotine pattern for bin k and let A_k be the area of item k. Let W and H be the width and height of the bin respectively. Let N be the total number of bins used in the layout, then

$$Eff_k = \frac{\sum_i \in N_k \ A_i}{HW} \tag{5.13}$$

$$F_5(X) = \frac{\sum_{i=1}^{N} Eff_i}{N}. \tag{5.14}$$

5.7 Summary

In this chapter a new design of the genetic algorithm (GA) is presented and demonstrated on the 2D bin packing problem (the algorithm is modified later to tackle the 3D bin packing and the result sheet is given in chapter 6). The design of the algorithm is different from the normal GA, due to the fact that in this new design the offspring inherit important factors of their parents. Such a factor in this problem is the combination of items in a bin. Thus, our GA lays emphasis on the combinations of items. Furthermore, heuristic methods (specifically the FFDH), are introduced into our GA for obtaining a better solution. The new method is called the "evolutionary algorithm" due to the fact that it evolves the normal GA with heuristic algorithms in order to find better solutions.

Chapter 6

Computational Experiments

6.1 2D bin packing problem

We considered a realistic example of the 2D bin packing problem. A list of items are given below in Table 6.1 and they are to be cut out from rectangular bins of dimensions 260 x 160 mm. For convenience the list is already sorted according to the requirements described above. The numbers on the items in the example correspond to those numbers in the table. The dimensions given in the table are in millimetres. There are 36 different types of items and the quantity of each type of item is ranging from one to ten.

Appendix B shows the final optimal solution found using the proposed algorithm for the two-dimensional bin (strip) packing algorithm. The numbers on the items correspond to their sequence number in Table 6.1. The thickness of the cutting blade was ignored in the example. The items first were placed using the heuristic first fit and best-fit decreasing height algorithm covered in sections 3.3.1.2 and 3.3.1.3. The items were placed into the bins in the order of appearance in the list. The longest items were placed first and, where possible, grouped. It should be noted that there is a fundamental difference between the known examples in the literature (see for example [71]) and the proposed approach. In these models the width of the strips is determined using an optimization technique, thus it is not fixed from the beginning.

This means that a strip can be formed from a number of items spread across it starting from the bottom of the bin. After a number of numerical tests we found that the packing algorithm is much faster and efficient when the strip width is determined only by one item placed at the bottom. The first stage of packing was done using only the heuristic algorithm and the overall efficiency of the packing achieved was 79:5%, which is already a relatively

No.	H	W	Qty	No.	H	W	Qty
1	39	38	4	19	30	9	4
2	39	26	4	20	30	7	6
3	39	7	10	21	28	20	9
4	38	28	3	22	25	14	9
5	38	21	4	23	25	12	9
6	38	20	2	24	25	10	8
7	37	28	9	25	25	8	9
8	37	23	1	26	24	10	5
9	37	9	10	27	21	12	8
10	33	33	10	28	20	20	6
11	36	27	10	29	20	11	10
12	35	32	4	30	20	5	2
13	34	27	8	31	19	6	10
14	34	5	5	32	17	10	3
15	34	5	2	33	17	10	11
16	32	14	1	34	14	12	6
17	32	7	4	35	11	10	3
18	30	13	7	36	8	7	9

Table 6.1: A set of 36 different rectangular items for packing, into a 260 x 160 mm rectangular bin. The dimensions of items are given in millimetres.

good result. The efficiency is calculated the same way as the fitness function, as a ratio of the total area of the packed items divided by the total area used in the container. In simple cases the heuristic algorithms usually result in optimum or near optimum solution. However, it is unlikely in more complex problems, such as the one considered here. Then, further improvement is desirable. The genetic algorithm was used for this purpose in this example. The chromosome string contains only the information about the sequence of the items and their numbers. In our case it was possible to code the whole set of the parameters into one string. Otherwise, some fixed amount of items or bins is used at one time. The application of the genetic algorithm has resulted in fast improvement of the packing efficiency up to 92:8 %, which can be considered as a very good result, and seems to be a near optimal one, taking into account a complexity of the problem.

6.2 3D bin packing problem

In order to evaluate the performance of our algorithm, a set of experiments were implemented based on the famous benchmarks, proposed by Bischoff E and Ratcliff M [72]. The developed EA was tested on several small experiments at first in order to find a relativity good parameter setting, and observe the performance of the new components.

In the end of this section, an overall test is arranged to our algorithm, and the results produced by this algorithm are compared with several optimizers presented in table 6.3, they are the most efficient in the literatures.

In section 6.2.1, The detail of the test benchmark is introduced. Section 6.2.2 presents the different parameter configuration and in section 6.2.4, the final experiment results are shown.

In order to observe a practical packing situation of the solution, the relevant location data of each layer is organized into a table, and a 3D figure is used to illustrate the packing pattern in appendix C.

6.2.1 Benchmark Description

Currently, there are seven benchmarks used in this research work (BR1-BR7), which totally include 700 problem instances. This test set was proposed by Bischoff E and Ratcliff M. They are widely used to evaluate the container loading optimization algorithm in many literatures. The benchmarks can either be downloaded in (http://paginas.fe.up.pt/ esicup/tiki-index.php) or generated by a problem generator which can be implemented by following the algorithm 6.1.

Algorithm 6.1 Deepest-Bottom-Left Sorting (S) Initialization:

```
Set Cargo target volume $Tc$;
Set Number of different Box type $n$;
Set Lower and upper limits on box dimensions $a_j, b_j, j \in [1, 3]$;
Set Box stability limit $L$;
Set Seed number $s$; //using in the random number generator;
Set box type index $i = 1$;
Initialized random number generator and discard first 10 random number;
Let $InstanceNub$ be the total number of the instance for each problem.
```

Iteration:

```
1: // the main loop used to generate different instances for each problem;
2:         for $j = 1$ to $InstanceNub$ do
3:          //Generate $n$ number type box;
4:          for $i = 1$ to n do
5:          repeat
6:    Generate 3 random number $r_j, j \in [1;3]$
7:          Determine box dimensions using:
8:          $d_{ij} = a_j + [r_j\text{x}(b_j - a_j + 1)], j \in [1;3]$;
9:  until (All $[d_{ij} = min(d_{ij})] < L$)
10: Initialize box quantity $m_i$ for box type $i$ : $m_i = 1$;
11:          Let the box volume $v_i = \prod_{j=1}^{3} d_{ij}$ ;
12: end for
13:  //Give the quantity to each box type $i$;
14:      repeat
15:    Calculate cargo volume: $C = \sum_{i=1}^{n} m_i v_i$
16:  Generate the next random number $r$
    and set box type indicator $k = 1 + [r\text{x}n]$
17: $m_k = m_k + 1$;
18: until $(T_c > C + v_k)$
19:   end for.
```

Output: Problem set;

In order to generate these seven benchmarks, the parameters in algorithm 6.1 should be set by following the table 6.2, in which the T_c is defined as the container volume, which can be represented by L x W x H. In addition, there are 100 instances generated for each problem. Thus, there are seven hundreds instances included in the test set. The random number generator in this algorithm is a special variant, suggested in [68], of the multiplicative congruential method and, as pointed out in [31], can be implemented in most programming languages on almost any computer. The seed number s is used in this random number generator in order to reproduce the individual problems without generating the complete set. To calculate the seed s we use the formula $s = 2502505 + 100(p - 1)$, where p is the problem number.

PID	Container				Box Type				Instance Number
	TC			L	Dimension			Quantity	
	L	W	H		(a_1, b_1)	(a_2, b_2)	(a_3, b_3)		
1								3	
2								5	
3								8	
4	586	233	220	2	$(30, 120)$	$(25, 100)$	$(20, 80)$	10	100
5								12	
6								15	
7								20	

Table 6.2: Parameters Setting of Problem Generator

6.2.2 Optimizer Parameter Configuration

Finding the best parameter setting for the GA is also an optimization problem. Normally, the Evolutionary Strategy (ES) can be used to tackle the parameter tuning optimization problem. In this work, we just arrange a set experiments by tuning the parameter in our algorithm in order to find a relatively good parameter setting. The corresponding parameter settings in this algorithm are presented in table 6.3.

Prarameter	Interval
POS	20
TOP	0.1 - 0.7
BOT	0.1 - 0.5
PC	0.1 - 0.5
ROTA	YES - NO

Table 6.3: The Algorithm Parameter Setting

The POS represents the population size, the PC is the crossover rate, and the ROTA is a fag that indicates whether the box will be allowed to rotate in this algorithm. According to the above table, we can see that there are totally 192 possible configurations for each problem, and each problem also has 100 instances. By following the above setting, there are an enormous number of experiments needed. Thus, in order to reduce the running time of the total experiments, two computers with Intel 5-core CPU running the Window 8 operating system were used to carry out these experiments. For each computer, three independent runs of the algorithm were made.

6.2.3 Different Components Setting

In this project, the original initialization component and mutation operator of the developed evolutionary algorithms (EA) were modified. For the original initialization, the position pattern generator was modified by a new method, which can generate a quite random position pattern. For the mutation operator, the original one will be replaced by a standard evolutionary system (ES) type mutation operator. In order to observe the effect of these modifications, the experiments in this section will be built up by running the developed EA with the new components and old version. The results generated are compared between the new version and the old version. The details of these part experiments will be seen in the subsequent sections.

6.2.4 Experiment Implementation

The average utility rate of 100 instances of each problem that are produced by this optimizer will be compared with eight efficient methods in table 6.4. Since we cannot obtain the code of these algorithms, the results that are produced by these approaches are directly collected from its corresponding literatures.

Algorithm	Source	Method
GH_L	Lim	Greedy Heuristic
PTS_BO	Bortfeldt	Parallel Tabu Search
PST_M	Mack	Parallel SA/TS
GRA(200000)_P	Parreno	GRASP
GRA(5000)_P	Parreno	GRASP
VNS_P	Parreno	VNS
TRS_FB	Fanslau and Bortfeldt	TRS
HBS_HH	He and Huang	Heuristic Beam Search

Table 6.4: The Approaches in Comparison

6.2.4.1 The Components Modification

The experiments in this section are divided into two subsections. The tests in the first subsections were organized by implementing the modified position pattern generator into the developed EA while other components were set with the unmodified position pattern generator. In experiments of the second subsection, the ES type mutation operator are used to replace the previously used position pattern generators. All mentioned experiments were implemented based on the seven benchmarks. The parameters fixed in this algorithm were the TOP with 0.15, the BOT with 0.15, the POS with 500, and the number of generation with 500.

- **New Position Pattern Generator**
  According to analysing the result shown in Fig.6.1, it can be seen that the results produced by using the modified "new" pattern generator on benchmarks 1, 2, 3, and 4 were better than using the unmodified pattern generator. While these four benchmarks are the weakly heterogeneous problem. Thus, it is considered that a more random position pattern generator could improve the final results on the weakly heterogeneous benchmarks. However, for the strongly heterogeneous benchmarks, this pattern generator cannot produce a satisfactory result.

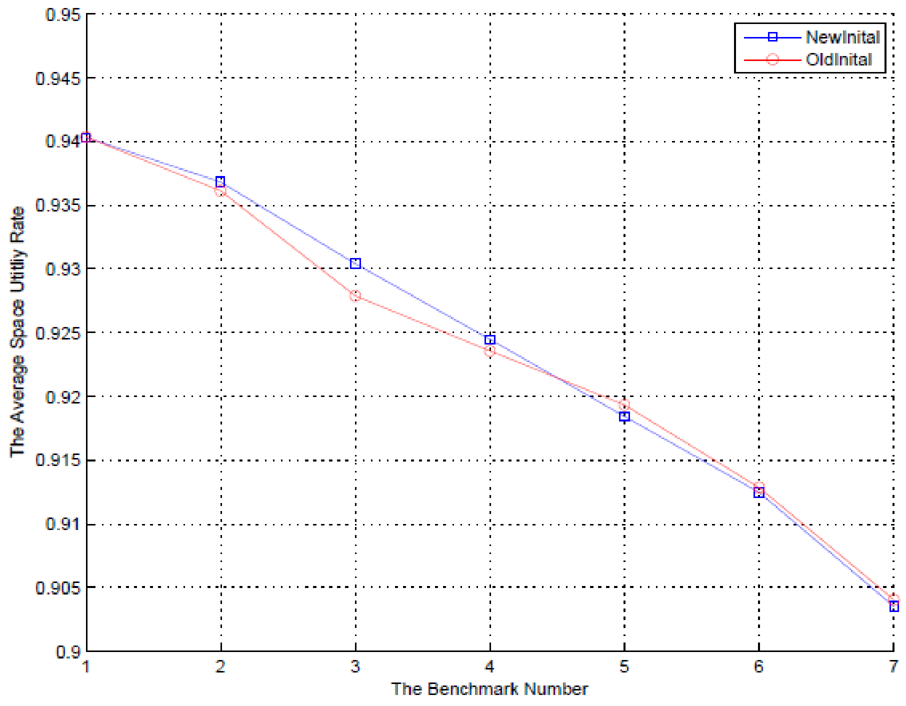


Figure 6.1: Modifed Position Pattern Generator vs Unmodified Position Pattern Generator.

- **ES Type Mutation operator**

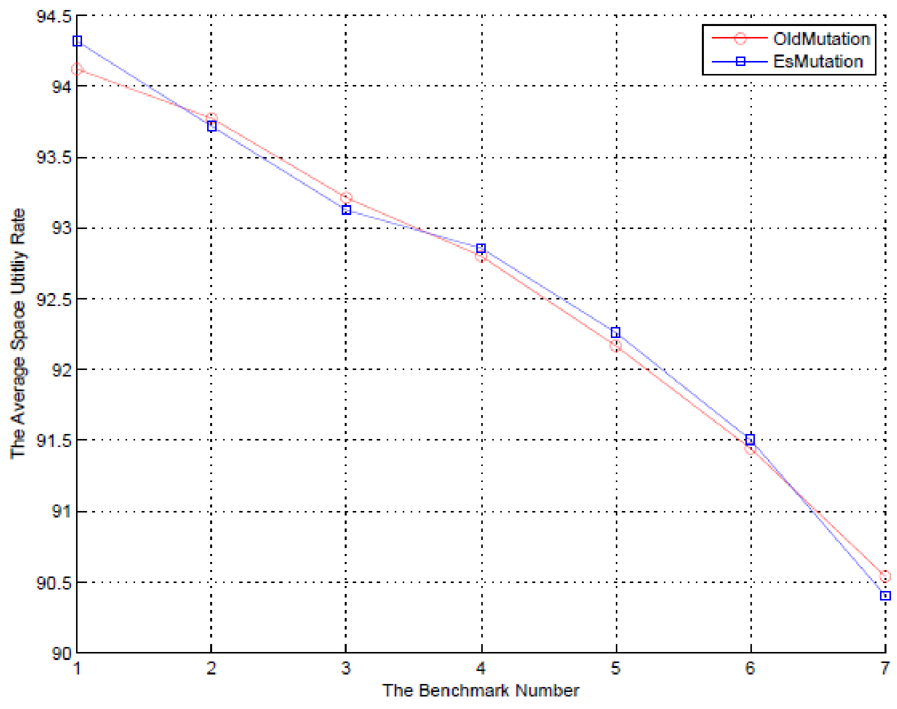


Figure 6.2: New Mutation Operator vs Old Mutation Operator

According to above experiment results the ES type mutation operator provided a positive effect for the the developed EA because the algorithm with the ES type mutation operator can generate a good result on most benchmarks, especially in benchmark 1. Thus, in experiments of later sections, the original mutation operator will be replaced by the ES type mutation operator.

6.2.4.2 Parameter Tuning

In order to find a relatively good parameter setting before running the overall tests, we started with implementing several small experiments, which were organized by tuning the parameters in table 6.2, but the population size was fixed with 500 iterations in order to reduce the experiment time. For each parameter portfolio in each benchmark, the algorithm runs five times, and each time had 500 iterations.

- **Crossover Rate Tuning**

The first group experiments were produced by applying nine different crossover rates from 0.1 to 0.7 within 0.05 intervals. Here the population size was set to be 500, the TOP block was 0.15%, the BOT was 0.15%, and the ROTA was allowed.

The second group experiments were included in Fig.6.3, which summarised the results that were produced by implementing this algorithm with different crossover rates and benchmarks. In order to find a suitable crossover rate, there were nine experiments illustrated in Fig.6.3(a), which were generated by applying this algorithm with tuning crossover rate from 0.1 to 0.7, to process the benchmark 5. According to the Fig.6.3(a), there was a slightly increasing trend from C01 to C65. However, after that the results dropped below the average level. Thus, the crossover rate was set to be 0.65% and applied it to process all benchmarks in order to observe the overall performance on all benchmarks.

To implement the tests in Fig.6.3(b), two other crossover settings which are close to 0.65% were introduced, in order to see the comparison result by running on all benchmarks. Although there was not any significant difference between the results that were generated by using these crossover rates, the performance of the algorithm with the crossover rate 0.65% on all benchmarks was slightly better than others. For the experiments in Fig.6.3(b), there is one more point, and we should touch on, that to handle the weakly heterogeneous benchmarks a smaller crossover rate would be appreciated, as mentioned before the heterogeneous strength of benchmarks was raised by increasing the number of box type in the problem. As an example of on first five benchmarks, the results of crossover rate with 0.7 are almost always lower than two others because the heterogeneous strength of rest two benchmarks is significantly stronger than the first five benchmarks. For a number of box type in first five benchmarks, they are from 3 to 12. For the rest of two benchmark (6,7), the number of box type are 15 and 20 respectively. Otherwise, for the strongly heterogeneous problems, a higher crossover rate can be a good choice.

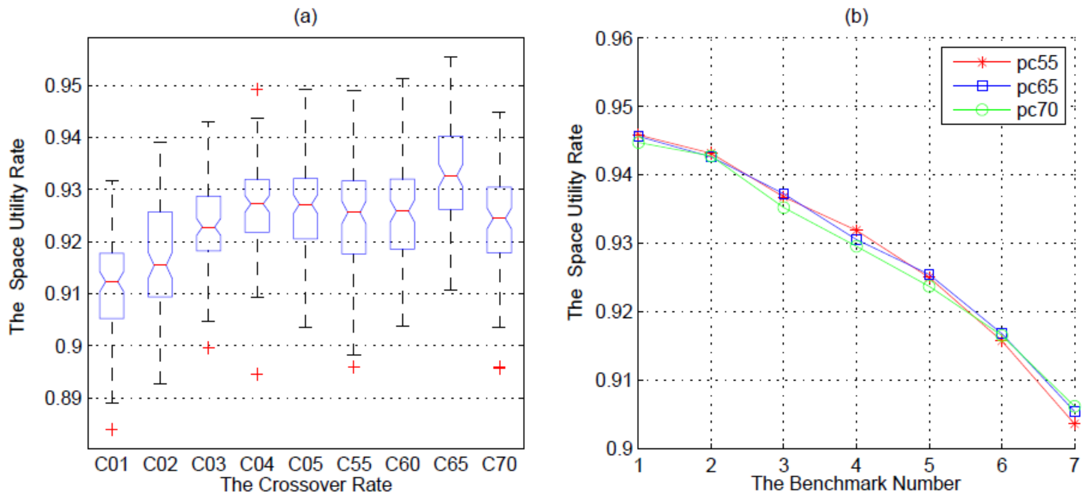


Figure 6.3: Different Crossover Comparison

(a): The labels on the x-axis represent the different crossover rates. For example, C01 means that the crossover rate is 0:1. Here, all experiments run on benchmarks 5.
(b): The labels on the x-axis represent the different benchmarks. Here, all experiments are implemented by applying different benchmarks.

- **BOT Size Tuning**

  The individuals in the BOT block are only generated by the mutation operator. Thus, the size of the BOT set can be considered as a control parameter of this genetic algorithm. By tuning this parameter could build up the experiments in this section. Here, the range of this parameter was set from 0.1 to 0.5 while the rest of the parameters were fixed based on the result of previous experiments.

  This part of experiments were divided into two steps. The first step of experiments was to implement this algorithm on a fixed benchmark with tuning the size of the BOT in order to find a reasonable setting. Here, the benchmark 5 was still used for generating the tests in the first step because the best results of different tuning steps were compared in the later part of this section. For the second step, the algorithm with three fixed BOT sizes, one of which was the best setting which was found through the first step experiments, was applied on all benchmarks in order to observe the overall performance of this best setting on seven benchmarks.

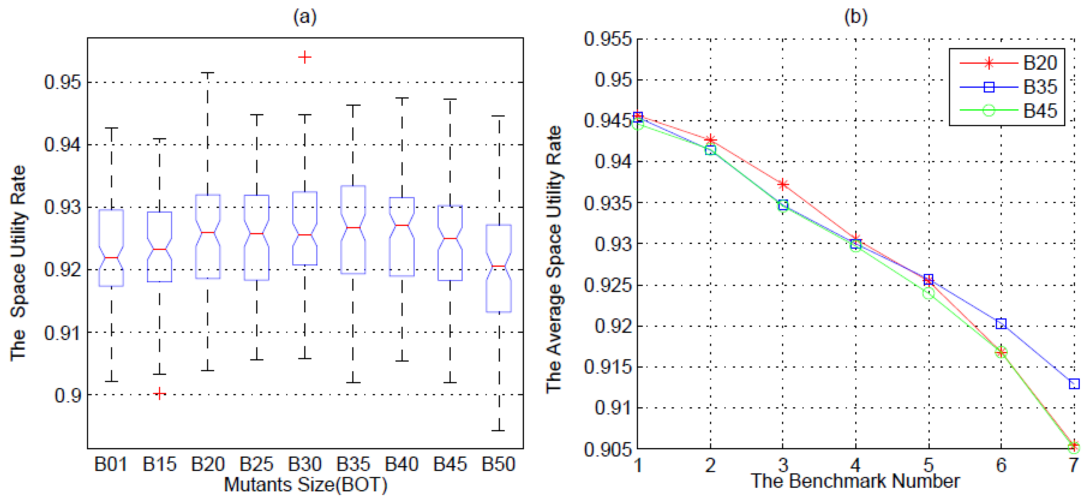


Figure 6.4: Different Crossover Comparison

(a): The labels on the x-axis represent the different BOT sizes. For example, B01 means that the mutants size is 10% of whole population size. Here, all experiments run on benchmark 5.
(b): The labels on the x-axis represent the different benchmarks. Here, all experiments with different BOT sizes are implemented by applying different benchmarks.

The results of the two steps tests were demonstrated by Fig.6.4. Fig.6.4(a), illustrates the nine experiments which were arranged according to applying this algorithm on benchmark 5 by tuning the size of the BOT from 0.1 to 0.5. Based on observing the median and the main body of the box in the box-plot, there was a slight increasing from B01 to B35 while the point B35 can be considered as a divide of the data after it illustrated a decreasing trend. Thus, the size of the BOT set to 0.35 was a reasonable setting.

For the second step in Fig.6.4, three different BOT settings, 0.2, 0.35 and 0.45, were used in this algorithm to process the seven benchmarks. The reason to choose the size with 0.2 was that the performance of this setting in benchmark 5 can be seen as a second favourable setting. It could have a big opportunity to produce a good result on different problems,. The size of the BOT with 0.45 was selected because this setting is close to the best one, and the solution that were generated by this setting can cover a more wide range. The result, shown in Fig.6.4(b), was the average space utility rate of each benchmark. We can see that the setting with 0.35 had reasonably good performance on overall views, and mainly it can produce a better result than two others when the benchmark becomes more strongly heterogeneous such as the problems from 5 to 7. For the weakly heterogeneous problem such as 2 and 3, the algorithm worked out unsatisfactorily.

Therefore, depending on the result analysis, it was found that the settings with low values are rather appreciated by more weakly heterogeneous problems. However, for more strongly heterogeneous problems, the settings does not have such discipline.

- **The Size of Elites Set Tuning (TOP)**

  To organize the experiments in this portion the algorithm was applied on the benchmark 5 by setting up with two fixed parameters and a variable one. The two fixed parameters were the crossover rate and the size of the BOT, which were set with 0.65 and 0.35 respectively by referred the previous experiments. The size of the TOP was selected as the variable one, which was tuned from 10% of the whole population size to 50%. After finding the best TOP size in the first phase, the setting was used by this algorithm to process the seven benchmarks. The final results were used to make a comparison with the results that were generated by applying two different settings within the same type parameter.

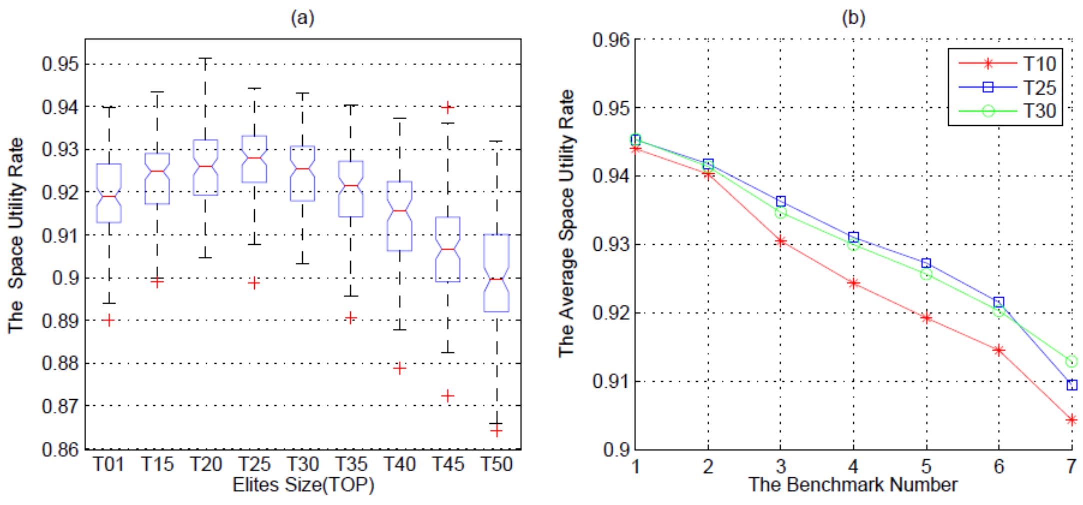


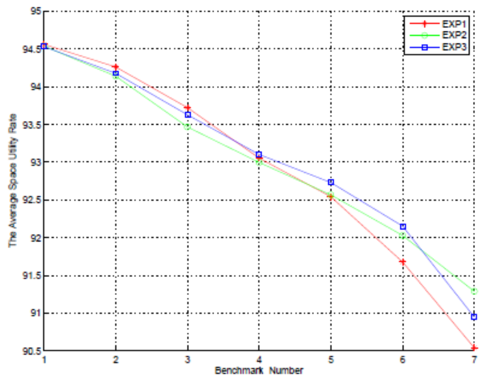
Figure 6.5: Different Elites Size Comparison

(a): The labels on the x-axis represent the different Elites Size. For example, the T01 means that the Elites Size is 10% of whole population size. Here, all experiments run on benchmark 5.
(b): The labels on the x-axis represent the different benchmarks. Here, all experiments with different Elites size are implemented by applying different benchmarks.

The Fig.6.5 manifested the result of the two aspects experiments. In Fig.6.5(a), the overall trend of the distribution of relevant result was slightly similar with a parabola. Although the best solution was obtained by implementing the setting with 20%, the reasonable peak of this parabola should be a setting with 25%. Since the average space utility of the benchmark five that was generated by the setting with 25% was the biggest, the TOP size with 25% of population size should be a reasonable setting. For the Fig.6.5(b), it illustrated three groups of results that were generated by three different TOP sizes, which were 10%; 25% and 30%, on seven benchmarks. It was clearly seen that the result produced by the setting with 25% was almost owing above the two others on every benchmarks, except the benchmark 7, so to fix the parameter with this value could be a desirable choice. According to the result of two other settings, it can also be proved that the settings that are higher or lower than 25% cannot give a valuable help to our algorithm on these benchmarks.

- **Parameter Overall Comparison**

  According to the above experiments, the best parameter settings had already been found as seen in the below table. In order make an overall view about the performance of this setting, TOP with 25%, the mutants size with 35%, and the crossover rate with 65%, the best results in each step were illustrated by the Fig.6.6. Through this figure, we can see that the parameter settings that were found by the experiments 6.5 had more stability than two others. Thus, we will implement this parameter setting in the later final experiments to generate the overall final result.



The Best Parameter Setting

Parameter	Interval
POS	20
TOP	0.25
BOT	0.35
PC	0.65
ROTA	YES

Figure 6.6: Three Tuning Steps Comparison

6.2.5 The Overall Comparison Experiments

To implement the experiments in this section, this algorithm will be set with the parameter found in above sections to solve the seven benchmarks. The overall results in table 6.4 were the average space utility rate of 100 instances of each problem, which are referred from [68].

BID	GH	PTS	PST	GRA(a)	GRA(b)	VNS	TRS	HBS	GA	EA
1	88.70	93.52	93.70	93.85	93.27	94.93	95.05	87.57	95.28	**95.38**
2	88.17	93.77	94.30	94.22	93.38	95.19	95.43	89.12	95.90	95.69
3	87.52	93.58	94.54	94.25	93.39	94.99	95.47	90.32	96.13	95.61
4	87.58	93.05	94.27	94.09	93.16	94.71	95.18	90.57	96.01	95.38
5	87.30	92.34	93.83	93.87	92.89	94.33	95.00	90.78	95.84	95.03
6	86.86	91.72	93.34	93.52	92.62	94.04	94.79	90.91	95.72	94.79
7	87.15	90.55	92.50	92.94	91.86	93.53	94.24	90.88	95.29	94.26

Table 6.5: The Final Comparison Result

According to observing the results in table 6.5, the the developed evolutionary algorithm (EA) shows the best result on BID 1. For rest of test cases, without considering the GA, EA defected all other algorithms on overall view. However, for the benchmark 6, result of our optimizer was same as the TRS.

	GH	HBS	PTS	PST	GRA(a)	VNS	TRS	EA	GA
Average (1-7)	87.61	90.02	92.65	93.78	93.82	94.53	95.02	95.16	95.74

Table 6.6: Average Utility Rate(B1-B7)Result

In order to analyse the performance of each algorithm in all problems, the overall average utility rate were calculated, and the algorithms will be ranked into an increasing order by using this result, as shown in table 6.6. Two worst results were produced by the GH and the HBS, they are much less than rest algorithms. by analysing the two algorithms, it is found that the GH and the HBS packs each item independently. In GH, boxes are placed on the wall of the container first so as to construct a base, after that other boxes will be placed on top of these boxes. For the HBS, in each packing iteration, a packing item will be put into the packing space first in order to detect a packing layer, after that the algorithm will based on this layer to pack the different type items. This kind of packing heuristic will produce too much piecemeal spaces which cannot use to pack the item, so this two algorithms cannot produce a good result. to compare the results of this two algorithms, HBS had a better result than the GH because the HBS will pack the item into layer first, which can reduce the number of piecemeal spaces. For the rest of algorithms, they will put same items into a same block or layer first, which won not generate too much piecemeal spaces. Thus, it is considered that the block and layer concept are more appropriate to handle the 3 dimensional container loading problem.

However, according to observing the PTS, it it found that to put the same items into a same block can reduce the dimension of the search space, but it will reduce the diversity of the search space as well. Thus, if the algorithm only depend on the block or layer concept, the result will easily be trapped into a local optima. From the PST to GA, all of them have a mechanism to add the diversity into the search space. For instance, the PST maintain multiple search paths simultaneously. and the GRA will add a randomization strategy to obtain different solution at each iteration. For the VNS and the TRS, they can combine two blocks or layers to produce new type blocks so that to add the diversity. In fact, in these algorithms, they try to find a way to balance this two aspect, low dimensional of search space and diversity of search space. The more suitable balance point it finds, The better result it obtains.

Therefore, for the GA, it is considered that there are three reasons to make it a better result in all benchmarks. The first one is that its representation includes two parts, namely BTPS and VLT. The BTPS can be seen as a packing plan. The item in here can be treated independently, which can offer the diversity of search space. For the VLT, the block and layer concept can be operated in this part, by which the dimension of search space will be reduced. The second one is that the GA is a population based algorithm. It can maintain multiple search paths simultaneously, which means that this kind of algorithms can have bigger chance to find a better result. The final one is the random key. The normal representation of solution for the combinatorial problem is the permutation. the algorithms in table 6.5 used this kind of representation, except the GA and EA. To use the permutation in the 3 dimensional container loading problem will involve an issue, in which the items in the permutation can not be packed completely. In this case, the representation can not provide a complete order relationship to the algorithm. To use the random key representation can avoid this defect. As mentioned, the items in the GA and the EA will ranged into a position pattern in the initialization component. This pattern never change. The random key just tag the each position in this pattern. All the operations in this kind of algorithms are just on the random keys. During the evolutionary process, the algorithm will find the relationship between the random key and corresponding position while the balance point between low dimensional and diversity of search space will be found as well. In this case, even though certain items can not be packed, the representation still can provide the useful information.

Chapter 7

Conclusion

7.1 Conclusion

In this research, a framework and methodology of using the GA with the heuristic algorithm (called EA) have been developed to solve the 2D and 3D bin packing problems. This research work taken up the problem of bins of constant sizes and rectangular shapes. To arrive with feasible bin packing pattern, many constraints were considered (e,g boundary crossing constraint, overlapping constrain, orientation constraint, etc). The developed EA module was also validated for its performance with the mathematical functions and found satisfactory.

The genetic parameters have been set by conducting various test cases and genetic operators have also been modified to suit the need. Decimal encoding has been applied to reduce the computational complexity where compared to the binary encoding methodology. The crossover operator has been developed to improve the convergence rate and it was found that the performance was better when compared to other methods in table 6.5.

The mutation operator has been developed to change the gene at random, thereby the overloaded mutation operator avoids the stagnation and increases the search space. The best orientation of bins inside the container has been identified by overloading the mutation operator with overloading constraints. Thus it is proved that the overloaded mutation operator performs better when compared to traditional mutation operators.

Bin packing inside the container has been achieved by layer-by-layer bin packing concept. The EA developed has yielded a better performance which could not have been possible with traditional simple GA.

7.2 Advantages

The principal advantage of the developed EA is to minimize the empty space inside the bin by considering the bin shape and satisfying the bin packing constraints. The other advantages of the EA are given as follows.

- The user input can be manually entered or by feeding directly from the database which makes it ease for the user.
- Each and every parameter is stored in the database for further analysis.
- The genetic parameters can be varied by the user based on the number of bins available.
- Separated algorithms for changing the orientation of the bins are eliminated, because the mutation operator is overloaded with the orientation constraints.
- The developed methodology overcomes the problem namely convergence rate, computational complexity, etc. associated with simple methods.
- The major constraints are considered in this work, which lead to feasible bin packing patterns.
- The solution reduce the variable costs in transportation of the bins and thereby reduces the overall product costs.

7.3 Applications

The developed EA module can be used to optimize the flowing problems:

- 1D, 2D and 3D bin packing problems
- Pallet loading problems
- Strip packing problems.

7.4 Future Work

The other interesting research openings for further investigation include the following:

- Time could be saved further if the algorithm was running on two or more computers. This is due to that one computer can create new generations, while the other computer checks the bin packing problem validity. It is suggested that a *search tree* be created and run the algorithm parallel on the same solution space. This need a huge amount of resources, but can increase the effectiveness of the algorithm.

- The research can be extended to multi-bin packing problems with modifications in the fitness function.

7.5 Research Output

As a result of this research work, a conference paper was produced. The paper addressed the 2D bin packing problem and provided efficient solutions for solving this class of optimization problems. The paper was presented at the 6th IASTED African Conference on Modelling and Simulation. AfriMS2016. September 5 - 7, 2016. Gaborone Botswana. url-http://www.actapress.com/Abstract.aspx?paperId=456312

Bibliography

[1] B Akay and D Karaboga. A modified artificial bee colony algorithm for real-parameter optimization. 192:120–142, 2012.

[2] M Amintoosi, H.S Yazdi, M Fathy, and R Monsefi. Using pattern matching for tiling and packing. *European Journal of Operational Research 183(3)*, (950-960), 2007.

[3] J S Arora. *Introduction to Optimum Design*. 3rd edition, 2012.

[4] T Bayraktar. A memory-integrated artificial bee algorithm for heuristic optimisation, 2014.

[5] JE Beasley. An exact two-dimensional non-guillotine cutting tree search procedure. *33(1)*, pages 49–64, 1985.

[6] Murat Ercsen Berberler, Urfat Nuriyev, and Ahmet Yildirim. A software for the one-dimensional cutting stock problem. *Journal of King Saud University-Science*, 23(1):69–76, 2011.

[7] Judith O Berkey and Pearl Y Wang. A parallel approximation algorithm for solving one-dimensional bin packing problems. In *Parallel Processing Symposium*, pages 138–143. IEEE, 1991.

[8] Marco A Boschetti. New lower bounds for the three-dimensional finite bin packing problem. *Discrete Applied Mathematics*, 140(1):241–258, 2004.

[9] Marco Cavazzuti. *Optimization methods: from theory to design scientific and technological aspects in mechanics*. Springer Science and Business Media, 2012.

[10] JWM Chan and KK Lai. Developing a simulated annealing algorithm for the cutting stock problem. *32*, pages 115–27, 1997.

[11] Tinggui Chen and Renbin Xiao. Enhancing artificial bee colony algorithm with self-adaptive searching strategy and artificial immune network operators for global optimization. *The Scientific World Journal*, 2014, 2014.

[12] Edward G Coffman, Jr, Peter J Downey, and Peter Winkler. Packing rectangles in a strip. *Acta Informatica*, 38(10):673–693, 2002.

[13] Edward G Coffman, Jr, Michael R Garey, David S Johnson, and Robert Endre Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9(4):808–826, 1980.

[14] EG Coffman and Peter W Shor. Packings in two dimensions: Asymptotic average-case analysis of algorithms. *Algorithmica*, 9(3):253–277, 1993.

[15] EG Coffman and PW Shor. Average-case analysis of cutting and packing in two dimensions. *European Journal of Operational Research*, 44(2):134–144, 1990.

[16] Edward G Coffman Jr, Michael R Garey, and David S Johnson. Approximation algorithms for bin-packingan updated survey. In *Algorithm design for computer system design*, pages 49–106. Springer, 1984.

[17] C C Columbus and and S P Simon K Chandrasekaran. Nodal and calony optimization for solving profit based unit commitment problem for gencos. 12:145–160, 2012.

[18] Isabel Correia, Luis Gouveia, and Francisco Saldanha-da Gama. Solving the variable size bin packing problem with discretized formulations. *Computers & Operations Research*, 35(6):2103–2113, 2008.

[19] Teodor Gabriel Crainic, Guido Perboli, and Roberto Tadei. Extreme point-based heuristics for three-dimensional bin packing. *Informs Journal on computing*, 20(3):368–384, 2008.

[20] Janos Csirik, JBG Frenk, Gabor Galambos, and AHG Rinnooy Kan. Probabilistic analysis of algorithms for dual bin packing problems. *Journal of Algorithms*, 12(2):189–203, 1991.

[21] E Goldberg D. *Genetic algorithms in search, optimization and machine learning.* Addison- Wesley,Reading, MA, 1989.

[22] S Das, R Mallipeddi, and D Maity. Adaptive evolutionary programming with p-best mutation strategy. pages 58–68, 2013.

[23] T Dereli and G S Das. A hybrid bees algorithm for solving container loading problems. 11:2854–2862, 2011.

[24] Ugur Dogrusoz. Two-dimensional packing algorithms for layout of disconnected graphs. *Information Sciences*, 143(1):147–158, 2002.

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

[26] M Eley. A bottlenneck assignment approach to the multiple container loading problem. *25(1)*, pages 45–60, 2003.

[27] Henrik Esbensen. A genetic algorithm for macro cell placement. In *Design Automation Conference*, pages 52–57. IEEE, 1992.

[28] John A Ford and Adel F Saadallah. On the conditioning of the hessian approximation in quasi-newton methods. *Journal of Computational and Applied Mathematics*, 35(1-3):197–206, 1991.

[29] G Fuellerer, K F Doerner, R F Hartl, and M Iori. Ant colony optimization for the two-dimensional loading vehicle routing problem. 36(3):655–673, 2009.

[30] W Gao and S Liu. Improved artificial bee colony algorithm for global optimization. 111:871–882, 2011.

[31] T. Gau and G. Wascher. Cutgen1: A problem generator for the standard one-dimensional cutting stock problem. *84(3)*, pages 572 –579, 1995.

[32] John A George. Multiple container packing: a case study of pipe packing. *Journal of the Operational Research Society*, 47(9):1098–1109, 1996.

[33] Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting stock problempart ii. *Operations research*, 11(6):863–888, 1963.

[34] J F Goncalves and M G C Resende. A biased random key genetic algorithm for 2d and 3d bin packing problems. pages 500–510, 2013.

[35] JF Hair, WC Black, BJ Babin, RE Anderson, and RL Tatham. Pearson prentice hall upper saddle river, 2006.

[36] K He and W Huang. A caving degree approach for the single container loading problem. pages 196(1):93–101, 2009.

[37] Eva Hopper and Brian CH Turton. A review of the application of meta-heuristic algorithms to 2d strip packing problems. *Artificial Intelligence Review*, 16(4):257–300, 2001.

[38] Chin-Yuan Hsu, Fu-Yao Ko, Chia-Wei Li, Kuni Fann, and Juh-Tzeng Lue. Magnetoreception system in honeybees (apis mellifera). *PloS one*, 2(4):e395, 2007.

[39] Wenqi Huang and Duanbing Chen. An efficient heuristic algorithm for rectangle-packing problem. *Simulation Modelling Practice and Theory*, 15(10):1356–1365, 2007.

[40] Shian-Miin Hwang, Cheng-Yan Kao, and Jorng-Tzong Horng. On solving rectangle bin packing problems using genetic algorithms. In *Systems, Man, and Cybernetics*, volume 2, pages 1583–1590. IEEE, 1994.

[41] Richard Johnsonbaugh and Marcus Schaefer. *Algorithms*, volume 2. Pearson Education Upper Saddle River, NJ, 2004.

[42] Leonardo Junqueira, Reinaldo Morabito, and Denise Sato Yamashita. Threedimensional container loading models with cargo stability and load bearing constraints. *39(1)*, pages 74 –85, 2012.

[43] F Kang, J Li, and H Li. Artificial bee colony algorithm and pattern search hybridized for global optimization. 13:1781–1791, 2013.

[44] D Karaboga. *An idea based on honey bee swarm for numerical optimization*. PhD thesis, 2005.

[45] D Karaboga. Yapay zeka optimizasyon algoritmalari. *Bolum 1 - Giris*, pages 1–19, 2011.

[46] D Karaboga and B Akay. A comparative study of artificial bee colony algorithm. 214:108–132, 2009.

[47] Nihat Kasap and Anurag Agarwal. Augmented-neural-networks approach for the bin-packing problem. 2004.

[48] A Kaveh and M Khayatazad. A new meta-heuristic method: Ray optimization. pages 283–294, 2012.

[49] Jong-Kyou Kim, Hyung Lee-Kwang, and Seung W Yoo. Fuzzy bin packing problem. *Fuzzy Sets and Systems*, 120(3):429–434, 2001.

[50] G Laporte, S Martello, M Gendreau M, and MLori. A tabu search algorithm for a routing and container loading problem. *40(3)*, 2006.

[51] KY Lee and MA El-Sharkawi. Modern heuristic optimization tehcniques, pp, 2008.

[52] Z Li, Z Tian, Y Xie, R Huang, and J Tan. A knowledge-based heuristic particle swarm optimization approach with the adjustment strategy for the weighted circle packing problem. pages 1758–1769, 2013.

[53] Ming-Hua Lin, Jung-Fa Tsai, and Chian-Son Yu. A review of deterministic optimization methods in engineering and management. *Mathematical Problems in Engineering*, 12, 2012.

[54] Lauro Lins, Sostenes Lins, and Reinaldo Morabito. An n-tet graph approach for non-guillotine packings of n-dimensional boxes into an n-container. *European Journal of Operational Research*, 141(2):421–439, 2002.

[55] DS Liu, Kay Chen Tan, SY Huang, Chi Keong Goh, and Weng Khuen Ho. On solving multiobjective bin packing problems using evolutionary particle swarm optimization. *European Journal of Operational Research*, 190(2):357–382, 2008.

[56] W Liu and L Xing. The double layer optimization problem to express logistics systems and its heuristic algorithm. pages 237–245, 2014.

[57] Andrea Lodi, Silvano Martello, and Daniele Vigo. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 123(1):379–396, 2002.

[58] D Mack and A Bortfeldt. A heuristic for the three-dimensional strip packing problem. *European journal of operational Research*, pages 183(3):1267–79, 2007.

[59] N Maculan, JLC Silva, and NY Soma. A greedy search for the three-dimensional bin packing problem: the packing static stability case. *123(2)*, pages 141–53, 2003.

[60] M Kovacevic Madic and M Radovanovic. Software prototype for validation of machining optimization solutions obtained with meta-heuristic algorithms. 40(17):6985–6996., 2013.

[61] GHA Martins. Packing in two and three dimensions, 2003.

[62] Z Michalewicz and D.B Forgel and. *How to Solve It: Modern Heuristics.* Springer-Verlag Berlin Heidelberg, 2000.

[63] B Min, J Lewis, E T Matson, and A H Smith. Heuristic optimization techniques for self-orientation of directional antennas in long-distance point-to-point broadband networks. pages 2252–2263, 2013.

[64] Marcel Mongeau and Christian Bes. Optimization of aircraft container loading. *IEEE Transactions on Aerospace and Electronic Systems*, 39(1):140–150, 2003.

[65] Walter Murray. Newton-type methods. *Wiley Encyclopedia of Operations Research and Management Science*, 2010.

[66] L Ozbakir, A Baykasoglu, and P Tapkan. Bees algorithm for generalized assignment problem. 215:3782–3795, 2010.

[67] Roy P Pargas and Rajat Jain. A parallel stochastic optimization algorithm for solving 2d bin packing problems. In *Artificial Intelligence for Applications*, pages 18–25. IEEE, 1993.

[68] S. K. Park and K. W. Miller. Random number generators: Good ones are hard to find. *Commun. ACM, 31(10)*, 1998.

[69] D T Pham, A Ghanbarzadeh, E Koc, S Otri, S Rahim, and M Zaidi. The bees algorithm - a novel tool for complex optimisation problems. 2006.

[70] D Pisinger. Heuristics for the container loading problem. *European journal of operational Research*, 2002.

[71] J Puchinger and J R Gnter. Models and algorithms for three-stage two-dimensional bin packing, 2004.

[72] MSW Ratcliff and EE Bischoff. Issues in the development of approaches to container loading. *23(3)*, pages 77–90, 1995.

[73] A Sahu, Panigrahi, and S Pattnaik. Fast convergence particle swarm optimization for functions optimization. 4:319–324, 2012.

[74] SQ Shen, CS Chen, and SM Lee. An analytical model for the container loading problem. *80(1)*, pages 68–76, 1995.

[75] Mirko Skiborowski, Marcel Rautenberg, and Wolfgang Marquardt. A novel approach to hybrid evolutionary-deterministic optimization in process design. 2013.

[76] H Smith. Bin packing problem, 2001.

[77] U Sommerweiss, J Riehme, J Terno J, and G Scheithauer. An efficient approach for the multi-pallet loading problem. *European Journal of Operational Research*, pages 123(2): 372–81, 2000.

[78] Oliver Stein, Jan Oldenburg, and Wolfgang Marquardt. Continuous reformulations of discrete–continuous optimization problems. *Computers and chemical engineering*, 28(10):1951–1966, 2004.

[79] E Sulaiman, MZ Ahmad, T Kosaka, and N Matsui. Design optimization studies on high torque and high power density hybrid excitation flux switching motor for hev. *Procedia Engineering*, 53:312–322, 2013.

[80] Hao Sun and Raimondo Betti. Simultaneous identification of structural parameters and dynamic input with incomplete output-only measurements. *Structural Control and Health Monitoring*, 21(6):868–889, 2014.

[81] W Y Szeto, Y Wu, and S C Ho. An artificial bee colony algorithm for the capacitated vehicle routing problem. 215:126–135, 2011.

[82] M F N Tajuddin, S M Ayob, Z Salam, and M S Saad. Evolutionary based maximum power point tracking technique using differential evolution algorithm. 67:245–252, 2013.

[83] Gracca Trindade and Jorge Ambrosio. An optimization method to estimate models with store-level data: A case study. *European Journal of Operational Research*, 217(3):664–672, 2012.

[84] Y Wakabayashi and FK Miyazawa. Approximation algorithm for the orthogonal zoriented three dimensional packing problem. *SIAM Journal on Computing*, 29(3), 1999.

[85] G Wascher, H Haubner, and H Schumann. An improved typology of cutting and packing problems, 2006.

[86] Gerhard Wascher, Heike Haussner, and Holger Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130, 2007.

[87] C Whitlock and N Christofides. An algorithm for two-dimensional cutting problems. *44(2)*, pages 45–59, 1977.

[88] EC Xavier and Flávio Keidi Miyazawa. A one-dimensional bin packing problem with shelf divisions. *Discrete Applied Mathematics*, 156(7):1083–1096, 2008.

[89] W Xiang and M An. An efficient and robust artificial bee colony algorithm for numerical optimization. 40:1256–1265, 2013.

[90] B Xu, KK Lai, and J Xue. Container packing in a multi-customer delivering operation. *Computers and Industrial Engineering*, pages 35(1–2):323–6, 1998.

[91] S Xu, Z Ji, D T Pham, and F Yu. Bio-inspired binary bees algorithm for a two-level distribution optimisation problem. 7:161–167, 2010.

[92] De-Fu Zhang, CHEN Sheng-Da, and LIU Yan-Juan. An improved heuristic recursive strategy based on genetic algorithm for the strip rectangular packing problem. *Acta Automatica Sinica*, 33(9):911–916, 2007.

[93] Wenbin Zhu, Zhaoyi Zhang, Wee-Chong Oon, and Andrew Lim. Space defragmentation for packing problems. *European Journal of Operational Research*, 222(3):452–463, 2012.

Appendix A

Evolutionary Algorithm – Evolutionary Optimization Code

```
% Bin sizes:
a=200;
b=200;

Nb=9; % number of boxes

mrg=0.45; % for margine
aa=[78     150      180    150   160  120  110  145  75   ];
bb=[80     110      100    110   45   25   39    112  50  ];
aa=aa*mrg;
bb=bb*mrg;

m2=min([aa bb]/2); % smallest half-size
AA=aa.*bb; % boxes areas

 penalty=0.2*a*b;
 nac=0.8; % negative area coefficient


N=100; % population size
ng=2000; % number of generations
pmpe=0.1; % places exchange mutation probability
pmbj=0.02; % big gauss jump
pmsj=0.04; % small gauss jump
```

```
pmrr=0.1; % random rotation
pmvi=0.1; % random visible/invisible
pmne=0.2; % move to nearest adge

figure;
%ha1=axes;
ha1=subplot(2,1,1);
plot([0 a a 0 0], [0 0 b b 0],'b-');
xlim([-0.1*a 1.1*a]);
ylim([-0.1*b 1.1*b]);
set(ha1,'NextPlot','add');
ht=title(ha1,'start');
ha2=subplot(2,1,2);
drawnow;


set_cl; % set color table cl to plot boxes with different colors



% random initial population:
G=zeros(N,4*Nb);
Gch=zeros(N,4*Nb); % children
for Nc=1:N % for each individual
    G1=zeros(4,Nb); % one individual
    % G1(1,i)=1 if i-box is visible
    % G1(2,i)=1 if i-box is rotated at 90 degrees
    % G1(3,i) - x-coordinate of i-box center
    % G1(4,i) - y-coordinate of i-box center

    G1(1,:)=double(rand(1,Nb)<0.2);
    G1(2,:)=double(rand(1,Nb)<0.5);

    G1(3,:)=m2+(a-m2)*rand(1,Nb);
    G1(4,:)=m2+(b-m2)*rand(1,Nb);


    G(Nc,:)=(G1(:))'; % (G1(:))' converts matrix to row-vector
end
```

```

hi=imagesc(G,'parent',ha2);
drawnow;



Gpr1=zeros(4,Nb);
Gpr2=zeros(4,Nb); % two parents
Gch1=zeros(4,Nb);
Gch2=zeros(4,Nb); % two children
for ngc=1:ng % generations counting
    % find fitnesses:
    fitnesses=zeros(N,1);
    for Nc=1:N % for each individual
        G1(:)=(G(Nc,:))';
        vis=G1(1,:);
        ind=find(vis);
        L=length(ind);
        if L>0
            % only visible:
            rot=G1(2,ind);
            x=G1(3,ind);
            y=G1(4,ind);
            if L==1
                aaa=aa(ind);
                bbb=bb(ind);
                if rot
                    tmp=aaa;
                    aaa=bbb;
                    bbb=tmp;
                end
                A0=AA(ind); % box area
                x1=max([x-aaa/2  0]);
                y1=max([y-bbb/2  0]);
                x2=min([x+aaa/2  a]);
                y2=min([y+bbb/2  b]);
                % x1 - x2,  y1 - y2 is box (part of current box)
                   that inside main box
                if (x1>=x2)||(y1>=y2)
                    A=0; % box that inside main box area
                else
                    A=(x2-x1)*(y2-y1); % box that inside main box
                        area
                end
                %if A<A0 % if not fully inside main box
                if (aaa/2<=x)&&(x<=a-aaa/2)&&(bbb/2<=y)&&(y<=b-bbb
                   /2) % if filly inside
                    fitness=A;
                else
                    fitness=A-nac*(A0-A)-penalty;
                end

            else
                fitness=0;
                ispen=false; % true if penality

                % check cross with main box:
                % add boxes arreas and strong subtract out areas:
                for n=1:L % for each box
                    ind1=ind(n);
                    aaa=aa(ind1);
                    bbb=bb(ind1);
                    if rot(n)
                        tmp=aaa;
                        aaa=bbb;
                        bbb=tmp;
                    end
                    A0=AA(ind1); % box area
                    x1=max([x(n)-aaa/2   0]);
                    y1=max([y(n)-bbb/2   0]);
                    x2=min([x(n)+aaa/2   a]);
                    y2=min([y(n)+bbb/2   b]);
                    % x1 - x2,  y1 - y2 is box (part of current
                        box) that inside main box
                    if (x1>=x2)||(y1>=y2)
                        A=0; % box that inside main box area
                    else
```

```
                    A=(x2-x1)*(y2-y1); % box that inside main box area
                end
                %if A<A0 % if not fully inside main box
                    %fitness=fitness + A-nac*(A0-A);
                    %ispen=true; % penality
                %else
                    %fitness=fitness + A;
                %end

                if (aaa/2<=x(n))&&(x(n)<=a-aaa/2)&&(bbb/2<=y(n))&&(y(n)<=b-bbb/2) % if filly inside
                    fitness=fitness + A;
                else
                    fitness=fitness + A-nac*(A0-A);
                    ispen=true; % penality
                end

            end

            % for each pair of boxes:
            for n1=1:L-1
                ind1=ind(n1);
                aaa1=aa(ind1);
                bbb1=bb(ind1);
                if rot(n1)
                    tmp=aaa1;
                    aaa1=bbb1;
                    bbb1=tmp;
                end
                A1=AA(ind1);
                x1=x(n1);
                y1=y(n1); % position of 1st box of pair
                for n2=n1+1:L
                    ind2=ind(n2);
                    aaa2=aa(ind2);
                    bbb2=bb(ind2);
                    if rot(n2)
                        tmp=aaa2;
                        aaa2=bbb2;
                        bbb2=tmp;
                    end
                    A2=AA(ind2);
                    x2=x(n2);
                    y2=y(n2); % position of 2nd box of pair
                    dx=abs(x1-x2);
                    dy=abs(y1-y2); % distancies
                    a12=(aaa1/2+aaa2/2);
                    b12=(bbb1/2+bbb2/2);
                    if (dx<a12)&&(dy<b12) % if cross
                        ispen=true;
                        Ac=(a12-dx)*(b12-dy); % area of cross
                        fitness=fitness-Ac-Ac; % becuse area of n1 and n2 was added fully
                        fitness=fitness-2*nac*Ac;
                    end

                end
            end

            if ispen
                fitness=fitness-penalty;
            end

        end
    else
        fitness=0;
    end
    fitnesses(Nc)=fitness;
end

[fb bi]=max(fitnesses); % best

% plot best:
G1(:)=(G(bi,:))';
Gb=G(bi,:); % best
if mod(ngc,10)==0
    cla(ha1);
```

```
        Atmp=0;
        for Nbc=1:Nb
            vis1=G1(1,Nbc);
            if vis1
                rot1=G1(2,Nbc);
                aaa=aa(Nbc);
                bbb=bb(Nbc);
                if rot1
                    tmp=aaa;
                    aaa=bbb;
                    bbb=tmp;
                end
                x=G1(3,Nbc);
                y=G1(4,Nbc);
                plot([x-aaa/2 x+aaa/2 x+aaa/2 x-aaa/2 x-aaa
                    /2],...
                     [y-bbb/2 y-bbb/2 y+bbb/2 y+bbb/2 y-bbb
                        /2],...
                     '-','color',cl(Nbc,:),...
                     'parent',ha1);
                hold on;
                Atmp=Atmp+aaa*bbb;
            end
        end
        plot([0 a a 0 0], [0 0 b b 0],'b-','parent',ha1);
        xlim(ha1,[-0.1*a 1.1*a]);
        ylim(ha1,[-0.1*b 1.1*b]);

        set(hi,'Cdata',G);

        nvb=length(find(G1(1,:))); % number of visible boxes

        set(ht,'string',[' generation: ' num2str(ngc) ', boxes: '
            num2str(nvb) ', area: ' num2str(fb)]);

        drawnow;
    end


```

```
% prepare for crossover, selection:
fmn=min(fitnesses);
fst=std(fitnesses);
if fst<1e-7
    fst=1e-7;
end
fmn1=fmn-0.01*fst; % little low then minimum
P=fitnesses-fmn1; % positive values
p=P/sum(P); % probabilities
ii=roulette_wheel_indexes(N,p);
Gp=G(ii,:); % parents

% crossover:
for n=1:2:N
    pr1=Gp(n,:);
    pr2=Gp(n+1,:); % two parents
    % in matrix form:
    Gpr1(:)=pr1';
    Gpr2(:)=pr2';

    for Nbc=1:Nb

        % visibility:
        if rand<0.5
            Gch1(1,Nbc)=Gpr1(1,Nbc);
        else
            Gch1(1,Nbc)=Gpr2(1,Nbc);
        end
        if rand<0.5
            Gch2(1,Nbc)=Gpr1(1,Nbc);
        else
            Gch2(1,Nbc)=Gpr2(1,Nbc);
        end

        % rotation:
        if rand<0.5
            Gch1(2,Nbc)=Gpr1(2,Nbc);
        else
            Gch1(2,Nbc)=Gpr2(2,Nbc);
        end
        if rand<0.5
            Gch2(2,Nbc)=Gpr1(2,Nbc);
        else
            Gch2(2,Nbc)=Gpr2(2,Nbc);
        end

        % position:
        % child 1:
        %i3=ceil(3*rand);
        %i3=roulette_wheel_indexes(1,[0.2 0.4 0.4]);
        i3=1+ceil(2*rand);
        switch i3
            case 1 % get mean position
                Gch1(3,Nbc)=(Gpr1(3,Nbc)+Gpr2(3,Nbc))/2;
                Gch1(4,Nbc)=(Gpr1(4,Nbc)+Gpr2(4,Nbc))/2;
            case 2 %get position of parent 1
                Gch1(3,Nbc)=Gpr1(3,Nbc);
                Gch1(4,Nbc)=Gpr1(4,Nbc);
            case 3 %get position of parent 2
                Gch1(3,Nbc)=Gpr2(3,Nbc);
                Gch1(4,Nbc)=Gpr2(4,Nbc);
        end
        % child 2:
        %i3=ceil(3*rand);
        %i3=roulette_wheel_indexes(1,[0.2 0.4 0.4]);
        i3=1+ceil(2*rand);
        switch i3
            case 1 % get mean position
                Gch2(3,Nbc)=(Gpr1(3,Nbc)+Gpr2(3,Nbc))/2;
                Gch2(4,Nbc)=(Gpr1(4,Nbc)+Gpr2(4,Nbc))/2;
            case 2 %get position of parent 1
                Gch2(3,Nbc)=Gpr1(3,Nbc);
                Gch2(4,Nbc)=Gpr1(4,Nbc);
            case 3 %get position of parent 2
                Gch2(3,Nbc)=Gpr2(3,Nbc);
                Gch2(4,Nbc)=Gpr2(4,Nbc);
        end


        end
        ch1=(Gch1(:))';
        ch2=(Gch2(:))';
        Gch(n,:)=ch1;
        Gch(n+1,:)=ch2;


    end
    G=Gch; % now children

    % mutations:
    % places exchange
    for Nc=1:N % for each individual
        if rand<pmpe
            G1(:)=(G(Nc,:))';
            ir1=ceil(Nb*rand);
            ir2=ceil(Nb*rand);
            tmp1=G1(3:4,ir1);
            G1(3:4,ir1)=G1(3:4,ir2);
            G1(3:4,ir2)=tmp1;
            G(Nc,:)=(G1(:))';
        end
    end

    % big gauss jump:
    for Nc=1:N % for each individual
        if rand<pmbj
            G1(:)=(G(Nc,:))';
            ir=ceil(Nb*rand);
            G1(3:4,ir)=G1(3:4,ir)+[0.05*a*randn;
                                   0.05*b*randn];
            G(Nc,:)=(G1(:))';
        end
    end

    % small gauss jump:
    for Nc=1:N % for each individual
        if rand<pmsj
```

```
                G1(:)=(G(Nc,:))';
                ir=ceil(Nb*rand);
                G1(3:4,ir)=G1(3:4,ir)+[0.005*a*randn;
                                        0.005*b*randn];
                G(Nc,:)=(G1(:))';
            end
        end

    % random rotation:
    for Nc=1:N % for each individual
        if rand<pmrr
            G1(:)=(G(Nc,:))';
            ir=ceil(Nb*rand);
            G1(2,ir)=double(rand<0.5);
            G(Nc,:)=(G1(:))';
        end
    end

    % random visible/invisible:
    for Nc=1:N % for each individual
        if rand<pmvi
            G1(:)=(G(Nc,:))';
            ir=ceil(Nb*rand);
            G1(1,ir)=double(rand<0.5);
            G(Nc,:)=(G1(:))';
        end
    end

    % move to nearest edge:
    for Nc=1:N % for each individual
        if rand<pmne
            G1(:)=(G(Nc,:))';
            ir=ceil(Nb*rand); % random small box
            rv=find((G1(1,:))&((1:Nb)~=Nc)); % find rest visible
            if rand<0.5
                % to veritcile edge
                eax=[G1(3,rv)-aa(rv)/2  G1(3,rv)+aa(rv)/2  0  a];
                    % edge xs
                deax=[(G1(3,ir)-aa(ir)/2) - eax  (G1(3,ir)+aa(ir)
                    /2) - eax]; % distancies
                [dmn indm]=min(abs(deax));
                G1(3,ir)=G1(3,ir)-deax(indm);
            else
                % to horizontal edge
                eay=[G1(4,rv)-bb(rv)/2  G1(4,rv)+bb(rv)/2  0  b];
                    % edge ys
                deay=[(G1(4,ir)-bb(ir)/2) - eay  (G1(4,ir)+bb(ir)
                    /2) - eay]; % distancies
                [dmn indm]=min(abs(deay));
                G1(4,ir)=G1(4,ir)-deay(indm);
            end
        end
    end



    % ellitism:
    G(1,:)=Gb;





end
```

Appendix B

Evolutionary Algorithm – 2D Practical Packing Solution and Pattern

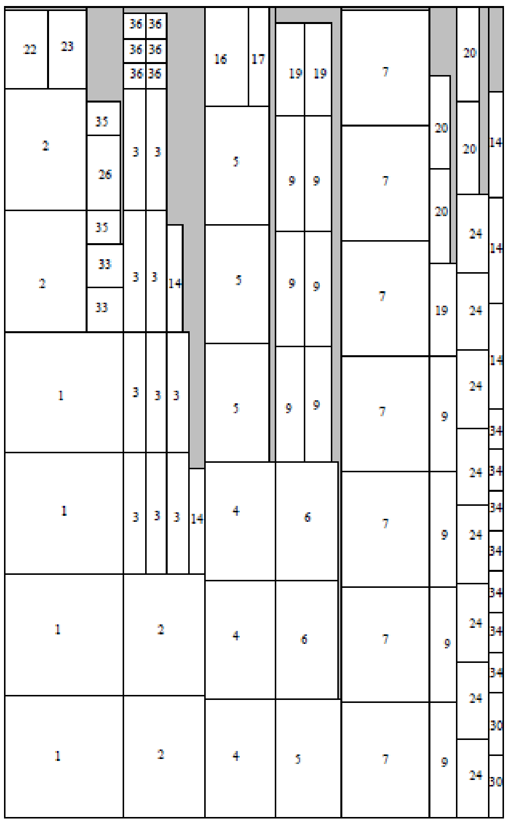


Figure B.1: Optimal placement of items in the first bin

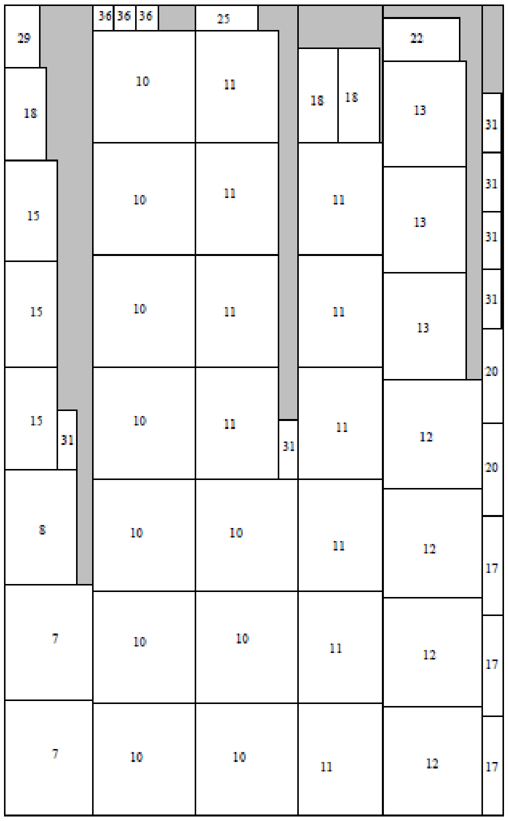


Figure B.2: Optimal placement of items in the second bin

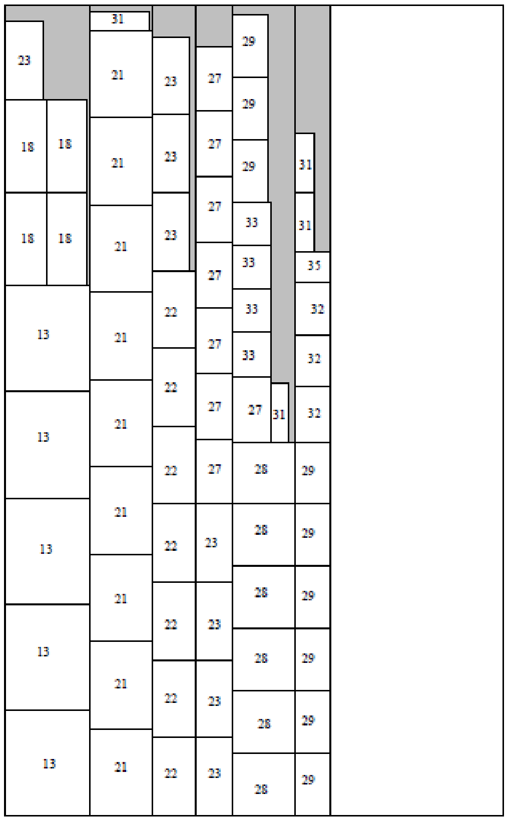
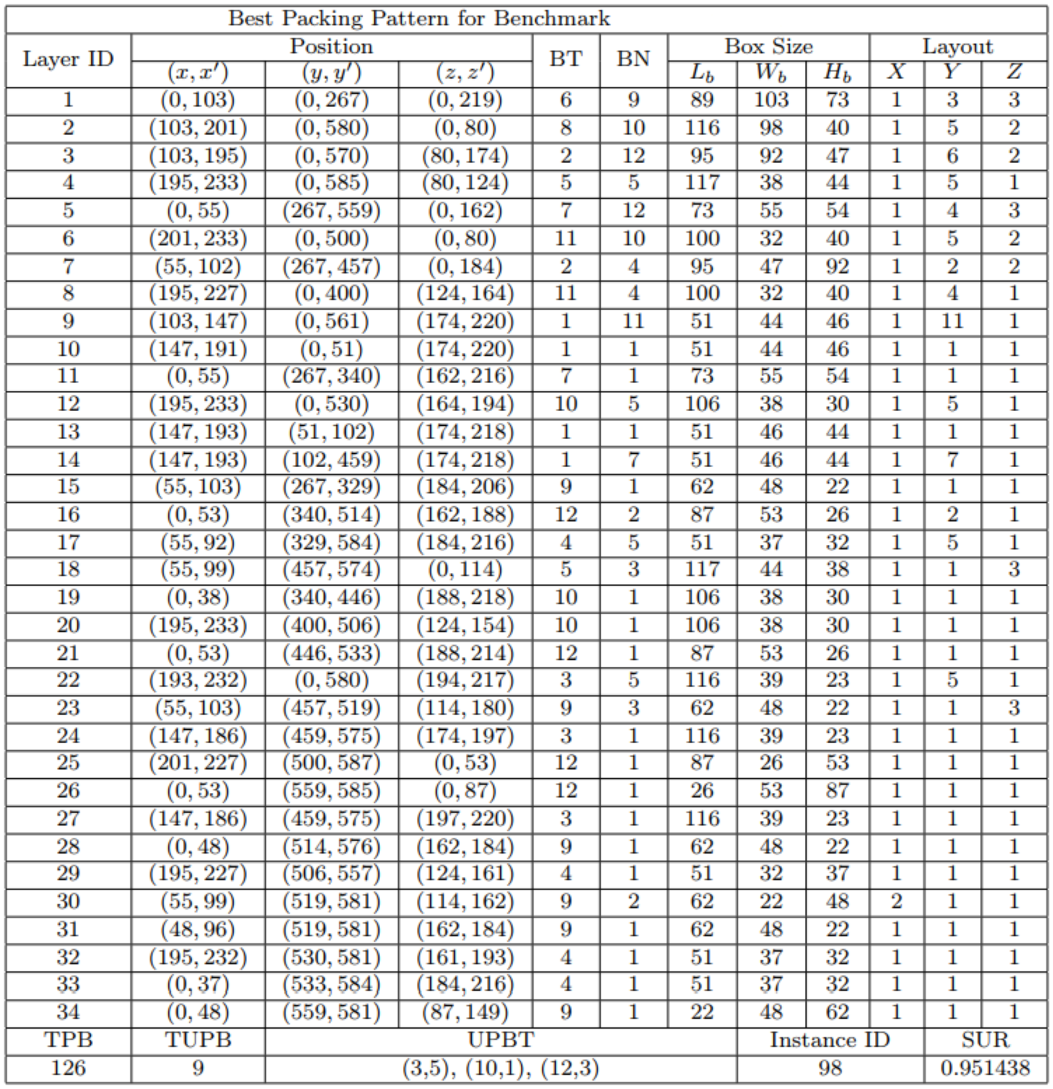


Figure B.3: Optimal placement of items in the third bin

Appendix C

Evolutionary Algorithm – 3D Practical Packing Solution and Pattern

The best results produced in the experiments section for each benchmarks are organized within a table, and the final packing pattern is demonstrated by a 3D figure. There are two figures for each table. The left one shows the result in current table, and the right one is the worst case in the same benchmark, which is used to do the comparison with the best one.



Best Packing Pattern for Benchmark											
Layer ID	Position			BT	BN	Box Size			Layout		
	(x, x')	(y, y')	(z, z')			L_b	W_b	H_b	X	Y	Z
1	$(0, 103)$	$(0, 267)$	$(0, 219)$	6	9	89	103	73	1	3	3
2	$(103, 201)$	$(0, 580)$	$(0, 80)$	8	10	116	98	40	1	5	2
3	$(103, 195)$	$(0, 570)$	$(80, 174)$	2	12	95	92	47	1	6	2
4	$(195, 233)$	$(0, 585)$	$(80, 124)$	5	5	117	38	44	1	5	1
5	$(0, 55)$	$(267, 559)$	$(0, 162)$	7	12	73	55	54	1	4	3
6	$(201, 233)$	$(0, 500)$	$(0, 80)$	11	10	100	32	40	1	5	2
7	$(55, 102)$	$(267, 457)$	$(0, 184)$	2	4	95	47	92	1	2	2
8	$(195, 227)$	$(0, 400)$	$(124, 164)$	11	4	100	32	40	1	4	1
9	$(103, 147)$	$(0, 561)$	$(174, 220)$	1	11	51	44	46	1	11	1
10	$(147, 191)$	$(0, 51)$	$(174, 220)$	1	1	51	44	46	1	1	1
11	$(0, 55)$	$(267, 340)$	$(162, 216)$	7	1	73	55	54	1	1	1
12	$(195, 233)$	$(0, 530)$	$(164, 194)$	10	5	106	38	30	1	5	1
13	$(147, 193)$	$(51, 102)$	$(174, 218)$	1	1	51	46	44	1	1	1
14	$(147, 193)$	$(102, 459)$	$(174, 218)$	1	7	51	46	44	1	7	1
15	$(55, 103)$	$(267, 329)$	$(184, 206)$	9	1	62	48	22	1	1	1
16	$(0, 53)$	$(340, 514)$	$(162, 188)$	12	2	87	53	26	1	2	1
17	$(55, 92)$	$(329, 584)$	$(184, 216)$	4	5	51	37	32	1	5	1
18	$(55, 99)$	$(457, 574)$	$(0, 114)$	5	3	117	44	38	1	1	3
19	$(0, 38)$	$(340, 446)$	$(188, 218)$	10	1	106	38	30	1	1	1
20	$(195, 233)$	$(400, 506)$	$(124, 154)$	10	1	106	38	30	1	1	1
21	$(0, 53)$	$(446, 533)$	$(188, 214)$	12	1	87	53	26	1	1	1
22	$(193, 232)$	$(0, 580)$	$(194, 217)$	3	5	116	39	23	1	5	1
23	$(55, 103)$	$(457, 519)$	$(114, 180)$	9	3	62	48	22	1	1	3
24	$(147, 186)$	$(459, 575)$	$(174, 197)$	3	1	116	39	23	1	1	1
25	$(201, 227)$	$(500, 587)$	$(0, 53)$	12	1	87	26	53	1	1	1
26	$(0, 53)$	$(559, 585)$	$(0, 87)$	12	1	26	53	87	1	1	1
27	$(147, 186)$	$(459, 575)$	$(197, 220)$	3	1	116	39	23	1	1	1
28	$(0, 48)$	$(514, 576)$	$(162, 184)$	9	1	62	48	22	1	1	1
29	$(195, 227)$	$(506, 557)$	$(124, 161)$	4	1	51	32	37	1	1	1
30	$(55, 99)$	$(519, 581)$	$(114, 162)$	9	2	62	22	48	2	1	1
31	$(48, 96)$	$(519, 581)$	$(162, 184)$	9	1	62	48	22	1	1	1
32	$(195, 232)$	$(530, 581)$	$(161, 193)$	4	1	51	37	32	1	1	1
33	$(0, 37)$	$(533, 584)$	$(184, 216)$	4	1	51	37	32	1	1	1
34	$(0, 48)$	$(559, 581)$	$(87, 149)$	9	1	22	48	62	1	1	1
TPB	TUPB	UPBT					Instance ID		SUR		
126	9	(3,5), (10,1), (12,3)					98		0.951438		

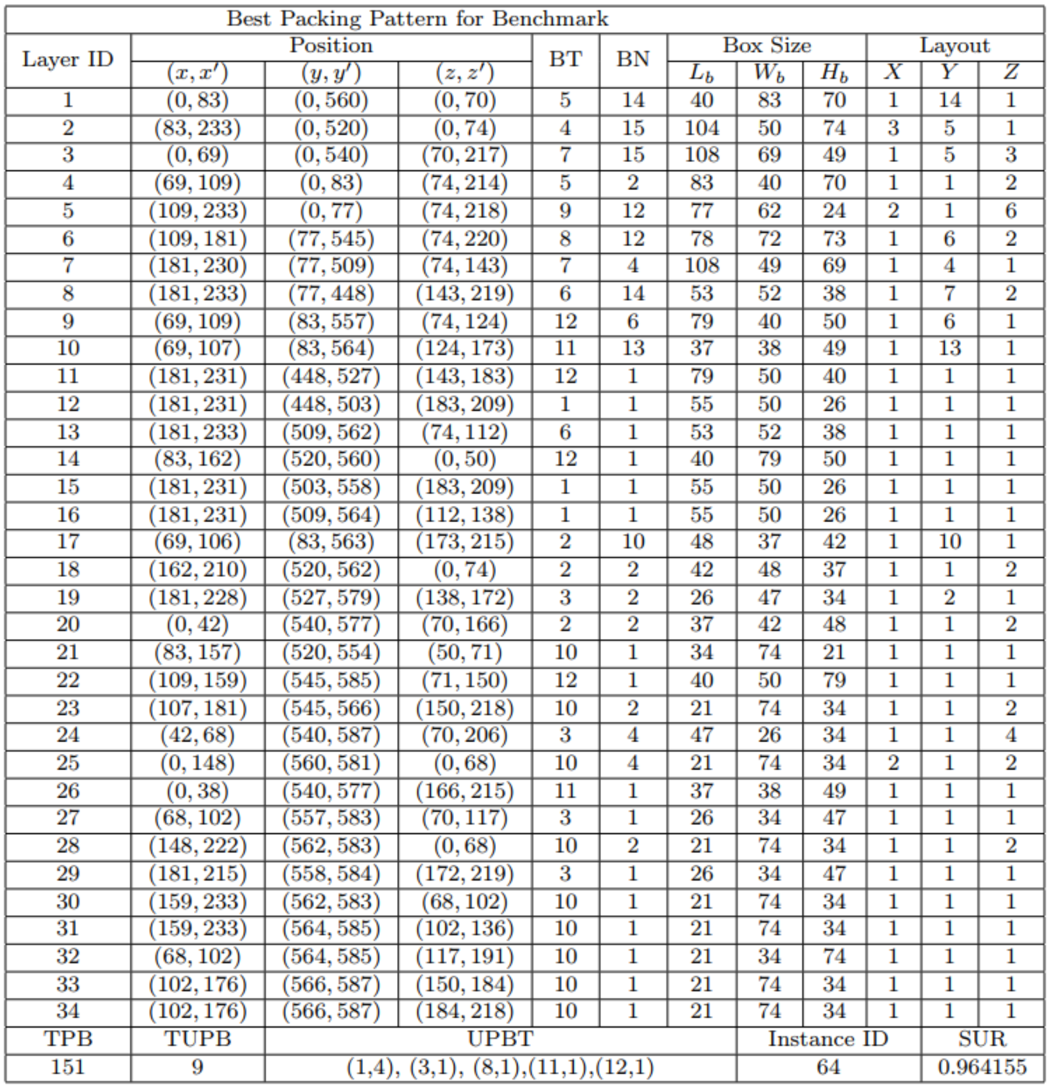
Figure C.1: The Best Result in Experiment 6.3

- BT: Box type ID.
- BN: Packed box number.
- TPB: Total number of Packed box.
- TUPB: Total number of unpacked box.
- UPBT: Unpacked box type and corresponding number.
- SUR: Space Utility Rate.

Best Packing Pattern for Benchmark											
Layer ID	Position			BT	BN	Box Size			Layout		
	(x, x')	(y, y')	(z, z')			L_b	W_b	H_b	X	Y	Z
1	$(0, 182)$	$(0, 246)$	$(0, 104)$	4	12	41	91	104	2	6	1
2	$(0, 210)$	$(0, 567)$	$(104, 178)$	2	14	81	105	74	2	7	1
3	$(182, 232)$	$(0, 532)$	$(0, 73)$	3	7	76	50	73	1	7	1
4	$(0, 72)$	$(0, 456)$	$(178, 219)$	11	12	38	72	41	1	12	1
5	$(72, 226)$	$(0, 456)$	$(178, 220)$	1	8	114	77	42	2	4	1
6	$(182, 232)$	$(0, 576)$	$(73, 102)$	10	8	72	50	29	1	8	1
7	$(0, 178)$	$(246, 421)$	$(0, 54)$	7	14	25	89	54	2	7	1
8	$(0, 73)$	$(246, 550)$	$(54, 104)$	3	4	76	73	50	1	4	1
9	$(73, 146)$	$(246, 474)$	$(54, 104)$	3	3	76	73	50	1	3	1
10	$(0, 89)$	$(421, 446)$	$(0, 54)$	7	1	25	89	54	1	1	1
11	$(210, 230)$	$(0, 93)$	$(102, 169)$	6	1	93	20	67	1	1	1
12	$(210, 233)$	$(93, 543)$	$(102, 172)$	5	6	75	23	70	1	6	1
13	$(146, 178)$	$(246, 582)$	$(54, 102)$	12	8	84	32	24	1	4	2
14	$(89, 156)$	$(421, 514)$	$(0, 40)$	6	2	93	67	20	1	1	2
15	$(0, 91)$	$(456, 560)$	$(178, 219)$	4	1	104	91	41	1	1	1
16	$(0, 89)$	$(446, 586)$	$(0, 46)$	8	5	28	89	46	1	5	1
17	$(156, 180)$	$(421, 505)$	$(0, 32)$	12	1	84	24	32	1	1	1
18	$(91, 225)$	$(456, 549)$	$(178, 198)$	6	2	93	67	20	2	1	1
19	$(73, 143)$	$(474, 549)$	$(46, 92)$	5	2	75	70	23	1	1	2
20	$(89, 164)$	$(514, 584)$	$(0, 46)$	5	2	70	75	23	1	1	2
21	$(91, 158)$	$(456, 549)$	$(198, 218)$	6	1	93	67	20	1	1	1
22	$(158, 225)$	$(456, 549)$	$(198, 218)$	6	1	93	67	20	1	1	1
23	$(178, 228)$	$(532, 561)$	$(0, 72)$	10	1	29	50	72	1	1	1
24	$(0, 93)$	$(567, 587)$	$(46, 180)$	6	2	20	93	67	1	1	2
25	$(93, 227)$	$(567, 587)$	$(102, 195)$	6	2	20	67	93	2	1	1
26	$(93, 129)$	$(549, 584)$	$(46, 94)$	9	2	35	36	24	1	1	2
27	$(91, 231)$	$(549, 585)$	$(195, 219)$	9	4	36	35	24	4	1	1
28	$(0, 84)$	$(560, 584)$	$(180, 212)$	12	1	24	84	32	1	1	1
29	$(164, 200)$	$(561, 585)$	$(0, 35)$	9	1	24	36	35	1	1	1
30	$(178, 213)$	$(561, 585)$	$(35, 71)$	9	1	24	35	36	1	1	1
TPB	TUPB	UPBT					Instance ID			SUR	
129	9	(5,4), (8,4), (10,1)					83			0.957639	

Figure C.2: The Best Result in Experiment 6.4

- BT: Box type ID.
- BN: Packed box number.
- TPB: Total number of Packed box.
- TUPB: Total number of unpacked box.
- UPBT: Unpacked box type and corresponding number.
- SUR: Space Utility Rate.



Best Packing Pattern for Benchmark												
Layer ID	Position			BT	BN	Box Size			Layout		
	(x, x')	(y, y')	(z, z')			L_b	W_b	H_b	X	Y	Z
1	$(0, 83)$	$(0, 560)$	$(0, 70)$	5	14	40	83	70	1	14	1
2	$(83, 233)$	$(0, 520)$	$(0, 74)$	4	15	104	50	74	3	5	1
3	$(0, 69)$	$(0, 540)$	$(70, 217)$	7	15	108	69	49	1	5	3
4	$(69, 109)$	$(0, 83)$	$(74, 214)$	5	2	83	40	70	1	1	2
5	$(109, 233)$	$(0, 77)$	$(74, 218)$	9	12	77	62	24	2	1	6
6	$(109, 181)$	$(77, 545)$	$(74, 220)$	8	12	78	72	73	1	6	2
7	$(181, 230)$	$(77, 509)$	$(74, 143)$	7	4	108	49	69	1	4	1
8	$(181, 233)$	$(77, 448)$	$(143, 219)$	6	14	53	52	38	1	7	2
9	$(69, 109)$	$(83, 557)$	$(74, 124)$	12	6	79	40	50	1	6	1
10	$(69, 107)$	$(83, 564)$	$(124, 173)$	11	13	37	38	49	1	13	1
11	$(181, 231)$	$(448, 527)$	$(143, 183)$	12	1	79	50	40	1	1	1
12	$(181, 231)$	$(448, 503)$	$(183, 209)$	1	1	55	50	26	1	1	1
13	$(181, 233)$	$(509, 562)$	$(74, 112)$	6	1	53	52	38	1	1	1
14	$(83, 162)$	$(520, 560)$	$(0, 50)$	12	1	40	79	50	1	1	1
15	$(181, 231)$	$(503, 558)$	$(183, 209)$	1	1	55	50	26	1	1	1
16	$(181, 231)$	$(509, 564)$	$(112, 138)$	1	1	55	50	26	1	1	1
17	$(69, 106)$	$(83, 563)$	$(173, 215)$	2	10	48	37	42	1	10	1
18	$(162, 210)$	$(520, 562)$	$(0, 74)$	2	2	42	48	37	1	1	2
19	$(181, 228)$	$(527, 579)$	$(138, 172)$	3	2	26	47	34	1	2	1
20	$(0, 42)$	$(540, 577)$	$(70, 166)$	2	2	37	42	48	1	1	2
21	$(83, 157)$	$(520, 554)$	$(50, 71)$	10	1	34	74	21	1	1	1
22	$(109, 159)$	$(545, 585)$	$(71, 150)$	12	1	40	50	79	1	1	1
23	$(107, 181)$	$(545, 566)$	$(150, 218)$	10	2	21	74	34	1	1	2
24	$(42, 68)$	$(540, 587)$	$(70, 206)$	3	4	47	26	34	1	1	4
25	$(0, 148)$	$(560, 581)$	$(0, 68)$	10	4	21	74	34	2	1	2
26	$(0, 38)$	$(540, 577)$	$(166, 215)$	11	1	37	38	49	1	1	1
27	$(68, 102)$	$(557, 583)$	$(70, 117)$	3	1	26	34	47	1	1	1
28	$(148, 222)$	$(562, 583)$	$(0, 68)$	10	2	21	74	34	1	1	2
29	$(181, 215)$	$(558, 584)$	$(172, 219)$	3	1	26	34	47	1	1	1
30	$(159, 233)$	$(562, 583)$	$(68, 102)$	10	1	21	74	34	1	1	1
31	$(159, 233)$	$(564, 585)$	$(102, 136)$	10	1	21	74	34	1	1	1
32	$(68, 102)$	$(564, 585)$	$(117, 191)$	10	1	21	34	74	1	1	1
33	$(102, 176)$	$(566, 587)$	$(150, 184)$	10	1	21	74	34	1	1	1
34	$(102, 176)$	$(566, 587)$	$(184, 218)$	10	1	21	74	34	1	1	1
TPB	TUPB	UPBT					Instance ID			SUR	
151	9	(1,4), (3,1), (8,1),(11,1),(12,1)					64			0.964155	

Figure C.3: The Best Result in Experiment 6.5

- BT: Box type ID.
- BN: Packed box number.
- TPB: Total number of Packed box.
- TUPB: Total number of unpacked box.
- UPBT: Unpacked box type and corresponding number.
- SUR: Space Utility Rate.

Best Packing Pattern for Benchmark 1 in Overall Experiment											
Layer ID	Position			BT	BN	Box Size			Layout		
	(x, x')	(y, y')	(z, z')			L_b	W_b	H_b	X	Y	Z
1	$(0, 231)$	$(0, 378)$	$(0, 37)$	2	66	63	21	37	11	6	1
2	$(0, 208)$	$(0, 25)$	$(37, 217)$	1	20	25	52	36	4	1	5
3	$(0, 100)$	$(25, 565)$	$(37, 220)$	3	45	36	100	61	1	15	3
4	$(100, 136)$	$(25, 525)$	$(37, 220)$	3	15	100	36	61	1	5	3
5	$(136, 172)$	$(25, 525)$	$(37, 220)$	3	15	100	36	61	1	5	3
6	$(172, 208)$	$(25, 525)$	$(37, 220)$	3	15	100	36	61	1	5	3
7	$(208, 233)$	$(0, 572)$	$(37, 73)$	1	11	52	25	36	1	11	1
8	$(0, 61)$	$(378, 578)$	$(0, 36)$	3	2	100	61	36	1	2	1
9	$(208, 233)$	$(0, 572)$	$(73, 217)$	1	44	52	25	36	1	11	4
10	$(61, 211)$	$(378, 586)$	$(0, 36)$	1	24	52	25	36	6	4	1
11	$(100, 204)$	$(525, 550)$	$(36, 216)$	1	10	25	52	36	2	1	5
12	$(100, 200)$	$(550, 586)$	$(36, 219)$	3	3	36	100	61	1	1	3
13	$(211, 232)$	$(378, 567)$	$(0, 37)$	2	3	63	21	37	1	3	1
14	$(0, 63)$	$(565, 586)$	$(36, 73)$	2	1	21	63	37	1	1	1
15	$(63, 100)$	$(565, 586)$	$(36, 99)$	2	1	21	37	63	1	1	1
16	$(0, 37)$	$(565, 586)$	$(73, 136)$	2	1	21	37	63	1	1	1
17	$(37, 100)$	$(565, 586)$	$(99, 210)$	2	3	21	63	37	1	1	3
18	$(0, 37)$	$(565, 586)$	$(136, 199)$	2	1	21	37	63	1	1	1
TPB	TUPB	UPBT					Instance ID			SUR	
280	4	(1,3), (3,1)					12			0.986502	

Figure C.4: The Best Result in Benchmark One

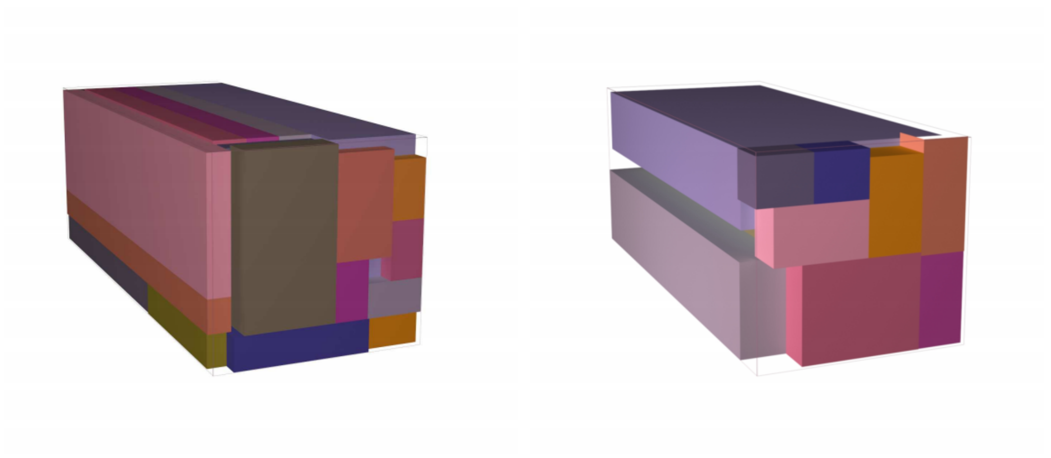


Figure C.5: The Benchmark One

Best Packing Pattern for Benchmark 2 in Overall Experiment											
Layer ID	Position			BT	BN	Box Size			Layout		
	(x, x')	(y, y')	(z, z')			L_b	W_b	H_b	X	Y	Z
1	$(0, 72)$	$(0, 304)$	$(0, 220)$	4	40	76	72	22	1	4	10
2	$(72, 128)$	$(0, 366)$	$(0, 216)$	3	36	61	56	36	1	6	6
3	$(128, 200)$	$(0, 31)$	$(0, 220)$	2	4	31	72	55	1	1	4
4	$(128, 202)$	$(31, 580)$	$(0, 198)$	5	27	61	74	66	1	9	3
5	$(0, 66)$	$(304, 526)$	$(0, 183)$	5	9	74	66	61	1	3	3
6	$(202, 233)$	$(0, 72)$	$(0, 220)$	2	4	72	31	55	1	1	4
7	$(202, 233)$	$(72, 576)$	$(0, 55)$	2	7	72	31	55	1	7	1
8	$(202, 233)$	$(72, 576)$	$(55, 110)$	2	7	72	31	55	1	7	1
9	$(202, 233)$	$(72, 576)$	$(110, 220)$	2	14	72	31	55	1	7	2
10	$(128, 202)$	$(31, 581)$	$(198, 220)$	1	20	55	37	22	2	10	1
11	$(0, 66)$	$(304, 579)$	$(183, 220)$	1	15	55	22	37	3	5	1
12	$(66, 128)$	$(366, 438)$	$(0, 220)$	2	8	72	31	55	2	1	4
13	$(66, 127)$	$(438, 586)$	$(0, 198)$	5	6	74	61	66	1	2	3
14	$(0, 66)$	$(526, 587)$	$(0, 148)$	5	2	61	66	74	1	1	2
15	$(66, 121)$	$(438, 586)$	$(198, 220)$	1	4	37	55	22	1	4	1
16	$(0, 55)$	$(526, 563)$	$(148, 170)$	1	1	37	55	22	1	1	1
TPB	TUPB	UPBT					Instance ID			SUR	
204	5	(1,2), (2,1), (3,1), (4,1)					84			0.981847	

Figure C.6: The Best Result in Benchmark Two

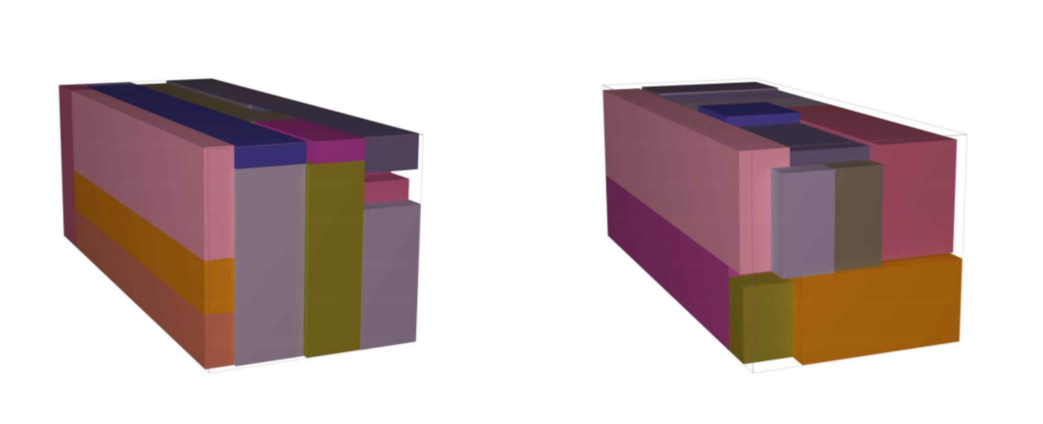


Figure C.7: The Benchmark Two

Best Packing Pattern for Benchmark 3 in Overall Experiment											
Layer ID	Position			BT	BN	Box Size			Layout		
	(x, x')	(y, y')	(z, z')			L_b	W_b	H_b	X	Y	Z
1	$(0, 195)$	$(0, 33)$	$(0, 219)$	4	15	33	39	73	5	1	3
2	$(0, 54)$	$(33, 529)$	$(0, 177)$	7	24	62	54	59	1	8	3
3	$(54, 143)$	$(33, 537)$	$(0, 180)$	1	36	56	89	45	1	9	4
4	$(143, 196)$	$(33, 523)$	$(0, 220)$	8	25	98	53	44	1	5	5
5	$(143, 196)$	$(523, 567)$	$(0, 98)$	8	1	44	53	98	1	1	1
6	$(143, 197)$	$(523, 585)$	$(98, 157)$	7	1	62	54	59	1	1	1
7	$(196, 233)$	$(0, 583)$	$(0, 34)$	2	11	53	37	34	1	11	1
8	$(196, 233)$	$(0, 578)$	$(34, 87)$	2	17	34	37	53	1	17	1
9	$(197, 232)$	$(0, 575)$	$(87, 141)$	3	23	25	35	54	1	23	1
10	$(197, 230)$	$(0, 584)$	$(141, 219)$	4	16	73	33	39	1	8	2
11	$(0, 29)$	$(33, 583)$	$(177, 219)$	6	11	50	29	42	1	11	1
12	$(29, 105)$	$(33, 544)$	$(180, 219)$	5	14	73	38	39	2	7	1
13	$(105, 143)$	$(33, 471)$	$(180, 219)$	5	6	73	38	39	1	6	1
14	$(105, 138)$	$(471, 544)$	$(180, 219)$	4	1	73	33	39	1	1	1
15	$(143, 197)$	$(523, 548)$	$(157, 192)$	3	1	25	54	35	1	1	1
16	$(0, 50)$	$(529, 587)$	$(0, 168)$	6	8	29	50	42	1	2	4
17	$(50, 137)$	$(537, 587)$	$(0, 168)$	6	12	50	29	42	3	1	4
18	$(138, 192)$	$(523, 558)$	$(192, 217)$	3	1	35	54	25	1	1	1
19	$(29, 137)$	$(544, 579)$	$(168, 218)$	3	4	35	54	25	2	1	2
20	$(137, 191)$	$(548, 583)$	$(157, 182)$	3	1	35	54	25	1	1	1
21	$(137, 191)$	$(558, 583)$	$(182, 217)$	3	1	25	54	35	1	1	1
TPB	TUPB	UPBT				Instance ID				SUR	
229	3	(4,1),(8,2)				38				0.97522	

Figure C.8: The Best Result in Benchmark Three

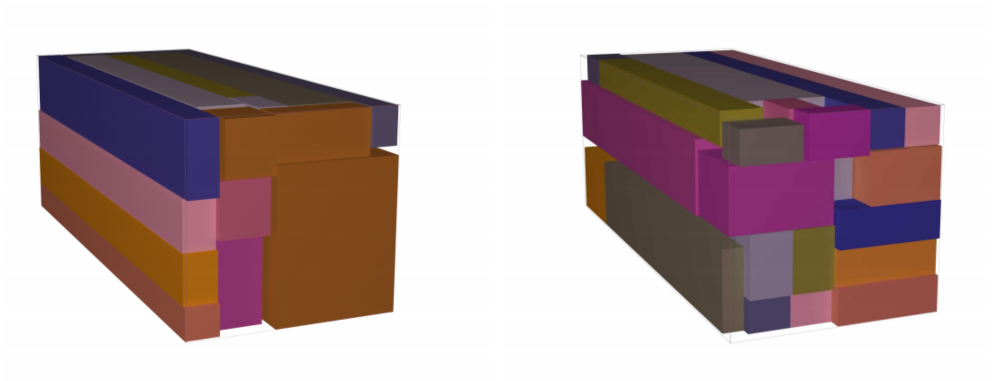


Figure C.9: The Benchmark Three

Best Packing Pattern for Benchmark 4 in Overall Experiment											
Layer ID	Position			BT	BN	Box Size			Layout		
	(x, x')	(y, y')	(z, z')			L_b	W_b	H_b	X	Y	Z
1	$(0, 192)$	$(0, 79)$	$(0, 220)$	10	24	79	32	55	6	1	4
2	$(0, 68)$	$(79, 503)$	$(0, 189)$	5	24	53	68	63	1	8	3
3	$(68, 158)$	$(79, 361)$	$(0, 220)$	4	15	94	90	44	1	3	5
4	$(192, 223)$	$(0, 42)$	$(0, 217)$	3	7	42	31	31	1	1	7
5	$(192, 233)$	$(42, 586)$	$(0, 180)$	1	32	34	41	90	1	16	2
6	$(192, 233)$	$(42, 222)$	$(180, 214)$	1	2	90	41	34	1	2	1
7	$(158, 189)$	$(79, 439)$	$(0, 192)$	7	15	72	31	64	1	5	3
8	$(0, 64)$	$(79, 151)$	$(189, 220)$	7	1	72	64	31	1	1	1
9	$(68, 124)$	$(361, 565)$	$(0, 150)$	8	12	34	56	75	1	6	2
10	$(124, 158)$	$(361, 586)$	$(0, 168)$	8	9	75	34	56	1	3	3
11	$(68, 124)$	$(361, 586)$	$(150, 218)$	8	6	75	56	34	1	3	2
12	$(158, 192)$	$(439, 514)$	$(0, 168)$	8	3	75	34	56	1	1	3
13	$(0, 68)$	$(151, 585)$	$(189, 220)$	9	14	62	34	31	2	7	1
14	$(158, 190)$	$(79, 527)$	$(192, 220)$	2	7	64	32	28	1	7	1
15	$(190, 232)$	$(222, 277)$	$(180, 210)$	6	2	55	21	30	2	1	1
16	$(190, 232)$	$(277, 587)$	$(180, 211)$	3	10	31	42	31	1	10	1
17	$(124, 158)$	$(361, 547)$	$(168, 199)$	9	3	62	34	31	1	3	1
18	$(124, 154)$	$(361, 581)$	$(199, 220)$	6	4	55	30	21	1	4	1
19	$(158, 188)$	$(439, 549)$	$(168, 189)$	6	2	55	30	21	1	2	1
20	$(0, 64)$	$(503, 587)$	$(0, 96)$	2	9	28	64	32	1	3	3
21	$(0, 68)$	$(503, 565)$	$(96, 189)$	9	6	62	34	31	2	1	3
22	$(158, 188)$	$(514, 535)$	$(0, 165)$	6	3	21	30	55	1	1	3
23	$(158, 192)$	$(535, 566)$	$(0, 62)$	9	1	31	34	62	1	1	1
24	$(158, 192)$	$(535, 566)$	$(62, 124)$	9	1	31	34	62	1	1	1
25	$(124, 186)$	$(549, 583)$	$(168, 199)$	9	1	34	62	31	1	1	1
26	$(154, 184)$	$(527, 582)$	$(199, 220)$	6	1	55	30	21	1	1	1
27	$(64, 94)$	$(565, 586)$	$(0, 110)$	6	2	21	30	55	1	1	2
28	$(94, 124)$	$(565, 586)$	$(0, 110)$	6	2	21	30	55	1	1	2
29	$(158, 189)$	$(535, 566)$	$(124, 166)$	3	1	31	31	42	1	1	1
30	$(0, 55)$	$(565, 586)$	$(96, 186)$	6	3	21	55	30	1	1	3
31	$(55, 110)$	$(565, 586)$	$(110, 140)$	6	1	21	55	30	1	1	1
32	$(158, 188)$	$(566, 587)$	$(0, 165)$	6	3	21	30	55	1	1	3
TPB	TUPB	UPBT					Instance ID		SUR		
226	7	(2,1),(3,1),(9,5)					55		0.975254		

Figure C.10: The Best Result in Benchmark Four

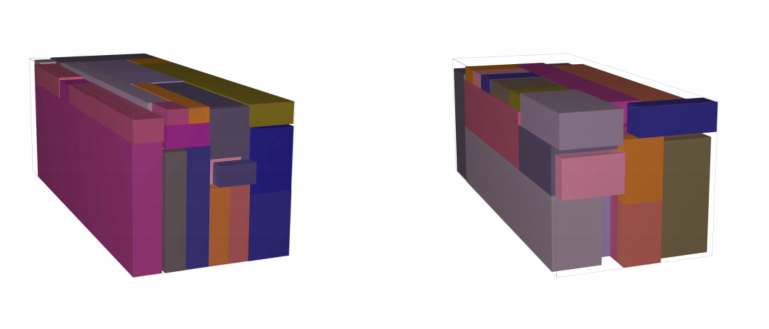


Figure C.11: The Benchmark Four

Best Packing Pattern for Benchmark 5 in Overall Experiment											
Layer ID	Position			BT	BN	Box Size			Layout		
	(x, x')	(y, y')	(z, z')			L_b	W_b	H_b	X	Y	Z
1	$(0, 43)$	$(0, 570)$	$(0, 123)$	12	15	114	43	41	1	5	3
2	$(43, 123)$	$(0, 434)$	$(0, 218)$	10	14	62	80	109	1	7	2
3	$(123, 232)$	$(0, 195)$	$(0, 219)$	8	9	65	109	73	1	3	3
4	$(123, 231)$	$(195, 492)$	$(0, 98)$	7	11	27	108	98	1	11	1
5	$(0, 40)$	$(0, 582)$	$(123, 215)$	9	6	97	40	92	1	6	1
6	$(123, 197)$	$(195, 525)$	$(98, 220)$	5	10	66	74	61	1	5	2
7	$(197, 233)$	$(195, 500)$	$(98, 210)$	3	10	61	36	56	1	5	2
8	$(43, 119)$	$(434, 500)$	$(0, 216)$	4	9	22	76	72	1	3	3
9	$(119, 228)$	$(492, 565)$	$(0, 65)$	8	1	73	109	65	1	1	1
10	$(43, 116)$	$(500, 565)$	$(0, 109)$	8	1	65	73	109	1	1	1
11	$(43, 117)$	$(500, 561)$	$(109, 175)$	5	1	61	74	66	1	1	1
12	$(119, 229)$	$(492, 564)$	$(65, 96)$	2	2	72	55	31	2	1	1
13	$(197, 229)$	$(500, 543)$	$(96, 213)$	6	3	43	32	39	1	1	3
14	$(40, 118)$	$(500, 564)$	$(175, 218)$	6	4	32	39	43	2	2	1
15	$(117, 174)$	$(525, 582)$	$(96, 141)$	11	1	57	57	45	1	1	1
16	$(174, 196)$	$(525, 580)$	$(96, 207)$	1	3	55	22	37	1	1	3
17	$(118, 173)$	$(525, 587)$	$(141, 213)$	2	2	31	55	72	1	2	1
18	$(196, 233)$	$(543, 587)$	$(96, 206)$	1	4	22	37	55	1	2	2
19	$(43, 117)$	$(561, 583)$	$(109, 164)$	1	2	22	37	55	2	1	1
20	$(40, 77)$	$(564, 586)$	$(164, 219)$	1	1	22	37	55	1	1	1
21	$(43, 119)$	$(565, 587)$	$(0, 72)$	4	1	22	76	72	1	1	1
22	$(77, 114)$	$(564, 586)$	$(164, 219)$	1	1	22	37	55	1	1	1
23	$(119, 230)$	$(565, 587)$	$(0, 55)$	1	3	22	37	55	3	1	1
24	$(119, 174)$	$(565, 587)$	$(55, 92)$	1	1	22	55	37	1	1	1
25	$(174, 229)$	$(565, 587)$	$(55, 92)$	1	1	22	55	37	1	1	1
26	$(43, 98)$	$(565, 587)$	$(72, 109)$	1	1	22	55	37	1	1	1
TPB	TUPB	UPBT					Instance ID			SUR	
117	9	(2,3),(6,3),(11,3)					84			0.965213	

Figure C.12: The Best Result in Benchmark Five

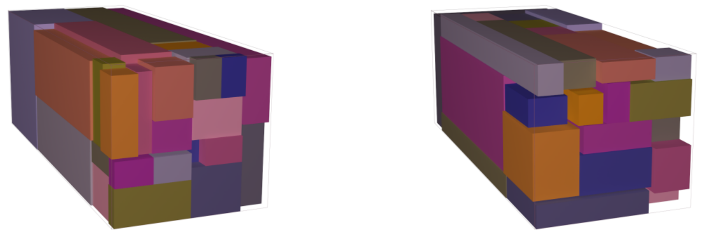


Figure C.13: The Benchmark Five

Best Packing Pattern for Benchmark 6 in Overall Experiment											
Layer ID	Position			BT	BN	Box Size			Layout		
	(x, x')	(y, y')	(z, z')			L_b	W_b	H_b	X	Y	Z
1	$(0, 101)$	$(0, 350)$	$(0, 180)$	15	14	50	101	90	1	7	2
2	$(101, 209)$	$(0, 444)$	$(0, 47)$	7	12	37	108	47	1	12	1
3	$(101, 209)$	$(0, 486)$	$(47, 113)$	2	9	54	108	66	1	9	1
4	$(101, 170)$	$(0, 480)$	$(113, 200)$	14	12	120	69	29	1	4	3
5	$(170, 232)$	$(0, 363)$	$(113, 187)$	3	11	33	62	74	1	11	1
6	$(0, 99)$	$(0, 553)$	$(180, 219)$	9	7	79	99	39	1	7	1
7	$(0, 99)$	$(350, 429)$	$(0, 117)$	9	3	79	99	39	1	1	3
8	$(0, 64)$	$(350, 563)$	$(117, 179)$	5	6	71	64	31	1	3	2
9	$(170, 231)$	$(0, 69)$	$(187, 220)$	4	1	69	61	33	1	1	1
10	$(170, 231)$	$(69, 552)$	$(187, 220)$	4	7	69	61	33	1	7	1
11	$(64, 97)$	$(350, 557)$	$(117, 178)$	4	3	69	33	61	1	3	1
12	$(170, 211)$	$(363, 563)$	$(113, 185)$	8	4	100	41	36	1	2	2
13	$(209, 231)$	$(0, 83)$	$(0, 110)$	10	2	83	22	55	1	1	2
14	$(209, 231)$	$(83, 166)$	$(0, 112)$	12	2	83	22	56	1	1	2
15	$(0, 66)$	$(429, 585)$	$(0, 44)$	13	3	52	66	44	1	3	1
16	$(0, 99)$	$(429, 478)$	$(44, 116)$	11	2	49	99	36	1	1	2
17	$(66, 210)$	$(444, 544)$	$(0, 41)$	8	4	100	36	41	4	1	1
18	$(210, 232)$	$(166, 498)$	$(0, 112)$	12	8	83	22	56	1	4	2
19	$(0, 99)$	$(478, 527)$	$(44, 116)$	11	2	49	99	36	1	1	2
20	$(99, 168)$	$(0, 505)$	$(200, 220)$	1	5	101	69	20	1	5	1
21	$(211, 231)$	$(363, 565)$	$(112, 181)$	1	2	101	20	69	1	2	1
22	$(99, 168)$	$(480, 581)$	$(113, 153)$	1	2	101	69	20	1	1	2
23	$(99, 160)$	$(480, 549)$	$(153, 186)$	4	1	69	61	33	1	1	1
24	$(99, 207)$	$(486, 585)$	$(41, 90)$	11	3	99	36	49	3	1	1
25	$(66, 166)$	$(544, 580)$	$(0, 41)$	8	1	36	100	41	1	1	1
26	$(99, 163)$	$(505, 576)$	$(186, 217)$	5	1	71	64	31	1	1	1
27	$(66, 98)$	$(527, 557)$	$(41, 117)$	6	2	30	32	38	1	1	2
28	$(99, 209)$	$(486, 569)$	$(90, 112)$	10	2	83	55	22	2	1	1
29	$(210, 232)$	$(498, 581)$	$(0, 56)$	12	1	83	22	56	1	1	1
30	$(0, 64)$	$(527, 558)$	$(44, 115)$	5	1	31	64	71	1	1	1
31	$(209, 231)$	$(498, 581)$	$(56, 111)$	10	1	83	22	55	1	1	1
32	$(0, 83)$	$(558, 580)$	$(44, 99)$	10	1	22	83	55	1	1	1
33	$(0, 55)$	$(563, 585)$	$(99, 182)$	10	1	22	55	83	1	1	1
34	$(168, 223)$	$(565, 587)$	$(112, 195)$	10	1	22	55	83	1	1	1
35	$(166, 198)$	$(544, 574)$	$(0, 38)$	6	1	30	32	38	1	1	1
36	$(99, 163)$	$(549, 587)$	$(153, 183)$	6	2	38	32	30	2	1	1
37	$(64, 96)$	$(553, 583)$	$(178, 216)$	6	1	30	32	38	1	1	1
38	$(0, 32)$	$(553, 583)$	$(182, 220)$	6	1	30	32	38	1	1	1
39	$(32, 64)$	$(553, 583)$	$(182, 220)$	6	1	30	32	38	1	1	1
40	$(64, 96)$	$(557, 587)$	$(99, 175)$	6	2	30	32	38	1	1	2
TPB	TUPB	UPBT					Instance ID			SUR	
145	10	(6,4),(11,1),(11,2),(13,3)					94			0.964554	

Figure C.14: The Best Result in Benchmark Six

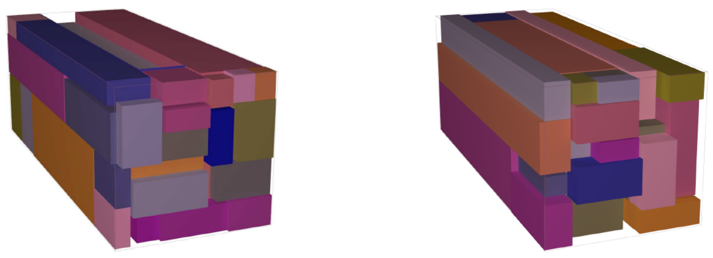


Figure C.15: The Benchmark Six

Best Packing Pattern for Benchmark 7 in Overall Experiment												
Layer ID	Position			BT	BN	Box Size			Layout		
	(x, x')	(y, y')	(z, z')			L_b	W_b	H_b	X	Y	Z
1	(0, 200)	(0, 116)	(0, 98)	8	5	116	40	98	5	1	1
2	(0, 103)	(0, 511)	(98, 187)	6	7	73	103	89	1	7	1
3	(103, 198)	(0, 564)	(98, 190)	2	12	47	95	92	1	12	1
4	(0, 80)	(116, 491)	(0, 75)	14	5	75	80	75	1	5	1
5	(200, 232)	(0, 500)	(0, 40)	11	5	100	32	40	1	5	1
6	(200, 232)	(0, 555)	(40, 128)	15	5	111	32	88	1	5	1
7	(198, 230)	(0, 111)	(128, 216)	15	1	111	32	88	1	1	1
8	(0, 38)	(0, 530)	(187, 217)	10	5	106	38	30	1	5	1
9	(80, 197)	(116, 420)	(0, 44)	5	8	38	117	44	1	8	1
10	(80, 153)	(116, 556)	(44, 98)	7	8	55	73	54	1	8	1
11	(153, 199)	(116, 512)	(44, 95)	1	9	44	46	51	1	9	1
12	(38, 76)	(0, 424)	(187, 217)	10	4	106	38	30	1	4	1
13	(76, 166)	(0, 564)	(190, 220)	20	6	94	90	30	1	6	1
14	(198, 228)	(111, 205)	(128, 218)	20	1	94	30	90	1	1	1
15	(166, 198)	(0, 560)	(190, 220)	19	5	112	32	30	1	5	1
16	(198, 230)	(205, 317)	(128, 218)	19	3	112	32	30	1	1	3
17	(198, 233)	(317, 562)	(128, 185)	13	7	35	35	57	1	7	1
18	(80, 196)	(420, 529)	(0, 42)	16	4	109	29	42	4	1	1
19	(0, 78)	(116, 580)	(75, 98)	3	8	116	39	23	2	4	1
20	(0, 78)	(491, 578)	(0, 53)	12	3	87	26	53	3	1	1
21	(0, 100)	(511, 551)	(98, 130)	11	1	40	100	32	1	1	1
22	(0, 100)	(511, 551)	(130, 162)	11	1	40	100	32	1	1	1
23	(38, 75)	(424, 577)	(187, 219)	4	3	51	37	32	1	3	1
24	(78, 194)	(529, 552)	(0, 39)	3	1	23	116	39	1	1	1
25	(0, 87)	(551, 577)	(98, 151)	12	1	26	87	53	1	1	1
26	(78, 178)	(552, 584)	(0, 40)	11	1	32	100	40	1	1	1
27	(196, 233)	(500, 551)	(0, 32)	4	1	51	37	32	1	1	1
28	(153, 190)	(512, 544)	(42, 93)	4	1	32	37	51	1	1	1
29	(178, 231)	(555, 581)	(0, 87)	12	1	26	53	87	1	1	1
30	(0, 92)	(511, 576)	(162, 185)	18	1	65	92	23	1	1	1
31	(0, 62)	(491, 587)	(53, 75)	9	2	48	62	22	1	2	1
32	(198, 233)	(317, 562)	(185, 214)	17	5	49	35	29	1	5	1
33	(0, 35)	(530, 587)	(185, 220)	13	1	57	35	35	1	1	1
34	(78, 140)	(556, 578)	(40, 88)	9	1	22	62	48	1	1	1
35	(140, 175)	(556, 585)	(40, 89)	17	1	29	35	49	1	1	1
36	(87, 179)	(564, 587)	(89, 154)	18	1	23	92	65	1	1	1
37	(92, 184)	(564, 587)	(154, 219)	18	1	23	92	65	1	1	1
38	(179, 227)	(564, 586)	(87, 149)	9	1	22	48	62	1	1	1
39	(184, 232)	(564, 586)	(149, 211)	9	1	22	48	62	1	1	1
TPB	TUPB	UPBT					Instance ID			SUR	
137	14	(12,3),(13,1),(16,2),(17,1),(18,1)					98			0.962306	

Figure C.16: The Best Result in Benchmark Seven

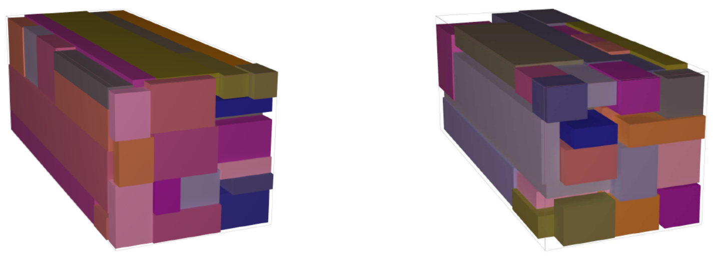


Figure C.17: The Benchmark Seven