Comparing approximate and optimal solution algorithms for the Multi-Level Bin Packing problem

J.J. Groenheide
Supervisors: M.G. Horn, N. Yorke-Smith
EEMCS, Delft University of Technology, The Netherlands

June 17, 2022

A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

Abstract

Multi-Level variants of classic optimisation problems are becoming more noteworthy as the complexity of real life applications increases. In this research we investigate the Multi-Level Bin Packing optimisation problem, which models, for example, global logistics and part manufacturing. We will look at the performance of solving Integer Linear Programming formulations of the Multi-Level Bin Packing problem using IBM ILOG CPLEX Optimization Studio, and compare these results to the performance of simple heuristic-based algorithms to reach conclusions about the usefulness of optimal-solution algorithms for NP-Hard problems.
We ultimately find that the simple heuristics leave a large gap in optimality, while the solving time for finding optimal solutions is still too large for practical instances. As such, we conclude that more specialised algorithms are needed that can balance the time cost and optimality, depending on the application.

1 Introduction

The research of this project revolves around the Multi-Level Bin Packing problem (MLBP). MLBP is a generalisation of the Bin Packing problem, a classic computer science optimisation problem where the goal is to distribute items among bins such that the number of required bins is minimised. Similarly, in the Multi-Level Bin Packing problem the goal is to distribute entities such that the total cost is minimised. These entities include items, intermediate bins, and top-level bins, each with a size and a cost associated with them.
Research on the standard Bin Packing problem dates back to studies by Johnson and Ullman in the early 1970s, investigating the existence of efficient approximation algorithms for the problem. Because of the attention the Bin Packing problem has received, sophisticated algorithms exist that can compute near-optimal solutions for even large instances of the problem with polynomial worst-case performance. [14, 15, 19]
In addition to the standard Bin Packing problem, numerous additional constraints have been researched. This research often includes the development of optimised algorithmic solutions, with respective upper and lower bounds for computation. [2, 13, 16]

In contrast to the high interest in the Bin Packing problem, the Multi-Level variant of the problem has received disproportionately little attention in the literature, despite the similar applicability in practical scenarios like logistics and process optimisation. This has left a large amount of research that can still be done on different optimal and approximate solution algorithms for this family of problems. [4, 11] By expanding the literature around MLBP and investigating the application of ILP models to the problem, we aim to create new interest in the practical applications of these algorithms and provoke further research.

In this research we will describe and optimise an Integer Linear Programming (ILP) formulation of the Multi-Level Bin Packing problem, as well as creating a formulation with additional Fragmentation Constraints.
The final conclusion of the research will indicate the time cost for solving instances of different sizes using an ILP model, and how this compares to the performance of heuristic-based algorithms. To make this comparison, we investigate the current state of approximation algorithms, the properties of the ILP model, and how optimisation software finds optimal solutions using this model.

For the experimental contribution of this paper, two models will be created for solving MLBP; A standard model, and a Network Flow representation of MLBP. The standard model will also be expanded to handle Fragmentation Constraints, which penalise item assignments where items from the same group are placed in different top-level bins, to investigate how a stricter objective function impacts the performance of the model. The performance of the models will be compared to that of two simple heuristic algorithms, based on the First Fit and Best Fit algorithms, followed by conclusions and recommendations.

2 Formal Definition

The problems discussed in this paper are all classified as optimisation problems. An optimisation problem is generally defined as a fourtuple $(\mathcal{I}, sol, obj, opt)$. The following definition is compiled from the works of G. Ausiello. For an optimisation problem A, the fourtuple is defined such that:

- $\mathcal{I}$ is the set of instances for the problem A. These are recognisable in polynomial time.
- Given an instance $I \in \mathcal{I}$, $sol(I)$ denotes the set of feasible solutions of the instance; for every $S \in sol(I)$, the size of S is polynomial in the size of I; given any I and any S polynomial in $|I|$, one can decide in polynomial time if $S \in sol(I)$;
- Given $I \in \mathcal{I}$ and $S \in sol(I)$, the objective function $obj(I, S)$ denotes the value of S; the value of $obj(I, S)$ is computable in polynomial time;
- $opt \in \{min, max\}$ defines whether obj should be minimised or maximised.

In his 2011 work "Complexity and Approximation in Reoptimization", Ausiello also defines approximation algorithms for optimisation problems as any algorithm that, given an instance I of a problem A, can return a feasible solution $S \in sol(I)$. The quality of an approximation algorithm is defined by the ratio between the returned solution S and the true optimal solution S'. [1]

The Multi-Level Bin Packing problem is an example of such an optimisation problem. An instance of MLBP defines a set of items $\mathcal{I}$ (not to be confused with the set of instances $\mathcal{I}$ of a problem), each with a size $s(\mathcal{I}_i)$, and a set of bins $\mathcal{B}^k$ for each level $1 \leq k \leq m$. For each bin j, we are given a size $s(\mathcal{B}^k_j) \in \mathbb{N}_{>0}$, capacity $w(\mathcal{B}^k_j) \in \mathbb{N}_{>0}$, and cost $c(\mathcal{B}^k_j) \in \mathbb{N}_{>0}$. The term 'top-level bins' refers to the set $\mathcal{B}^m$, while all other levels are considered 'intermediate bins'. 'Entities' refers to the set of items and intermediate bins, but not top-level bins.

MLBP has the following constraints.

- All items must be packed into exactly one bin of level 1.
- All active intermediate bins of level k must be placed in a bin of level $k + 1$.
- The total size of items in a bin must not exceed the capacity of that bin.

The objective function considers the sum of the costs of all used top-level and intermediate bins. As such, the goal is to minimise the total cost, while adhering to these constraints.

In addition to the standard MLBP constraints, the fragmentation constraint states that each item $\mathcal{I}_i \in \mathcal{I}$ belongs to a group $g(\mathcal{I}_i) \in G = 1, 2, \ldots, q$, where $q \in \mathbb{N}_{>0}$ denotes the maximum number of groups. q is calculated as a percentage of the total number of items. Items of the same group should be packed into the same top-level bin. To enforce this, a penalty value p is added to the cost function for every n_r bins containing items from group $r \in G$. The task is to find a packing that minimises the changed objective function such that every bin satisfies the capacity constraints.

Optimisation strategies, like dynamic programming and heuristic search, can be applied to significantly improve the time cost. Unfortunately, there are very few of these optimisation techniques that have been discovered for MLBP. Instead, we make use of existing optimisation software for Integer Programming problems by creating a reduction from MLBP to IP. The mathematical programming formulation is then used to find optimal solutions.
Integer Programming is closely related to Linear Programming. For both problems the goal is to optimise an objective function by setting variables subject to a number of constraints. The difference between the two is that the decision variables of a Linear Program can be any real number $x \in \mathbb{R}$, while for Integer Programming the decision variables are restricted to be integers. Despite the seemingly small difference in constraints, Linear Programming is solvable in polynomial time, while Integer Programming is NP-Hard. For a more detailed introduction to IP we refer to the 1988 textbook "Integer and Combinatorial Optimization" by G. L. Nemhauser and L. A. Wolsey. [18]

The Integer Programming problem is more suitable for dedicated optimisation software than other problems, like the Bin Packing problem, because there exist a number of optimisation techniques for finding optimal solutions. The most impactful of these is the "Branch-and-Bound" algorithm. [17] Branch-and-Bound can be applied to a multitude of optimisation problems, though we shall focus here on how it is applied to Integer Programming problems. The Branch-and-Bound algorithm relies on the inherent connection between Integer Programming and Linear Programming to perform an intelligent traversal of the solution space. By removing the integrality constraint from the model, we can find its LP solution in polynomial time. This solution is then used to partition the solution space into smaller sub-problems; the branching step. By keeping track of the optimality bound, sub-problems can be pruned if their LP solution value falls outside of this bound; the bounding step. [20] By applying Branch-and-Bound, the solution space can be traversed in an intelligent and guided manner, making the average time cost much lower than for problems for which no such traversal algorithm exists.

The general form of the Integer Linear Programming problem is as follows:

$$\begin{aligned} \min\ & c'x && \text{(linear objective function)} \\ \text{s.t. } & Ax \geq b && \text{(linear equality constraints)} \\ & x \in \mathbb{Z}^n \end{aligned}$$

$$\begin{aligned} x &\in \mathbb{Z}^n && n\text{-dimensional vector of decision variables} \\ c &\in \mathbb{R}^n && n\text{-dimensional cost vector} \\ A &\in \mathbb{R}^{m \times n} && m \times n \text{ constraint matrix} \\ b &\in \mathbb{R}^m && m\text{-dimensional vector} \end{aligned}$$

3 Background Information and Methodology

The introduction of computational complexity classes is generally attributed to Juris Hartmanis and Richard E. Stearns, in their 1965 paper "On the Computational Complexity of Algorithms". [9, 12] Their research formed the basis for the division of problems into classes based on their relative complexity, and defined the first notions of algorithmic "hardness". The Bin Packing problem is part of the NP-Hard complexity class, which informally means that it is at least as hard as the hardest problems in NP. Because BP is NP-Hard, there does not exist a polynomial-time solving algorithm for it.

The lack of a polynomial-time solving algorithm for BP has not discouraged researchers from investigating different polynomial-time approximation algorithms, however. Examples of such algorithms for BP are the Best Fit and First Fit algorithms, as well as more advanced near-optimal approximation algorithms, as described by Johnson in the introduction of his 1973 thesis "Near-optimal bin packing algorithms". [14]
In his thesis, Johnson already ponders about the trade-off between computational cost and optimality of the results. Johnson describes the following situation for a carpenter trying to minimise the number of planks used for an order, stating that "your carpenter's common sense is probably correct when it tells you to forget about figuring out the absolute minimum number of boards required, and to just use some simple heuristics that intuitively would seem to be of help in keeping the wastage down.". (D. Johnson, 1973, p. 9)

This situation was described in 1973. However, in the current world of 2022, we can paint a widely different picture from the perspective of an international shipping and delivery company. Let us imagine a generic logistics company, tasked with moving items from warehouses to customers all around the globe. They are dealing with a large number of items, which need to be distributed among boxes. These boxes are then placed in containers of varying sizes, before being placed on a number of ships. In this scenario, the ships form the top-level bins, while boxes and containers are intermediate bins. To maximise profits, the company will want to minimise the total cost of each step, which means finding an optimal distribution of items. A company of sufficient scale might benefit from finding the optimal solution to this instance of the Multi-Level Bin Packing problem, if the cost of wasting space at any stage is high enough.

With the ever-increasing performance of modern computers, the cost of finding optimal solutions for complex problems decreases. Meanwhile, the cost of sub-optimal solutions increases as resources become more scarce. The question that remains is where the balance lies between the two costs. When does the increased optimality start to outweigh the time cost? To answer this question, we gather performance results for calculating optimal and approximate solutions for MLBP instances of different sizes. The algorithms for calculating both the optimal and approximate solutions will be implemented within the provided C++ Codebase, using IBM ILOG CPLEX Optimization Studio for finding optimal solutions. The heuristic-based algorithms will be developed from scratch within the same codebase to maintain a fair comparison.
Before implementation starts, research is done on the existing approximation and optimal-solution algorithms for the Bin Packing problem, to ensure the implemented algorithms form an accurate representation of the performance for their respective approaches.

4 Related Research

Research on the Multi-Level Bin Packing problem is almost completely absent from the literature. The most promising literature is by Chen et al. [4], investigating dynamic programming and fuzzy-matching algorithms for the Multi-Level Bin Packing problems where the properties of the entities must be estimated from historical records. As a result of this added uncertainty factor, the focus of this research was on optimising the balance between optimality and consistency in suggesting feasible assignments. This differs from the focus of this research, which is more concerned with computational cost.
Hao et al. [11] studied the practical application of the Multi-Level Lot-Sizing and Bin Packing problem for manufacturing composite aeronautic products. IBM ILOG CPLEX Optimization Studio was used to solve formulations of the integrated bin packing and lot-sizing problem, the results of which were used to quantify the solution quality and computational efficiency of their proposed heuristic algorithm. This research shows the power and potential of high-efficiency dynamic programming algorithms for finding near-optimal solutions for NP-Hard optimisation problems.
Other noteworthy research on multi-level variants of optimisation problems is found in a series of research papers investigating the Multi-Level Network Optimisation problem (MLNO) in the late 90s by Cruz et al. [5–7]. In this research, a Multi-Level Network Flow problem is formulated as a mixed-integer programming model. The Branch-and-Bound algorithm is then used to find optimal solutions.

Other variants of the Bin Packing problem have seen more attention in the literature, like the Two-Dimensional Bin Packing problem. Baker et al. introduce the problem as "problems of packing an arbitrary collection of rectangular pieces into an open-ended, rectangular bin so as to minimize the height achieved by any piece.", and later present a $5/4$-approximation algorithm for the problem. [2, 3]
Once again the need for approximation algorithms highlighted, as the inherent complexity of the problem makes finding optimal solutions computationally expensive. Similar situations arise in the research by Coffman et al. on the Dynamic Bin Packing problem and by Friesen et al. on the Variable-Sized Bin Packing problem. [8, 10] Following this trend, it would only be reasonable to assume that a heuristic-based method would be the most promising approach for solving the Multi-Level Bin Packing problem as well.

5 Experimental Setup

5.1 Approximate Solutions

Two of the earliest approximation algorithms that were developed for BP are the First Fit and Best Fit algorithms. They both use a simple on-line heuristic to determine in which bin an item is placed once it is encountered. We assume that bins are sorted by an index, and that items are processed in the order in which they are received.
For First Fit, the heuristic says that an item should be placed in the bin with the lowest index with sufficient remaining capacity to fit the item. For the Best Fit algorithm, we check all bins with sufficient capacity and choose the one with the lowest remaining capacity. In the case of a tied capacity, the bin with the lowest index is used for simplicity. Despite the additional complexity of considering all possible assignments, the worst-case performance of Best Fit for BP is equivalent to that of First Fit, as shown by Johnson. [14, 15]

Algorithm 1 First Fit Algorithm for MLBP

1: Initialise list A with $\mathcal{I}$
2: Initialise list S of size $|\mathcal{B}|$.
3: $total\_cost \leftarrow 0$
4: **for all** $k \in M$ **do**
5: **for all** $\mathcal{B}_i^{k-1} \in \{\mathcal{B}^{k-1} \cap A\}$ **do**
6: **for all** $\mathcal{B}_j^k \in \mathcal{B}^k$ **do**
7: **if** $w(\mathcal{B}_j^k) - s(\mathcal{B}_i^{k-1}) \geq 0$ **then**
8: $w(\mathcal{B}_j^k) \leftarrow w(\mathcal{B}_j^k) - s(\mathcal{B}_i^{k-1})$
9: Add $(\mathcal{B}_i^{k-1} \rightarrow \mathcal{B}_j^k)$ to S
10: Add $\mathcal{B}_j^k$ to A
11: **break**
12: **end if**
13: **end for**
14: **end for**
15: **for all** $\mathcal{B}_j^k \in A$ **do**
16: $total\_cost \leftarrow total\_cost + c(\mathcal{B}_j^k)$
17: **end for**
18: **end for**
19: **return** heuristic solution S and objective value $total\_cost$

The aforementioned First Fit and Best Fit heuristics can be applied to MLBP with minimal adjustments. We make use of the values defined by the instance. Namely:

- The number of bin-levels m and the index set $\mathrm{M} = \{1, 2, \ldots, m\}$.
- The set of items $\mathcal{I} = \{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_{n^0}\}$ and the set of bins $\mathcal{B} = \mathcal{I} \cup \{\mathcal{B}_j^k \in \mathcal{B}^k \mid k \in \mathrm{M}\}$.
- The bin properties size $s(\mathcal{B}_j^k) \in \mathbb{N}_{>0}$, capacity $w(\mathcal{B}_j^k) \in \mathbb{N}_{>0}$, and cost $c(\mathcal{B}_j^k) \in \mathbb{N}_{>0}$ for all bins $\mathcal{B}_j^k \in \mathcal{B}^k$ at all levels $k \in \mathrm{M}$. The size property $s(\mathcal{B}_j^k)$ is also defined for level $k = 0$, meaning items.

The set of items is added to the set of bins at index $k = 0$ to generalise over the distinction between the two entity types, which simplifies the code. Items do not have a capacity or cost property attached to them, which means we need to ensure these properties are only called on level $k \in \mathrm{M}$, and not $k - 1$, since the index set M starts at index 1.

The First Fit algorithm for MLBP is shown in Algorithm 1. List A represents the active bins, and is initialised with the item set since items are always considered active. List S contains, for each level $1 \leq k \leq m$, the entity assignments from level $k - 1$ to level k. An assignment is represented by $(\mathcal{B}_i^{k-1} \rightarrow \mathcal{B}_j^k)$, meaning entity i at level $k-1$ is assigned to bin j at level k. For each level k, the size of every active bin at level $k-1$ is checked against the remaining capacity of bins $\mathcal{B}^k$ (line 7). If there is sufficient capacity in a bin, the assignment $(\mathcal{B}_i^{k-1} \rightarrow \mathcal{B}_j^k)$ is added to S, the remaining capacity of the bin is adjusted, and the bin is added to A to denote it being in use. The iteration is then terminated and the next bin from $\mathcal{B}^{k-1}$ is processed (line 5).

Algorithm 2 Best Fit Algorithm for MLBP

1: Initialise list A with $\mathcal{I}$
2: Initialise list S of size $|\mathcal{B}|$.
3: $total\_cost \leftarrow 0$
4: **for all** $k \in M$ **do**
5: **for all** $\mathcal{B}_i^{k-1} \in \{\mathcal{B}^{k-1} \cap A\}$ **do**
6: Initialise list R of size $|\mathcal{B}^k|$
7: **for all** $\mathcal{B}_j^k \in \mathcal{B}^k$ **do**
8: **if** $w(\mathcal{B}_j^k) - s(\mathcal{B}_i^{k-1}) \geq 0$ **then**
9: Add $w(\mathcal{B}_j^k) - s(\mathcal{B}_i^{k-1})$ to R
10: **end if**
11: **end for**
12: $min \leftarrow$ index of min(R)
13: $w(\mathcal{B}_{min}^k) \leftarrow w(\mathcal{B}_{min}^k) - s(\mathcal{B}_i^{k-1})$
14: Add $(\mathcal{B}_i^{k-1} \rightarrow \mathcal{B}_{min}^k)$ to S
15: Add $\mathcal{B}_{min}^k$ to A
16: **end for**
17: **for all** $\mathcal{B}_j^k \in A$ **do**
18: $total\_cost \leftarrow total\_cost + c(\mathcal{B}_j^k)$
19: **end for**
20: **end for**
21: **return** heuristic solution S and objective value $total\_cost$

The Best Fit algorithm is slightly more complex, because it keeps track of a set of values to find the assignment with the tightest fit. The Best Fit algorithm for MLBP is shown in Algorithm 2. The BF algorithm diverts from the previously discussed First Fit algorithm at line 9, where the remaining capacity is added to list R. Since we are looking for the tightest fit, we find the index of the assignment with the smallest remaining capacity in R (line 12). Updating the remaining capacity of $\mathcal{B}_{min}^k$, and the bookkeeping of sets A and S, is done in the same way as for FF. The objective value is calculated as the total cost of all active bins of levels $1 \leq k \leq m$, which is then returned alongside the assignment set S, representing the heuristic solution. These steps are identical to the ones in the FF algorithm.

The performance of the algorithms can be improved by pre-processing the input. By sorting the entities in non-increasing order of size before they are assigned to a bin, the heuristic can find a more optimal assignment. This can be done for both algorithms, by sorting the entities at level $k-1$ before iterating over them (line 5 for both algorithms). If pre-processing is applied, the algorithms are considered to be off-line algorithms, as they are able to scan the entire input before the processing step takes place.
Similar pre-processing steps can be applied to the bins, since bins have varying properties in MLBP. We gather results for sorting bins based on two properties. One method will rely on sorting the bins in non-increasing order of capacity, such that the largest bins are used first. The second method relies on the cost property, sorted in non-decreasing order, such that the cheapest bins are used first. Since capacity and cost are generally linked, sorting the bins in descending order of capacity implies descending order of cost, and vice versa.

5.2 Optimal Solutions

Polynomial-time approximation algorithms can never guarantee an optimal result. If they did, that would prove N=NP, which would be quite spectacular to say the least. Instead we make use of the optimisation techniques for mathematical programming to find optimal solutions with an optimised average time cost. To create a reduction from MLBP to IP we must define Decision Variables, Constraints, and an Objective Function. The Instance remains unchanged, since it already contains all the necessary information.

Decision variables:

- $x_{ijk} \in \{0, 1\}$, $\forall k \in \mathrm{M}$, $\forall i \in \mathcal{B}^{k-1}$
  One ($x_{ijk} = 1$) if entity i at level $k-1$ is inserted into bin j at level k, zero otherwise.
- $y_{jk} \in \{0, 1\}$, $\forall k \in \mathrm{M}$, $\forall j \in \mathcal{B}^{k}$
  One ($y_{jk} = 1$) if bin j at level k is active, zero otherwise.

Objective function:

- $\min \Sigma_{k \in \mathrm{M}} \Sigma_{j \in \mathcal{B}^k} \; y_{jk} \; c(\mathcal{B}_j^k)$

Constraints:

- $\Sigma_{j \in \{\mathrm{A} \cap \mathcal{B}^k\}} x_{ijk} = 1$, $\forall k \in \mathrm{M}$, $\forall i \in \mathcal{B}^{k-1}$
  Each active entity at level $k-1$ must be inserted into exactly one bin at level k.
- $\Sigma_{i \in \mathcal{B}^{k-1}} (x_{ijk} \; s(\mathcal{B}_i^{k-1})) \leq y_{jk} \; w_{jk}$, $\forall k \in \mathrm{M}$, $\forall j \in \mathcal{B}^{k}$
  The total size of all items in a bin j must not exceed the capacity of that bin.

By adding additional constraints, we can limit the solution space and push the solver to choose more appropriate optimisation strategies. To test the applicability of this, we formulate our problem as a network flow problem. The goal is to have the performance gain from the additional flow constraints outweigh the cost of the additional decision variables for some instances.

Additional Decision Variables:

- $f_{ijk} \in \mathbb{N}_{>0}$, $\forall k \in \mathrm{M}$, $\forall i \in \mathcal{B}^{k-1}$
  The amount of flow ($= f_{ijk}$) between entity i at level $k-1$ and bin j at level k.
  Represents the number of items entering a bin, with each item producing 1 flow.

Additional Constraints:

- $\Sigma_{j \in \mathcal{B}^k} f_{ijk} = 1$, $\forall i \in \mathcal{I}$, where $k = 1$
  Every item i must send out exactly one flow to a bin j at level 1.
- $\Sigma_{p \in \mathcal{B}^{k-1}} f_{pjk} = \Sigma_{q \in \mathcal{B}^{k+1}} f_{jq(k+1)}$, for all $2 \leq k \leq (m-1)$, $\forall j \in \mathcal{B}^k$
  The total amount of flow from all bins p at level $k-1$ entering a bin j at level k must be equal to the amount of flow bin j sends out to all bins q at level $k+1$.
- $\Sigma_{j \in \mathcal{B}^k} \Sigma_{i \in \mathcal{B}^{k-1}} f_{ijk} = |\mathcal{I}|$, where $k = m$
  The total amount of flow from all entities i at level $m-1$ entering top-level bins $\mathcal{B}_j^m$ must be equal to the amount of produced flow $|\mathcal{I}|$.

The network flow representation and the standard formulation both solve the same problem, despite the additional decision variables and constraints. Additional constraints can also be added in the form of objective function penalties. This creates a new problem, since the optimal solution is different from the one found for standard MLBP. The effects of this are explored by adding Fragmentation Constraints to the standard MLBP formulation.

We gain access to a number of additional values defined by the instance. Namely:

- The penalty value p.
- The number of groups q and the index set $\mathrm{G} = \{1, 2, \ldots, q\}$.
- The item property group $g(\mathcal{I}_i) \in \mathrm{G}$ for all items $\mathcal{I}_i \in \mathcal{I}$.

The new objective function will be defined as follows:

- $\min \Sigma_{k \in \mathrm{M}} \Sigma_{j \in \mathcal{B}^k} \; y_{jk} \; c(\mathcal{B}_j^k) + \Sigma_{r \in \mathrm{G}} \Sigma_{j \in \mathcal{B}^m} \; z_{rjm} \; p$

The objective function adds the penalty value p to the objective function for every top-level bin $\mathcal{B}_j^m$ that contains one or more items of group $r \in \mathrm{G}$. The objective function relies on a decision variable z to determine how many bins contain items from a group r.
The additional Decision Variable z is defined as follows:

- $z_{rjk} \in \{0, 1\}$, $\forall r \in \mathrm{G}$, $\forall k \in \mathrm{M}$, $\forall j \in \mathcal{B}^k$
  One ($z_{rjk} = 1$) if bin j at level k contains an item from group r, zero otherwise.

Decision variable z is not subject to any additional constraints, but is merely linked to x through the following constraints.

- $x_{ijk} \leq z_{rjk}$, $\forall i \in \mathcal{I}$, $\forall j \in \mathcal{B}^k$, where $k = 1$ and $r = g(\mathcal{I}_i)$
  If item i is assigned to bin j at level $k = 1$, then bin j contains an item from group r.
- $z_{ri(k-1)} \leq z_{rjk} + M(1 - x_{ijk})$, $\forall r \in \mathrm{G}$, $\forall j \in \mathcal{B}^k$, where $2 \leq k \leq m$ and $M = 1$
  If an item of group r is assigned to bin i at level $k - 1$, and bin i is assigned to bin j at level k, then bin j contains an item from group r.
  The constraint is deactivated by the big-M component when $x_{ijk} = 0$.

5.3 Results

Results were gathered over a set of 500 instances of 50 instance classes. Instances were generated using the provided Python file for all combinations of $m = \{1, 2, 3, 4, 5\}$ levels and $n = \{10, 15, 20, 25, 30, 35, 40, 45, 50, 100\}$ items. The following rules apply to these instances:

- If an entity i cannot be assigned to any bin after applying the First Fit heuristic to a random shuffle of the entities in layer $k - 1$, a new bin is added to layer k with a capacity from the discrete uniform distribution $\mathcal{U}\{s(\mathcal{B}_i^{k-1}), 1.9s(\mathcal{B}_i^{k-1})\}$, meaning between 1 and 1.9 times the size of the item that could not be assigned.
- The size of a bin j at level k is randomly sampled from the uniform distribution $\mathcal{U}\{w(\mathcal{B}_j^k), 1.5w(\mathcal{B}_j^k)\}$, meaning between 1 and 1.5 times the capacity of the bin.
- The cost of a bin is equal to the square root of its size, multiplied by 100.

The set of all test instances, as well as the scripts for generating new instances, have been made available on the project repository.[1] The algorithms were implemented using GCC 10.2.0. All tests were done on a single core of an Intel Core i7-8750H with 2.21 GHz with a CPU-time limit of 900s. The Integer Programming formulations were solved using CPLEX 22.1. Test results were gathered and aggregated through a Python script. This code has also been made available on the repository.

Table 1 lists the aggregated results for the on-line and off-line algorithms of First Fit and Best Fit, as well as the optimal solutions found by ILP/CPLEX, for each instance class as returned by the aforementioned Python script. The results are averaged over 10 instances per instance class. The obj columns show the achieved objective value. Columns %-gap list the average optimality gap between the achieved objective value and the optimal objective value as determined by CPLEX. The CPU time of the heuristic-based algorithms was less than 0.1s for all instances, and as such was deemed negligible for the purposes of this research. The time columns for ILP/CPLEX present the amount of CPU time needed for finding the optimal solution. CPU times are averaged over the instances that were completed within the time limit, or not mentioned if less than 8 instances could be solved within the time limit of 900s seconds.

Roughly 40% of the solutions produced by the on-line First Fit algorithm were infeasible. For Best Fit, this percentage is only 20%. The standard Bin Packing problem uses an endless supply of bins, all with the same capacity, while the Multi-Level Bin Packing problem has a constrained number of bins. This means that if a packing is really sub-optimal, the required number of bins can exceed the available number of bins. In this case, the algorithm does not assign these entities to any bin, which makes the final solution infeasible. Because these algorithms cannot guarantee a feasible solution, they fall outside of the definition of an approximation algorithm as defined by Ausiello. Instead, they are referred to as heuristic-based algorithms in this paper.
By pre-processing the entities based on size, the number of infeasible solutions is reduced to less than 1% for both algorithms. It is clear from these results that pre-processing the input leads to a large performance improvement.

The results of applying the additional pre-processing rules are have been omitted from the results, but a complete list of results can be found on the repository. The results will be discussed briefly here. For First Fit, both pre-processing methods lead to performance loss. This indicates that First Fit benefits from a combination of high and low capacity bins, without any formal ordering. When sorting the bins by capacity, the number of infeasible solutions even rises to roughly 35%. Applying either of the pre-processing methods to Best Fit shows small improvements for most instances.

For ILP/CPLEX, the results are as expected; The time cost for finding the optimal solution quickly increases with the size of the instances. ILP/CPLEX was unable to complete any of the instances with 100 items for $m > 1$. These results are even worse for the Network Flow formulation of the problem. Through different optimisations, these results could still be improved, however, as there are many intricacies to the way CPLEX solves problems which were not able to be fully understood within the timeframe of this research.

[1] `https://github.com/jgroenheide/mlbp-research.git`

Table 1: Average results for MLBP

		First Fit				Best Fit				ILP/CPLEX		
		On-Line		Off-Line		On-Line		Off-Line		Standard		NF
n	m	obj	%-gap	obj	%-gap	obj	%-gap	obj	%-gap	obj	time	time
10	1	3141.4	12.89%	3196.1	14.85%	3527.1	26.75%	3258.1	17.08%	2782.8	<0.1	<0.1
15	1	4394.5	18.73%	4261.3	15.14%	4606.5	24.46%	4278.5	15.60%	3701.1	<0.1	<0.1
20	1	6057.4	22.83%	5802.7	17.66%	6271.1	27.16%	5909.9	19.84%	4931.6	0.24	0.25
25	1	6961.8	21.48%	6645.0	15.96%	7047.5	22.98%	6738.5	17.59%	5730.6	0.37	0.36
30	1	9047.6	20.10%	8368.5	11.09%	9343.3	24.03%	8594.7	14.09%	7533.2	0.64	0.67
35	1	9838.6	15.26%	9435.9	10.54%	10101.9	18.35%	9667.1	13.25%	8535.9	0.70	0.70
40	1	11265.4	15.05%	10780.8	10.10%	11503.5	17.48%	10905.6	11.37%	9791.8	1.55	1.72
45	1	12745.2	16.48%	12216.4	11.65%	12648.9	15.60%	12272.9	12.17%	10941.6	1.79	1.81
50	1	13322.9	12.37%	12845.2	8.34%	13368.6	12.75%	13158.7	10.98%	11856.6	1.79	1.84
100	1	26578.4	9.82%	25544.8	5.55%	26492.0	9.47%	25867.2	6.88%	24201.0	18.15	15.69
10	2	6945.9	27.73%	6653.8	22.36%	7201.8	32.43%	6760.3	24.31%	5438.1	<0.1	<0.1
15	2	9242.1	34.63%	8875.8	29.29%	10098.2	47.10%	9420.4	37.23%	6864.8	0.28	0.47
20	2	13042.8	34.12%	12189.6	25.34%	13882.7	42.75%	12439.2	27.91%	9724.9	0.99	1.38
25	2	15280.7	30.02%	14571.9	23.99%	15877.8	35.10%	14921.2	26.96%	11752.6	5.34	12.39
30	2	17813.6	33.59%	16238.8	21.78%	18410.3	38.07%	16956.1	27.16%	13334.4	5.55	8.32
35	2	20120.8	26.39%	19090.5	19.92%	20879.0	31.15%	19135.5	20.20%	15919.4	22.34	43.31
40	2	22558.5	26.47%	21213.8	18.93%	23442.7	31.43%	21651.3	21.39%	17836.8	27.07	54.62
45	2	25404.5	27.83%	23581.6	18.66%	26320.1	32.44%	23641.9	18.97%	19872.9	40.96	65.91
50	2	28491.2	25.13%	26759.6	17.53%	29592.5	29.97%	27075.5	18.92%	22768.5	61.78	139.48
100	2	53553.0	-	49904.7	-	53621.1	-	50500.6	-	-	-	-
10	3	11670.4	43.38%	11604.5	42.57%	12941.9	59.00%	12181.9	49.66%	8139.6	<0.1	<0.1
15	3	16457.4	53.42%	15420.0	43.75%	17705.9	65.06%	15827.1	47.55%	10726.8	0.64	1.09
20	3	18314.8	40.68%	17581.9	35.05%	20206.7	55.22%	18259.7	40.26%	13018.5	0.89	1.33
25	3	24113.4	49.38%	22133.9	37.12%	25587.5	58.52%	22433.0	38.97%	16142.0	6.61	19.13
30	3	27802.6	46.86%	24699.3	30.47%	29489.9	55.77%	26861.7	41.89%	18931.6	18.94	32.31
35	3	31011.7	42.26%	28545.5	30.95%	32276.5	48.06%	28890.4	32.53%	21798.9	69.69	85.58
40	3	36803.0	44.34%	33718.7	32.24%	39091.7	53.32%	34588.6	35.66%	25497.3	84.87	152.32
45	3	39279.9	40.81%	36441.7	30.64%	40361.0	44.69%	36435.4	30.61%	27895.5	120.05	-
50	3	41900.0	37.19%	38551.1	26.22%	43491.6	42.40%	39792.9	30.29%	30542.3	180.46	-
100	3	79350.5	-	72527.7	-	80785.7	-	74212.7	-	-	-	-
10	4	16293.8	55.50%	16016.1	52.85%	17641.6	68.36%	15748.8	50.30%	10478.5	<0.1	<0.1
15	4	22502.0	60.99%	22647.4	62.03%	25313.5	81.10%	22741.9	62.71%	13977.3	0.39	0.46
20	4	26852.9	57.12%	26106.6	52.76%	31335.0	83.35%	26433.8	54.67%	17090.2	2.27	3.93
25	4	34334.1	68.93%	31808.1	56.50%	36279.1	78.50%	31922.7	57.07%	20324.3	9.72	11.26
30	4	39653.7	61.20%	36255.4	47.39%	44103.6	79.29%	38035.5	54.63%	24598.4	56.49	65.17
35	4	41687.7	48.53%	38961.7	38.82%	47426.6	68.98%	40762.4	45.24%	28066.5	61.52	86.85
40	4	47006.2	52.65%	43391.4	40.91%	51308.1	66.62%	45438.0	47.56%	30793.0	142.50	-
45	4	49858.3	50.55%	44912.1	35.61%	52877.0	59.66%	46974.2	41.84%	33118.2	178.24	-
50	4	56774.5	53.76%	51269.9	38.86%	61721.4	67.16%	52928.0	43.35%	36923.2	197.16	-
100	4	112813.7	-	99763.9	-	114925.4	-	101871.6	-	-	-	-
10	5	21963.2	81.25%	20593.1	69.94%	24395.4	101.32%	21594.9	78.21%	12117.7	<0.1	<0.1
15	5	31425.2	94.74%	29270.7	81.39%	37872.1	134.69%	29420.1	82.32%	16136.8	1.01	1.43
20	5	36667.5	80.52%	31303.1	54.11%	41163.6	102.66%	34586.6	70.28%	20311.6	1.76	2.48
25	5	39058.8	78.17%	36096.0	64.65%	44463.3	102.82%	38738.6	76.71%	21922.7	5.02	6.36
30	5	48954.9	75.52%	45574.8	63.40%	56279.8	101.78%	45763.9	64.08%	27891.1	48.78	37.11
35	5	58800.6	81.62%	50442.3	55.81%	64215.0	98.35%	55320.7	70.88%	32374.8	121.18	145.24
40	5	61794.6	75.12%	53993.0	53.01%	66230.2	87.69%	56696.6	60.67%	35287.6	117.92	-
45	5	74254.2	79.58%	63919.5	54.58%	77061.8	86.37%	65950.6	59.50%	41349.4	211.85	-
50	5	75647.3	-	65861.9	-	82303.5	-	69866.3	-	-	-	-
100	5	145898.5	-	126962.0	-	149414.8	-	127387.0	-	-	-	-

Table 2: Average results of ILP/CPLEX for MLBPFC

			20% groups		40% groups		60% groups	
p	n	m	obj	time	obj	time	obj	time
40	5	1	1469.4	<0.1	1523.8	<0.1	1490.7	<0.1
40	10	1	2925.0	<0.1	2595.0	<0.1	2756.1	<0.1
40	15	1	4029.1	0.52	3920.9	0.89	4290.4	1.82
40	20	1	5199.3	3.26	5394.8	4.45	5387.2	2.75
40	5	2	2670.3	<0.1	3202.8	<0.1	3161.6	<0.1
40	10	2	5631.3	0.21	4899.6	0.24	5288.3	0.51
40	15	2	7316.0	7.59	7453.7	17.04	7394.5	44.22
40	20	2	10364.3	135.51	-	-	-	-
40	5	3	4245.7	<0.1	4936.6	<0.1	4938.8	<0.1
40	10	3	7837.4	0.72	8060.4	0.73	8452.4	1.89
40	15	3	10375.9	10.28	10489.6	51.74	10154.8	42.59
40	20	3	-	-	-	-	-	-
120	5	1	1935.0	<0.1	1781.1	<0.1	1822.8	<0.1
120	10	1	3099.5	<0.1	3110.9	<0.1	3587.9	<0.1
120	15	1	4672.0	2.05	4738.8	0.55	5042.7	0.44
120	20	1	6102.9	9.25	6296.1	4.17	6274.9	3.89
120	5	2	3279.5	<0.1	2952.5	<0.1	3262.5	<0.1
120	10	2	6165.7	0.69	5833.3	0.45	5840.9	0.69
120	15	2	8168.9	64.54	7776.5	117.57	8202.6	147.12
120	20	2	-	-	-	-	-	-
120	5	3	4285.1	<0.1	5396.7	<0.1	4706.4	<0.1
120	10	3	7729.8	0.63	7415.8	0.86	7812.1	0.97
120	15	3	11047.0	64.64	10664.4	77.66	-	-
120	20	3	-	-	-	-	-	-

Table 2 shows the results of applying ILP/CPLEX to the Multi-Level Bin Packing problem with Fragmentation Constraints. The results have been split into 3 categories, based on the value q that was used to calculate the maximum number of groups as a percentage of the number of item n. The results present a few interesting insights. Most obviously, the additional complexity of the Fragmentation Constraints means the size of instances that can be solved decreases drastically. We also see that the number of items is a much larger factor in the time cost, with the addition of groups. In many cases, CPLEX was unable to solve instances with 20 items within the time limit. The size of the penalty factor p and the group percentage q were less impactful than expected, however.

6 Responsible Research

Considering the ethical implications of a new piece of research is an important part of doing responsible research. It is the responsibility of the scientific community to protect the ethical and moral standards of the industry by actively working to avoid fraud and other forms of misconduct.
For this particular research, which is founded on a highly abstract field of science, foreseeing potential negative implications is a tricky endeavour. The research does not use a data set with potential for bias, and was performed by independent researchers from the TU Delft. The results of this research can, however, influence future research. Therefore, it is highly important that the research and results discussed in this paper are clear, true, and reproducible.
This research was performed in collaboration with a group of peers. As such, each member was held accountable by the research group, as well as the supervisors, to ensure the results are correct and reproducible. All code that was written for this project has been made available on a public repository, and has been explained in the paper to ensure results can be verified. The full set of results has also been made available there for further verification.

7 Conclusions and Recommendations

In this paper we introduced the Multi-Level variant of the Bin Packing problem, where items must be packed into top-level bins through a number of intermediate packing layers. The problem was introduced alongside the concept of heuristic-based approximation algorithms. The Best Fit and First Fit heuristics were used to gather results on the performance of heuristic-based algorithms for MLBP. These results were then compared to the objective value of the optimal solution produced by applying optimisation software to an Integer Linear Programming formulation of the problem. In doing so, we have shown the potential for using optimisation software like CPLEX to solve Integer Programming formulations of the Multi-Level Bin Packing problem. The time cost for finding optimal solutions using ILP/CPLEX is quite large, especially when compared against the negligible time cost of the heuristic-based approach.
Additionally, we showed that simple heuristic-based algorithms can produce solutions within 25% of the optimal solution for small instances, and within 50% of medium instances. For larger instances the optimality gap grows rapidly, however. Lastly, we showed the impact of additional complexity in the form of penalty values on the performance of the optimisation software. As expected, the time cost for finding optimal solutions increases with the complexity of the problem, leading to an even greater need for sophisticated approximation algorithms.

Future research will focus on further optimising both the optimal and approximate solution methods, to ultimately present a better comparison of what the two approaches are capable of. Particularly for the approximation algorithms, more advanced heuristics can be developed. In its current state, there does not appear to be a place for optimal solution solving algorithms outside of a few dedicated applications and areas of research, as the time cost for solving realistically sized instances is still too large. However, it is hard to imagine these problems not finding purpose in the future, when taking into account how widely applicable the problem is to real-life scenarios.

References

[1] Giorgio Ausiello, Vincenzo Bonifaci, and Bruno Escoffier. Complexity and approximation in reoptimization, 2 2011.

[2] Brenda S Baker, Donna J Brown, and Howard P Katseff. A 5/4 algorithm for two-dimensional packing. *Journal of Algorithms*, 2:348–368, 12 1981.

[3] Brenda S Baker, E G Coffman, and Ronald L Rivest. Orthogonal packings in two dimensions. *SIAM J. COMPUT*, 9, 1980.

[4] Lei Chen, Xialiang Tong, Mingxuan Yuan, and Jia Zeng. A data-driven approach for multi-level packing problems in manufacturing industry. pages 1762–1770. ACM, 7 2019.

[5] F. R.B. Cruz, G. R. Mateus, and J. MacGregor Smith. A branch-and-bound algorithm to solve a multi-level network optimization problem. *Journal of Mathematical Modelling and Algorithms 2003 2:1*, 2:37–56, 2003.

[6] F.R.B. Cruz, J.MacGregor Smith, and G.R. Mateus. Solving to optimality the uncapacitated fixed-charge network flow problem. *Computers Operations Research*, 25:67–81, 1 1998.

[7] F.R.B. Cruz, J.MacGregor Smith, and G.R. Mateus. Algorithms for a multi-level network optimization problem. *European Journal of Operational Research*, 118:164–180, 10 1999.

[8] Jr. E. G. Coffman, M. R. Garey, and D. S. Johnson. Dynamic bin packing. *SIAM Journal on Computing*, 12:227–258, 5 1983.

[9] Lance Fortnow and Steven Homer. A short history of computational complexity. *Bulletin of the EATC*, 80:95–133, 2003.

[10] D. K. Friesen and M. A. Langston. Variable sized bin packing. *SIAM Journal on Computing*, 15:222–230, 2 1986.

[11] Xinye Hao, Li Zheng, Na Li, and Canrong Zhang. Integrated bin packing and lot-sizing problem considering the configuration-dependent bin packing process. *European Journal of Operational Research*, 3 2022.

[12] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285, 5 1965.

[13] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3:299–325, 12 1974.

[14] David S Johnson. Near-optimal bin packing algorithms, 6 1973.

[15] David S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8:272–314, 6 1974.

[16] David S Johnson and Michael R Garey. A 71/60 theorem for bin packing. *Journal of Complexity*, 1:65–106, 10 1985.

[17] E. L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14:699–719, 8 1966.

[18] George L. Nemhauser and Laurence A. Wolsey. Integer and combinatorial optimization. page 763, 1988.

[19] J D Ullman. The performance of a memory allocation algorithm., princeton university, department of electrical engineering. *Computer Science Laboratory*, 47, 1971.

[20] H. Paul Williams. *Logic and Integer Programming*, volume 130. Springer US, 2009.