universität wien

DIPLOMARBEIT

Titel der Diplomarbeit

"Solving the two-dimensional bin packing problem"

Verfasser

Lukas Baumgartner

angestrebter akademischer Grad

Magister rerum socialium oeconomicarumque
(Mag. rer. soc. oec.)

Wien, im April 2011

Studienkennzahl lt. Studienblatt:	A 157
Studienrichtung lt. Studienblatt:	Internationale Betriebswirtschaftslehre
Betreuer:	O. Univ.-Prof. Dipl.-Ing. Dr. Richard F. Hartl

I want to thank the Chair of Production and Operations Management for giving me the opportunity to write this thesis. Especially Dr. Verena Schmid for supporting and mentoring me. She is such a good mentor that all she needed to explain the two-dimensional bin packing problem to me was a post-it note. Further I want to thank my family for supporting and inspiring me.

Contents

List of Figures

List of Tables

1 Introduction

At first view a newspaper editor and a carpenter do not have a lot in common. One is responsible for publishing a newspaper every day, the other is building a variety of things mostly out of wood. But by doing this they face very similar problems. The editor has to place articles on the different pages of the newspaper, which can be considered a packing problem and the carpenter has to cut a set of pieces out of wooden planks, which can be considered a cutting problem. The cutting and the packing problem are actually the same. The difference is that for a cutting problem one wants to cut a set of small items out of a set of large objects and for the packing problem one wants to pack a set of small items into a set of large objects. This thesis will focus on packing problems.

A packing problem is the problem of placing a set of small items in a set of large objects, so that the small items are completely within the large objects and do not overlap. For this five sub-problems have to be solved:

1. Selecting the small items.
2. Selecting the large objects.
3. Grouping the small items.
4. Assigning the groups of small items to the large items.
5. Placement of the small items within the large items.

To distinguish the different packing problems five criteria are used to categorize packing problems [52]. These categories are dimensionality, kind of assignment, assortment of small items, assortment of large objects and shape of small items.

Dimensionality, as the name already indicates, distinguishes between one-, two- and three-dimensional problems. Even problems with more then three dimensions are discussed in the literature. The one-dimensional bin packing problem is a special case of the two-dimensional bin packing problem. It can be interpreted as two-dimensional bin packing problem where all items have the width of the bin width. This is the same for the two- and three-dimensional bin packing problem. The two-dimensional problem can

be formulated as a three-dimensional one, where all items have the depth/length of the bin.

There are two kinds of assignments, output maximization and input minimization. For output maximization the set of large objects is given and the number of selected small items which are packed have to be maximized. With input minimization the set of small items are given and the large objects have to be selected so that their number is minimized.

Assortment of small items means if the items are more (identical small items) or less (strongly heterogeneous assortment) alike or something in between (weakly heterogeneous assortment), regarding size and shape.

The criteria of assortment of large objects is the same as assortment of small items with the additional case of one large object.

The last criterium, shape of small items, distinguishes between regular and irregular shaped items. Regular shaped items are items such as rectangles, circles, balls, etc.

Bin packing is classified as a problem with input minimization and strongly heterogeneous small items.

Concerning dimensionality the literature mainly distinguishes between one-, two- and three-dimensional problems, where the last two are more often discussed. The one-dimensional case has been solved using a wide variety of approaches such as a weighted annealing heuristic [35], ant colony optimization [11], a hybrid improvement heuristic [2] or a variable neighborhood search [8]. The two-dimensional bin packing problem has been solved with ant colony optimization [22], using an exact approach [15], with a heuristic placement routine [53] and with Tabu Search [31] to name a few. There are four general cases of the two-dimensional bin packing problem depending if the items are oriented or not and if guillotine cutting is required or free. A linear programming approach [28], a hybrid GRASP/VND algorithm [41], a two-level tabu search [18] and an extreme point-based heuristic [17] have been applied to the three-dimensional bin packing problem. A variation of the three-dimensional bin packing problem is when the supporting areas of the item(s) on which another item is packed have to be considered [6]

Further the classic bin packing problem contains identical large objects but heterogeneous assortments (variable sized bin packing problems) have also been discussed [16] [27] [42].

The most common small item shape would be rectangles for the two-dimensional case and boxes for the three-dimensional case but other cases (irregular shaped items) have also been discussed [10] [37] [49]. Also fragile items have been considered in [4]. A good overview of cutting and packing problems can be found in [48].

There is a wide variety of real world applications for the bin packing problem. The most common application of heuristics for bin packing is found in the cutting and transportation industry. But bin packing algorithms has also been used for routing and wavelength assignment in optical networks [47], applied to problems related to video-on-demand [54] and used to improve operating room efficiency [29].

The first topic discussed in this thesis will be the two-dimensional bin packing problem in Section 2, including a brand new ILP model for the two-dimensional bin packing problem with orientation and free guillotine cutting (Section 2.3). Section 3 introduces the probabilistic LGFi heuristic, which is an improved version of the heuristic LGFi. This is what this thesis mainly is about and also includes a small example to illustrate how it works. The probabilistic LGFi heuristic was applied using three different methods (Multi-start approach, Beam Search and Variable Neighborhood Search), which are presented in Section 4. The experimental evaluation of these three methods are presented in Section 5. This section contains the introduction of the instances, on which the heuristics have been tested, the results of the extensive parameter testing and the overall results. The heuristic has shown to be very effective and outperforms other heuristics by $1.1\% - 5.7\%$. It also was able to find three new best solutions for the 500 instances it was tested on. Further conclusions can be found in Section 6.

2 Two-dimensional bin packing

In this section an overview of the 2BP is presented. First the problem is defined in Section 2.1 and then related work will be discussed in Section 2.2. In Section 2.3 one finds the new ILP model developed. At last lower bounds for the 2BP are presented in Section 2.4.

2.1 Problem Formulation

The two-dimensional bin packing problem (2BP) consists in packing a set of n rectangular items $i \in \mathcal{Q} = \{1, \ldots, n\}$ into bins of height H and width W. The total number of bins is unlimited. Each item i is characterized by its height h_i and its width w_i. Items have to be packed so that they do not overlap. The goal is to minimize the number of used bins. Many real world applications exist for the 2BP such as, for example, cutting glass, wood or metal and packing in the context of transportation or warehousing.

According to [32] there are four different cases of the 2BP. The differences between these four cases are derived from two aspects: (1) the 90° rotation of items may be allowed, or not, and (2) guillotine cutting may be required or free. Guillotine cutting means that only straight cuts through the whole bin are allowed. So one cannot cut up to a certain point, make a 90° turn and continue cutting. The four problem cases can be characterized as follows:

- 2BP|O|G: The items are oriented and guillotine cutting is required.
- 2BP|O|F: The items are oriented and guillotine cuttings is free.
- 2BP|R|G: The items can be rotated by 90° and guillotine cutting is required.
- 2BP|R|F: The items can be rotated by 90° and guillotine cutting is free.

This thesis exclusively focuses on the 2BP|O|F case, that is, in the remainder of the paper the abbreviation 2BP will refer to this problem version. Concerning the complexity of the 2BP, the problem is classified as NP-hard [24]. For further reading [33] [34], Lodi [30] and [19] provide a good overview over the 2BP by presenting different models, heuristics, exact algorithms, metaheuristics, lower and upper bounds.

2.2 Related Work

Concerning heuristic solution methods one mainly distinguishes between one-phase and two-phase approaches. One-phase algorithms pack the items directly into the bins, whereas two-phase algorithms first pack the items into levels of one infinitely high strip with width W and then stack these levels into the bins.

Level-packing algorithms place items next to each other in each level. Hereby, the bottom of the first level is the bottom of the bin. For the next level the bottom is a horizontal line coinciding with the tallest item of the level below. Items can only be placed besides each other in each level, in contrast to packing items on top of each other.

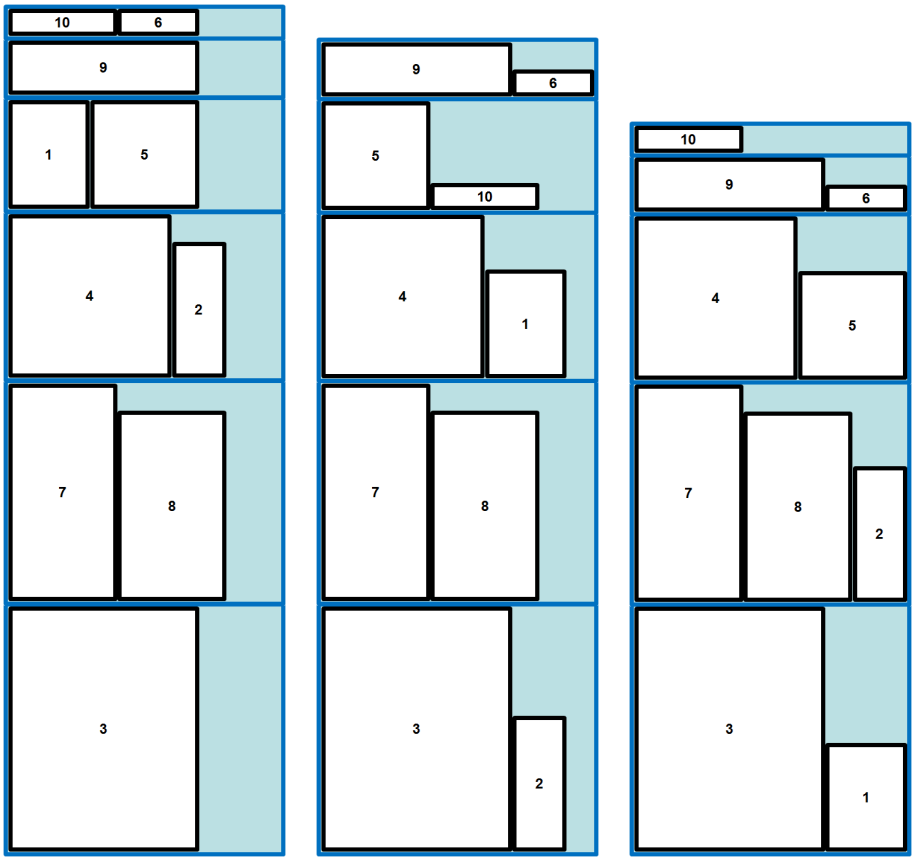
Well known level-packing algorithms are Next-Fit Decreasing Height (NFDH), First-Fit Decreasing Height (FFDH) and Best-Fit Decreasing Height (BFDH). [20] These strategies were originally developed as algorithms for the one-dimensional bin packing problem, but have also been adapted to strip packing problems and as components of heuristics for the two-dimensional bin packing problem, which we will present in the following. For the strip packing problem one has to pack a given set of small items into on large object with a given width and unlimited height, where the aim is to minimized the used height of the large object. For all three heuristics the items must first be sorted by non-increasing height. Then they are packed in this order.

NFDH packs the current item in the leftmost position of the current level, unless it does not fit. In this case, it creates a new level, which becomes the new current level, where the item will be packed in the leftmost position. In contrast, FFDH packs the current item as follows. Starting from the first level (among the currently available levels), FFDH tries to accommodate the current item, which is finally packed into the first level in which it still fits. As in the case of NFDH, the current item is always placed in the leftmost position possible. If no level can accommodate the current item a new level is created. Finally, BFDH works as follows. For the current item, BFDH chooses among the available levels the one where the distance from the right side of the item to the right side of the bin is the smallest. If the current item does not fit in any available level, a new level is created. This can be illustrated in a small example using the items in Table 1. Using NFDH the items would be packed as shown in Figure 1(a), FFDH

would result in Figure 1(b) and BFDH would pack the items as illustrated in Figure 1(c). In general, NFDH is the fastest among these three heuristics, but it produces the worst solutions. The opposite is the case for BFDH, while FFDH is a compromise between these two.

Table 1: Items for level-packing algorithms

i	1	2	3	4	5	6	7	8	9	10
w_i	3	2	7	6	4	3	4	4	7	4
h_i	4	5	9	6	4	1	8	7	2	1

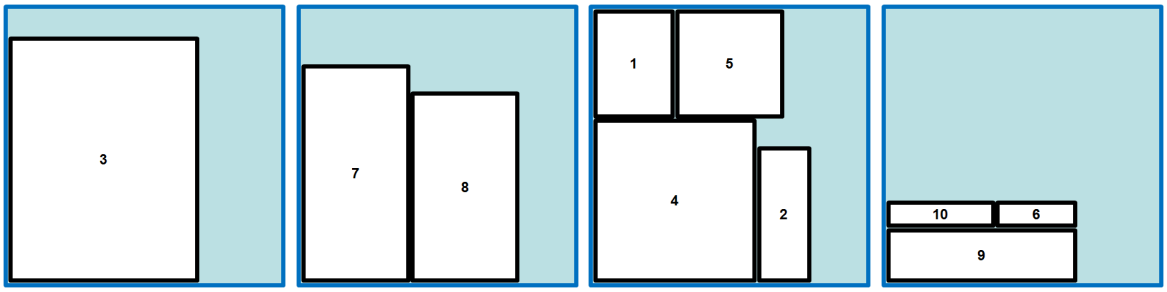


(a) NFDH (b) FFDH (c) BFDH

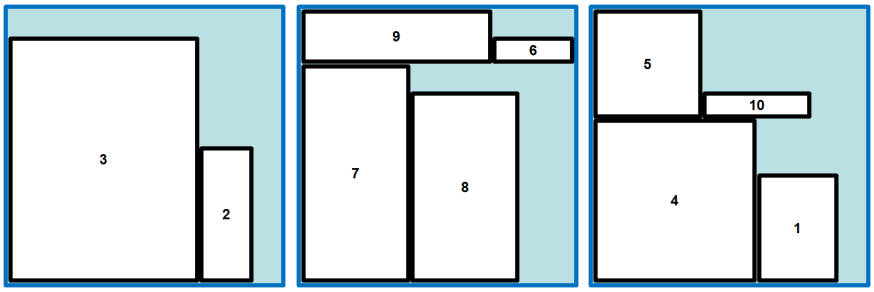
Figure 1: Level packing algorithms

Based the three heuristics described above, the following two-phase level-packing algorithms have been developed. Hybrid Next-Fit (HNF) [21] is based on NFDH, Hybrid First-Fit (HFF) [14] on FFDH and Finite Best-Strip (FBS) [5], which is also sometimes referred to as Hybrid Best-Fit, is based on BFDH. In the first

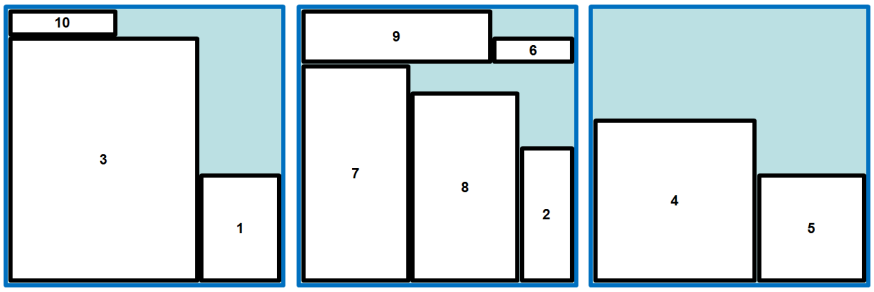
phase of all three algorithms the levels are created by the algorithm on which they are based. Then the levels are packed into bins. This is done using the same strategy as was used for the packing of the items into the levels (Figure 2).



(a) HNF



(b) HFF



(c) HBF

Figure 2: Two-phase level packing algorithms

Further two-phase level-packing algorithms are FLOOR CEILING (FC) [32] and KNAPSACK PACKING (KP) [32]. In the first phase of KP the levels are packed by solving a knapsack problem. In the second phase these levels are packed into bins. For the first phase the tallest unpacked item, say i, initializes the level. In terms of the knapsack problem the remaining horizontal distance up to the right bin border, $W - w_i$, is the capacity. Moreover, the width w_i of an unpacked item i is regarded as its weight, while the items' area $w_i \cdot h_i$ is regarded as its value (or profit). This results in a knapsack problem which is then solved. This procedure is repeated until all items are packed into

levels. In the second phase the remaining one-dimensional bin packing problem is solved by using a heuristic such as BEST-FIT DECREASING HEIGHT or an exact algorithm.

The FC algorithm can be seen as an improvement over the FBS algorithm. Again items are packed into levels in the first phase, and these levels are packed into bins in the second phase. First, the items are sorted by non-increasing height. The tallest unpacked item initializes the level and a horizontal line coinciding with the top edge of this item is the ceiling of that level. Remaining items are packed from left to right on the floor and from right to left on the ceiling. The first item on the ceiling must not fit on the floor of that level. FC tries to pack the current item first on a ceiling (if allowed) following a best-fit strategy. If not possible it tries to pack it on a floor and if that is not possible it initializes a new level. The second phase is the same as in *KP*.

One-phase non-level-packing algorithms are ALTERNATE DIRECTION (AD) [32], BOTTOM-LEFT FILL (BLF) [13], IMPROVED LOWEST GAP FILL (LGFi) [53] and TOUCHING PERIMETER (TP) [32]. In the following these techniques will be described shortly.

AD sorts the items by non-increasing heights and initializes L bins, where L is the lower bound of the two-dimensional bin packing problem. It then fills the bottom border of the bins from left to right using a best-fit decreasing strategy. After that one bin after another is being filled. In this context items are packed in bands from left to right and from right to left until no items can be packed into the current bin anymore.

BLF initializes bins by placing the first item at the bottom left corner. The top left and bottom right corners of already placed items are positions where new items could be inserted. BLF tries to place the items starting from the lowest to the highest available position. When positions with an equal height are encountered, the position closer to the left is tried first.

LGFi has a preprocessing and a packing stage. In the preprocessing stage, items are sorted by non-increasing area as a first criterion, and in a case of tie by non increasing absolute difference between height and width of the item. In the packing stage a bin is initialized with the first unpacked item, which is placed at the bottom left corner. Now items are packed on the bottom leftmost position. If possible, an item is chosen such that either the horizontal gap, or the vertical gap to the top, is filled. If this is

not possible, the largest fitting item is placed at this position. This is repeated until all items are packed.

TP first sorts the items by non-increasing area and initializes L bins, where L is the computed lower bound for the related two-dimensional bin packing problem. Furthermore, depending on a certain position, a score is associated to each item: the percentage of the edges of the item touching either an edge of another item or the border of the bin. Each item is now tried on different positions in the bin and for each position the corresponding score is calculated. The item is then placed at the position at which the score is highest.

Tabu Search (TS) [31] [33] is a meta-heuristic and therefor cannot be classified as a one- or two-phase algorithm.

Tabu Search uses lists containing moves which are considered forbidden to use again for a certain amount of iterations. First a starting solution is created using a heuristic such as FBS, KP, AD...etc. and a lower bound for the problem instance is calculated. TS then selects a target bin b, which it tries to empty. For that it defines a subset S containing an item i from bin b and k other bins. Using a heuristic, such as the ones mentioned before, it now repacks the subset S and if it can be packed in k or less bins the move is executed and added to the tabu list. This is repeated with all combinations of i and k, where k can be increased up to a fixed number, until either the lower bound is reached or the algorithms is considered stuck and has to be restarted by randomly moving packed items into empty bins.

Extreme Point-based Heuristic (C-EPBFD) [17] is a heuristic originally designed for the three-dimensional bin packing problem, but was also applied to the two-dimensional problem.

This heuristic uses extreme points to determine all points in the bin where items can be placed. Extreme points can either be corners of the already placed items or points generated by the extended edges of the placed items. These points are updated every time an item is placed into the bin. For placing the items a modified version of BFDH is used.

Concerning the performance of the heuristics two-phase level-packing heuristics pro-

vide the worst results. One can expect the average gap to the lower bound to be above 8%. Their big advantage is that, because of their simple nature, they solve 2BP problems very fast. One-phase non-level heuristics perform slightly better, as one can expect that the average gap to the lower bound will be above 7%, except for BLF which performs quite poorly with a gap above 10%. More specialized heuristics such as metaheuristics or heuristics developed for the three-dimensional case provide even better results with with results between 2% and 7%, but due to their more sophisticated approach they need the longest to solve the 2BP problem.

2.3 A New ILP Model

Inspired by the models proposed in [43] and [45] in the following a new ILP model for the 2BP is presented. For this purpose, the set of all items and the set of all bins are denoted by $\mathcal{Q} = \{1, \ldots, n\}$. W and H refer to the bin-width and the bin-height, while w_i and h_i refer to the width and the height of item $i \in \mathcal{Q}$.

The binary decision variable α_{ik} evaluates to 1 if item i is packed into bin k, and 0 otherwise. Without loss of generality, only variables α_{ik} where $i \geq k$ are created so that only $\frac{n^2+n}{2}$ instead of n^2 have to be initialized. Furthermore decision variable α_{ik} decides if bins are opened or not. Bins are considered open if the item with the same index as the bin is placed in that bin. For example item 1 cannot be placed in bin 3 but only in bin 1. Item 3 can be placed in bin 3, in bin 2 if item 2 is placed in bin 2 or in bin 1, which is always open as item 1 can only be placed in bin 1. It is easy to see that, even with this restricted variable set, all combinations of items packed into one bin are still possible. The integer variables x_i and y_i model the x- and y-coordinates of the bottom left corner of each item within a bin. For the overlapping constraints, which will be introduced in the next paragraph, the binary variables $ul_{ij}, ua_{ij}, ur_{ij}$ and uu_{ij} are needed. Each one of these four variables decides if item i has to be to the left (ul_{ij}), above (ua_{ij}), to the right (ur_{ij}) or underneath (uu_{ij}) item j. Only variables for $i < j$ are created so that only $\frac{n^2-n}{2}$ instead of n^2 have to be initialized for each variable. This can be done because if item i has to be to the left of item j, item j automatically has to be to the right of item i which makes it unnecessary to initialize the corresponding variable of item j.

$$Z = \sum_{i=0}^{n} \alpha_{ii} \rightarrow min \tag{1}$$

$$\sum_{k=0}^{n} \alpha_{ik} = 1 \qquad i,k \in \mathcal{Q}; i \geq k \tag{2}$$

$$\alpha_{ik} \leq \alpha_{kk} \qquad i,k \in \mathcal{Q}; i \geq k \tag{3}$$

$$x_i + w_i \leq W \qquad i \in \mathcal{Q} \tag{4}$$

$$y_i + h_i \leq H \qquad i \in \mathcal{Q} \tag{5}$$

$$ul_{ij} + ua_{ij} + ur_{ij} + uu_{ij} = 1 \qquad i,j \in \mathcal{Q}; i < j \tag{6}$$

$$x_i + w_i \leq x_j + W \cdot (3 - ul_{ij} - \alpha_{ik} - \alpha_{jk}) \qquad i,j,k \in \mathcal{Q}; k \leq i < j \tag{7}$$

$$y_i + H \cdot (3 - ua_{ij} - \alpha_{ik} - \alpha_{jk}) \geq y_j + h_j \qquad i,j,k \in \mathcal{Q}; k \leq i < j \tag{8}$$

$$x_i + W \cdot (3 - ur_{ij} - \alpha_{ik} - \alpha_{jk}) \geq x_j + w_j \qquad i,j,k \in \mathcal{Q}; k \leq i < j \tag{9}$$

$$y_i + h_i \leq y_j + H \cdot (3 - uu_{ij} - \alpha_{ik} - \alpha_{jk}) \qquad i,j,k \in \mathcal{Q}; k \leq i < j \tag{10}$$

$$\alpha_{ik} \in \{0,1\} \tag{11}$$

$$x_i \geq 0 \tag{12}$$

$$y_i \geq 0 \tag{13}$$

The objective function (1) minimizes the number of bins used. The constraint (2) ensures that each item is assigned to one bin. That an item i can only be assigned to an open/initialized bin is ensured by (3). That each item is placed within the bin is ensured by inequations (4) and (5). Equation (6) states that item i has to be placed either to the left, above, to the right or underneath item j. The next four equations (7)-(10) ensure that two items do not overlap if assigned to the same bin. That α_{ik} is binary is ensured in (11). The last two equations (12)-(13) state that neither x_i nor y_i can take negative values.

The equations (1)–(2) and the use of the binary variable α_{ik} are from [45]. The constraint (3) was derived from other constraints presented in [45]. The idea for using binary variables to ensure no overlapping is from [43]. The constraints (8)–(11) are modified versions of constrains also presented in [43].

2.4 Lower Bounds

The lower bound for the 2BP gives the smallest number of bins needed to pack all items. The closer it is to the actuall number of bins needed the better it is. The trivial lower bound would be the Continuous Lower Bound:

$$L_1 = \left\lceil \frac{\sum_{i=1}^{n} w_i \cdot h_i}{W \cdot H} \right\rceil \tag{14}$$

In [38] it was proven that the worst-case behavior of L_1 is

$$L_1(I) \geq \frac{1}{4} \cdot OPT(I) \tag{15}$$

$L_1(I)$ denotes the lower bound for a given instance I and $OPT(I)$ gives the optimal solution for instance I. As the lower bound can be as low as one forth of the optimal solution it does not provide a very good lower bound for computational experiments.

A more sophisticated lower bound was proposed by [38]. For all pairs of the integers (p, q) with $1 \leq p \leq \frac{H}{2}$ and $1 \leq q \leq \frac{W}{2}$ the following three sets of items are generated:

$$I_1 = i \in Q : h_i > H - p \text{ and } w_i > W - q \tag{16}$$

$$I_2 = i \in Q \backslash I_1 : h_i > \frac{1}{2} \cdot H \text{ and } w_i > \frac{1}{2} \cdot W \tag{17}$$

$$I_3 = i \in Q : \frac{1}{2} \cdot H \geq h_i \geq p \text{ and } \frac{1}{2} \cdot W \geq w_i \geq q \tag{18}$$

Using the formula

$$m(i,p,q) = \left\lfloor \frac{H}{p} \right\rfloor \cdot \left\lfloor \frac{W - w_i}{q} \right\rfloor + \left\lfloor \frac{W}{q} \right\rfloor \cdot \left\lfloor \frac{H - h_i}{p} \right\rfloor - \left\lfloor \frac{H - h_i}{p} \right\rfloor \cdot \left\lfloor \frac{W - w_i}{q} \right\rfloor \tag{19}$$

a lower bound for each pair of (p, q) can be computed with

$$L_2(p,q) = |I_1 \cup I_2| + max\left\{0, \left\lceil \frac{|I_3| - \sum_{i \in I_2} m(i,p,q)}{\left\lfloor \frac{H}{p} \right\rfloor \left\lfloor \frac{W}{q} \right\rfloor} \right\rceil \right\} \tag{20}$$

which is used to compute the overall lower bound:

$$L_2 = \max_{\substack{1 \le p \le \frac{H}{2} \\ 1 \le q \le \frac{W}{2}}} \{L_2(p,q)\} \tag{21}$$

This lower bound was improved by [9] and one was created that dominates L_2. For all pairs of the integers (p,q) with $1 \le p \le \frac{1}{2} \cdot H$ and $1 \le q \le \frac{1}{2} \cdot W$ the following five instead of three sets of items are generated:

$$A_{large} = i \in Q : h_i > H - p \text{ and } w_i > W - q \tag{22}$$

$$A_{med} = i \in Q \backslash A_{large} : h_i > \frac{1}{2} \cdot H \text{ and } w_i > \frac{1}{2} \cdot W \tag{23}$$

$$A_s^t = i \in Q : h_i > \frac{1}{2} \cdot H \text{ and } \frac{1}{2} \cdot W \ge w_j \ge q \tag{24}$$

$$A_s^w = i \in Q : \frac{1}{2} \cdot H \ge h_i \ge p \text{ and } w_i > frac12 \cdot W \tag{25}$$

$$A_s^s = i \in Q : \frac{1}{2} \cdot H \ge h_i \ge p \text{ and } \frac{1}{2} \cdot W \ge w_i \ge q \tag{26}$$

Using the formulas

$$\begin{aligned} m(i,p,q) = &\left\lfloor \frac{W - w_i}{q} \right\rfloor \cdot \left\lfloor \frac{H - h_i}{p} \right\rfloor - \left\lfloor \frac{W - w_i}{q} \right\rfloor \cdot \left\lfloor \frac{H}{p} \right\rfloor \\ &- \left\lfloor \frac{W}{q} \right\rfloor \cdot \left\lfloor \frac{H - h_i}{p} \right\rfloor \qquad \forall i \in A_{med} \end{aligned} \tag{27}$$

$$m(i,p,q) = \left\lfloor \frac{w_i}{q} \right\rfloor \cdot \left\lfloor \frac{H}{p} \right\rfloor - \left\lfloor \frac{w_i}{q} \right\rfloor \cdot \left\lfloor \frac{H - h_i}{p} \right\rfloor \qquad \forall i \in A_s^t \tag{28}$$

$$m(i,p,q) = \left\lfloor \frac{W}{q} \right\rfloor \cdot \left\lfloor \frac{h_i}{p} \right\rfloor - \left\lfloor \frac{W - w_i}{q} \right\rfloor \cdot \left\lfloor \frac{h_i}{p} \right\rfloor \qquad \forall i \in A_s^w \tag{29}$$

$$m(i,p,q) = \left\lfloor \frac{w_i}{q} \right\rfloor \cdot \left\lfloor \frac{h_i}{p} \right\rfloor \qquad \forall i \in A_s^s \tag{30}$$

again a lower bound for each pair of (p,q) can be computed with

$$L_3(p,q) = |A_{large} \cup A_{med}| + max\left\{0, \left\lceil \frac{\sum_{i \in A \setminus A_{large}} m(i,p,q)}{\left\lfloor \frac{H}{p} \right\rfloor \left\lfloor \frac{W}{q} \right\rfloor} \right\rceil\right\} \tag{31}$$

which is used to compute the overall lower bound:

$$L_3 = \max_{\substack{1 \le p \le \frac{1}{2} \cdot H \\ 1 \le q \le \frac{1}{2} \cdot W}} \{L_3(p,q)\} \tag{32}$$

As this lower bound proved to work very well it was used for the computational experiments for this thesis.

3 Probabilistic LGFi

This section will first introduce the method called probabilistic LGFi, presented in Section 3.1. The application of this method is illustrated in a small example in Section 3.2.

3.1 The Heuristic

In the following the probabilistic way of using LGFi developed will be outlined. This concerns in particular the preprocessing stage. The aim of the preprocessing stage is to order the items in a list which is then used for the packing stage. The classic LGFi approach did this using a deterministic approach. It sorted the items by non-increasing area. The probabilistic LGFi approach uses a more sophisticated approach. Instead of using a deterministic sorting mechanism, a probabilistic one is used. Therefor the area of the items determines the probability of each item getting picked and sorted first in the list. This leads to a different sorting each time the preprocessing stage is done. The packing stage stayed the same as in LGFi and is described after the preprocessing stage.

3.1.1 The Preprocessing Stage

The aim of the preprocessing stage is to sort the items using a probabilistic approach. Therefore it is necessary to calculate the probability p_i of each item getting picked and sorted in the next position of the list for each item i. These p_i depend on the value v_i which is calculated for each item i and these v_i again depend on the area a_i and absolute difference of width and height d_i of each item i.

First, for each item i the area (a_i) and the absolute difference between height and width (d_i) must be calculated:

$$a_i = w_i \cdot h_i \tag{33}$$

$$d_i = |w_i - h_i| \tag{34}$$

Then, on the basis of a_i and d_i, a value v_i is computed for each item i:

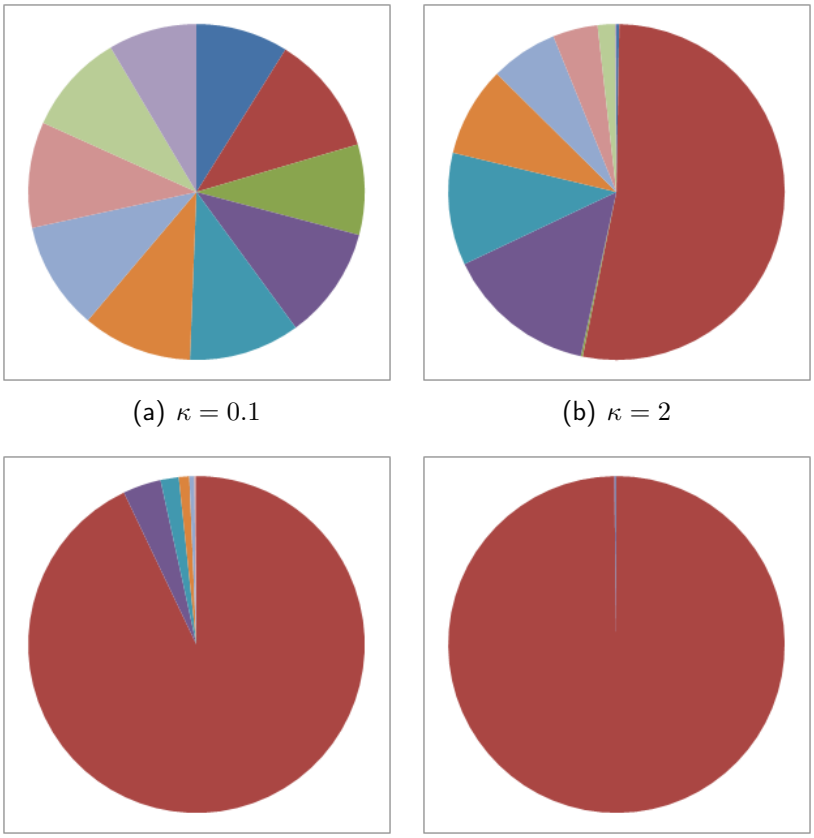
$$v_i = (\lambda \cdot a_i - d_i)^\kappa \tag{35}$$

Hereby, λ and κ are parameters. Larger values of λ result in the fact that items with larger areas receive higher v-values, that is, with increasing λ the importance of the area grows in comparison to the absolute difference between width and height. Also important is that λ is large enough so that when calculating v_i the value within the brackets does not turn negative, which is the case when $\lambda \geq 1$. With $\lambda \geq 100$ it ensures that if $a_i > a_j$ then $v_i > v_j$. Increasing λ over values than 100 does not significantly change the resulting p_i values. Therefor λ should be between 1 and 100. Note that for the computational experiments presented in the following section $\lambda = 100$ was used. Concerning κ, larger values of κ increase the difference between the v-values of different items. In other words, when $\kappa = 0.1$ the v-values of all items will be very similar to each other, while when $\kappa = 10$, for example, the v-values are characterized by large differences. Figure 3 illustrates the different distribution of v_i values for different κ values. It shows that it makes sense to use κ values between 0.1 and 10, because with $\kappa = 0.1$ all items have almost the same v_i values and with $\kappa = 10$ the largest item makes up for almost the entire area in 3(d). With a larger κ the selection probabilities of the items would be too biased.

The sequence of the items is determined randomly, depending on the probability of the items getting picked, which are based on the v-values. At each step, let $\mathcal{I} \subseteq \mathcal{Q}$ be the set of items that are not yet assigned to the list. An item $i \in \mathcal{I}$ is chosen according to probabilities p_i (for all $i \in \mathcal{I}$) by roulette-wheel-selection and becomes the next item in the list. The probabilities p_i are calculated proportional to the v-values:

$$p_i = \frac{v_i}{\sum_{i \in \mathcal{I}} v_i} \tag{36}$$

This is repeated until all items are moved to the list. Every time an item is chosen it is removed from the set $\mathcal{I}$. The result of this process is a list of all items, which is used for the packing stage.



(a) $\kappa = 0.1$ (b) $\kappa = 2$

(c) $\kappa = 5$ (d) $\kappa = 10$

Figure 3: Resulting v_i distribution for different κ

3.1.2 The Packing Stage

In the packing stage the sorted items are packed into the bins. The first bin is initialized by placing the first item from the list obtained in the preprocessing stage at the bottom left corner of the bin.

Now the bottom leftmost point in the bin, on which no item is placed, is chosen as the current point. From this current point there are two gaps, one horizontal (gap_h) and one vertical (gap_v). The horizontal gap is the distance between the current point and the right border of the bin or the left edge of the first item between the point and the right border of the bin. The distance between the point and the upper border of the bin defines the value of the vertical gap. Which ever one of those two is smaller is the current gap (gap_c):

$$gap_c = min(gap_h, gap_v) \tag{37}$$

The current gap is compared to either the widths of the items for the horizontal

gap or to the heights of the items for the vertical gap. The heuristics compares the current gap against the width and height of all unpacked items. If any item fills the gap completely it is packed with its bottom left corner on the current point and the next bottom leftmost point is determined. If no item is able to fill the gap completely the heuristic looks at the items one more time and picks the first item whose height is less or equal than the vertical gap and whose width is less or equal than the horizontal gap. If still no item fits on this position a certain area has to be declared as wastage, which works as following. A wastage area with width of the horizontal gap is created. The height of it is chosen so that the area continuously touches either an edge of an item or the border of the bin on both sides. The heuristic now searches for a new point and tries to place an unpacked item from the list again the same way as described before. This is done until the bin is completely filled with items and areas declared as wastage. A new bin is initialized with the first unpacked item placed on the bottom left corner of the new bin. This is repeated until all items are packed into bins.

3.2 Example

For this example five items, presented in Table 2, with $1 \leq w_i \leq 10$ and $1 \leq h_i \leq 10$ have to be packed in bins with $W = 10$ and $H = 10$. First they will be sorted in the preprocessing stage and after that they will be packed into the bins in the packing stage. The parameters for the preprocessing stage are set to $\lambda = 100$ and $\kappa = 2$.

Table 2: Items

i	w_i	h_i
1	2	1
2	5	2
3	10	8
4	2	8
5	3	3

3.2.1 The Preprocessing Stage

Iteration 1: First the area (a_i), using Formula (33), and the absolute difference between width and height (d_i), using Formula (34), of each item i have to be calculated. This is

shown in Table 3.

Table 3: Area and abs. difference values

i	w_i	h_i	a_i	d_i
1	2	1	2	1
2	5	2	10	3
3	10	8	80	2
4	2	8	16	6
5	3	3	9	0

Using the values a_i and d_i, v_i (Formula (35)) and therefore p_i (Formula (36)) can be calculated. Further the lower (rw_i^{lb}) and upper bound (rw_i^{ub}) for the roulette wheel can be derived for each item i from p_i. These four values are shown in Table 4, where the p_i values are presented in as %-values. The distribution of the p_i values shown in Figure 4.

Table 4: Values for iteration 1

i	v_i	p_i	rw_i^{lb}	rw_i^{ub}
1	39601	0.06%	0.000	0.001
2	994009	1.45%	0.001	0.015
3	63968004	93.59%	0.015	0.951
4	2540836	3.72%	0.951	0.988
5	810000	1.19%	0.988	1.000

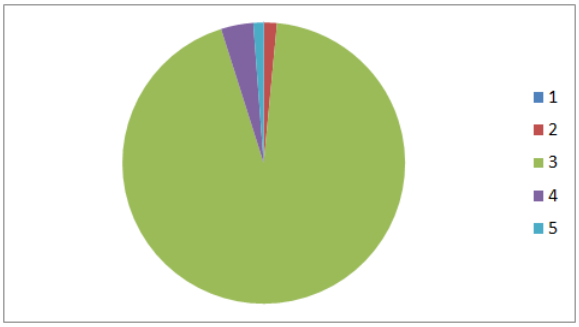


Figure 4: Distribution of p_i (Iteration 1)

Now a random number is generated (0.6285) which puts item 3 as the next item in the list ($L : \{3\}$).

Iteration 2: For all remaining items i the values for v_i, p_i, rw_i^{lb} and rw_i^{ub} have to be updated (Table 5), which of course changes the distribution of p_i values (Figure 5).

Table 5: Values for iteration 2

i	v_i	p_i	rw^lb_i	rw^ub_i
1	39601	0.90%	0.000	0.009
2	994009	22.67%	0.009	0.236
4	2540836	57.95%	0.236	0.815
5	810000	18.47%	0.815	1.000

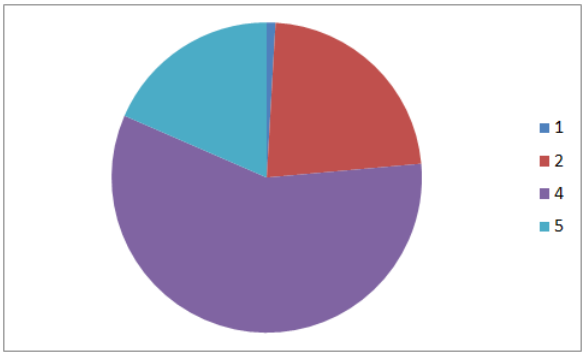


Figure 5: Distribution of p_i (Iteration 2)

Again a random number is generated (0.3843) which puts item 4 as the next item in the list ($L : \{3; 4\}$).

Iteration 3: As in iteration 2, for all remaining items i the values for v_i, p_i, rw_i^{lb} and rw_i^{ub} have to be updated (Table 6), which again changes the distribution of p_i values (Figure 6).

Table 6: Values for iteration 3

i	v_i	p_i	rw^lb_i	rw^ub_i
1	39601	2.15%	0.000	0.021
2	994009	53.92%	0.021	0.561
5	810000	43.94%	0.561	1.000

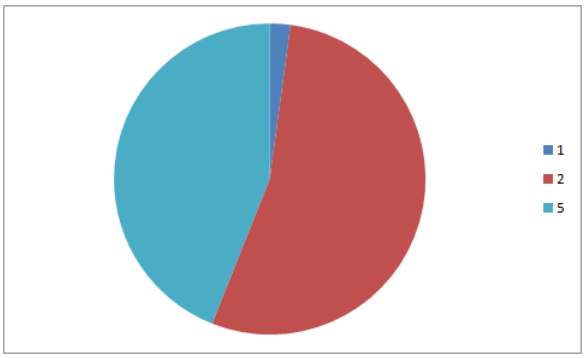


Figure 6: Distribution of p_i (Iteration 3)

With the randomly generated number 0.7941 item 5 is put next in the list (L : $\{3; 4; 5\}$).

Iteration 4: For the last iteration all remaining items i the values for v_i, p_i, rw_i^{lb} and rw_i^{ub} have to be updated (Table 7), which again changes the distribution of p_i values (Figure 7).

Table 7: Values for iteration 4

i	v_i	p_i	rw^lb_i	rw^ub_i
1	39601	3.83%	0.000	0.038
2	994009	96.17%	0.038	1.000

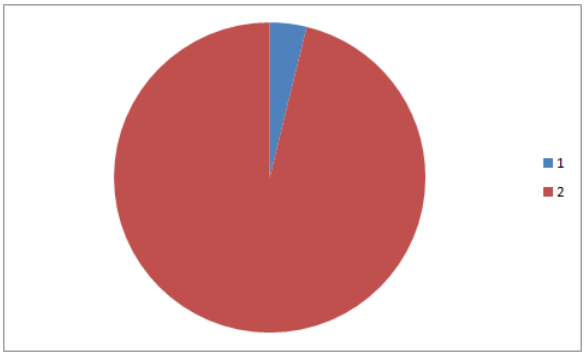


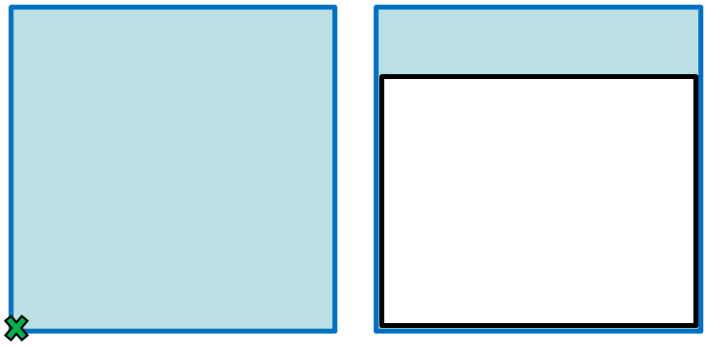
Figure 7: Distribution of p_i (Iteration 4)

The randomly generated number 0.6111 puts item 2 in the list and as only item 1

is not yet assigned to the list, it is placed on the next and last position generating the output of the preprocessing stage, which is the List $L : \{3; 4; 5; 2; 1\}$.

3.2.2 The Packing stage

Iteration 1: For the first iteration of the packing stage no item has been placed yet. The first bin is initialized by placing the first item of the list $L : \{3; 4; 5; 2; 1\}$ on the bottom leftmost position (Figure 8). In this case this is item 3, which is then removed from the list $L : \{4; 5; 2; 1\}$.



(a) (b)

Figure 8: Initialization of bin 1

Iteration 2: From the bottom leftmost unoccupied point in bin 1 (Figure 9(a)) the horizontal (gap_h) and vertical (gap_v) gap have to be calculated (Figure 9(b)). With $gap_v = 2$ and $gap_h = 10$, the shorter gap, gap_v, is selected as the current gap (gap_c) (Figure 9(c)). Now the first item of the list, which completely fills gap_c, is placed on the current point. With a height of 2 item 2 is placed on the current point (Figure 9(d)) and removed from the list $L : \{4; 5; 1\}$.

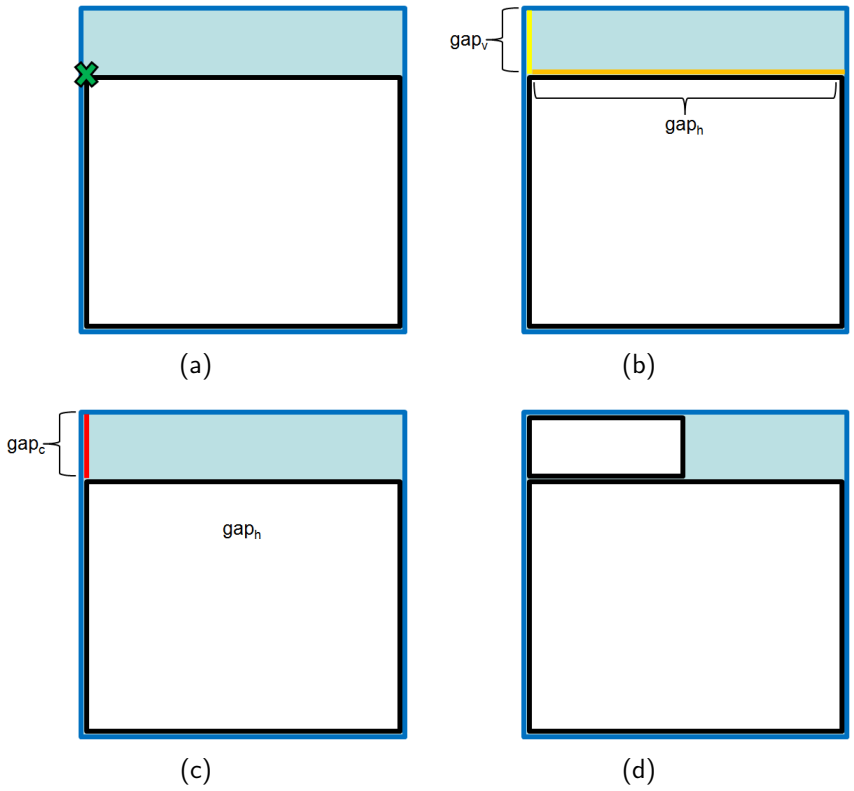


Figure 9: Placing 2nd item in bin 1

Iteration 3: From the bottom leftmost unoccupied point in bin 1 (Figure 10(a)), again the horizontal gap_h and gap_v have to be calculated (Figure 10(b)). The shorter gap ($gap_v = 2$) becomes gap_c (Figure 10(c)). Now the first item of the list, which completely fills gap_c, is placed on the current point. Because no item fulfills this condition the first item, which fits on the current point is placed there. With a height of 1 and a width of 2 item 1 is placed on the current point (Figure 10(d)) and removed from the list $L : \{4; 5\}$.

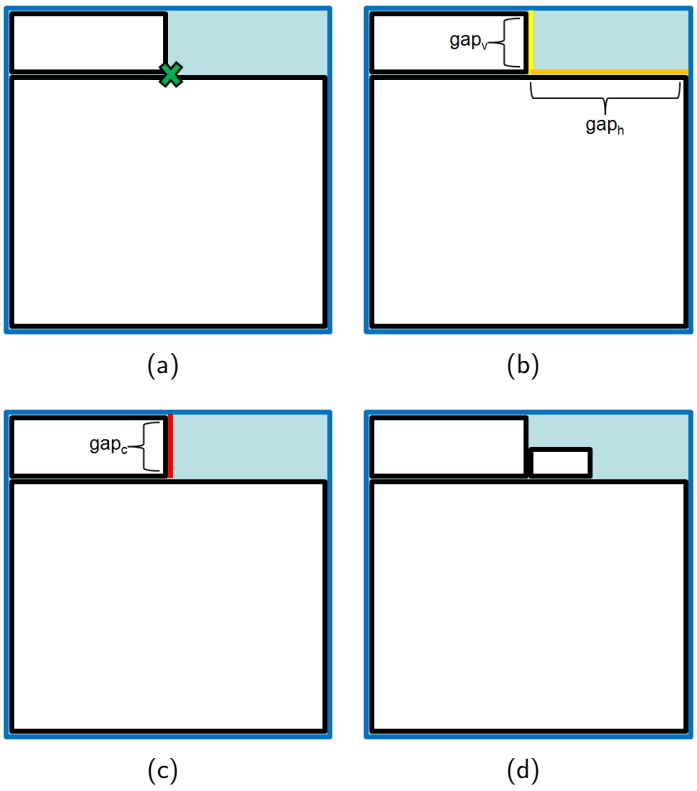


Figure 10: Placing 3rd item in bin 1

Iteration 4: As in iterations 2 and 3, gap_h and gap_v are calculated and gap_c is chosen (Figure 11(a)–Figure 11(c)). Because no item of the list can be placed on that position an area with width 3 and height 1 is declared wastage (Figure 11(d)).

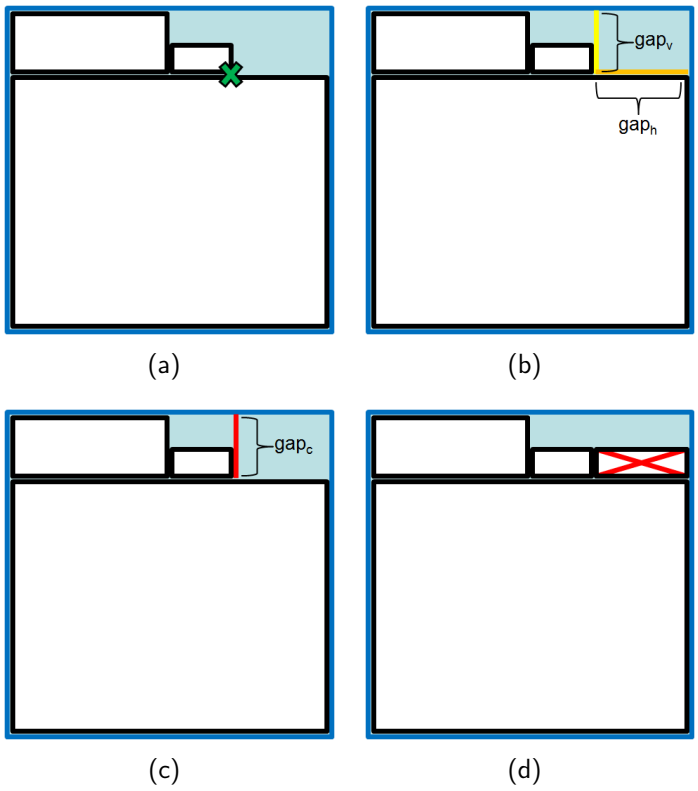


Figure 11: Declaring 1st wastage area in bin 1

Iteration 5: Like iteration 4 no item can be placed and the remaining area is declared wastage (Figure 12). The bin is now closed and a new bin will be opened in the next iteration.

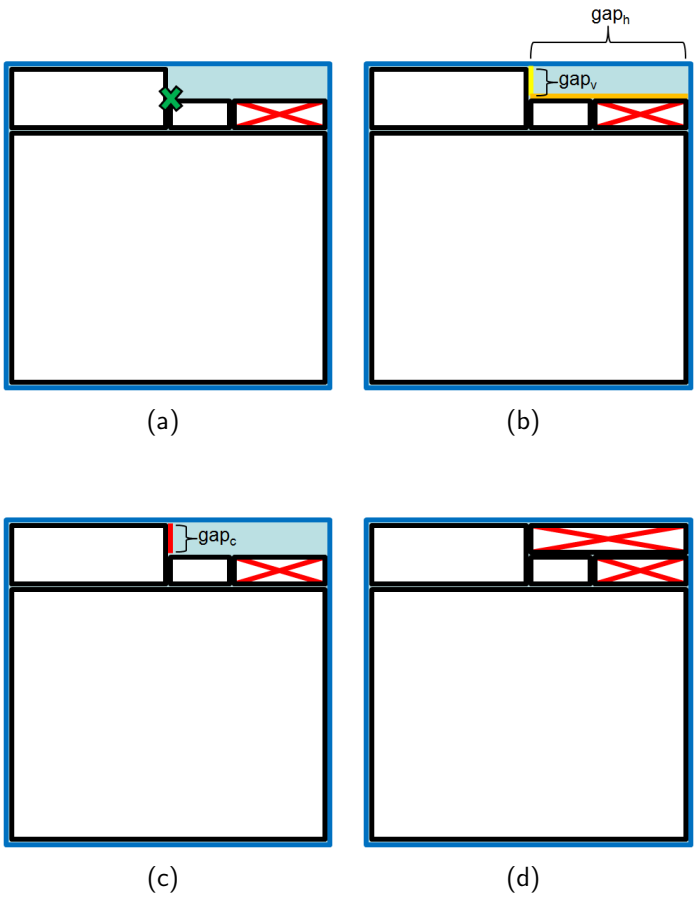


Figure 12: Declaring 2nd wastage area in bin 1

Iteration 6: As with bin 1, the first item of the list $L : \{4; 5\}$ is placed on the bottom leftmost position of bin 2 (Figure 13). Therefore item 4 is placed in bin 2 and then removed from the list $L : \{5\}$.

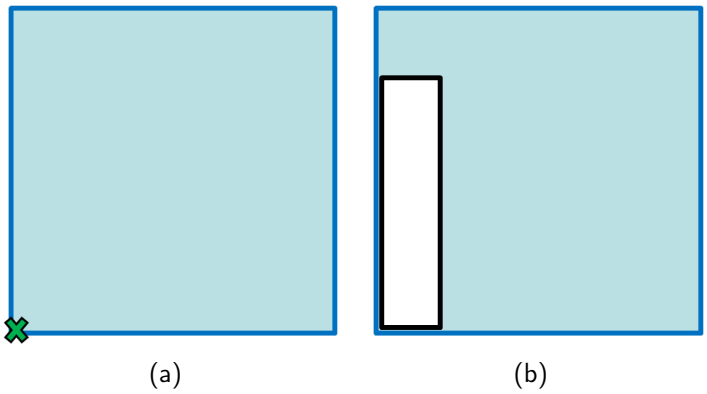


Figure 13: Initialization of bin 2

Iteration 7: Item 5, which is the last remaining item, is placed on the bottom leftmost position of bin 2 (Figure 14).

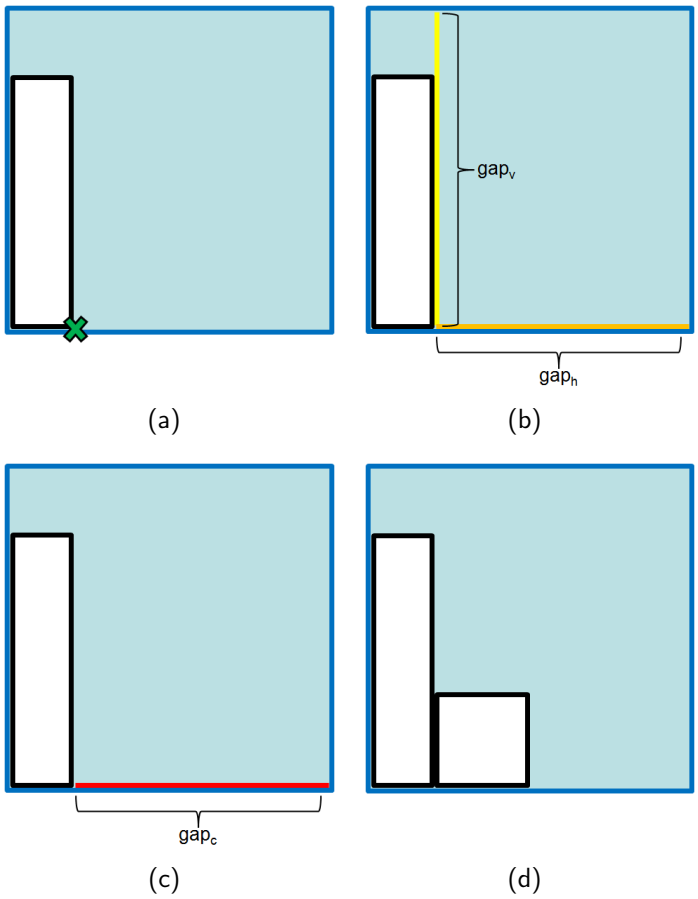


Figure 14: Placing 2nd item in bin 2

All items have now been placed in bins and therefore the heuristic is finished.

4 Solution Methods

In this section the three solution methods used to solve the 2BP are presented. In Section 4.1 a multi-start approach is presented, followed by Beam search in subsection 4.2. The last of these three approaches, namely Variable Neighborhood Search, is presented in Section 4.3.

4.1 Multi-Start Approach

For the multi-start approach the probabilistic LGFi heuristic is executed once at each iteration. The best solution obtained in this way is stored and provided as output of the algorithm when the stopping criterion has been reached. In this work a fixed number of iterations was used as stopping criterion. This algorithm is denoted in the following as Multi-start Probabilistic Improved Lowest Gap Fill (MP-LGFi). The results of this method are presented in Section 5.2.

4.2 Beam Search

Beam search (BS) was first used by [36] for the speech recognition problem and can be seen as an improvement of best-first search using breadth-first search with no back-tracking. It was mainly applied to the speech recognition problem [1] [39] [25] but also to other problems such as the job shop scheduling [46], in combination with ant colony optimization to the open shop scheduling [7] or the berth allocation problem [51].

Best-first search uses a tree to find a solution. Each node of the tree represents a partial solution. The tree itself consists at the beginning of a root, which is the starting point. Then all successors of this root are created and evaluated by how close they are to a complete solution. Then the best solution of all stored solutions is chosen and its successors are generated and evaluated. This leads to a large amount of partial solutions, which have to be stored. The main difference between beam search and best-first search is that for beam search not all partial solutions are stored but only a certain number of them and the other ones are pruned permanently. For this chosen nodes again the succeeding nodes are created, and from these succeeding nodes again the most promising

are chosen. This goes on and on until no succeeding nodes can be created anymore. As for all chosen nodes successors are created simultaneously is is a breadth-first search.

As mentioned before unlike best-first, not all nodes are explored further but only a certain amount of nodes. This number of nodes is referred to as the breadth width β. Nodes on the same level which are not considered are pruned permanently (Figure 15).

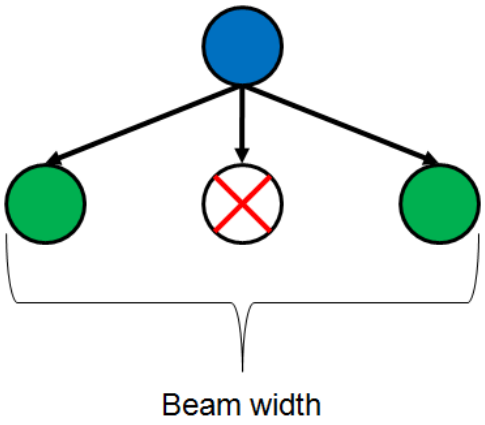


Figure 15: Beam width β

For the decision which nodes are chosen to be kept and which ones are pruned beam search uses a evaluation function. This function can either take a local or a global view on the respective node. With the local view the current state of the solution is evaluated whereas with the global view the remaining, not yet solved part, is valued. The first method is generally faster but the second method tends to generate better solutions. Further either the overall best β nodes can be chosen or for each node the best succeeding node, which also results in β nodes. In Figure 16 the exact same tree is shown twice. If one now picks the two overall best nodes of the six generated ones it can lead to a result where all chosen nodes are the successor of one and the same node (Figure 16(a)). Picking the best successor of each node chosen in the level before leads to a result illustrated in Figure 16(b). Here is a tradeoff between diversity and quality of the chosen nodes.

The filtered beam search was introduced by [40]. Unlike the normal beam search not all possible succeeding nodes for each chosen node are created but only a certain amount, which is called the filter width, denoted as γ (Figure 17). Here again there is a trade-off between computational time and result quality.

To summarize beam search starts with one node, called the root, and for this node

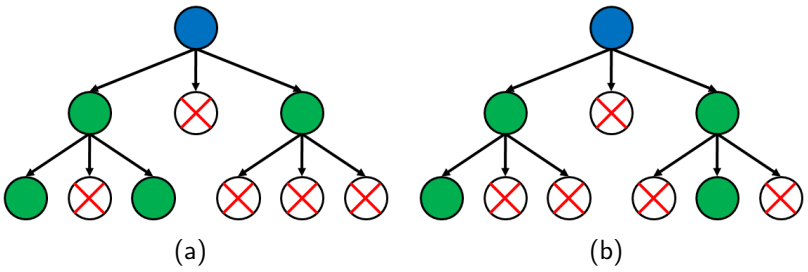


Figure 16: Overall vs. local best nodes

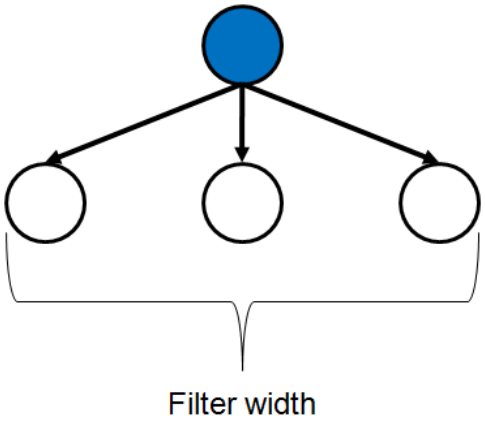


Figure 17: Filter width γ

γ succeeding nodes are created. From this level the β best nodes are chosen and the other ones are pruned permanently. For all the chosen nodes, again γ succeeding nodes are created, from which the best β nodes are picked. This is continued until all the remaining nodes represent feasible complete solutions.

For solving the 2BP always at least β nodes succeeding the root were created, so that the quality of the result would not depend too strongly on the quality of the first level. Each node represents one packed bin. So on the first level one bin is packed, on the second level two bins are packed and so on. The root marks the starting point where no bins are packed. As evaluation function the lower bound of the remaining/unpacked items was calculated, using formula (32) and for each level the β bins with the lowest lower bound were chosen. The results of this method are presented in Section 5.3

4.3 Variable Neighborhood Search

Variable neighborhood search (VNS) was introduced by Hansen and Mladenovic [26]. They define it as *"a descent, first improvement method with randomization"* [26]. It was already applied to a wide variety of different problems such as the graph coloring [3], the p-median [23], the multi depot vehicle routing [44] and the asymmetric traveling salesman problem [12]. It has proven to be very efficient and applicable to a large number of different problems.

VNS consists of three main parts. First a shaking move, which creates a solution within the neighborhood of the incumbent solution. Then a local search, which is applied on the solution gained by the shaking move. At last the decision if that solution is accepted and becomes the new incumbent solution.

First k, the neighborhood index, is set to 1 and the following steps are repeated until k reaches k_{max}. The first step is the shaking step. In this step a point of the k^{th} neighborhood of the incumbent solution is created. Next is the local search step. Here a local search is applied to the point created in the step before improving this solution. If this new solution is better than the incumbent solution the new solution becomes the incumbent solution and k is set to 1. Otherwise the incumbent solutions stays the same and k is increased by 1. The algorithm is illustrated in Figure 18.

Initialization. Select the set of neighborhood structures $N_\kappa(\kappa = 1, \ldots, \kappa_{max})$, that will be used in the search; find an initial solution x; choose a stopping condition; *Repeat* the following until the stopping condition is met:

1. Set $\kappa \leftarrow 1$;
2. *Repeat* the following steps until $\kappa = \kappa_{max}$:
   (a) *Shaking*. Generate a point x' at random from κ^{th} neighborhood of x $(x' \in N_\kappa(x))$;
   (b) *Local search*. Apply some local search method with x' as initial solution; denote with x'' the so obtained local optimum;
   (c) *Move or not*. If this local optimum x'' is better than the incumbent, or if some acceptance criterion is met, move there $(x \leftarrow x'')$, and continue the search with N_1 $(\kappa \leftarrow 1)$; otherwise, set $\kappa \leftarrow \kappa + 1$;

Figure 18: Steps of the VNS (cf. [26])

Shaking: The function of the shaking operator is to move to a neighbor of the incumbent solution so that with local search a better solution can be obtained. Even though the shaking move temporarily worsens the solution quality it avoids getting stuck in local

optima like variable neighborhood descent (VND) tends to as it only moves to neighbors which improves the solution quality. This in mind the shaking move for the VNS applied to the 2BP has to be big enough so that a new solution with less bins used can be found. For that it is necessary that at enough items are moved so that after repacking them local search can obtain a solution with less bins used. What has proven to work best was the approach to take a certain amount of bins, starting with 1, and moving the items packed within them each to an empty bin. To determine how many bins are emptied using the approach described before the formula (38) was used, where ϑ stands for the amount of bins to be emptied, ϑ_{limit} for the maximum number of bins selected, k for the current neighborhood, x for the amount of bins used by the incumbent solution and δ represents the parameter which controls how large the increment from one neighborhood to the next one is.

$$\vartheta = 1 + \frac{(k-1) \cdot x}{\delta} \tag{38}$$

Local search: For the local search two approaches are used. The first one was repacking a number of bins with MP-LGFi with the aim of achieve a packing which needs less bins than needed before. For that three things have to be considered. Which bins are selected to merge, how many bins and what is the maximum number of bins selected.

First the bins have to be selected. For that they have to be evaluated so that it is possible to compare and rank them. Knowing that one has to take a look at what distinguished the different bins, two criteria can be considered. The first one is how much of the bin area is occupied by packed items. The second one is how many items are packed within the respective bin. In [50] [32] a formula which measures the easiness of emptying the respective bin, denoted by φ, is presented. In this formula, presented in Formula (39), ρ denotes a positive number weighing the importance of area occupied, S_l a subset of the items i packed into bin l and n the total number of items. The smaller φ the easier it is to empty the respective bin.

$$\varphi(S_l) = \rho \cdot \frac{\sum_{j \in S_l} w_i \cdot h_i}{W \cdot H} - \frac{|S_l|}{n} \tag{39}$$

Next the number of bins repacked and the increment per unsuccessful iteration have to be defined. As the local search is performed after the shaking move starting with two bins works best. This is because there are a number of bins containing only one item which makes it fairly easy to merge two bins. Also the more bins are selected the longer it takes to solve the related 2BP problem for the subset. If it is not possible to merge the two selected bins a higher number of bins have to be selected to merge. The question here is how many more items should be considered. A constant incrementation proved to work best in this case. The incrementation was done using formula (40), where τ represents the number of bins selected to merge and υ the parameter defining the rate of incrementation.

$$\tau_{m+1} = \tau_m + \upsilon \tag{40}$$

So τ_0^1 is set to 2, as it is the first iteration. Now either the solution improves or stays the same after the first iteration. If it improves τ is not incremented but if it does not improve it is incremented using formula (40). So for each iteration τ is either set back to 2 or incremented. This is done until a certain limit is reached, which is denoted as τ_{limit}. This limit was depending on the current solution as the number of bins could vary from 1 up to values as high as 80 bins.

The second local search approach is applied after the first one reaches its limit. Instead of merging a number of bins it tries to pack one item of a selected bin into another bin. Here again the bin which is to be emptied has to be selected, the receiving bin has to be picked and a limit of bins, denoted as ω, which are emptied has to be considered.

The selecting procedure of the bin which is to be emptied is the same as for the first approach, using formula (39).

Now this local search operator, again using MP-LGFi, tries to pack all items, one by

one, from the selected bin into the bin which is the second easiest to empty according to formula (39). If no items can be packed it tries to pack all items, again one by one, into the third easiest to empty bin. Then the forth, fifth, sixth bin and so on. This is repeated until no item of that bin fits into any other bin or the selected bin is emptied. If the selected bin is emptied the new easiest bin to empty is selected and the procedure is repeated as before. If the bin is not emptied the second easiest to empty bin is picked to be emptied in the same way the first one was. If an item is packed into another bin the iteration stops and continues with the easiest bin again. If no item can be packed into another bin the third bin is tried to be emptied. This goes on until no item from the last bin smaller than ω can be packed into another bin.

Acceptance: After the shaking operator and the local search operators are applied on the incumbent solution the new solution can either be rejected or accepted as the new incumbent solution. The three possible outcomes and the resulting actions are shown in Table 8, where x represents the bins used in the incumbent solution and x'' the number of bins used in the solution obtained after the shaking and local search operates were applied to the incumbent solution x.

Table 8: Acceptance procedure for VNS

Outcome	Action
$x < x''$	x'' is rejected, x stays the incumbent solution and the neighborhood k is increased by 1.
$x = x''$	x'' is accepted and the neighborhood k is increased by 1.
$x > x''$	x'' is accepted and the neighborhood k is set to 1.

The results of this method are presented in Section 5.4.

5 Experimental Evaluation

All three solution methods were implemented using Microsoft Visual C++ 2008. All experiments were performed on an Intel® Xeon® X5500 @ 2.67 GHz with 3 GB of RAM. The proposed algorithms were tested on instances provided in the literature, presented in Section 5.1. In the Sections 5.2 - 5.4 the three solutions approaches will be discussed in detail. The results are then compared to other meta-heuristics and heuristics in Section 5.5

5.1 Problem Instances

Ten classes of problem instances for the 2BP are provided in the literature. A first instance set, containing six classes (I-VI), was proposed by [5]. For each of these classes, the widths and heights of the items were chosen uniformly at random from the intervals presented in Table 9. Moreover, the classes differ in the width (W) and the height (H) of the bins. Instance sizes, in terms of the number of items, are taken from $\{20, 40, 60, 80, 100\}$.10 instances for each combination of a class with an instance size were provided. This results in a total of 300 problem instances.

Table 9: Specification of instance classes I-VI (as provided by [5]).

Class	w_j	h_j	W	H
I	[1,10]	[1,10]	10	10
II	[1,10]	[1,10]	30	30
III	[1,35]	[1,35]	40	40
IV	[1,35]	[1,35]	100	100
V	[1,100]	[1,100]	100	100
VI	[1,100]	[1,100]	300	300

The second instance set, consisting of classes VII-X, was introduced by [38]. In general, they considered four different types of items, as presented in Table 10. The four item types differ in the limits for the width w_i and the height h_i of an item. Then, based on these four item types, four classes of instances were introduced which differ in the percentage of items they contain from each type. As an example, let us consider an instance of class VII. 70% of the items of such an instance are of type 1, 10% of the items are of type 2, further 10% of the items are of type 3, and the remaining 10% of

the items are of type 4. These percentages are given per class in Table 11. As in the case of the first instance set, instance sizes are taken from $\{20, 40, 60, 80, 100\}$. The instance set consists of 10 instances for each combination of a class with an instance size. This results in a total of 200 problem instances.

Table 10: Item types for classes VII-X (as introduced in [38]).

Item type	w_j	h_j	W	H
1	$[\frac{2}{3} \cdot W, W]$	$[1, \frac{1}{2} \cdot H]$	100	100
2	$[1, \frac{1}{2} \cdot W]$	$[\frac{2}{3} \cdot H, H]$	100	100
3	$[\frac{1}{2} \cdot W, W]$	$[\frac{1}{2} \cdot H, H]$	100	100
4	$[1, \frac{1}{2} \cdot W]$	$[1, \frac{1}{2} \cdot H]$	100	100

Table 11: Specification of instance classes VII-X (as provided by [38]).

Class	Type 1	Type 2	Type 3	Type 4
VII	70%	10%	10%	10%
VIII	10%	70%	10%	10%
IX	10%	10%	70%	10%
X	10%	10%	10%	70%

These all together 500 instances can be downloaded from `http://www.or.deis.unibo.it/research.html`.

5.2 Multi-Start Approach

Before conducting a full experimental evaluation of MP-LGFi, it was necessary to first understand certain aspects of the behavior of the algorithm. More specifically, the influence of the value of parameter κ, presented in formula (35) as well as the run-time behavior of the algorithm. Concerning κ, remember that rather high values result in random sequences of all the items that are very similar to the deterministic sequence generated by LFGi. This means that the higher the value of κ, the less probabilistic is this probabilistic version of LGFi. Intuitively, it was to expect that values close to zero do not work very well, because the degree of stochasticity is too high. Also values that are too high were not expected to work very well, because the resulting sequences are too similar to the deterministic sequence of LGFi. In order to confirm this intuition, MP-LGFi was applied with a limit of 100 iterations three times to each of the 500 instances. This

was done for $\kappa \in \{0.1, 0.5, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. For each κ the average percentage deviation of the corresponding results with respect to the best known lower bounds was calculated. The obtained results are graphically shown in Figure 19. They show indeed that our initial intuition appears to be true: MP-LGFi seems to work best for intermediate values of κ, that is, for values in $\{4, 5, 6\}$. Therefore, the setting of $\kappa = 5$ was chosen for all the remaining experiments.

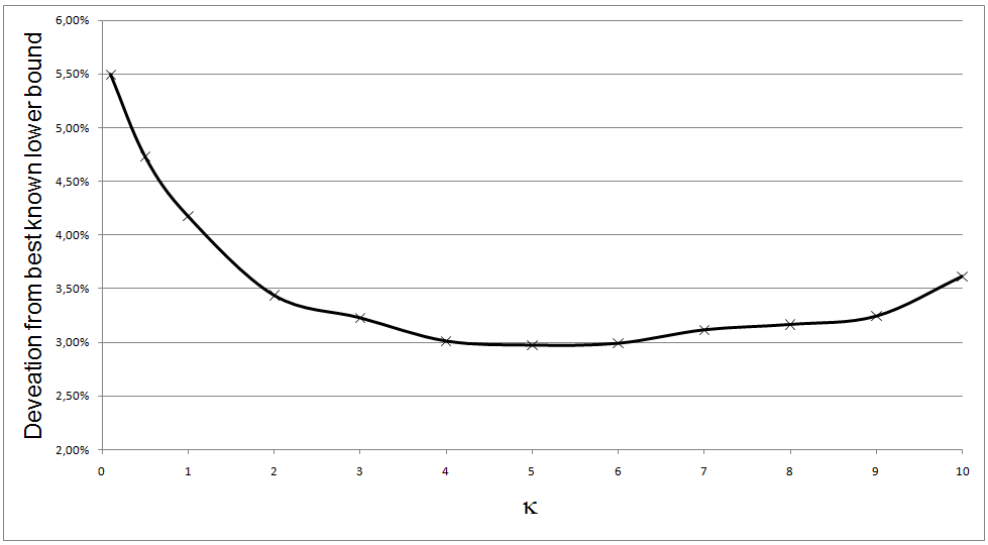


Figure 19: Results averaged over all 500 instances for different values of κ (x-axis). The y-axis provides the average percent deviation of the corresponding results with respect to the best known lower bounds.

As mentioned above, the run-time behavior of the algorithm was also given a closer look. For this purpose MP-LGFi (with $\kappa = 5$) was applied thrice to each of the 500 problem instances, using an iteration limit of up to 20000 iterations. The aggregated results are shown graphically in Figure 20. The results show that most improvements are obtained during the first 100 iterations. Further significant improvements are achieved until around 5000 iterations. After that the results almost do not improve. Given this behavior, an iteration limit of 10000 iterations was picked for the final set of experiments.

Table 12 provides numerical results of MP-LGFi, showing the results averaged over the 10 instances for each combination of class and instance size. The values in the columns with heading (q) are the ratio between the obtained solution and the lower bound of the respective two-dimensional bin packing problem. Therefore, the lower a value in the columns with heading (q) the better. Also note that in a case in which for all

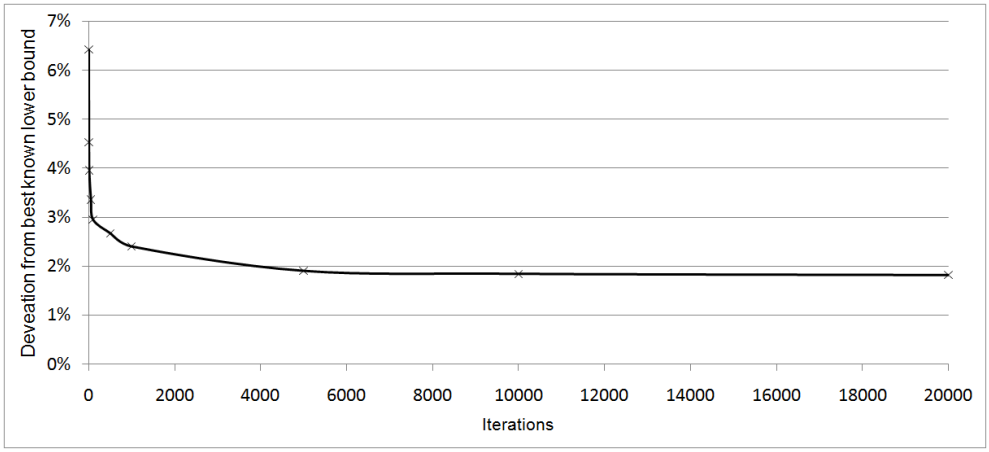


Figure 20: Results averaged over all 500 instances for different iteration limits (x-axis). The y-axis provides the average percent deviation of the corresponding results with respect to the best known lower bounds.

10 instances a solution was obtained whose value matches the one of the lower bound, the corresponding q-value is 1.000. In other words, 1.000 is the best possible q-value as it represents a gap to the best known lower bound of 0%. As the average results of three runs are shown, the table also provides information about the corresponding standard deviations (columns with heading σ), the average time when the best solution was found (columns with heading t_b), and the total runtime in seconds (columns with heading t). Moreover, the last line for each class gives the average of each algorithm for all instance sizes. In the last line the aggregated results for all 500 instances are shown.

Table 12: Numerical results for all 500 instances. The results are shown as averages over the 10 instances for each combination of instance size and class for the heuristic MP-LGFi

	MP-LGFi					MP-LGFi			
	q	σ	t_b	t		q	σ	t_b	t
Class I					**Class VI**				
20	1.000	0.000	0.0	0.1	20	1.000	0.000	0.0	0.0
40	1.000	0.000	0.0	0.3	40	1.200	0.000	10.5	124.4
60	1.017	0.000	0.0	0.5	60	1.000	0.000	9.0	4.5
80	1.004	0.000	0.0	0.4	80	1.000	0.000	0.1	0.1
100	1.000	0.000	0.0	0.8	100	1.067	0.000	0.3	234.8
Average	1.004	0.000	0.0	0.4	Average	1.053	0.000	4.0	72.8
Class II					**Class VII**				
20	1.000	0.000	0.0	0.0	20	1.000	0.000	0.0	9.0
40	1.000	0.000	0.0	0.0	40	1.020	0.000	4.3	17.0
60	1.000	0.000	0.0	0.0	60	1.019	0.000	0.7	38.0
80	1.000	0.000	0.0	0.0	80	1.037	0.000	0.0	128.4
100	1.000	0.000	0.0	0.0	100	1.009	0.002	43.8	63.8
Average	1.000	0.000	0.0	0.0	Average	1.017	0.000	9.8	51.3
Class III					**Class VIII**				
20	1.022	0.019	0.0	1.3	20	1.000	0.000	0.4	13.8
40	1.033	0.007	0.3	1.2	40	1.009	0.000	0.1	7.2
60	1.032	0.000	0.0	4.0	60	1.013	0.000	5.8	25.0
80	1.030	0.003	1.4	6.4	80	1.005	0.000	3.0	12.0
100	1.026	0.003	1.4	9.1	100	1.015	0.000	1.0	58.7
Average	1.029	0.006	0.6	4.4	Average	1.008	0.000	2.1	23.3
Class IV					**Class IX**				
20	1.000	0.000	0.0	0.0	20	1.000	0.000	0.0	0.0
40	1.000	0.000	0.0	0.0	40	1.000	0.000	0.0	52.4
60	1.100	0.000	0.0	5.7	60	1.000	0.000	0.0	53.2
80	1.033	0.000	0.1	3.6	80	1.000	0.000	0.1	110.0
100	1.000	0.000	2.3	1.1	100	1.000	0.000	0.1	87.9
Average	1.027	0.000	0.5	2.1	Average	1.000	0.000	0.0	60.7
Class V					**Class X**				
20	1.000	0.000	0.1	22.7	20	1.000	0.000	0.0	7.8
40	1.000	0.000	3.6	25.8	40	1.000	0.000	0.0	9.7
60	1.009	0.004	13.7	49.6	60	1.053	0.000	0.0	57.9
80	1.026	0.000	10.2	103.4	80	1.056	0.000	0.0	74.7
100	1.035	0.000	1.2	146.4	100	1.054	0.007	0.2	94.7
Average	1.014	0.001	5.8	69.6	Average	1.033	0.001	0.1	49.0
					Total Average	1.018	0.001	2.3	33.4

5.3 Beam Search

As discussed before in Section 4.2 There are two parameters which influence the performance of beam search. First there is the breadth width β, which determines how many nodes on each level are explored further. Then there is the filter width γ, which determines how many succeeding nodes are created.

To determine what a good ratio between β and γ is a wide range of ratios was tested. To be able to compare the different ratios the product of β and γ has to be the same so that for each level the same amount of nodes are generated which leads to comparable results as the computation times are similar. In Table 13 both parameters and the resulting ratio which have been tested on the 500 instances are presented.

Table 13: Beam search parameters and ratios

combination number	1	2	3	4	5	6	7	8
β	1	2	3	4	6	8	12	24
γ	24	12	8	6	4	3	2	1
$\frac{\beta}{\gamma}$	0.04	0.17	0.38	0.67	1.50	2.67	6.00	24.00

Each parameter setting was run three times on all 500 instances. The average gap, between the found solution and the best known lower bound, for each parameter setting is shown in Figure 21. It shows that the best results are achieved with a ratio between 1.5 and 6 with the overall best results for the ratio of 6 with 3.5%. For future experiments β was set to 12 and γ was set to 2.

Table 14 provides numerical results of beam search, showing the results averaged over the 10 instances for each combination of class and instance size. The form in which the data is presented in this table is the same as in Table 12.

Table 14: Numerical results for all 500 instances. The results are shown as averages over the 10 instances for each combination of instance size and class for the metaheuristic beam search

	Beam Search					Beam Search			
	q	σ	t_b	t		q	σ	t_b	t
Class I					**Class VI**				
20	1.000	0.000	0.0	0.0	20	1.000	0.000	13.4	13.4
40	1.003	0.005	0.1	0.1	40	1.400	0.000	129.0	129.0
60	1.024	0.003	0.3	0.3	60	1.050	0.000	332.6	332.6
80	1.005	0.002	0.5	0.5	80	1.000	0.000	775.7	775.7
100	1.013	0.004	1.1	1.1	100	1.100	0.000	1211.1	1211.1
Average	1.009	0.003	0.4	0.4	Average	1.110	0.000	492.4	492.4
Class II					**Class VII**				
20	1.000	0.000	0.0	0.0	20	1.008	0.014	2.1	2.1
40	1.100	0.000	0.3	0.3	40	1.035	0.005	6.1	6.1
60	1.050	0.000	0.8	0.8	60	1.027	0.004	15.6	15.6
80	1.033	0.000	1.6	1.6	80	1.037	0.000	23.6	23.6
100	1.033	0.000	3.0	3.0	100	1.020	0.002	42.2	42.2
Average	1.043	0.000	1.2	1.2	Average	1.025	0.005	17.9	17.9
Class III					**Class VIII**				
20	1.040	0.012	0.1	0.1	20	1.007	0.012	1.9	1.9
40	1.047	0.010	0.6	0.6	40	1.015	0.005	6.8	6.8
60	1.032	0.000	1.6	1.6	60	1.021	0.004	14.4	14.4
80	1.030	0.006	2.9	2.9	80	1.015	0.003	22.5	22.5
100	1.032	0.015	4.6	4.6	100	1.024	0.002	36.9	36.9
Average	1.036	0.008	2.0	2.0	Average	1.016	0.005	16.5	16.5
Class IV					**Class IX**				
20	1.000	0.000	0.6	0.6	20	1.000	0.000	1.4	1.4
40	1.000	0.000	6.7	6.7	40	1.000	0.000	6.8	6.8
60	1.100	0.000	16.7	16.7	60	1.000	0.000	18.4	18.4
80	1.100	0.000	33.2	33.2	80	1.000	0.000	35.3	35.3
100	1.078	0.019	52.8	52.8	100	1.000	0.000	59.6	59.6
Average	1.056	0.004	22.0	22.0	Average	1.000	0.000	24.3	24.3
Class V					**Class X**				
20	1.013	0.012	1.6	1.6	20	1.013	0.012	1.7	1.7
40	1.000	0.000	5.1	5.1	40	1.000	0.000	6.5	6.5
60	1.017	0.004	13.9	13.9	60	1.051	0.005	14.5	14.5
80	1.033	0.007	20.8	20.8	80	1.056	0.000	26.7	26.7
100	1.037	0.002	35.4	35.4	100	1.052	0.007	43.0	43.0
Average	1.020	0.005	15.4	15.4	Average	1.034	0.005	18.5	18.5
					Total Average	1.035	0.003	61.0	61.0

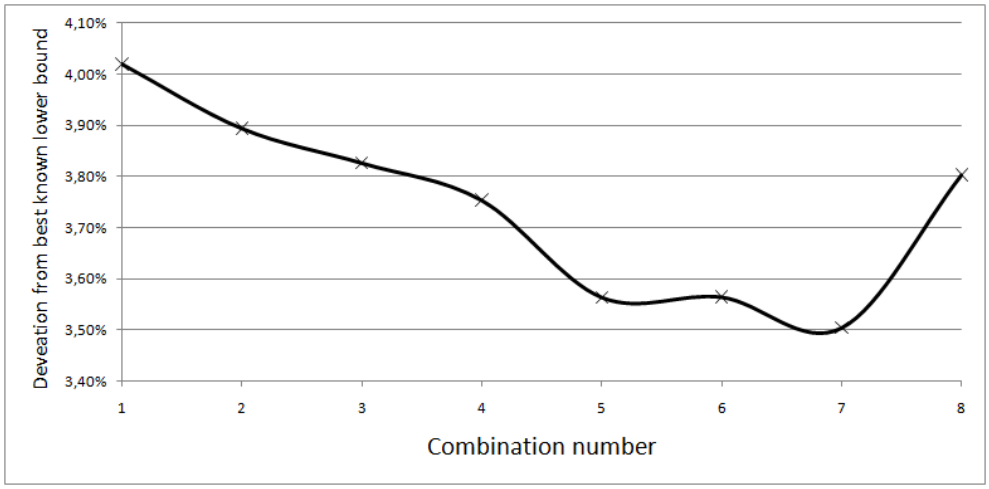


Figure 21: Results averaged over all 500 instances for the different combinations of β and γ, where the x-axis represents the combination number. The y-axis provides the average percent deviation of the corresponding results with respect to the best known lower bounds..

5.4 Variable Neighborhood Search

Before running more extensive computational evaluations the effects of the different parameters had to be tested. Some had nearly no impact on the solution quality, others had a greater influence. The stopping criterium for the VNS was set to 100 iterations and it was tested 3 times on all 500 instances presented in Section 5.1. First the ones which have not gotten that much influence and then the ones which have more influence on the result are discussed.

ϑ_{limit}, which defines the maximum amount of bins selected by the shaking operator was set to $\frac{x}{2}$. ρ, weighing the area in formula (39) was set to 5 as recommended in the literature. The incrementation rate υ in formula (38) was set to 1.

The first influental parameter to discuss is δ, setting the increment of bins selected for the shaking move per neighborhood. It was tested with the values {2;3;4;5;6;7;8;9;10;15;20}. Figure 22 illustrates the results for the different parameter settings, where the x-axis denotes the parameter value and the y-axis the average gap to the best known lower bound.

Figure 22 illustrates that neither a too small nor a too large value for δ provided

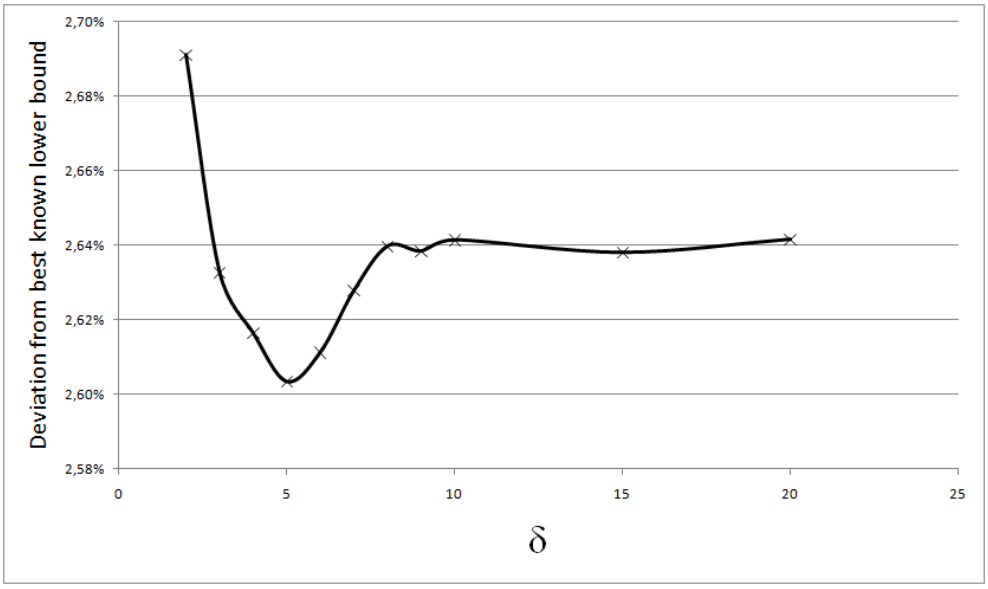


Figure 22: Results averaged over all 500 instances for different values of δ (x-axis). The y-axis provides the average percent deviation of the corresponding results with respect to the best known lower bounds.

good results, but δ set to 5 produced the results with the average lowest gap. For future experiments δ was set to 5.

The limit of bins selected to merge by the first local search operator τ_{limit} had the largest impact on the obtained results. The limit, in % of the bins used in the current solution was tested on the instances with values from 10%–90%. The results are illustrated in Figure 23.

One can see that VNS provides the best results if the first local search operator tries to merge up to 50% of easiest to empty bins of the incumbent solution. For future experiments τ_{limit} was set to 0.5.

The last more influential parameter is ω which denotes the maximum number of bins which are tried to empty using the second local search operator. It was tested for the values {1;2;3;4;5;10;15} and the results are illustrated in Figure 24

Figure 24 shows that as ω increases results tend to improve. For further research ω was set to 5. Higher values did provide slightly better results but with significant higher computational times.

After setting the parameters for the different operators the effect of using the local

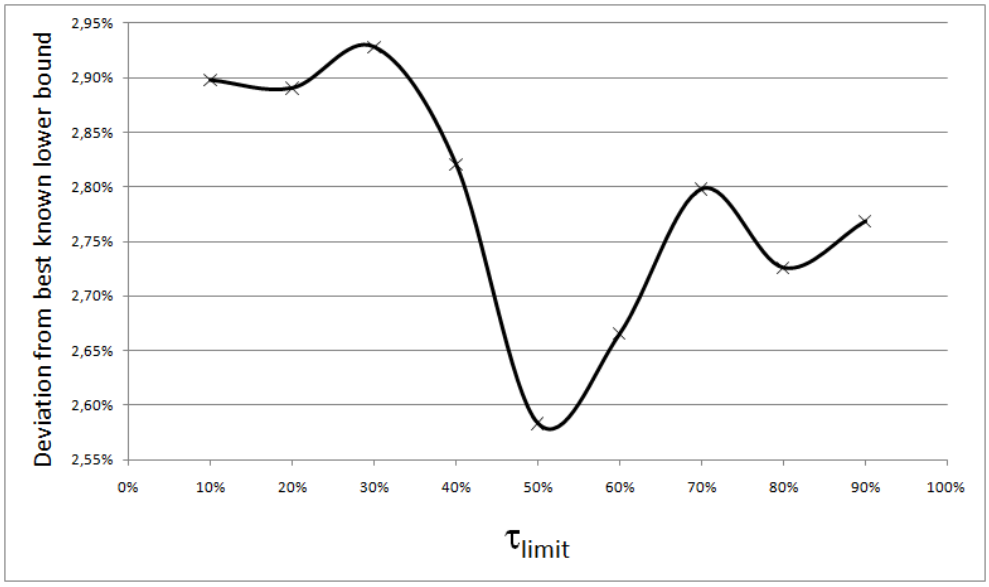


Figure 23: Results averaged over all 500 instances for different values of τ_{limit} (x-axis). The y-axis provides the average percent deviation of the corresponding results with respect to the best known lower bounds.

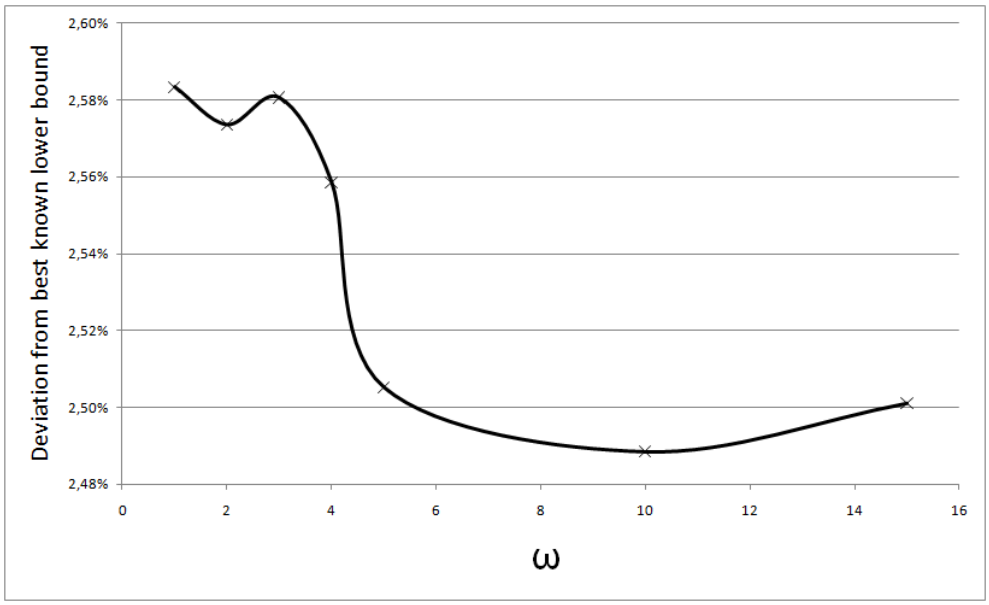


Figure 24: Results averaged over all 500 instances for different values of ω (x-axis). The y-axis provides the average percent deviation of the corresponding results with respect to the best known lower bounds.

search operators in reverse order and also applying them separately was tested. The four different possibilities tested were using them normally as described above (N), in reverse order (R), only the first local search operator (I) and only the second local search operator (II). The average results of 3 runs with a stopping criterium of 1500 iterations

are shown in Figure 25.

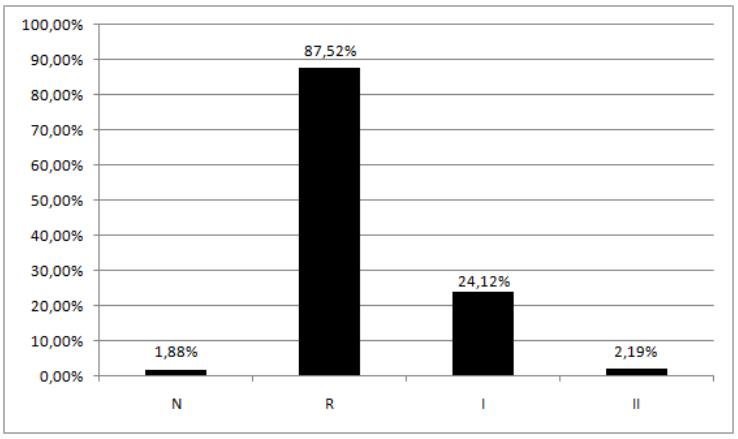


Figure 25: Results averaged over all 500 instances for different settings of local search operators (x-axis). The y-axis provides the average percent deviation of the corresponding results with respect to the best known lower bounds.

In Figure 25 one can see that the worst results, with 87.52%, are produced with the reverse order of the local search operators. With 24.12% the first local search operator (merging bins) also generates rather bad results. Good results are obtained if only using the second local search operator (moving single items) and using both of them in normal order with 2.19% and 1.88%. This shows that the normal order of the local search operators works best and was used for further tests.

Now that the parameters and the order of the local search operators was fixed the number of iterations was to be tested next. For that VNS was tested with {100;1000;2000 ;3000;4000;5000;1000} iterations. The results are shown in Figure 26.

Figure 26 shows that more than 1000 iterations does not pay of as the results only improve marginally and therefor 1000 iterations were used for the final experiments.

Table 15 provides numerical results of VNS, showing the results averaged over the 10 instances for each combination of class and instance size. The form in which the data is presented in this table is the same as in Table 12.

Table 15: Numerical results for all 500 instances. The results are shown as averages over the 10 instances for each combination of instance size and class for the metaheuristic VNS

	VNS					VNS			
	q	σ	t_b	t		q	σ	t_b	t
Class I					**Class VI**				
20	1.000	0.000	0.0	0.2	20	1.000	0.000	0.8	0.8
40	1.000	0.000	0.0	1.0	40	1.300	0.000	39.0	396.2
60	1.017	0.000	0.0	2.3	60	1.000	0.000	73.4	73.4
80	1.004	0.000	0.0	2.5	80	1.000	0.000	5.9	5.9
100	1.000	0.000	0.1	7.2	100	1.067	0.000	12.7	595.8
Average	1.004	0.000	0.0	2.6	Average	1.073	0.000	26.3	214.4
Class II					**Class VII**				
20	1.000	0.000	0.0	0.0	20	1.000	0.000	0.1	10.5
40	1.033	0.058	0.1	0.2	40	1.020	0.000	4.0	34.7
60	1.000	0.000	0.0	0.0	60	1.019	0.000	1.7	125.4
80	1.000	0.000	0.0	0.0	80	1.037	0.000	1.2	555.3
100	1.000	0.000	0.1	0.1	100	1.008	0.000	43.4	208.8
Average	1.007	0.012	0.0	0.1	Average	1.017	0.000	10.1	186.9
Class III					**Class VIII**				
20	1.000	0.000	0.1	1.4	20	1.000	0.000	0.5	12.5
40	1.033	0.007	0.5	2.9	40	1.009	0.000	0.4	15.4
60	1.029	0.005	0.1	11.6	60	1.013	0.000	12.0	70.7
80	1.023	0.007	2.5	18.7	80	1.005	0.000	4.1	47.5
100	1.022	0.003	5.0	39.1	100	1.015	0.000	2.3	299.1
Average	1.021	0.004	1.6	14.7	Average	1.008	0.000	3.9	89.0
Class IV					**Class IX**				
20	1.000	0.000	0.0	0.0	20	1.000	0.000	0.1	0.1
40	1.000	0.000	0.1	0.1	40	1.000	0.000	0.6	149.0
60	1.100	0.000	0.1	10.0	60	1.000	0.000	1.3	201.9
80	1.033	0.000	1.8	11.4	80	1.000	0.000	2.1	512.3
100	1.033	0.000	0.4	10.7	100	1.000	0.000	3.8	512.4
Average	1.033	0.000	0.5	6.4	Average	1.000	0.000	1.6	275.1
Class V					**Class X**				
20	1.000	0.000	0.1	25.7	20	1.000	0.000	0.2	6.8
40	1.000	0.000	4.7	70.7	40	1.000	0.000	0.2	20.3
60	1.007	0.000	25.5	159.9	60	1.048	0.005	6.8	173.6
80	1.026	0.000	9.1	492.6	80	1.056	0.000	0.7	323.6
100	1.030	0.002	49.5	826.2	100	1.048	0.004	9.5	432.0
Average	1.012	0.000	17.8	315.0	Average	1.030	0.002	3.5	191.3
					Total Average	1.021	0.002	6.5	129.6

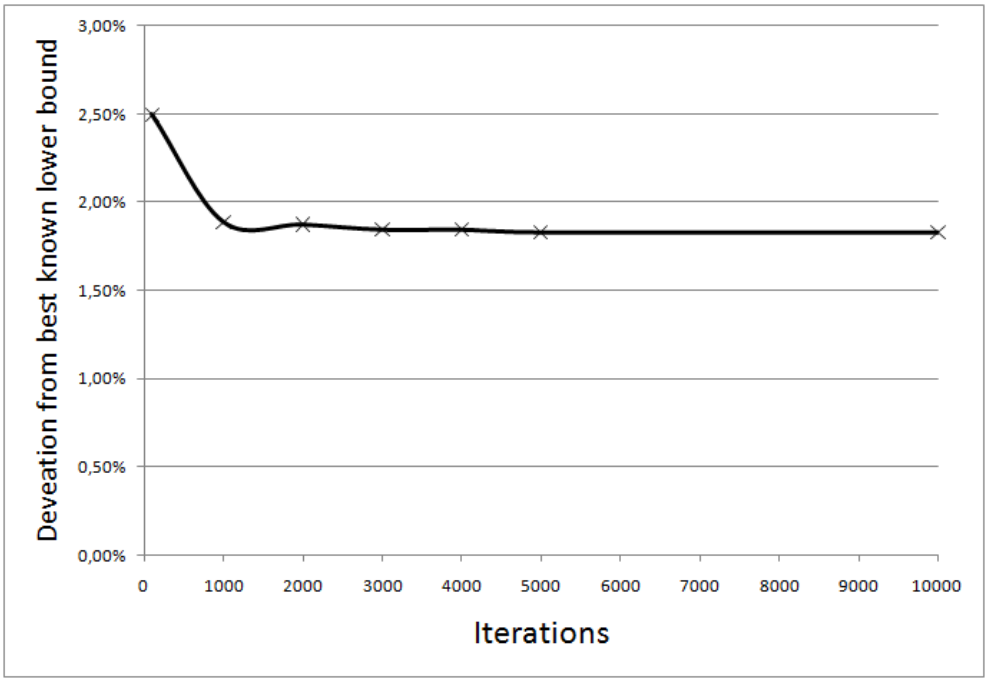


Figure 26: Results averaged over all 500 instances for different iteration limits (x-axis). The y-axis provides the average percent deviation of the corresponding results with respect to the best known lower bounds.

5.5 Comparison of Results

In this section the results of the different approaches are compared to each other. First the results of the three approaches presented (MP-LGFi, BS and VNS) are compared to FC, AD, LGFi and TS. Then the three approaches are compared to C-EPBFD. This is done separately because the results for C-EPBFD have been calculated in a different way than for FC, AD, LGFi and TS. After that the ranks of the different approaches are compared and also illustrated using boxplot diagrams. At last the results are discussed verbally.

Tables 16–17 compare the results of Floor Ceiling (FC), Alternate Directions (AD), Improved Least Gap Fill (LGFi) and Tabu Search using Alternate Directions as local search operator (TS) with the three solution approaches presented in this thesis, namely Multi-start Probabilistic Improved Least Gap Fill (MP-LGFi), Beam Search (BS) and Variable Neighborhood Search (VNS). The results for FC, AD and LGFi are [53]. TS results were presented in [32] and was run on a Silicon Graphics INDY R10000sc 195Mhz. The values in the columns

are the ratio between the obtained solution and the lower bound of the respective two-dimensional bin packing problem. Therefore, the lower a value the better. Also note that in a case in which for all 10 instances a solution was obtained whose value matches the one of the lower bound, the corresponding value is 1.000. In other words, 1.000 is the best possible value. Further all bold numbers represent the best value for each combination of class and. In the last line the aggregated results for all 500 instances are shown.

Table 16: Numerical results for all 500 instances. The results are shown as averages over the 10 instances for each combination of instance size and classes I–V for the (meta)heuristics FC, AD, LGFi, TS, MP-LGFi, BS and VNS

	FC	AD	LGFi	TS	MP-LGFi	BS	VNS
Class I							
20	1.120	1.120	1.110	1.060	**1.000**	**1.000**	**1.000**
40	1.080	1.090	1.060	1.060	**1.000**	1.003	**1.000**
60	1.070	1.070	1.050	1.040	**1.017**	1.024	**1.017**
80	1.060	1.060	1.040	1.050	**1.004**	1.005	**1.004**
100	1.060	1.050	1.030	1.040	**1.000**	1.013	**1.000**
Average	1.078	1.078	1.059	1.050	**1.004**	1.009	**1.004**
Class II							
20	1.100	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
40	1.100	1.100	1.100	1.100	**1.000**	1.100	1.033
60	1.100	1.100	1.100	1.100	**1.000**	1.050	**1.000**
80	1.070	1.070	1.030	1.070	**1.000**	1.033	**1.000**
100	1.030	1.030	1.030	1.030	**1.000**	1.033	**1.000**
Average	1.080	1.060	1.053	1.060	**1.000**	1.043	1.007
Class III							
20	1.180	1.200	1.230	1.200	1.022	1.040	**1.000**
40	1.140	1.150	1.170	1.110	**1.033**	1.047	**1.033**
60	1.110	1.130	1.100	1.050	1.032	1.032	**1.029**
80	1.100	1.100	1.070	1.080	1.030	1.030	**1.023**
100	1.090	1.090	1.090	1.090	1.026	1.032	**1.022**
Average	1.124	1.134	1.131	1.106	1.029	1.036	**1.021**
Class IV							
20	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
40	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
60	**1.100**	1.150	**1.100**	1.150	**1.100**	**1.100**	**1.100**
80	1.100	1.100	1.100	1.100	**1.033**	1.100	**1.033**
100	1.100	1.030	1.070	1.030	**1.000**	1.078	1.033
Average	1.060	1.056	1.053	1.056	**1.027**	1.056	1.033
Class V							
20	1.140	1.140	1.110	1.110	**1.000**	1.013	**1.000**
40	1.110	1.110	1.100	1.040	**1.000**	**1.000**	**1.000**
60	1.100	1.100	1.090	1.060	1.009	1.017	**1.007**
80	1.090	1.090	1.080	1.060	**1.026**	1.033	**1.026**
100	1.090	1.090	1.090	1.080	1.035	1.037	**1.030**
Average	1.106	1.106	1.092	1.070	1.014	1.020	**1.013**

Table 17: Numerical results for all 500 instances. The results are shown as averages over the 10 instances for each combination of instance size and classes VI–X for the (meta)heuristics FC, AD, LGFi, TS, MP-LGFi, BS and VNS

	FC	AD	LGFi	TS	MP-LGFi	BS	VNS
Class VI							
20	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
40	1.400	1.400	1.400	1.400	**1.200**	1.400	1.300
60	1.100	1.050	1.100	1.050	**1.000**	1.050	**1.000**
80	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
100	1.100	1.070	1.100	1.070	**1.067**	1.100	**1.067**
Average	1.120	1.104	1.120	1.104	**1.053**	1.110	1.073
Class VII							
20	1.080	1.100	1.100	1.040	**1.000**	1.008	**1.000**
40	1.090	1.100	1.070	1.060	**1.020**	1.035	**1.020**
60	1.070	1.070	1.040	1.050	**1.019**	1.027	**1.019**
80	1.060	1.060	1.060	1.040	**1.037**	**1.037**	**1.037**
100	1.040	1.040	1.030	1.030	1.009	1.020	**1.008**
Average	1.068	1.074	1.059	1.044	**1.017**	1.025	**1.017**
Class VIII							
20	1.160	1.130	1.120	1.060	**1.000**	1.007	**1.000**
40	1.070	1.080	1.080	1.030	**1.009**	1.015	**1.009**
60	1.060	1.060	1.060	1.020	**1.013**	1.021	**1.013**
80	1.060	1.060	1.040	1.020	**1.005**	1.015	**1.005**
100	1.060	1.060	1.050	1.040	**1.015**	1.024	**1.015**
Average	1.082	1.078	1.068	1.034	**1.008**	1.016	**1.008**
Class IX							
20	1.010	1.010	1.010	**1.000**	**1.000**	**1.000**	**1.000**
40	1.020	1.020	1.010	1.010	**1.000**	**1.000**	**1.000**
60	1.020	1.020	1.010	1.010	**1.000**	**1.000**	**1.000**
80	1.020	1.020	1.010	1.010	**1.000**	**1.000**	**1.000**
100	1.010	1.010	1.010	1.010	**1.000**	**1.000**	**1.000**
Average	1.016	1.016	1.012	1.008	**1.000**	**1.000**	**1.000**
Class X							
20	1.140	1.100	1.130	1.100	**1.000**	1.013	**1.000**
40	1.090	1.090	1.090	1.060	**1.000**	**1.000**	**1.000**
60	1.080	1.110	1.110	1.070	1.053	1.051	**1.048**
80	1.110	1.100	1.090	1.060	**1.056**	**1.056**	**1.056**
100	1.090	1.100	1.080	1.080	1.054	1.052	**1.048**
Average	1.102	1.100	1.100	1.074	1.033	1.034	**1.030**
Total Average	1.084	1.081	1.075	1.061	**1.018**	1.035	1.021

Table 18 compares the results of Extreme Point-based Heuristic (C-EPBFD) [17] with the results of MP-LGFi, Beam Search (BS) and Variable Neighborhood Search (VNS). The values in the columns are computed as $(mean_H - mean_{LB})/LB + 1$, where $mean_H$ represents the average of 10 instances for each combination of number of items and class for the respective heuristic. It is the same for $mean_{LB}$ but only for the best known lower bounds. Again 1.000 stands for the best possible value, as it represents a gap of 0% to the best known lower bound.

Table 18: Numerical results for all 500 instances. The results are shown as averages over the 10 instances for each combination of instance size and class for the (meta)heuristics C-EPBFD, MP-LGFi, BS and VNS

	C-EPBFD	MP-LGFi	BS	VNS		C-EPBFD	MP-LGFi	BS	VNS
	q	q	q	q		q	q	q	q
Class I					Class VI				
20	**1.000**	**1.000**	**1.000**	**1.000**	20	**1.000**	**1.000**	**1.000**	**1.000**
40	1.038	**1.000**	1.001	**1.000**	40	1.267	**1.133**	1.267	1.200
60	1.025	**1.015**	1.021	**1.015**	60	1.095	**1.000**	1.048	**1.000**
80	1.007	**1.004**	**1.004**	**1.004**	80	**1.000**	**1.000**	**1.000**	**1.000**
100	1.022	**1.000**	1.012	**1.000**	100	1.125	**1.063**	1.094	**1.063**
Average	1.019	**1.004**	1.009	**1.004**	Average	1.093	**1.037**	1.074	1.046
Class II					Class VII				
20	**1.000**	**1.000**	**1.000**	**1.000**	20	1.018	**1.000**	1.007	**1.000**
40	1.053	**1.000**	1.053	1.018	40	1.037	**1.018**	1.037	**1.018**
60	1.040	**1.000**	1.032	**1.000**	60	1.032	**1.019**	1.028	**1.019**
80	1.065	**1.000**	1.032	**1.000**	80	**1.036**	**1.036**	**1.036**	**1.036**
100	1.026	**1.000**	1.026	**1.000**	100	1.022	1.011	1.017	**1.007**
Average	1.040	**1.000**	1.031	1.003	Average	1.030	1.020	1.026	**1.018**
Class III					Class VIII				
20	1.039	**1.000**	1.031	**1.000**	20	1.017	**1.000**	1.010	**1.000**
40	1.054	**1.022**	1.039	1.025	40	1.027	**1.009**	1.013	**1.009**
60	1.044	1.029	1.031	**1.027**	60	1.025	1.015	1.020	**1.013**
80	1.054	1.025	1.030	**1.021**	80	1.018	**1.004**	1.015	**1.004**
100	1.041	1.024	1.028	**1.020**	100	1.022	**1.015**	1.023	**1.015**
Average	1.047	1.023	1.031	**1.021**	Average	1.022	**1.010**	1.018	**1.010**
Class IV					Class IX				
20	**1.000**	**1.000**	**1.000**	**1.000**	20	**1.000**	**1.000**	**1.000**	**1.000**
40	1.053	**1.000**	**1.000**	**1.000**	40	**1.000**	**1.000**	**1.000**	**1.000**
60	1.130	**1.087**	**1.087**	**1.087**	60	**1.000**	**1.000**	**1.000**	**1.000**
80	1.100	**1.033**	1.100	**1.033**	80	**1.000**	**1.000**	**1.000**	**1.000**
100	1.135	**1.000**	1.059	1.027	100	**1.000**	**1.000**	**1.000**	**1.000**
Average	1.101	**1.025**	1.061	1.034	Average	**1.000**	**1.000**	**1.000**	**1.000**
Class V					Class X				
20	**1.000**	**1.000**	1.012	**1.000**	20	1.024	**1.000**	1.019	**1.000**
40	1.043	1.003	**1.000**	**1.000**	40	**1.000**	**1.000**	**1.000**	**1.000**
60	1.017	1.007	1.017	**1.006**	60	1.071	1.048	1.049	**1.044**
80	1.037	1.026	1.031	**1.025**	80	1.081	**1.054**	1.057	1.057
100	1.036	1.032	1.033	**1.027**	100	1.072	1.057	1.051	**1.048**
Average	1.031	1.019	1.023	**1.017**	Average	1.059	1.041	1.042	**1.038**
					Total Average	1.023	**1.012**	1.018	**1.012**

Table 19–20 shows how often each (meta)heuristic has achieved which rank, with rank 1 for the best and 7 or 4 for the worst ranking, compared to the other heuristics presented in the respective Table. The heuristics are ranked 50 times, once for each combination of number of items and class.

Table 19: Distribution of Rankings I

	FC	AD	LGFi	TS	MP-LGFi	BS	VNS
1	5	5	6	6	41	16	47
2	0	1	0	1	7	4	2
3	4	6	5	7	2	25	0
4	4	4	14	29	0	2	1
5	11	7	18	6	0	1	0
6	20	23	5	1	0	1	0
7	6	4	2	0	0	1	0

Table 20: Distribution of Rankings II

	C-EPBFD	MP-LGFi	BS	VNS
1	13	39	16	45
2	0	9	2	5
3	7	2	30	0
4	30	0	2	0

The distribution of these ranks is again represented as a boxplot in the Figures 27–28.

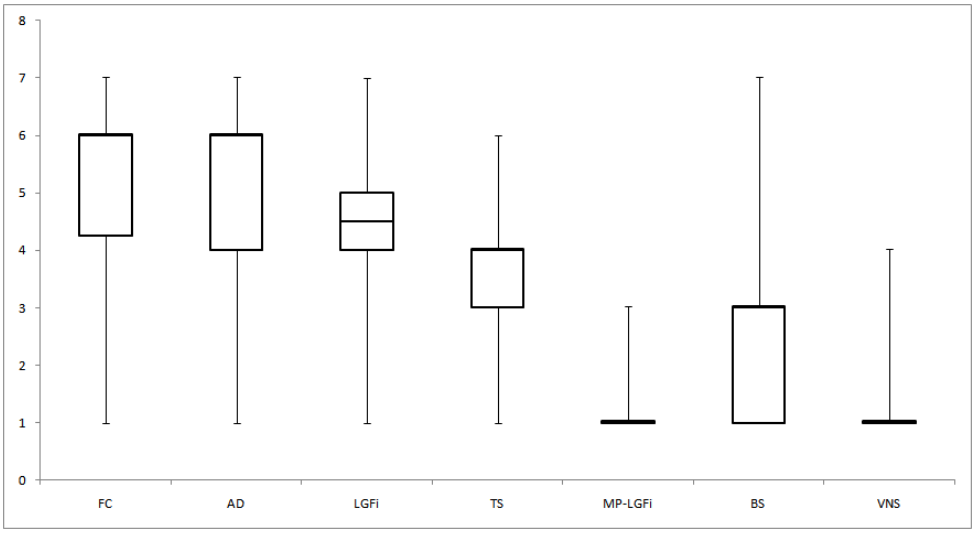


Figure 27: Boxplot of distribution of ranks I

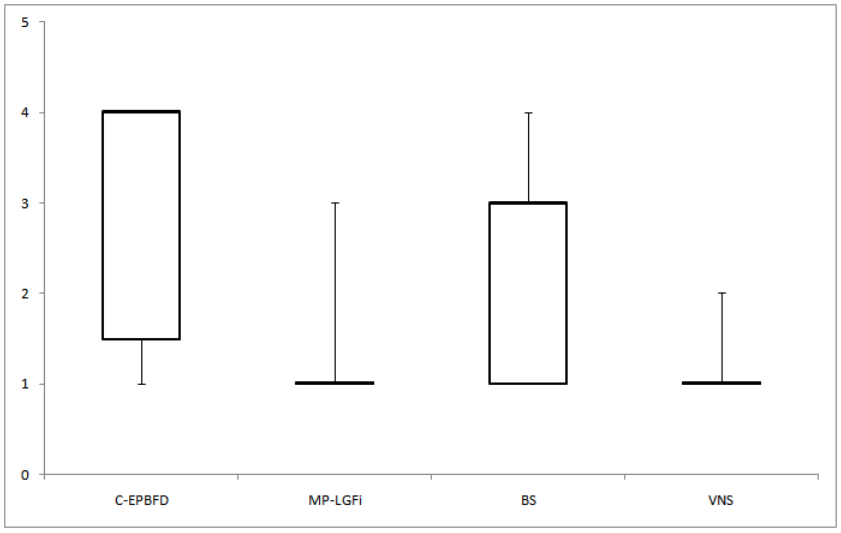


Figure 28: Boxplot of distribution of ranks II

Comparing the results of the three solution approaches presented in Table 16–17 to the results of the first four (meta)heuristics (FC, AD, LGFi and TS) provided by the literature, shows that MP-LGFi is never worse than any of those five. Beam Search provides better or equal results in 45 out of 50 cases for all combinations of classes and number of items. Looking at the average gap for each of the 10 classes, Beam Search generates 8 times equal or better results. Over all 500 instances Beam Search outperforms all four approaches. Except for one time, VNS is always equally good or better than the four other approaches, when looking at the 50 combinations of classes and number of items. For the classes and the overall average gap VNS never provides worse results.

Looking at the results for the different classes presented in Table 16–17 shows that most of the time MP-LGFi, BS and VNS outperform the other four approaches and MP-LGFi and VNS, being equally good, provide the best results, but MP-LGFi is averagely four times faster than VNS (see Tables 12 and 15). BS may provide worse results than VNS but is still averagely twice as fast as VNS (see Tables 14 and 15). Only in Class III VNS outperforms MP-LGFi clearly. VNS also seems to work better for the Classes V and X. Classes IV and IX can be considered as rather easy to solve.

The different performances are also clearly shown in Figure 27. FC and AD provide the worst performance. LGFi performs slightly better, and TS a bit better than LGFi. All three approaches presented in this thesis (MP-LGFi, BS and VNS) outperform the ones

mentioned before. BS provides slightly better results than TS. Comparing MP-LGFi and VNS shows that they almost perform equally well, but the worst rank MP-LGFi reaches is 3, compared to 4 for VNS.

Looking at the results provided in Table 18, it shows that the three approaches presented in this thesis almost always outperform C-EPBFD. Only twice out of 50 times BS performs worse than C-EPBFD. MP-LGFi and VNS always provide better or equally good results than C-EPBFD. However, C-EPBFD is much faster than MP-LGFI, BS and VS as indicated in [17].

Giving the classes a closer look shows that in general MP-LGFi and VNS provide the best solutions. VNS seems to provide better results in the Classes III and V. Classes II and IX tend to be rather easy to solve.

When ranking the performances of the 50 combinations of number of items and class, the distribution of these ranks, illustrated in Figure 28, again shows that C-EPBFD performs worse than MP-LGFi, BS and VNS. BS performs slightly better than C-EPBFD and MP-LGFi and VNS compute the best results. In this case VNS seems to be a bit better with a worst rank of 2 compared to MP-LGFi with 3.

Further MP-LGFi and VNS managed to find a new best upper bound for three of the 500 instances, reducing the number of instances where the upper bound does not match the lower bound from 68 to 65. The Upper bound was lowered for instance 398 (Class 8, Instance 8, 100 Items) from 29 to 28, 197 (Class 4, Instance 7, 100 Items) from 4 to 3 and 187 (Class 4, Instance 7, Items 80) from 4 to 3.

6 Conclusion

In this thesis the two-dimensional bin packing problem with oriented items and free guillotine cutting (2BP|O|F) was discussed. For this, a new ILP model has been developed. Moreover an existing heuristic was improved by adding a probabilistic sorting mechanism resulting in the heuristic called Probabilistic Improved Least Gap Fill. It was applied using three different approaches, namely a multi start approach, Beam Search and Variable Neighborhood Search. Of those three Beam Search, with and average gap of 3.5% to the best known lower bound performed worst, but still outperformed other (meta)heuristics compared too. VNS and the multi start approach performed better with an average gap of 2.1% and 1.8% to the best known lower bounds of the 500 instances provided by the literature. Further three new upper bounds for the 500 instances tested were found.

References

[1] S. Abdou and M. S. Scordilis. Beam search pruning in speech recognition using a posterior probability-based confidence measure. *Speech Communication*, 42(3-4):409–428, 2004.

[2] A. C. F. Alvim, C. C. Ribeiro, F. Glover, and D. J. Aloise. A hybrid improvement heuristic for the one-dimensional bin packing problem. *Journal of Heuristics*, 10:205–229, 2004.

[3] C. Avanthay, A. Hertz, and N. Zufferey. A variable neighborhood search for graph coloring. *European Journal of Operational Research*, 151(2):379–388, 2003.

[4] N. Bansal, Z. Liu, and A. Sankard. Bin-packing with fragile objects and frequency allocation in cellular networks. *Wireless Networks*, 15:821–830, 2009.

[5] J. O. Berkey and P. Y. Wang. Two dimensional finite bin packing algorithms. *Journal of the Operational Research Society*, 38(5):423–429, 1987.

[6] E. E. Bischoff. Three-dimensional packing of items with limited load bearing strength. *European Journal of Operational Research*, 168(3):952–966, 2006.

[7] C. Blum. Beam-aco–hybridizing ant colony optimization with beam search: an application to open shop scheduling. *Computers and Operations Research*, 32(6):1565–1591, 2005.

[8] C. Blum, V. Hemmelmayr, H. Hernández, and V. Schmid. Hybrid algorithms for the variable sized bin packing problem. In María Blesa, Christian Blum, Günther Raidl, Andrea Roli, and Michael Sampels, editors, *Hybrid Metaheuristics*, volume 6373 of *Lecture Notes in Computer Science*, pages 16–30. Springer Berlin / Heidelberg, 2010.

[9] M. Boschetti and A Mingozzi. The two-dimensional finite bin packing problem. part i: new lower bounds for the oriented case. *4OR: A Quarterly Journal of Operations Research*, 1(1):27–42, 2003.

[10] A. Bouganis and M. Shanahan. A vision-based intelligent system for packing 2-d irregular shapes. *Automation Science and Engineering, IEEE Transactions on*, 4(3):382–394, 2007.

[11] B. Brugger, K. F. Doerner, R. F. Hartl, and M. Reimann. Antpacking, an ant colony optimization approach for the one-dimensional bin packing problem. In J. Gottlieb and G. R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 3004 of *Lecture Notes in Computer Science*, pages 41–50. Springer Berlin Heidelberg, 2004.

[12] E. Burke, P. Cowling, and R. Keuthen. Effective local and guided variable neighbourhood search methods for the asymmetric travelling salesman problem. In Egbert Boers, editor, *Applications of Evolutionary Computing*, volume 2037 of *Lecture Notes in Computer Science*, pages 203–212. Springer Berlin / Heidelberg, 2001.

[13] B. Chazelle. The bottom-left bin packing heuristic: An efficient implementation. *IEEE Transactions on Computers*, 32(8):697–707, 1983.

[14] F. R. K. Chung, M. R. Garey, and D. S. Johnson. On packing two-dimensional bins. *SIAM Journal on Algebraic and Discrete Methods*, 3(1):66–76, 1982.

[15] F. Clautiaux, J. Carlier, and A. Moukrim. A new exact method for the two-dimensional bin-packing problem with fixed orientation. *Operations Research Letters*, 35(3):357 – 364, 2007.

[16] I. Correia, L. Gouveia, and F. Saldanha da Gama. Solving the variable size bin packing problem with discretized formulations. *Computers and Operations Research*, 35(6):2103–2113, 2008.

[17] T. G. Crainic, G. Perboli, and R. Tadei. Extreme point-based heuristics for three-dimensional bin packing. *Informs Journal on Computing*, 20(3):368–384, 2008.

[18] T. G. Crainic, G. Perboli, and R. Tadei. Ts2pack: A two-level tabu search for the three-dimensional bin packing problem. *European Journal of Operational Research*, 195(3):744 – 760, 2009.

[19] K. A. Dowsland and W. B. Dowsland. Packing problems. *European Journal of Operational Research*, 56(1):2–14, 1992.

[20] Jr. E. G. Coffman, M. R. Garey, D. S. Johnson, and R. E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9(4):808–826, 1980.

[21] J. B. G. Frenk and G. Galambos. Hybrid next-fit algorithm for the two-dimensional rectangle bin-packing problem. *Computing*, 39(3):201–217, 1987.

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

[23] F. García-López, B. Melián-Batista, J. A. Moreno-Pérez, and J. M. Moreno-Vega. The parallel variable neighborhood search for the p-median problem. *Journal of Heuristics*, 8:375–388, 2002. 10.1023/A:1015013919497.

[24] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[25] R. Haeb-Umbach and H. Ney. Improvements in beam search for 10000-word continuous-speech recognition. *Speech and Audio Processing, IEEE Transactions on*, 2(2):353–356, 1994.

[26] P. Hansen and N Mladenovic. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467, 2001.

[27] M. Haouari and M. Serairi. Heuristics for the variable sized bin-packing problem. *Computers and Operations Research*, 36(10):2877–2884, 2009.

[28] M. Hifi, I. Kacem, S. Nègre, and L. Wu. A linear programming approach for the three-dimensional bin-packing problem. *Electronic Notes in Discrete Mathematics*, 36:993–1000, 2010.

[29] M. Van Houdenhoven, J. M. Van Oostrum, E. W. Hans, G. Wullink, and G. Kazemier. Improving operating room efficiency by applying bin-packing and portfolio techniques to surgical case scheduling. *Anesthesia and analgesia*, 105(3):707–714, 2007.

[30] A. Lodi. *Algorithms for Two-Dimensional Bin Packing and Assignment Problems*. PhD thesis, Università degli Studio di Bologna, 1996-1999.

[31] A. Lodi, S. Martello, and D. Vigo. Approximation algorithms for the oriented two-dimensional bin packing problem. *European Journal of Operational Research*, 112(1):158–166, 1999.

[32] A. Lodi, S. Martello, and D. Vigo. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, 11(4):345–357, 1999.

[33] A. Lodi, S. Martello, and D. Vigo. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 123(1-3):379–396, 2002.

[34] A. Lodi, S. Martello, and D. Vigo. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241–252, 2002.

[35] K. Loh, B. Golden, and E. Wasil. Solving the one-dimensional bin packing problem with a weight annealing heuristic. *Computers and Operation Research*, 35(7):2283–2291, 2008.

[36] B. T. Lowerre. *The harpy speech recognition system*. PhD thesis, Carnegie Mellon University, 1976.

[37] E. López-Camacho, H. Terashima-Marín, and P. Ross. Defining a problem-state representation with data mining within a hyper-heuristic model which solves 2d irregular bin packing problems. In Angel Kuri-Morales and Guillermo Simari, editors, *Advances in Artificial Intelligence, IBERAMIA 2010*, volume 6433 of *Lecture Notes in Computer Science*, pages 204–213. Springer Berlin / Heidelberg, 2010.

[38] S. Martello and D. Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management Science*, 44(3):388–399, 1998.

[39] H. Ney, D. Mergel, A. Noll, and A. Paeseler. A data-driven organization of the dynamic programming beam search for continuous speech recognition. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '87.*, volume 12, pages 833–836, 1987.

[40] P. S. Ow and T. E. Morton. Filtered beam search in scheduling. *International Journal of Production Research*, 26:35–62, 1988.

[41] F. Parreño, R. Alvarez-Valdes, J. Oliveira, and J. Tamarit. A hybrid grasp/vnd algorithm for two- and three-dimensional bin packing. *Annals of Operations Research*, 179:203–220, 2010. 10.1007/s10479-008-0449-4.

[42] D. Pisinger and M. Sigurd. The two-dimensional bin packing problem with variable bin sizes and costs. *Discrete Optimization*, 2(2):154–167, 2005.

[43] D. Pisinger and M. Sigurd. Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem. *INFORMS Journal on Computing*, 19(1):36–51, 2007.

[44] M. Polacek, R. F. Hartl, K. Doerner, and M. Reimann. A variable neighborhood search for the multi depot vehicle routing problem with time windows. *Journal of Heuristics*, 10:613–627, 2004. 10.1007/s10732-005-5432-5.

[45] J. Puchinger and G. Raidl. Models and algorithms for three-stage two-dimensional bin packing. *European Journal of Operational Research*, 183(3):1304–1327, 2007.

[46] I. Sabuncuoglu and M. Bayiz. Job shop scheduling with beam search. *European Journal of Operational Research*, 118(2):390–412, 1999.

[47] N. Skorin-Kapov. Routing and wavelength assignment in optical networks using bin packing based algorithms. *European Journal of Operational Research*, 177(2):1167–1179, 2007.

[48] P. E. Sweeney and E. R. Paternoster. Cutting and packing problems: A categorized, application-orientated research bibliography. *The Journal of the Operational Research Society*, 43(7):691–706, 1992.

[49] H. Terashima-Marín, P. Ross, C. Farías-Zárate, E. López-Camacho, and M. Valenzuela-Rendón. Generalized hyper-heuristics for solving 2d regular and irregular packing problems. *Annals of Operations Research*, 179:369–392, 2010. 10.1007/s10479-008-0475-2.

[50] S. Voss, S. Martello, I. H. Osman, and C. Roucairol. *Neighborhood search algorithm for the guillotine non-oriented two-dimensional bin packing problem*, pages 125–139. Kluwer Academic Publishers, Boston, USA, 1998.

[51] F. Wang and A. Lim. A stochastic beam search for the berth allocation problem. *Decision Support Systems*, 42(4):2186–2196, 2007. Decision Support Systems in Emerging Economies.

[52] G. Wäscher, H. Haußner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109 – 1130, 2007.

[53] L. Wong and L. S. Lee. Heuristic placement routines for two-dimensional bin packing problem. *Journal of Mathematics and Statistics*, 5(4):334–341, 2009.

[54] E. C. Xavier and F. K. Miyazawa. The class constrained bin packing problem with applications to video-on-demand. *Theoretical Computer Science*, 393(1-3):240–259, 2008.

Abstract

The two-dimensional bin packing problem with oriented items and free guillotine cutting (2BP|O|F) has been discussed in this thesis. For the two-dimensional bin packing problem a set of small rectangular items have to be packed into an unlimited set of identical large objects. Oriented means that the items cannot be rotated and free guillotine cutting means that items can be placed everywhere as long as they are withing the bin and do not overlap. There are a large number of variations of the bin packing problem, such as different dimensionality, variable sized bins, irregular shaped items, rotatable items or guillotine cutting being required.

For this thesis a new ILP model was developed. Further an existing heuristic (LGFi) has been improved using a probabilistic approach. The heuristic consists of a preprocessing stage and a packing stage. The aim of the preprocessing stage is to sort the items and the aim of the packing stage is to pack these sorted items into bins. What was changed is that the items are not sorted in a deterministic way but using a probabilistic approach in the preprocessing stage.

This improved heuristic was applied to 500 instances provided by the literature using three different approaches. These three approaches are a multi-start approach, Beam Search and Variable Neighborhood Search. All three of them outperformed the already existing approaches, where Beam Search performed the worst and the multi-start approach and Variable Neighborhood Search are the best performing almost equally good. Additionally to outperforming the other approaches three new best solutions for the 500 instances have been found.

Zusammenfassung

Das "two-dimensional bin packing" Problem mit orientierten Elementen und freiem Schneiden (2BP|O|F) wurde in dieser Arbeit diskutiert. Für dieses Problem müssen ein Set kleiner, rechteckiger Elemente in ein unbegrenztes Set von einheitlichen großen Objekten gepackt werden. Orientiert heißt, dass die Elemente nicht gedreht werden dürfen und freies Schneiden heißt, dass die Elemente überall im großen Objekt platziert werden können, solange sie innerhalb von diesem platziert werden und sich dabei nicht überlappen. Es gibt eine große Anzahl an Variationen für das Problem, wie zum Beispiel eine unterschiedliche Dimensionalität, unterschiedlich große Objekte, unregelmäßig geformte Elemente, rotierbare Elemente oder dass nur Guillotineschnitte vorgenommen werden können.

Für diese Arbeit wurde ein neues ILP Modell entwickelt. Weiters wurde eine bereits existierende Heuristik (LGFi) verbessert, indem ein auf Wahrscheinlichkeiten basierender Ansatz verwendet wurde. Die Heuristik besteht aus einem Vorverarbeitungsschritt und einem zweiten Schritt in dem die Elemente gepackt werden. Das Ziel des Vorverarbeitungsschrittes ist es die Elemente zu sortieren und das Ziel des zweiten Schrittes ist es die sortierten Elemente zu packen. Was verändert wurde ist, dass die Elemente nicht mehr auf eine deterministische Weise sortiert werden sondern basierend auf Wahrscheinlichkeiten.

Diese verbesserte Heuristik wurde mit Hilfe von drei verschiedenen Ansätzen auf 500 Instanzen, die von der Literatur zur Verfügung gestellt wurden, angewendet. Diese drei sind ein multi-start Ansatz, Beam Search und Variable Neighborhood Search. Alle drei übertreffen die bisher dagewesenen Ansätze, wobei Beam Search die schlechteste ist und der multi-start Ansatz und Variable Neighborhood Search am besten und etwa gleich gut sind. Außerdem wurden drei neue beste Lösungen für die 500 Instanzen gefunden.

Curriculum Vitae

Name:	Lukas Baumgartner
Address:	Rembrandtstraße 16/36
	1020 Wien, Austria
Telephone number:	+4369912608723
Email:	Lukas.baumgarnter.1984@gmail.com
Date of birth:	December 21st, 1984
Place of birth:	Vienna, Austria
Nationality:	Austria

Education

Since October 2005	International Business Administration at the University of Vienna
	1st Diploma Examination: June 30th 2006
	Honored as 2nd best student of my year
	2nd Diploma Examination: June 29th, 2007
	Semester abroad in Groningen (NL) in 2008
	Specialization: Operations Research, Production & Logistics
1999-2004	Secondary College for Business Administration in Korneuburg, Austria
	Specialization: Business Informatics and Organization
	Graduated with honors
1995-1999	General high school in Stockerau, Austria
	Specialization: Informatics
1991-1995	Elementary school in Stockerau, Austria

Work Experience

University of Vienna	Tutor at the Chair of Production and Operations Management
	(March 2008-February 2009 and September 2009-August 2010)
Judoclub Stockerau	Support at organizing international and national tournaments
	Trainer since 1997
Wr. Städtische	Intern (September 2007)
Post AG	Intern (August-September 2005)
Military Service	November 2004-July 2005
BVA	Intern (August 2003, July 2004, September 2006, February 2007)

Skills

English:	Excellent language skills
	3-year residence in the USA
	1st foreign language at University
French:	Good language skills
	2nd foreign language at University
Computer:	Programming in C++
	Simulation software AnyLogic
	Good Knowledge in Latex, MS Office and SPSS
	Basic Knowledge in Linux
	Overall excellent knowledge (Software and Hardware)