Integer Programming Models for the Class Constrained Multi-Level Bin Packing Problem

Agata Natalia Kordyl
Supervisor(s): Neil Yorke-Smith, Matthias Horn
EEMCS, Delft University of Technology, The Netherlands

June 19, 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

The Multi-Level Bin Packing problem is a generalisation of the widely-known NP-hard Bin Packing problem. This work aims to investigate approaching this problem, as well as a version including Class Constraints, through integer programming. By modelling the problems in two ways: standard, and with a network flow approach. The studying of the performance of these formulations on various instances, provides a greater insight into the possibilities of integer programming for this purpose. Furthermore, evaluating the two approaches gives an initial suggestion for developing effective models for solving the aforementioned problems.

1 Introduction

This report will detail the solving and exploration of the Multi-Level Bin Packing problem, as well as of a variation of it with added Class Constraints.

The Multi-Level Bin Packing (MLBP) problem is a generalisation of the Bin Packing (BP) problem which is a well-known NP-hard optimisation problem [2]. In the BP problem, a set of items and an infinite set of bins is considered. Each item is characterised by a size and all bins have the same maximum capacity. The challenge is to insert all the items into the smallest possible amount of bins such that no bin's capacity is exceeded. The MLBP considers the same problem but with multiple levels of bins, where, firstly items are packed into bins on the first level, then, bins of the first level must be inserted into those on the next level. This is repeated until all items (in bins) are packed into bins on the top-level. The bins in MLBP also have different capacities which need to be accounted for, as well as costs, where the total cost of used bins has to be minimised. As opposed to the standard BP problem, which has been extensively studied, the Multi-Level generalisation has been hardly considered in literature [1], which is why it is worth solving. This problem can describe multiple real-life scenarios, such as, in the field of logistics where products are packed into packets or boxes before they are loaded into containers for shipping.

The Class Constrained Multi-Level Bin Packing problem (CCMLBP) is a variation of the aforementioned MLBP, with the addition of a "Class" constraint, where each item belongs to some class and the number of classes allowed in a bin is bounded on each level. This problem, similarly to MLBP has not been explored in literature, however, it's BP counterpart has [4]. Among others, the Class Constrained Bin Packing problem is useful in the context of allocating resources in multimedia storage systems [12]. The challenge in this sub-problem is to find a packing that minimises the total cost of the used bins such that every bin at level $i \in M$ satisfies the capacity constraints and contains items from no more than Q^i classes.

This work aims to explore approaching the MLBP and CCMLBP problems using Integer Programming. This is a worthwhile contribution due to the many possible applications of these problems, and since popular problems such as BP can be reduced to them.

The next section of this paper will provide a formal description of the problem and the methodology of solving it. This will be followed by a study of related problems in the literature. Next will be discussed the experimental setup and results, followed by a section on responsible research. Final sections will be dedicated to discussion and a conclusion.

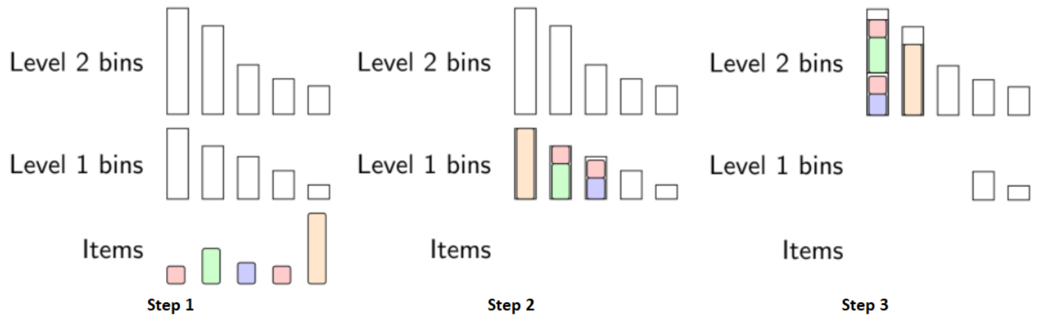

Figure 1: Visualisation of the MLBP problem being solved on an instance with 5 items and 2 levels.

2 Methodology

2.1 Formal Problem Description

The MLBP problem can be described as follows:

1. Given a set of items $B_0 = \{B_{0,1}, B_{0,2}, ..., B_{0,n^0}\}$ with size $s_{k,j} \in \mathbb{N}_{>0}$, $1 \leq j \leq n^0$.
2. Given a set of bins $B_i = \{B_{i,1}, B_{i,2}, \ldots, B_{i,n_0}\}$, $1 \leq i \leq m$, with m levels. Each bin having a size $s_{i,k} \in \mathbb{N}_{>0}$, a capacity $w_{i,k} \in \mathbb{N}_{>0}$, and cost $c_{i,k} \in \mathbb{N}_{>0}$, where $1 \leq k \leq n^i$, $1 \leq i \leq m$.

The task of this problem, as described in Section 1, is to first fit all the items into first level bins and then all used bins on level i, for $1 \leq i \leq m$, into a bin on level $i+1$ until the top level is reached. This must be accomplished while not surpassing any bin's capacity and minimising the overall cost of used bins. An illustration of a small instance of this problem can be seen in fig. 1

The CCMLBP problem has the same description as MLBP only with the addition of classes, which can be formalised as follows:

1. Each Item $B_{0,j} \in B_0$ belongs to a class $\kappa_j \in \{1, 2, ..., q\}$ where q is a positive integer.
2. Each level $i \in M$ is associated with a bound $Q^i \in \mathbb{N}_{>0}$ for the number of different classes within bin $B_{i,k} \in B_i$

The challenge in this sub-problem is to find a packing that minimises the total cost of the used bins such that every bin at level $i \in M$ satisfies the capacity constraints and contains items from no more than Q^i classes.

2.2 Method

These problems can be approached in a variety of ways, however due to the limitations of this project, this paper will focus on a method of mathematical optimization called Integer Programming (IP) to model the problems as. An Integer Program is a mathematical model which consists of an objective function which has to be minimized or maximized, and a finite number of constraints, both of which are "linear in a finite number of decision variables" [10, p.5]. In addition, as opposed to linear programming models, in integer programming models the decision variables are restricted to integer values.

Although IP problems are NP-hard, there exist powerful general purpose solvers that can solve a wide range of these problems in reasonable time. Therefore, it is a valuable method for modeling the problems at hand. Even when considering only IP, the MLBP and CCMLBP problems can be approached at different angles which will be explored in this work. The performance of these models w.r.t their efficacy in solving different instances of the MLBP and CCMLBP problems will be evaluated. For the implementation and testing of the Research Question, the C++ framework provied by the supervisor will be used. The CLPEX Optimization Studio will be used as the solver for the IP formulations.

2.3 Research Question

The following research question can be formulated based on these problems:

Which of the considered IP models of the MLBP and CCMLBP problems perform best?

In addition to finding the most effective models, other issues will be considered, such as:

1. How far can Integer Programming can be used to solve instances of the MLBP and CCMLBP problems?
2. How well does the CPLEX solver perform on these MLBP/CCMLBP formulations?
3. How large instances can be solved by CPLEX before having to resort to approximation algorithms?

3 Background

3.1 Related Work

As described in Section 1, the Bin Packing problem is a well-studied optimization problem, as opposed to it's Multi-Level generalisation. The literature that exists on the MLBP problem, namely [1], proposes a dynamic programming approach for normal-size instances and a heuristic multi-level fuzzy-matching algorithm for large instances, as opposed to the integer programming strategy found in this work.

There also exist multiple variations of the BP problem which incorporate some of the characteristics of MLBP and CCMLBP problems. One of them is the Variable-Size Bin Packing problem [5, 6], which is a generalised formulation of the BP problem, which considers a "finite collection of bin sizes and an inexhaustible supply of bins of each size" [5, p. 222]. This problem has been mainly tackled in the literature using approximation algorithms and heuristics. Another relevant problem is the Class Constrained Bin Packing (CCBP) problem, which considers the same scenario as the BP, however, each item belongs to some class and there are constraints as to the possible number of classes in each bin. This problem has been studied w.r.t both online and offline algorithms [4, 12], which were mostly approximation methods.

Studying the Multi-Level Bin Packing problem from an integer programming perspective can be a beneficient contribution to the existing literature because this approach has not yet been explored. In addition, not many similar works focus on exact algorithms and the extent of their usefulness. The problem is valuable because it is a generalisation of the BP and related problems. This means that these other problems can be formulated as the MLBP, and also new problems can be solved. Similarly, the Class Constrained MLBP is an even greater generalisation. The addition of layers can be helpful in expanding the possibilities of various applications. For instance, the CCBP problem is useful in the context of "data placement on video-on-demand servers" [12, p. 242], where a goal would be to store as many movies of one genre on one server as possible. In turn, the CCMLBP could be applied to a case where there are multiple layers to this storage system and the goal would be not to gave too many movies of different genres up to the top-level structure to make retrieval and searching more efficient.

3.2 Integer Programming

An Integer Program (IP) is a mathematical model which consists of am objective function which has to be minimized or maximized, a finite number of constraints, both of which are "linear in a finite number of decision variables" [10, p.5]. Integer Programs, as opposed to Linear Programs (LP), contain decision variables which are restricted to integer values. This small difference introduces a lot of complexity, as IP has been shown to be NP-complete [9], while one of the most widely used algorithms for solving LP, Simplex, is shown to usually perform in polynomial time [11]. A general form for an integer program is as follows:

$$\min(\mathbf{c}^\mathsf{T}\mathbf{x}) \tag{1}$$

$$s.t. \mathbf{A}\mathbf{x} \geq \mathbf{b} \tag{2}$$

$$\mathbf{x} \geq \mathbf{0} \tag{3}$$

$$\mathbf{x} \in \mathbb{R}^n \tag{4}$$

Although NP-hard, there exist many algorithms and solvers which utilise these algorithms for solving these problems in reasonable time. These include three main types: exact algorithms, approximation algorithms, and heuristic algorithms. In this work, we are interested in the extent of the performance of exact algorithms before having to employ approximation algorithms. The two main classes of exact algorithms for solving integer programs are the *Cutting Plane* methods and the *Branch-and-Bound* algorithms. A combination of these is called a *Branch-and-Cut* algorithm. [7].

The main idea of cutting plane methods is to solve the integer programming problem by solving a sequence of linear programming problems, This involves solving the LP relaxation of the problem. If the solution is integral then the problem is solved, otherwise a "cut" is performed. A "cut" means finding a linear constraint that excludes the LP solution but does not exclude any integer points. The cut constraint is added to the problem and the process is repeated until until an integral solution is found. To most efficiently find a solution defining constraints to help strengthen the LP relaxation is necessary.

On the other hand, the branch-and-bound algorithm is a tree-based search heuristic to aid in the search of the large solution space. This is done by keeping track of the upper and lower bounds of the optimal solution. Branches of the tree can be pruned to limit what needs to be searched. An advantage of the branch-and-bound algorithm is that it can be terminated early as long as at least one integral solution has been found, even though it may not be optimal. This method can also return multiple optimal solutions.

In reality many solvers, such as CPLEX, use a combination of these two methods to solve IP problems. A more detailed introduction to IP and various algorithms can be found in [8].

CPLEX Optimization Studio The CPLEX optimizer, which will be used in this work to solve instances of the MLBP and CCMLBP problems, is a powerful developed by IBM for solving a wide range of optimization problems including large linear programming problems, integer programming problems, and even quadratic programming problems. To solve integer programs, CPLEX uses a branch-and-bound algorithm coupled with "modern features like cutting planes and heuristics" [1]. Since CPLEX is such a powerful commercial solver with a number of features which are mostly a "black box", it is difficult to assess which heuristic or tactic was applied to each instance. This somewhat complicates the evaluation and comparison of different formulations.

4 Mathematical Model

This section will detail the two mathematical formulations for MLBP which were conceived. Since the Class constraints can be added interchangeably to both formulations, they will only be described once. The common instance variables used in all implementations are as follows:

1. The number of levels is m.
2. n_k is the number of items/bins on level k for all $k = 0, \dots, m$.
3. B_k is a set of all bins/items on level k for all $k = 0, \dots, m$.
4. $c_{k,j}$ is the cost for bin j on level k for all $k = 1, \dots, m$ and $j \in B_k$.
5. $w_{k,j}$ is the capacity for bin j on level k for all $k = 1, \dots, m$ and $j \in B_k$.
6. $s_{k,j}$ is the size of item/bin j on level k for all $k = 0, \dots, m$ and $j \in B_k$.

4.1 Standard Integer Programming MLBP Formulation

The following integer program (IP) models solutions to the MLBP problem in terms of two binary decision variables x and y. Where $x_{k,i,j} \in \{0, 1\}$ signifies if a bin/item i from level $k-1$ has been inserted into bin j on level k, in which case it is 1. While the variable $y_{k,j} \in \{0, 1\}$ signifies if bin j on level k was used or not.

$$\min \sum_{k=1}^{m} \sum_{j=0}^{n_k} y_{k,j} \cdot c_{k,j} \tag{5}$$

$$\sum_{j \in B_1} x_{1,i,j} = 1 \qquad i \in B_0 \tag{6}$$

$$\sum_{j \in B_k} x_{k,i,j} = y_{k-1,i} \qquad i \in B_{k-1}, k = 1, \dots, m \tag{7}$$

$$\sum_{i=1}^{n} x_{k,i,j} \cdot s_{k-1,i} \le y_{k,j} \cdot wk, j \qquad j \in B_k, k = 1, \dots, m \tag{8}$$

$$x_{k,i,j} \in \{0, 1\} \qquad j \in B_k,\ i \in B_{k-1}, k = 1, \dots, m \tag{9}$$

$$y_{k,j} \in \{0, 1\} \qquad j \in B_k, k = 1, \dots, m \tag{10}$$

Hereby, equation (6) ensures that on level 1, each item is packed into one bin, while equation (7) ensures that for all other levels, if a bin was used on level $k-1$, it must be inserted into a bin on level k. The capacity constraints are ensured by equation (8) by summing over the sizes $s_{k-1,i}$ of items, if they were added to bin j on level k, and limiting that total size to the capacity $w_{k,j}$ of bin j at level k. The objective function (11) aims to minimize the total cost C by accounting, for all bins, for all levels, for the cost $c_{k,j}$ of bin j on level k, if it was used.

[1] `https://documentation.aimms.com/platform/solvers/cplex.html` Accessed 9-06-2022

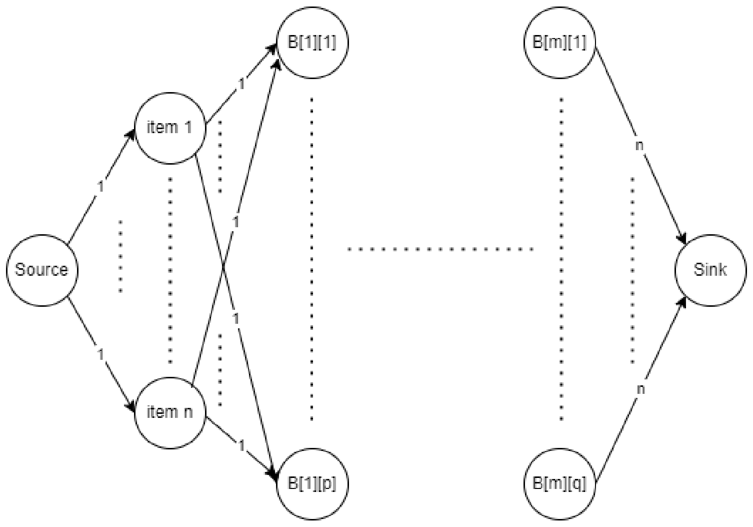

Figure 2: A generalised network flow representation of the MLBP problem.

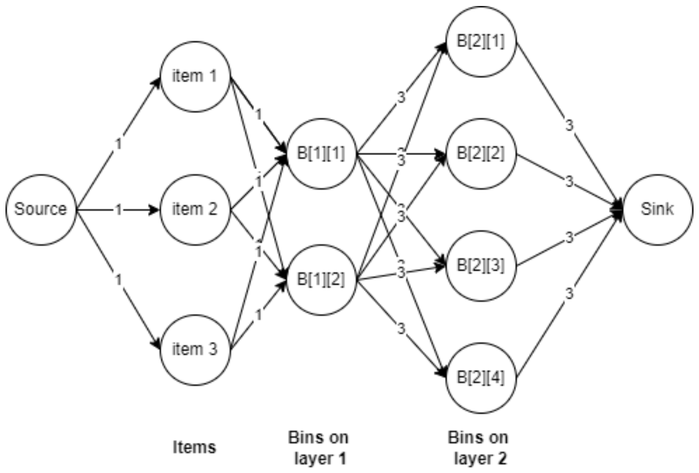

Figure 3: A representation of a small instance (3 items, 2 levels of bins) of the MLBP problem as a network flow problem.

4.2 MLBP Network Flow Formulation

The following integer program (IP) models solutions to the MLBP problem similarly to the first formulation (4.1), with the same decision variables and constraints for capacity, etc.

However, this formulation approaches the problem through a network flow lens. Here, items are nodes which are connected to a source which emits one unit of flow to each of them. Then, each item node is connected to each first level bin node with a connection of capacity of one unit of flow. After this, all bin nodes on level k are connected to all bin nodes on level $k+1$ with connections of capacity n_0 (number of items). On the last level, each node corresponding to a bin is connected to a sink with a connection of capacity n_0.

This representation of the MLBP problem can be seen in a generalised form in Fig.2. A visualisation of a small instance can be seen in Fig. 3. In this integer program, an additional integer variable $f_{k,i,j} \in \{0, n_0\}$ is present to signify the amount of 'flow' passing from item/bin i to bin j on level k .

$$\min \sum_{k=1}^{m} \sum_{j=0}^{n_k} y_{k,j} \cdot c_{k,j} \tag{11}$$

$$\sum_{j \in B_1} x_{1,i,j} = 1 \qquad i \in B_0 \tag{12}$$

$$\sum_{j \in B_k} x_{k,i,j} = y_{k-1,i} \qquad i \in B_{k-1}, k = 1, \dots, m \tag{13}$$

$$\sum_{i=1}^{n} x_{k,i,j} \cdot s_{k-1,i} \leq y_{k,j} \cdot wk, j \qquad j \in B_k, k = 1, \dots, m \tag{14}$$

$$\sum_{j \in B_1} f_{1,i,j} = 1 \qquad i \in B_0 \tag{15}$$

$$\sum_{p \in B_{k-1}} f_{k,p,j} = \sum_{q \in B_{k+1}} f_{k+1,j,q} \qquad k = 1, \dots, m-1 \tag{16}$$

$$\sum_{j \in B_m} \sum_{i \in B_{m-1}} f_{m,i,j} = n_0 \tag{17}$$

$$f_{k,i,j} \leq x_{k,i,j} \cdot n_0 \qquad k = 1, \dots, m, i \in B_{k-1}, j \in B_k \tag{18}$$

$$x_{k,i,j} \leq f_{k,i,j} \qquad k = 1, \dots, m, i \in B_{k-1}, j \in B_k \tag{19}$$

$$0 \leq f_{k,i,j} \leq \mathrm{n}_0 \qquad k = 1, \dots, m, i \in B_{k-1}, j \in B_k \tag{20}$$

$$x_{k,i,j} \in \{0, 1\} \qquad j \in B_k,\ i \in B_{k-1}, k = 1, \dots, m \tag{21}$$

$$y_{k,j} \in \{0, 1\} \qquad j \in B_k, k = 1, \dots, m \tag{22}$$

Hereby, equation (15) ensures that the flow leaving each item node is 1, which also ensures that the total flow in the system equals to the number of items n_0. The equation (16) guarantees that for each bin j on level k, the incoming flow from bins on level $k-1$ is equal to the outgoing flow to bins on level $k+1$. Finally, the equation (17) confirms that the total number of flow entering nodes on level m, and consequently entering the sink, is equal to the number of items n_0. Equations (18, 19) are meant to connect the x variable with the flow variable f so that the problem is correctly recognised by CPLEX. Equation (18) guarantees that for every item in every bin on each level, the flow must be less than the total number of items n_0. While equation (19) ensures that the flow from bin i on level $k-1$ to bin j on level k is greater than 0 if bin i was inserted into bin j.

Since this network flow representation does not account for capacity constraints it also needs the same variables and constraints as the previous program (4.1). The addition of the flow variable and other constraints is meant to strengthen the initial formulation, however this relies on speculation and a proof would be outside the scope of this work. However, some limited performance experiments can be seen in section 5.

4.3 CCMLBP Formulation

The following formulation can be added to both the classical IP (4.1) for MLBP and the modified flow formulation (4.2). In it, the class constraint is described using one additional binary decision variable, $c_{k,r,j} \in \{0,1\}$, which signifies if item(s) of class r are present in bin j on level k. In addition, κ_i signifies the class of item i, Q^k is the upper bound for number of classes allowed in one bin on level k. The following constraints are in addition considered for the CCMLBP problem:

$$x_{1,i,j} \leq c_{1,\kappa_i,j} \qquad j \in B_1, i \in B_0 \tag{23}$$

$$\text{IloIfThen}((x_{k,i,j} = 1 \wedge c_{k-1,r,i} = 1), c_{k,r,j} = 1) \quad k = 1, \ldots, m, r = 1, \ldots, q, j \in B_k, i \in B_{k-1} \tag{24}$$

$$\sum_{r=1}^{q} c_{k,q,j} \leq Q^k \cdot y_{k,j} \qquad k = 1, \ldots, m, j \in B_k \tag{25}$$

$$y_{k,j} \leq \sum_{r=1}^{q} c_{k,q,j} \qquad k = 1, \ldots, m, j \in B_k \tag{26}$$

$$c_{k,r,j} \in \{0,1\} \qquad r = 1, \ldots, q, k = 1, \ldots, m, j \in B_k \tag{27}$$

Hereby, the equation (23) ensures that on level 1, if item i was inserted into bin j, then the class of i must be in j. The statement (24) uses an ILOG constraint called $IloIfThen$ [2], which represents a conditional constraint between the two variables, essentially meaning $A \implies B$. This structure is similar to a 'Big M' constraint[3], so it is still a linear constraint. It ensures that on all other levels, if bin i is packed into bin j constraint on level k, and class r is present in i, then it is also present in j. Equation (25) ensures the class constraint itself, meaning that the amount of classes in bin j on level k is less than the upper bound Q^k for that level, if the bin is used. Finally, equation (26) ensures that if a bin is used, then there is at least one class in it.

5 Experimental Setup and Results

Having formulated the MLBP and CCMLBP in two different ways (Section 4) and implemented them. The performance of these formulations can be evaluated. This section will detail the setup of experiments performed and their results.

[2] `https://www.ibm.com/docs/en/icos/12.9.0?topic=c-iloifthen-1` Accessed 9-06-2022

[3] `https://www.ibm.com/support/pages/difference-between-using-indicator-constraints-and-big-m-formulation` Accessed 9-06-2022

formulation	no. runs	obj. mean	median CPU time	mean B&B nodes	ratio solved
MLBP	550	21222.737	4.2	10976.016	0.772
NFMLBP	550	21099.724	2.5	11117.514	0.658
CCMLBP	2200	13161.746	1.6	2835.116	0.362
NFCCMLBP	2200	18315.824	42.4	8596.704	0.425

Table 1: Summarised results for all four formulations.

5.1 Experiment Setup

To test the four formulations two sets of non-trivial random instances were generated. The instances were generated with a script provided by the supervisor. Firstly, for the standard Multi-Level Bin Packing (MLBP) problem, the set consists of 10 instances of each 'class', or combination of n items and m levels. Where $n \in \{5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 100\}$ and $m \in \{1, 2, 3, 4, 5\}$. While for the Class Constrained MLBP (CCMLBP), the same n and m were used but with an addition of $q \in \{25, 50, 75, 100\}$, which signifies the maximum number of different classes, based on the number of items in percent (for instance, 25% of 5 items = 2.5, so at most 3 different classes). All instance sets can be found in the GitHub repository, `https://github.com/agatak7/CCMLBP`.

The tests were performed on the DelftBlue[3] supercomputer, using one 2x Intel XEON E5-6248R 24C 3.0GHz CPU per task, with a 900s time limit for each instance. The formulations were implemented using GCC 10.2.0, and solved with CPLEX 20.1 with default settings.

The four formulations which were tested were: The Standard IP MLBP (MLBP, see 4.1), the Network Flow MLBP (NFMLBP, see 4.2), the Standard CCMBLP (CCMLBP, see 4.3), meaning the standard MLBP with added CCMLBP constraints, and the Network Flow formulation with added CCMLBP constraints (NFCCMLBP, see 4.3). The standard IP formulations without added flow variables and other constraints for both MLBP and CCMPLBP can be considered the control formulations. The goal is to assess the performance of all these formulations and also the differences between the standard and NF formulations. The relevant metrics are: the objective value attained for each instance, the amount of CPU time needed to solve each instance, the number of Branch-and-Bound nodes used by CPLEX, and the number of instances solved to optimality.

5.2 Results

After running all the aforementioned instances in the DelftBlue[3] cluster, the results for each formulation were processed and aggregated. Firstly, by combining all of the instance classes to obtain the mean objective value, median CPU time, mean number of branch-and-bound nodes, and the proportion of optimally solved instances for each class. These results can be seen summarised in table 1.

MLBP and NFMLBP With the MLBP formulation, CPLEX was able to optimally solve around 11% more instances. This can be seen in the gaps in NFMLBP compared to MLBP in the graph in fig. 4. Where missing results signifies that for that instance no feasible solution was found by CPLEX within the time limit. The objective values obtained from the two formulations also don't differ greatly and the difference that exists is possibly caused by the missing instances which NFMLBP was not able to solve.

The CPU time for both the MLBP and NFMLBP does not differ greatly. For different instances one performs better than the other without a clear pattern. The time for both formulations grows

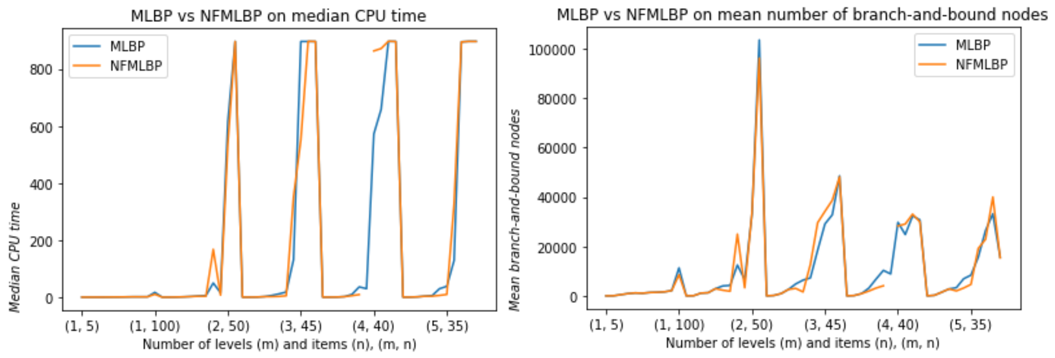

Figure 4: MLBP and NFMLBP compared on median CPU time and mean number of branch-and-bound nodes.

almost instantly up to the time limit of 900s when the number of items approaches a certain amount depending on the number of levels. At only one level, the time does not get too high even at 100 items, but as the number of levels grows, the amount of items which can be packed in the appointed time decreases. At 2 levels, up to 50 items can be efficiently packed, at 3 levels - around 40. At 4 and 5 levels, the increase in time happens around instances with 35 items. This can be seen in fig. 4.

The mean number of branch-and-bound nodes is greater for the NF formulation by 1%, which is not significant. As can be seen in fig. 4, in places one formulation performs better than the other without a clear pattern. Generally, more branch-and-bound nodes signifies that a larger space has to be searched for the optimal solution, ideally, the search space would be as limited as possible. Interestingly, considering the spike in branch-and-bound nodes around instances with two levels and 50 items in fig. 4, which is higher than for larger instances.

Comparing these two formulations using these metrics shows that, even though they perform similarly, overall the MLBP formulation appears more robust as it successfully solved more instances. The limits of both can also be clearly observed in the rapid increases in CPU time around a certain amount of items for each level.

CCMLBP and CCNFMLBP With the two CCMLBP formulations, the results are almost opposite to the MLBP ones. Here, CPLEX given the NFCCMLBP formulation was able to solve 6% more instances to optimality. However more significant differences can be seen in fig. 6 and fig. 5, where there are a lot of missing results for CMLBP. This is because if at least one feasible, even sub-optimal, solution is found within the time limit this result is still displayed, as opposed to when no solution is found.

For CPU time, the results for both formulations where both are present look very similar. However, looking at the median values in table 1, NFCCMLBP has a lot larger median time. This is because, with the standard CCMLBP formulation, CPLEX seemingly was not able to find a feasible solution within the time limit on many of the more difficult instances. It should also be noted that for instances where no feasible solution was found using the standard formulation, CPLEX was able to find a sub-optimal solution using the NFCCMLBP formulation, this can be seen on in instances where the time limit of 900s was reached. Overall, it can be said that there is a slight increase in number of 'peaks' as the number of q, so number of classes, grows. However, this does not affect the CPU time as much as an increase in number of items or levels.

The mean number of branch-and-bound nodes is around three times larger for the NFCCMLBP

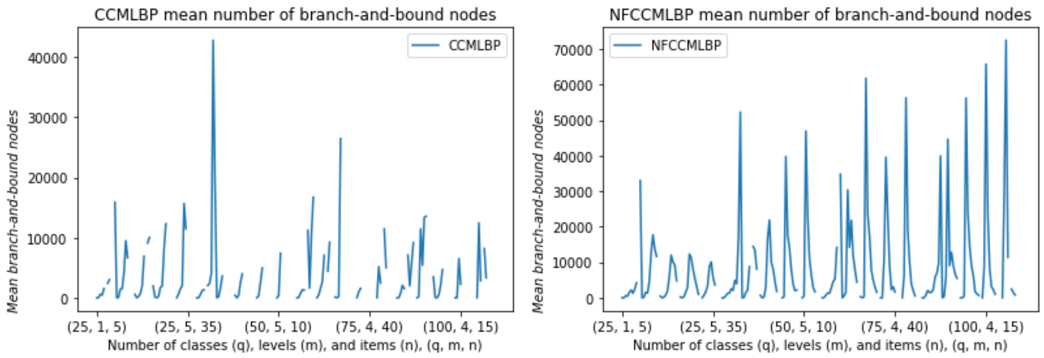

Figure 5: CCMLBP and NFCCMLBP compared on mean number of branch-and-bound nodes.

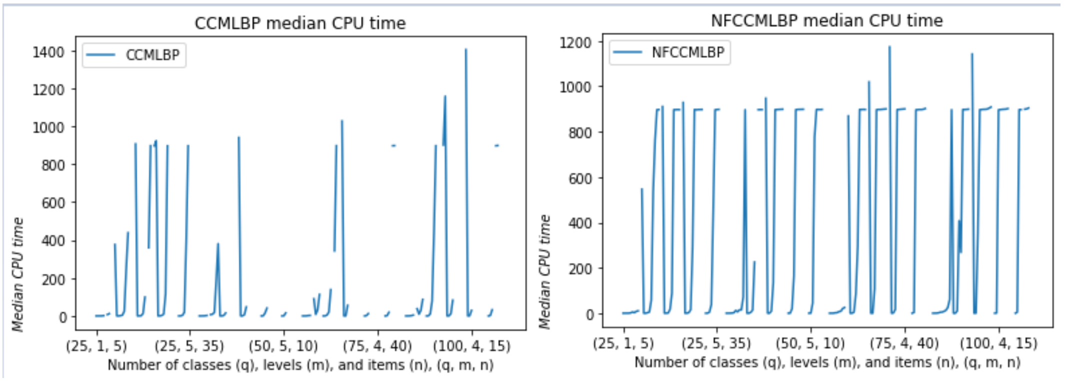

Figure 6: CCMLBP and NFCCMLBP compared on median CPU time for each class.

formulation. This, however, is mostly caused by the lack of solutions for CCMLBP. Where results are present, even for the simple instances, more branch-and-bound nodes are needed for the standard CCMLBP formulation.

Comparing these two formulations, significant differences can be seen. Here, the network flow formulation performs notably better than the standard one, as opposed to the MLBP formulations. The NF formulation seems a lot more robust, since with it CPLEX is able to find sub-optimal solutions for difficult instances which is better than not finding any. For NFCCMLBP the results are also more consistent in terms of the number of branch-and-bound nodes.

To conclude, a somewhat inconsistent behaviour can be observed between the standard and network flow formulations for the two problems. For MLBP, more results can be found with the standard formulation, although in general there are no significant differences. For the CCMLBP problem, however, the standard formulation performs much worse than the network flow one, in particular when it comes to more difficult instances. Somehow, with the NFCCMLBP formulation, CPLEX is able to find sub-optimal solutions for almost all of the instances which is not the case for the other formulation. This can be due to the internal operations of CPLEX, since many optimizations are performed by it which can differ based on even the order of adding constraints to the model. This makes determining the reasons for changes in performance difficult, however, it could be said that there is some potential in the network flow approach. The change in performance from MLBP to CCMLBP indicates that possibly for MLBP CPLEX perfers a simpler formulation to which it can apply it's own optimizations. This, perhaps, is not the case for the more difficult CCMLBP formulation, where the improvements of the network flow formulation can be observed.

6 Responsible Research

There are little ethical aspects which need to be considered in this work due to the abstract nature of it. Of course, as with most things, it could possibly be used to cause harm even though there is nothing inherently dangerous about studying problems like the multi-level bin packing problem. This, however, is an almost unavoidable consequence of any kind of research.

Reproducibility is a vital aspect of any research. In order to enable the reproducibility of this work to the fullest extent, a repository with all of the instances and source code was created[4]. There, provided are also the scripts used to generate instances and process/aggregate results. In order to reproduce the results, however, it is necessary to run the experiments on the DelftBlue [3] cluster using one standard node CPU per task, with a 1 hour total task timeout, and a 900 second CPLEX timeout. Otherwise, very likely different results would be obtained especially w.r.t CPU time.

7 Conclusions and Future Work

In conclusion, as demonstrated by the results, the integer programming approach can be used to solve a relatively large set of instances of the MLBP and CCMLBP problems. The largest instance class which can be solved to optimality in under 900 seconds for the MLBP problem was that with 5 levels and 35 items, but on average problems with under 5 levels and around 40 items can be efficiently solved. For the CCMLBP problem, the largest instance class solved to optimality within the time limit was that with the number of classes of at most 100% of the number of items, 5 levels, and 10 items. However, on average configurations with a smaller percentage of classes, under 5 levels, and up to 20 items were solved efficiently.

Comparing the standard and network flow formulations, it can be concluded that the standard formulation likely due to it's simplicity is able to be better used by CPLEX in the MLBP scenario. However, in the more complex CCMLBP problem, the network flow formulation shows it's strength and potential. In the CCMLBP case, the network flow formulation performed better on more difficult instances and using it CPLEX was able to find sub-optimal solutions within the time limit while no solution was found with the standard version. The results are, nevertheless, too limited to decisively conclude superiority of the network flow formulation.

More research and experimentation is needed to gain a larger understanding of how CPLEX interprets the various formulations and which optimizations are applied to each of them. This can be done by studying the CPLEX node log [5]. Furthermore, various CPLEX paramenters can be tuned in order to possibly obtain better results than those presented in this work. The formulations themselves can also be further improved. For instance, the constraint described in equation (18) of the network flow formulation (section 4.2) can be strengthened. In the current model the flow for any bin at any level is bounded by n_0, the number of items. A better estimate could be found by considering the largest capacity of a bin on each level.

[4]`https://github.com/agatak7/CCMLBP` Accessed 19-06-2022

[5]`https://www.ibm.com/support/pages/cplex-performance-tuning-mixed-integer-programs` Accessed 18-06-2022

References

[1] Lei Chen, Xialiang Tong, Mingxuan Yuan, Jia Zeng, and Lei Chen. A data-driven approach for multi-level packing problems in manufacturing industry. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1762–1770, 2019.

[2] EG Coffman Jr, MR Garey, and DS Johnson. Approximation algorithms for bin packing: A survey. *Approximation algorithms for NP-hard problems*, pages 46–93, 1996.

[3] Delft High Performance Computing Centre (DHPC). DelftBlue Supercomputer (Phase 1). `https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1`, 2022.

[4] Leah Epstein, Csanád Imreh, and Asaf Levin. Class constrained bin packing revisited. *Theoretical Computer Science*, 411(34-36):3073–3089, 2010.

[5] Donald K. Friesen and Michael A. Langston. Variable sized bin packing. *SIAM journal on computing*, 15(1):222–230, 1986.

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

[7] John E Mitchell. Branch-and-cut algorithms for combinatorial optimization problems. *Handbook of applied optimization*, 1:65–77, 2002.

[8] George L Nemhauser and Laurence A Wolsey. *Integer programming and combinatorial optimization*, volume 191. Springer, 1988.

[9] Christos H Papadimitriou. On the complexity of integer programming. *Journal of the ACM (JACM)*, 28(4):765–768, 1981.

[10] Gerard Sierksma. *Linear and integer programming: theory and practice*. CRC Press, 2001.

[11] Daniel A Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM (JACM)*, 51(3):385–463, 2004.

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