Als Manuskript gedruckt

Technische Universität Dresden
Herausgeber: Der Rektor

The Overflowing Bin Packing Problem: Theoretical Results and a New Flow Formulation

John Martinovic, Nico Strasdat

Preprint MATH-NM-01-2024

August 2024

The Overflowing Bin Packing Problem: Theoretical Results and a New Flow Formulation

J. Martinovic[a,*], N. Strasdat[a]

[a] *Institute of Numerical Mathematics, Technische Universität Dresden, Germany*

Abstract

We consider a recently proposed one-dimensional packing problem, called the overflowing bin packing problem (OBPP). In this scenario, we are given a set of items (of known sizes) and a set of bins (of known capacities). Roughly speaking, the task is to assign the items to the bins in such a way that the total load of every bin is as close as possible to its capacity. In this article, we first separate the problem under consideration from related partitioning problems, thus justifying its status as an independent branch of research. Subsequently, we review an assignment model known from the literature and investigate its theoretical properties. As a main result, we give a closed-form term for the associated LP bound and specify its worst-case performance ratio. In the second part, we then present a new flow-based approach for the OBPP and discuss its theoretical and numerical benefits, the latter being based on extensive computational tests with different benchmark sets. Overall, the new flow formulation typically leads to good-quality (mostly even optimal) solutions in short time, clearly outperforming the previous state of the art.

Keywords: Cutting, Bin Packing, Flow Formulation, Partition Problem, LP Bound

1. Introduction

In this article, we consider a multiset of m positive integers $c_1, \ldots, c_m$ that has to be partitioned into a sequence of n subsets subject to some given optimality criterion. From a theoretical point of view, such problems can be approached from various perspectives like combinatorics, scheduling, or cutting and packing, and they are commonly known to be $\mathcal{NP}$-hard, see [24]. Probably one of the earliest references dates back to the 1960s, see [25], introducing a makespan minimization problem in multiprocessor scheduling. In this specific context, the integers c_i, $i \in I := \{1, \ldots, m\}$, can be thought of the times needed to process given jobs, and the task is to assign these jobs to n machines (each of which being able to execute just one job in parallel), while minimizing the total completion time of the entire project, i.e., the largest sum of integers in any of the n subsets. Given a wide variety of further practically-relevant application areas, such as in logistics [3, 41], so far research has focussed mainly on heuristic algorithms, see [2, 19, 20] for some examples, whereas the literature on exact approaches is smaller. For the sake of exposition, here we just mention a branch-and-bound method proposed in [16, 17], a cutting plane approach introduced in [37], and a binary search strategy suggested in [15]. Some further advances have been regularly summarized in renowned survey articles, see [6, 7, 36] for some (by far not exhaustive) references.

Apart from makespan minimization, the relevant literature also deals with scenarios, in which the "fairness" or "efficiency" of an assignment is measured differently, e.g., by maximizing the

*Corresponding author

Email addresses: john.martinovic@tu-dresden.de (J. Martinovic), nico.strasdat@tu-dresden.de (N. Strasdat)

smallest sum of integers or by minimizing the discrepancy between the largest and the smallest sum of integers assigned to one of the subsets, see [22, 32]. Among these neighboring goals, the first objective is certainly the more common one, especially due to its relevance in

- economic applications to obtain fair allocations of indivisible goods among a set of agents, see [31],
- sequencing maintenance actions for a fleet of machines to keep the whole system operational for as long as possible, see [23].

Note that, according to [30, 43], these alternative objective functions lead to optimization problems with mutually disjoint solution sets, in general, hence representing independent branches of research. Further extensions to multidimensional input data can be found in [21, 29].

In this article, another optimality criterion recently introduced by the authors of [5] is considered. To be more precise, any of the n subsets is equipped with a target value C_k, $k = 1, \ldots, n$, and the integers assigned to any subset k have to sum up to some value close to C_k, meaning that any unit of deviation (i.e., under- and overutilization) is counted in the objective function. In [5], the problem under investigation is introduced as a novel bin packing variant, called the *overflowing bin packing problem (OBPP)*, in which the subsets are referred to as bins of capacity[1] C_k, $k = 1, \ldots, n$, and the integers $c_1, \ldots, c_m$ can be thought of given item sizes. Note that the connection between makespan minimization and bin packing is well established in the literature since the 1970s, see [8], and it was mainly applied to construct heuristic approaches in the field of multiprocessor scheduling [8, 28]. In fact, any bin packing problem (BPP) can be interpreted as a machine scheduling problem with fixed makespan, but an unlimited number of servers. Given this relationship, the BPP is also referred to as a "dual version" of the makespan minimization, see [15]. However, both problems are not dual to each other in the sense of mathematical optimization, but the research on partitioning problems can highly benefit from results obtained in bin packing. To this end, the interested reader is referred to the book [39], the survey articles [11, 42], and the references mentioned therein to obtain a more thorough overview of the most important classic solution approaches for the BPP.

Given the novelty of the OBPP, there is not yet a substantial body of works in this specific field. The only available preprint article, that is [5], mainly provides an introduction to some heuristic approaches and motivates the optimization problem to be investigated from a concrete application having rather homogeneous bin capacities. To be more precise, digital documents with a given file size are to be combined into packages (so-called archival information packages) for archiving, whereby it is important for the subsequent accessibility of the data that the size of the resulting archives is as identical as possible. The quality of any such partition is then evaluated by the objective function described above. Another area of application is given by server consolidation problems in computer science, because it is a well known fact that servers require a disproportionally high amount of energy whenever they are idle, underutilized, or overloaded, see [26] or [45, Fig. 2.2]. Hence, allocating a set of given jobs to executing units as fairly as possible is very important to keep operational costs low.

Overall, the main purpose of this article is (i) to show the relations of the OBPP to well-established neighboring problems, (ii) to introduce a new flow formulation tailored for the OBPP, and (iii) to investigate most important theoretical and numerical properties of that new ILP model. The main contributions can thus be summarized as follows:

- We establish the OBPP as an independent field of research in cutting and packing, separating it from three neighboring problem classes ($\rightarrow$ Sect. 2).

[1] In fact, the term "capacity" could be slightly misleading since the bins are allowed to carry items with total load larger than the given capacity.

- We provide a closed-form term for the LP bound of an assignment model introduced in [5]. In addition, we discuss the worst-case behavior of this lower bound ($\rightarrow$ Sect. 2).
- We introduce a new flow-based approach for the OBPP. In particular, this ILP formulation is theoretically shown to be superior to the assignment model in terms of the respective LP bounds ($\rightarrow$ Subsect. 3.1). Moreover, the challenges of constructing a competitive reflect formulation, as introduced in [18], are discussed ($\rightarrow$ Subsect. 3.2).
- The numerical behavior of the two ILP formulations is studied. At first, the flow model is shown to be able to solve all benchmarks sets introduced in [5] and three additional (more challenging) benchmark sets (gathered together for the variable-sized BPP in [18]) to proven optimality ($\rightarrow$ Subsect. 4.2). As a consequence of that, a new set of more challenging benchmark instances for the OBPP is designed to study the numerical performance of the flow model in more details ($\rightarrow$ Subsect. 4.3).

2. Preliminaries

Let us consider a set of item types $I := \{1, \ldots, m\}$, each characterized by a size c_i and an availability d_i, $i \in I$, and a set of bins $K := \{1, \ldots, n\}$, each associated with a capacity C_k, $k \in K$. By grouping these input data into integer vectors, we can refer to a specific OBPP by the following term.

Definition 1. *A tuple $E = (m, \boldsymbol{c}, \boldsymbol{d}, n, \boldsymbol{C})$ with $\boldsymbol{c}, \boldsymbol{d} \in \mathbb{Z}_+^m$ and $\boldsymbol{C} \in \mathbb{Z}_+^n$ is called an instance (of the OBPP).*

Remark 1. *Without loss of generality, we typically assume the items (bins) of an instance to be sorted with respect to non-increasing values of c_i (C_k), unless stated otherwise.*

We now look for an items-to-bins assignment minimizing the sum of the absolute difference between the capacity of each bin and the total size of the items assigned to it. In other words, we are looking for an assignment where the items are packed in a "balanced" way with respect to the bin capacities.

Definition 2. *Let E be an instance of the OBPP. Any items-to-bin assignment is referred to as a vector $\boldsymbol{a} \in \mathbb{Z}_+^m$ with a_i stating the number of items of type $i \in I$ appearing in the bin. Any such vector is called a pattern.*

In contrast to the ordinary BPP, the definition of a pattern is independent of the bin capacities, since bins are allowed to be underutilized and overloaded. To better keep track of the actual bin occupation (induced by a given pattern $\boldsymbol{a}$), let us define the following terminology.

Definition 3. *Let E be an instance of the OBPP and let $\boldsymbol{a} \in \mathbb{Z}_+^m$ denote a pattern for bin type $k \in K$. Then, we call this bin*

- *open, whenever $\boldsymbol{c}^\top \boldsymbol{a} < C_k$ holds, i.e., the items assigned to the bin do not use all capacity units,*
- *balanced, whenever $\boldsymbol{c}^\top \boldsymbol{a} = C_k$ holds, i.e., the items assigned to the bin perfectly use the capacity units,*
- *closed, whenever $\boldsymbol{c}^\top \boldsymbol{a} > C_k$ holds, i.e., the items assigned to the bin use more than C_k capacity units,*
- *overfull, whenever the bin is closed and there is an item (in the bin) which can be removed without changing the status of the bin. Any such item (in the pattern) is called redundant.*

From the objective function perspective, redundant items (in an overfull bin) will cause the largest costs and should be avoided (if possible). This leads to the following easy observation.

Lemma 2. *Let E be an instance of the OBPP, and let us consider an optimal solution of E. If that solution contains an overfull bin, then it does not contain any open bin.*

Proof. Let us assume the opposite by considering an optimal solution having both, (at least) one overfull and (at least) one open bin. Obviously, the objective function is reduced by shifting any redundant item from an overfull bin into an open bin. So, the given solution cannot be optimal. □

In other words, an overfull bin can only appear in an optimal solution if any other bin is balanced or closed. Another important property of the solution set is the following:

Lemma 3. *Let E be an instance of the OBPP with $c_i = C_k$ for some $(i,k) \in I \times K$. Then, there is an optimal solution in which bin k contains precisely one item of type i (and no other items).*

Proof. Let us assume that there is no such optimal solution. Then, there are two cases:

(i) There is an optimal solution in which bin k contains one item of type i and some further items.

(ii) There is no optimal solution at all in which one item of type i is assigned to bin k.

In the first case, we note that bin k is overfull. According to Lemma 2, the considered solution must not contain any open bin. Hence, we can shift all further items from bin k into bin k' without changing the objective value, and the claim is proved.

For the second case, let us consider an arbitrary bin $k' \neq k$ containing (at least) one item of type i. Then, this configuration can be modified as follows: One item of type i is shifted from bin k' to bin k. All other items from bin k are assigned to bin k'. Hence, we observe:

- Before: Bin k' is filled by one item of type i (size c_i) and possibly other items of total size, say, $W(k') \geq 0$. Bin k is filled with some items of total size, say, $W(k) \geq 0$.
- After: Bin k' is filled by some items of total size $W(k') + W(k)$. Bin k is filled by precisely one item of size c_i (and no other items).

To quantify the impact on the objective function, only the two bins k and k' have to be considered, since all other bins remain untouched. Due to the triangle inequality and $c_i = C_k$, we obtain:

$$\begin{aligned} & |W(k') + c_i - C_{k'}| + |W(k) - C_k| \geq |W(k') + c_i - C_{k'} + W(k) - C_k| \\ = \; & |W(k') + W(k) - C_{k'}| = |W(k') + W(k) - C_{k'}| + |c_i - C_k| \,. \end{aligned}$$

Hence, the proof is complete. □

Note that Lemma 3 directly leads to a preprocessing strategy to possibly remove some items and bins from a given instance. In fact, if there is some pair $(i,k) \in I \times K$ with $c_i = C_k$, then bin k and one item of type i can be removed from the instance. Moreover, the optimal value does not change by this operation since, effectively, a balanced bin (that is, a bin with no costs) is removed from the optimal solution.

In the following, we present two integer models for the OBPP known from the literature. To this end, we introduce integer variables $x_{ik} \in \mathbb{Z}_+$ denoting the number of items of type $i \in I$ that are assigned to bin $k \in K$. Then, the textbook model from [5] can be formulated as follows:

Nonlinear Assignment Model for the OBPP

$$z^{ass} = \sum_{k \in K} \left| C_k - \sum_{i \in I} c_i \cdot x_{ik} \right| \to \min$$

$$\text{s.t.} \quad \sum_{k \in K} x_{ik} = d_i, \qquad i \in I, \tag{1}$$

$$x_{ik} \in \mathbb{Z}_+, \qquad i \in I, k \in K. \tag{2}$$

In the objective function, the discrepancy of the actual bin load and the bin size is counted, and then summed up over all available bins. Furthermore, Constraints (1) ensure that all items are packed, whereas Constraints (2) define the integrality of the variables.

Remark 4. *In [5], the above model was originally introduced with binary (instead of integer) assignment variables. However, to possibly reduce the total number of variables in the case of identical items, we decided to use a CSP-like representation from the very beginning. This means that copies of identical items are grouped into one item type.*

Before taking a closer look at the optimization problem under consideration, let us first separate it from a number of related problems, and thus establish it as an independent branch of research. To be more precise, we remember that – for homogeneous bin sizes – the OBPP is close to the field of *multiway number partitioning* known from computer science, see also Sect. 1. In that scenario, a given list of integer numbers has to be partitioned into n classes, so that the associated n sums are "as identical as possible". To quantify this goal, common objectives used are

- to minimize the largest sum,
- to maximize the smallest sum,
- to minimize the difference between the largest and the smallest sum.

However, none of these objectives is identical to ours.

Theorem 5. *The OBPP is different from all the three aforementioned optimization problems.*

Proof. We prove this statement by specifying a concrete instance in which the solution sets of the OBPP and at least one of the other three problems are disjoint. To this end, we consider instances consisting of six items (originating from three item types) and three uniform bins.

(i) For the instance $E = (3, (13, 9, 6), (1, 2, 3), 3, (16, 16, 16))$, we observe the following:

- The partition $\mathcal{P}_1$ given by the patterns $\boldsymbol{a}^{(1)} = (1, 0, 0)^\top$, $\boldsymbol{a}^{(2)} = (0, 2, 0)^\top$, and $\boldsymbol{a}^{(3)} = (0, 0, 3)^\top$ minimizes the largest sum, and it is the unique solution with respect to this criterion.
- Since we have $C = 16$, the assignment described in $\mathcal{P}_1$ leads to $3 + 2 + 2 = 7$ units in the objective function of the OBPP. This is not optimal, since the partition $\mathcal{P}_2$ given by $\boldsymbol{a}^{(4)} = (1, 0, 1)^\top$, $\boldsymbol{a}^{(5)} = (0, 1, 1)^\top$, and $\boldsymbol{a}^{(6)} = (0, 1, 1)^\top$ only leads to an objective value of $3 + 1 + 1 = 5$ units. Hence, minimizing the makespan is different from the OBPP.

(ii) For the instance $E = (3, (13, 9, 6), (1, 2, 3), 3, (18, 18, 18))$, we observe the following:

- The partition $\mathcal{P}_2$ introduced before maximizes the smallest sum and also minimizes the difference between the largest and the smallest sum. In addition, it is the unique solution with respect to each of these criteria.
- Since we have $C = 18$, the assignment described in $\mathcal{P}_2$ leads to $1+3+3 = 7$ units in the objective function of the OBPP. This is not optimal, since the partition $\mathcal{P}_1$ only leads to an objective value of $5 + 0 + 0 = 5$ units. Hence, maximizing the smallest sum and minimizing the difference between largest and smallest sum are both different from the OBPP.

□

Hence, OBPP can be considered an independent optimization problem, even for the special case of homogeneous bin sizes. For the heterogeneous scenario, the difference is clear anyway.

It is possible to linearize the objective function of the above nonlinear model by introducing additional continuous variables $s_k \geq 0$, $k \in K$. Then, according to [5], we obtain

Linear Assignment Model for the OBPP

$$z^{ass} = \sum_{k \in K} s_k \to \min$$

$$\text{s.t.} \quad \sum_{k \in K} x_{ik} = d_i, \qquad i \in I, \tag{3}$$

$$\sum_{i \in I} c_i \cdot x_{ik} + s_k \geq C_k, \qquad k \in K, \tag{4}$$

$$\sum_{i \in I} c_i \cdot x_{ik} - s_k \leq C_k, \qquad k \in K, \tag{5}$$

$$x_{ik} \in \mathbb{Z}_+, \qquad i \in I, k \in K, \tag{6}$$

$$s_k \geq 0, \qquad k \in K. \tag{7}$$

This linearized version can now be dealt with by commercial ILP solvers. However, its computational performance is rather poor. On the one hand, the solution space can be highly symmetric, especially for rather homogeneous bin sizes. In such a scenario, permuting the indices k can lead to several feasible points having the same objective value. On the other hand, the more important negative feature is given by a rather poor LP[2] bound, as can be seen in the following theorem. Note that the rather technical proof is shifted to Appendix A.

Theorem 6. *The LP bound of the linear assignment model is given by*

$$z_{LP}^{ass,\star} = \left| \sum_{k \in K} C_k - \sum_{i \in I} c_i d_i \right|. \tag{8}$$

Remark 7. *As a direct consequence of Theorem 6, the LP value obtained by solving the relaxed version of the linear assignment model is always integral. Hence, there is no need for further rounding.*

As we saw in Theorem 6, the absolute gap between the supply of and the demand for bin units is an important indicator for a given instance E. For simplicity, we hence introduce the abbreviation

$$\varrho := \varrho(E) := \sum_{k \in K} C_k - \sum_{i \in I} c_i d_i. \tag{9}$$

On the one hand, the LP bound of a given instance can now be computed as $z_{LP}^{ass,\star} = |\varrho|$. On the other hand, we note that the overall problem may reduce to an empirically somewhat "easier" task, depending on the term ϱ. Indeed:

- For $\varrho > 0$ sufficiently large, there has to be a feasible solution in which every bin $k \in K$ is either open or balanced. In that situation, shifting one item to another bin cannot reduce the objective function value.
- For $\varrho < 0$ sufficiently small, there has to be a feasible solution in which every bin $k \in K$ is either balanced or closed. In that situation, shifting one item to another bin cannot reduce the objective function value.

[2]To obtain the LP problem, conditions (6) are replaced by $x_{ik} \geq 0$ for $(i, k) \in I \times K$.

Hence, in both these cases, the problem reduces to either finding a feasible point for a suitably defined ordinary cutting stock problem or skiving stock problem, respectively. However, although appearing to be easier from an empirical point of view, the problems to be solved still remain $\mathcal{NP}$-hard.

Remark 8. *Another reasoning is that, for $|\underline{\varrho}|$ sufficiently large, the integer optimal value always matches the LP bound, since there is an optimal solution for which*

- *all the bins are either overfull, or*
- *all the bins are open.*

In contrast, OBPP is particularly challenging whenever $|\underline{\varrho}|$ is not too large[3]. In fact, the tightness of the LP bound can be very poor in these scenarios.

Theorem 9. *The LP bound of the linear assignment model can be arbitrarily bad.*

Proof. Let us consider an even number $n \in \mathbb{N}$ and a sufficiently small $\varepsilon > 0$. We now study an instance containing n identical bins of size $C = 1$, $n/2$ items[4] of size $2 - \varepsilon$, and $n/2$ items of size ε. Then, the supply of and the demand for bin capacity units is identical, meaning that $z_{LP}^{ass,\star} = 0$ holds. On the other hand, the optimal value of the integer problem is given by $z^{ass,\star} = n \cdot (1 - \varepsilon)$, since one optimal solution is given by packing precisely one item into any of the bins. This proves the statement for both, the absolute and the relative performance of the LP bound. □

Due to the two reasons presented (possibly many symmetric solutions, and a typically poor LP bound), the assignment-based formulation cannot be assumed to perform very well in solving benchmark instances to proven optimality, see also Sect. 4.

3. Flow-based Formulations for the OBPP

In this section, we would like to introduce a more powerful compact ILP model for the OBPP which is based on a network flow structure. Note that the theoretical foundations of such approaches were already discussed in early publications like [44], but their numerical properties could only be verified with the progressing development of software tools and hardware components, see [42]. Meanwhile, flow-based models are among the most investigated and used solution approaches for fundamental problems such as the cutting stock problem [13, 18, 34], the skiving stock problem [33], temporal bin packing problems [35], and a wide variety of other related aspects, see [12]. From an overall point of view, flow-based approaches are generally convincing due to their great clarity in modeling, a (mostly) manageable number of variables, a well-structured set of constraints, and a very good LP bound. Observe that, mainly after the publication of the famous book by Nemhauser and Wolsey [38], in 1988, the latter aspect became a central concern in IP modeling.

3.1. A New Arcflow Formulation for the OBPP

Since there is no graph-based approach for the OBPP in the literature so far, we would first like to present a rather "classical" flow model for this optimization problem. To set up our flow model conveniently, let us slightly rephrase the representation of an instance of the OBPP. To be more precise, in this section, we typically also group the bins into bin types $k \in \bar{K}$, and introduce a number D_k stating how often bin type $k \in \bar{K}$ is available. With this at hand, the construction[5] of our network can be done as follows.

[3] As a consequence of that, this assumption will apply to all the benchmarks considered later in Sect. 4. Especially the new challenging instance sets designed in Subsect. 4.3 will always satisfy $\varrho = 0$.

[4] In a slight abuse of our assumptions, we note that the item sizes in this toy instance are not integral. However, by choosing ε as a sufficiently small unit fraction, all the item and bin sizes can be easily scaled to integer numbers.

[5] To facilitate understanding the construction of the graph, a detailed example can be found in Fig. 1.

- **Step 1:** In this step, we construct the standard arcs modeling the positioning of an item, hereinafter referred to as *item arcs*. Starting with $\mathcal{V}_{\text{item}} := \{0\}$ and $\mathcal{A}_{\text{item}} := \emptyset$, we go through the item types iteratively and update the sets of vertices and arcs in every step. In a specific iteration (corresponding to some fixed item type $i \in I$), for any $p \in \mathcal{V}_{\text{item}}$ we add the arcs $(p, p + c_i)$, $(p + c_i, p + 2c_i)$, ..., $(p + (d_i - 1)c_i, p + d_i c_i)$ to $\mathcal{A}_{\text{item}}$, as long as the tail is strictly below $\max_{k \in \bar{K}} C_k$. After that, we add all vertices that have been visited for the first time by these new arcs to $\mathcal{V}_{\text{item}}$. In fact, this construction is the same as generating the arcflow graph for an ordinary skiving stock problem with threshold $L := \max_{k \in \bar{K}} C_k$, see also [33, Algorithm 1] for a pseudocode.

- **Step 2:** In this step, we construct arcs modeling the use of the various bin types, hereinafter referred to as *bin arcs*. To this end, we first we introduce artificial sink nodes labeled t_k, $k \in \bar{K}$, and add them to $\mathcal{V}_{\text{bin}}$. Then, we generate two sets of bin arcs:

  - At first, we construct arcs (v, t_k) for every $k \in \bar{K}$ and for every $v \in \mathcal{V}_{\text{item}}$ that is the head of an item arc $(u, v) \in \mathcal{A}_{\text{item}}$ starting at $u < C_k$. These arcs are added to $\mathcal{A}_{\text{bin}}$, and they refer to a bin of size C_k being used to carry items the sizes of which sum up to v capacity units.
  - In a second step, for any $k \in \bar{K}$, we further add bin arcs $(0, t_k)$ to $\mathcal{A}_{\text{bin}}$. These arcs allow to leave a bin of capacity C_k empty.

  For any bin arc $(v, t_k) \in \mathcal{A}_{\text{bin}}$ the costs of this arc are defined by $\text{cost}(v, t_k) := |C_k - v|$ since these are precisely the units of over- or underutilization appearing in a path for bin type $k \in \bar{K}$ that ends at node $v \in \mathcal{V}_{\text{item}}$.

For any of the arcs $(u, v) \in \mathcal{A}_{\text{item}}$ or $(v, t_k) \in \mathcal{A}_{\text{bin}}$ generated by that procedure, we introduce an integer variable $\xi_{u,v} \in \mathbb{Z}_+$ or $\xi_{v,t_k} \in \mathbb{Z}_+$ modeling the units of flow carried by that specific arc. However, so far our network does not contain the possibility to build all overfull bins. To this end, we note that adding an item to a balanced or closed bin is equivalent to paying the "full price" of this item, since c_i units will be added to the objective function. Instead of allowing the paths in the network to be of arbitrary length, we can also introduce auxiliary variables $y_i \in \mathbb{Z}_+$ stating the number of *redundant* items of type $i \in I$, i.e., items that are not used efficiently at all. By that, the size of the network can be kept under control.

With these ingredients, we obtain the

Arcflow Model for the OBPP

$$z^{af} = \sum_{i \in I} c_i y_i + \sum_{(v,t_k) \in \mathcal{A}_{\text{bin}}} \text{cost}(v, t_k) \cdot \xi_{v,t_k} \to \min$$

$$\text{s.t.} \quad \sum_{(u,v) \in \mathcal{A}_{\text{item}} : v - u = c_i} \xi_{uv} + y_i = d_i, \qquad i \in I, \tag{10}$$

$$\sum_{(v,t_k) \in \mathcal{A}_{\text{bin}}} \xi_{v,t_k} = D_k, \qquad k \in \bar{K}, \tag{11}$$

$$\sum_{(u,v) \in \mathcal{A}_{\text{item}}} \xi_{uv} = \sum_{(v,w) \in \mathcal{A}_{\text{item}}} \xi_{vw} + \sum_{k \in \bar{K}} \xi_{v,t_k}, \qquad v \in \mathcal{V}_{\text{item}} \setminus \{0\}, \tag{12}$$

$$\xi_{uv} \in \mathbb{Z}_+, \qquad (u, v) \in \mathcal{A}_{\text{item}} \tag{13}$$

$$\xi_{v,t_k} \in \mathbb{Z}_+, \qquad (v, t_k) \in \mathcal{A}_{\text{bin}}, \tag{14}$$

$$y_i \in \mathbb{Z}_+, \qquad i \in I. \tag{15}$$

The objective function minimizes the total costs associated with the flow in the network. Here, costs either occur by redundant items (first sum) or as a consequence of under- or overutilized units in a bin (second sum). Moreover, Conditions (10) make sure that all items of type $i \in I$

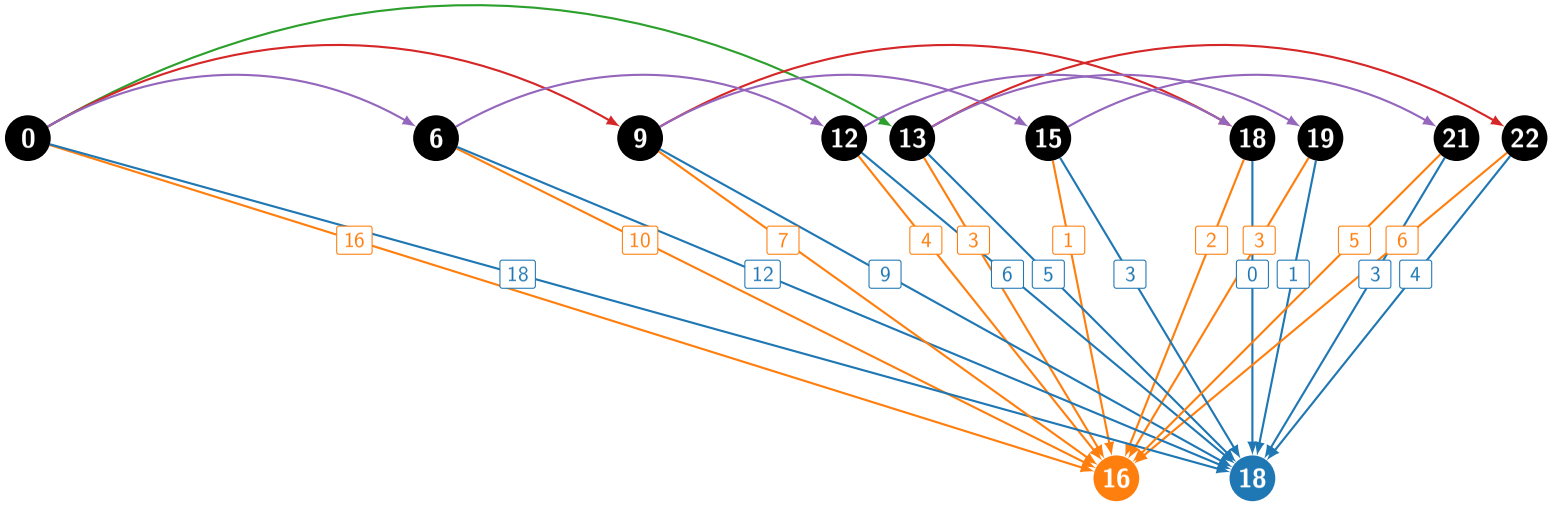

Figure 1: An illustration of the arcflow network for the instance $E = (3, (13, 9, 6), (1, 2, 3), 6, (16, 16, 16, 18, 18, 18)$ having six items and two bin types. The item arcs are colored according to the different item types: green arcs refer to item length $c_1 = 13$, red arcs refer to item length $c_2 = 9$, and purple arcs refer to item length $c_3 = 6$. Similarly, the bin arcs and the artificial sink nodes are colored according to the two different bin types: orange arcs refer to bin type $k = 1$ with $C_1 = 16$, blue color refers to bin type $k = 2$ with $C_2 = 18$. The small square-shaped labels attached to the bin arcs correspond to their associated costs. Note that this network contains both the two networks relevant for the instances considered in the proof of Theorem 5. To this end, precisely one of the two bin types has to be discarded.

appear, either directly as a part of a path (first term) or indirectly as a redundant item (second term) in some unspecified position. Constraints (11) state that all bins of type $k \in \bar{K}$ have to appear in a feasible solution. This also includes the possibility to use an empty bin. Furthermore, Conditions (12) model the flow conservation at every interior vertex. Note that, for the outgoing flow, both types of arcs (item arcs and bin arcs) have to be considered in that equation. Finally, Constraints (13)–(15) force all variables to be integer.

Remark 10. *To avoid a possibly large number of bin arcs in the arcflow model, we also tried an alternative construction using loss arcs between successive vertices in forward and backward direction, a variant that was partly inspired an approach used in [14] for a multiple knapsack problem. In our scenario, any such loss arc then has to be equipped by costs identical to the absolute value of its length. In addition, just one bin arc (C_k, t_k) per bin type $k \in \bar{K}$ would be required to keep track of the bin types used. However, although this alternative way of modeling led to slightly smaller ILP models, typically it did not lead to better numerical results in our internal tests.*

Remark 11. *Note that the implication*

$$\sum_{i \in I} y_i \geq 1 \Longrightarrow \forall k \in \bar{K} : \sum_{(v,t_k) \in \mathcal{A}_{bin} : v \geq C_k} \xi_{v,t_k} = D_k \tag{16}$$

has to hold for any integer[6] *optimal solution. In other terms, an auxiliary variable y_i, $i \in I$, can only be positive, meaning that there is at least one redundant item of that type, whenever all bins (of the considered solution) are balanced or closed. However, the logical implication given by (16) can only be linearized based on a big-M method which is not helpful to raise the LP bound.*

As alluded to at the beginning of this section, flow formulations are known to possess better LP bounds than assignment-based approaches. This favorable property can also be verified here in the following theorem. Note that the rather technical proof is shifted to Appendix B.

Theorem 12. *The LP bound of the arcflow model is greater than or equal to the LP bound of the linearized assignment model, i.e., we have*

$$z_{LP}^{af,\star} \geq \left| \sum_{k \in \bar{K}} C_k D_k - \sum_{i \in I} c_i d_i \right| . \tag{17}$$

Given this observation, the new flow formulation can be expected to perform better than the state-of-the-art model from the literature in the computational experiments conducted in Sect. 4.

3.2. An Open Problem: Constructing a Competitive Reflect Model for the OBPP

In [18], the authors proposed a novel view on flow formulations for one-dimensional cutting and packing problems, referred to as the *reflect model*. The main idea of that approach was given by (more or less) focussing on the vertex set corresponding to the first half of the bin only, and constructing feasible patterns by a combination of two subpaths that are connected by a special type of arc, called *reflect arc*. As shown for some bin packing variants in [14, 18], and later also for the skiving stock problem in [33], the resulting models are typically smaller in size, due to the improved representation of the required state space, and lead to convincing numerical results.

The question whether and, if so, how a reflect model can also be formulated efficiently for the OBPP, inevitably leads to some challenges collected in the list below. For some of them there are rather straightforward solutions, based on ideas already successfully applied in the literature.

[6] In fact, a similar implication can be formulated for the optimal solutions of the associated linear program. In this scenario, $\sum_{i \in I} y_i \geq 1$ has to be replaced by $\sum_{i \in I} y_i > 0$ on the left-hand side of (16).

(a) From a pattern perspective, the OBPP can be thought of a combination of a cutting stock problem (CSP) and a skiving stock problem (SSP), because bins are allowed to be under- and overutilized. Hence, any feasible network has to contain both, cutting stock patterns (with items summing up to at most the bin capacity) and skiving stock pattern (with items summing up to at least the bin capacity). In the literature, reflect was successfully applied to both these optimization problems using more or less the same network structure, see [18] and [33]. Hence, representing two types of patterns in one graph cannot be expected to be problematic, as long as any artificial reflect arc of type $(C/2, C/2, r)$ is allowed to carry negative units of flow. This is required to combine two subpaths of length larger than or equal to $C/2$ in the resulting network, see [33, Subsection 3.2].

(b) Compared to the aforementioned classic versions of the CSP and the SSP, the OBPP is more or less a "variable-sized" version of both these problems since different bin types are involved. Hence, a separate reflection point $R_k := C_k/2$, $k \in \bar{K}$, for any such bin type has to be considered, meaning that one single item arc (u, v) from the model presented in Subsect. 3.1 could lead to more than one reflect arc, namely if $u < R_k < v$ is true for several $k \in \bar{K}$. From a bin packing perspective, the general ideas of Algorithm 9 from the supplemental online material of [18], which can be found at `https://pubsonline.informs.org/doi/suppl/10.1287/ijoc.2018.0880/suppl_file/ijoc.2018.0880.sm1.pdf`, can be used to cope with that issue.

(c) In all bin packing variants of reflect, a symmetry breaking rule can be used to reduce the number of reflect arcs to be generated. This rule demands that any reflect arc with tail u, head v, and reflection point R_k, $k \in \bar{K}$, has to satisfy $R_k - u \geq R_k - v$, meaning that the "forward part" of such arc is at least as long as the "backward part". Unfortunately, whenever skiving stock patterns have to be modelled, this rule can no longer be applied[7]. Since items are allowed to be larger than bin capacities in the OBPP, in particular, this may also lead to vertices indexed with a negative number.

(d) In the construction proposed in [18], loss arcs in forward direction are introduced to keep track of the bin units not covered by items. In the OBPP, however, bins are allowed to be over- and underutilized, and both options are associated with costs. Hence, for our reflect network, we require loss arcs in forward and backward direction. Independent of the specific direction, any such arc has to be associated with costs equalling the absolute value of its arc length.

While all the aforementioned ideas help to come closer to a correct reflect formulation for the OBPP, the weak point and true open challenge is to deal with a variable-sized version of the skiving stock problem. At first, note that there are no corresponding approaches in the literature. In addition, a straightforward application of reflect, with the modifications mentioned before, does not lead to a feasible network, as can be seen in the Example 1.

Example 1. *Let us consider an instance having two bin types with $C_1 = 22$ (twice) and $C_2 = 18$ (once) as well as three item types $c_i = 14$, $c_2 = 11$, and $c_3 = 7$ each of which available twice. The resulting reflect network, corresponding to a variable-sized SSP, is depicted in Fig. 2. Note that this network is identical to what would be obtained with the ideas collected so far for the OBPP, except that the loss arcs in forward direction would be missing, because an SSP pattern will never consist of unused units of the bin capacity. In contrast, we have to use loss arcs in*

[7]For the sake of exposition, let us consider an instance with one bin of size $C = 10$ and three items with sizes $\boldsymbol{c} = (4, 4, 3)$. Of course, the unique solution is given by using the pattern $\boldsymbol{a} = (1, 1, 1)^\top$. However, due to $R := C/2 = 5$ neither the arc $(4, 2, R)$ (attaching one item of size 4 directly after the other item of the same size) nor the arc $(4, 3, R)$ (putting together an item of sizes 4 and 3) would exist in the network, if the symmetry breaking rule from was used.

backward direction to model the bin units used additionally (i.e., those beyond the bin size) in a skiving stock pattern.

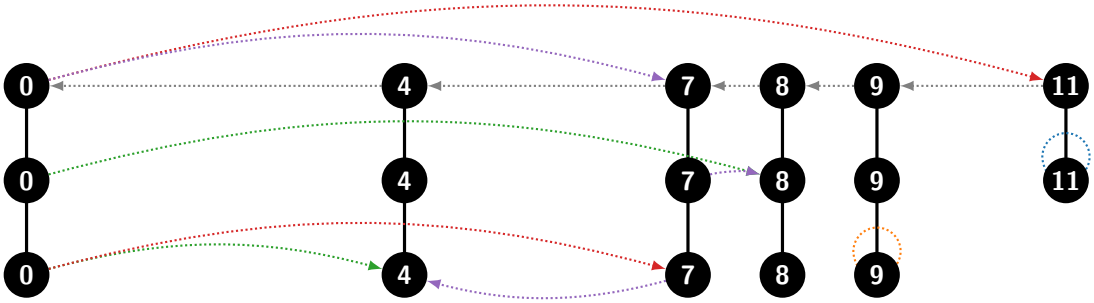

Figure 2: The reflect graph for the variable-sized skiving stock problem considered in Example 1. For the sake of a better visibility, the illustration of the graph is structured in layers. In the first layer, we display all standard arcs and all loss arcs (colored gray), whereas in the remaining layers, we draw all reflect arcs relevant for the considered bin type of size $C_1 = 22$ (second layer) or $C_2 = 18$ (third layer), respectively. The arcs modeling the placement of an item are colored as follows: green color is used for $i = 1$, red color is used for $i = 2$, and purple color is used for $i = 3$. Artificial reflect arcs, starting and ending at $C_k/2$, are colored blue (for $k = 1$) and orange (for $k = 2$).

In terms of flow conservation, a standard arc entering a node $v \neq 0$ can be continued by

- *a standard arc or reflect arc leaving node v (classic flow conservation),*
- *a loss arc entering node v (modeling additionally required bin units beyond the bin capacity),*
- *a reflect arc entering node v (merging two subpaths to one pattern).*

No matter whether a variable-sized SSP (with objective function coefficients equal to the bin sizes) or the OBPP is considered, the solution given by the network consists of the following nonzero flow variables:

$$\xi_{(0,7,s)} = 2,\ \xi_{(0,8,r_1)} = 2,\ \xi_{(9,9,r_2)} = -1,\ \xi_{(0,7,r_2)} = 2, \xi_{(9,8,l)} = 2.$$

In that notation, (u, v, κ) denotes an arc starting at u, ending at v, and having type κ, where $\kappa = s$ is used for standard arcs, $\kappa = l$ is used for loss arcs, and $\kappa = r_k$ is used for reflect arcs associated to bin type $k \in \bar{K}$.

From a pattern perspective, the two subpaths $(0, 8, r_1) \to (9, 8, l)$ are connected by the artificial reflect arc $(9, 9, r_2)$ carrying a negative flow (due to flow conservation). In addition, the subpaths $(0, 7, s)$ and $(0, 7, r_2)$ are combined twice. Hence, the network allows building two patterns associated with bin type $k = 2$, although that bin type is only available once. For this reason, the applied modeling and the network are infeasible.

The problem is that, in some situations, two subpaths (of a fixed bin type) already containing a reflect arc can be linked by an artificial reflect arc of another(!) smaller bin type. Due to the negative flow on that arc, this smaller bin type k can then be virtually used more often than allowed by the input parameter D_k. In a variable-sized SPP, this may help to seemingly create more patterns than possible, which increases the revenue obtained in the objective function. In the OBPP, this may help to shortcut the costs required to leave some portions of a large bin type empty. Indeed, for the "optimal solution" detected by the network, the total costs would be equal to two, since just one loss arc is used twice. However, any true optimal solution to the OBPP requires four units of over- or underutilization. For instance, such a configuration can be obtained by using the pattern $(0, 1, 1)^\top$ for bin type $k = 2$ (no costs) as well as $(1, 0, 1)^\top$ and $(1, 1, 0)^\top$ for bin type $k = 1$ (in total, $1 + 3 = 4$ cost units).

As we have explained in detail using the previous example, it is impossible to set up an efficient reflect model for the OBPP, as there is no valid graph-theoretic representation of the variable-sized SSP within a single graph according to the best of our knowledge. Here, artificial reflect arcs can complete subpaths of other bins, thus avoiding costs by virtually changing the availabilities of the bin types. To prevent such effects, it would therefore only be possible to set up a separate reflect graph for each bin type. However, this leads to a very large number of variables, typically containing several copies of one and the same state and/or decision, meaning that the resulting model would no longer be competitive. For the sake of exposition, some exemplary numerical results for that version of reflect will be discussed in Remark 15 at the end of Subsect. 4.3.

4. Computational Results

In this section, we compare the two models presented earlier from a numerical point of view. To this end, the ILP models were coded in Python and passed to the Gurobi solver (version 10.0, default settings). The hardware is given by a computer having an AMD Ryzen 9 5900X processor with 64 GB RAM. Unless stated otherwise, a time limit of 300 seconds was used to solve the integer programs.

4.1. Benchmark Sets from the Literature

Since the OBPP is a rather new problem in discrete optimization, there is not yet a comprehensive collection of benchmark instances. So far, the only sets tailored for the OBPP have been introduced in [5]. This benchmark set consists of three classes, called F1, F2, and F3, of 200 instances each. The main difference between these classes is given by the fact, whether the optimal solution is known by construction or not.

- By that, we mean that for F1 and F2, the authors first constructed a set of items, then assigned these items to some bins, and finally set the bin capacities equal to the actual bin fillings. Hence, all these instances have an optimal value of zero.
- For F3, the authors first constructed a set of items together with a suitable number n of total bins, and finally defined an almost perfectly homogeneous set of bins. To be more precise, all but possibly the very last bin possess the capacity $C = \left\lfloor \frac{\sum_{i \in I} c_i}{n} \right\rfloor$. Whenever the argument of the floor function is integer, all bins will have the same capacity. Otherwise, the very last bin will compensate for the (fractional) units missing in the previous bins, so that the total size of all bin capacities is equal to $\sum_{i \in I} c_i$. Note that the construction process ensures that there are at most two different bin types. However, note that the optimal solution is not known by construction, and it is not clear in advance, whether an integer solution with objective value zero exists.
- Any of the three subsets is further divided into packages each of which containing five instances. These packages are parametrized by $m = \sum_{i \in I} d_i$ and $n = \sum_{k \in \bar{K}} D_k$, i.e., by the total number of items and bins, respectively. To be more precise, we have $n \in \{10, 15, 20, 25, 30, 60, 90, 120\}$ and $m \in \{2n, 3n, 4n, 5n, 6n\}$.

An overview of some characteristic features of the three subsets is presented in Tab. 1. Besides the optimal value, the most remarkable difference between the three instance classes is the number of different bin types. By construction, in F3, almost all bins have the same capacity, so there are at most two different bin types. In F1 and F2, the bins are much more heterogeneous. To broaden the variety of the numerical experiments, we also make use of benchmark sets originating from a neighboring field, called the variable-sized bin packing problem, see [1, 4, 9, 42] for some associated publications. In this setting, a set of given items has to be assigned to a set of heterogeneous bins, so that the total sum of the bin capacities (of the required bins) is minimized. Contrary to the OBPP, unoccupied bins do not contribute to the objective function. Motivated by [18], the additional benchmark sets (in their original version) considered here are the following:

Class	$\lvert\bar{K}\rvert$	$\sum_{k\in\bar{K}} D_k$	m	$\sum_{i\in I} d_i$
F1	30.73	46.25	54.81	185.00
F2	38.17	46.25	54.81	185.00
F3	1.62	46.25	54.81	185.00

Table 1: Some characteristic features of the instances from [5]

- 'Crainic': We include three sets of rather easy instances presented in [10], forming a package of 840 instances. In their original form, these instances consist of up to 12 bin types and up to 1000 items in total, the latter of which possessing relatively small values of c_i.
- 'Hemmelmayr': We consider two sets of more difficult instances suggested in [27], forming a package of 200 instances. In their original form, these instances consist of up to seven bin types and 2000 items in total.
- 'Belov': We include four sets of rather difficult instances proposed in [4], forming a package of 200 instances. In their original form, these instances contain up to 5000 items in total, and the maximum number of bins is given by 2, 4, 8, and 16, respectively, for the four sets described before.

Note that most of these additional benchmark sets possess (much) more bins and/or items than those from [5]. Hence, they typically lead to optimization problems larger in size.

Given the fact that these instances were built to study a classic bin packing problem, the sum of the bin capacities is typically much larger than the sum of the item sizes. In the context of the OBPP, this situation would lead to rather easy optimization problems, as discussed in the final parts of Sect. 2. Hence, we had to modify these original instances by adjusting the set of bins according to the following rules:

(S1) At first, we sort the bin capacities of the original instance, respecting their multiplicity, with respect to non-increasing sizes.

(S2) We (repeatedly) iterate through this list and generate a bin of the associated size. This procedure stops when the sum of the bin capacities is larger than or equal to the total item size for the first time.

By that, we basically keep the characteristics of the original instances, but also create a more difficult scenario, where $\varrho > 0$ is relatively small by construction. The full details of the instances generated by these steps can be found at `https://github.com/wotzlaff/obpp-instances`. For the sake of exposition, however, some average characteristic data of the actual instances used in the computations can be found in Tab. 2:

4.2. Numerical Experiments and Discussion

For our computational experiments, we coded the following ILP formulations:

(I) the original linearized assignment model from [5] with binary variables, and no reduction according to Lemma 3,

(II) the linearized assignment model from Sect. 2 with integer variables and the reduction introduced in Lemma 3,

(III) the new flow formulation presented in Sect. 3 together with the reduction proposed in Lemma 3.

Class	$\lvert\bar{K}\rvert$	$\sum_{k\in\bar{K}} D_k$	m	$\sum_{i\in I} d_i$
Crainic 1	1.00	72.30	51.45	175.00
Crainic 2	3.08	19.97	19.50	450.00
Crainic 3	1.00	73.36	14.24	450.00
Hemmelmayr 1	7.00	596.02	186.43	760.00
Hemmelmayr 2	3.00	70.58	62.46	175.00
Belov 1	1.12	1927.86	99.14	4995.12
Belov 2	1.88	2058.42	99.10	5069.44
Belov 3	2.76	2078.52	99.24	5049.90
Belov 4	3.82	2068.10	99.10	5032.20

Table 2: Some characteristics of the postprocessed instances for the OBPP. The first two columns represent the number of bin types and the total number of bins, respectively. The other two columns state the number of item types and the total number of items, respectively.

Note that, in this comparison, Configuration (I) represents the state of the art from the literature. We highlight that the original calculation results tabulated in [5] have been obtained by Java (version 17) in combination with CPLEX (version 22.1.1). By way of example, only 189 of the 400 instances from F1 and F2 were solved to proven optimality by the implementation of the linearized assignment model in [5] within one hour of computation time. To enable a fair comparison to the contributions presented within this paper, we had to re-code that formulation, so that the same computational environment is used for any of the three ILP models.

In the subsequent computations, we mainly focus on the following performance indicators:

- The average time t taken to deal with the set of instances. Whenever an instance is not solved to proven optimality, the time limit is used as the solution time.
- The number *opt* of instances solved to proven optimality.
- An indicator called *value* measuring the average of the term

$$\frac{z_{best}}{\sum_{k\in\bar{K}} D_k C_k}$$

  per instance set. In this formula, z_{best} indicates the best objective value found within the time limit. Hence, the term *value* more or less displays the relative units of under- or overutilization of the bin capacities. Whenever $value = 0$ holds for a specific instance, then all items can be packed exactly, i.e., any optimal solution consists of balanced bins only.

Remark 13. *For a fixed instance E, the indicator called value can decrease or increase after having applied the reduction presented in Lemma 3. This is because both the numerator and denominator of the fraction (used to define it) are changed. Hence, a comparison based on that feature is difficult.*

Let us first have a look at the effects of the reduction rule presented in Lemma 3. To this end, we compare the average numbers collected in Tab. 1 and Tab. 2, without application of Lemma 3, with the respective numbers gathered for the same instances after having applied Lemma 3. The results of that comparison can be found in Tab. 3. We see that the effects of our preprocessing are highly instance-specific. To be more precise, we do not see any reduction for the instance sets F3, 'Crainic', and 'Belov'. To some extent, this is related to the properties of the original instance sets, and to the fact that the modified instances just contain a rather small number of different bin types. Hence, the probability that one item size matches a bin size has to be

small by construction. On the other hand, the reduction is able to successfully remove some items and bins for F1, F2, and 'Hemmelmayr'. Note that the total number of items removed is always equal to the total number of bins removed. The average numbers collected in Tab. 3 state that, on average, the reduction (if successful) of $\sum_{i \in I} d_i$ is between 1% and 4%, whereas the reduction in terms of $\sum_{k \in \bar{K}} D_k$ is between 3% and 16%.

Class	Lemma (3)	$\lvert\bar{K}\rvert$	$\sum_{k \in \bar{K}} D_k$	m	$\sum_{i \in I} d_i$
F1	✗	30.73	46.25	54.81	185.00
	✓	29.02	43.98	53.94	182.73
F2	✗	38.17	46.25	54.81	185.00
	✓	32.01	38.67	52.34	177.43
F3	✗	1.62	46.25	54.81	185.00
	✓	1.62	46.25	54.81	185.00
Crainic 1	✗	1.00	72.30	51.45	175.00
	✓	1.00	72.30	51.45	175.00
Crainic 2	✗	3.08	19.97	19.50	450.00
	✓	3.08	19.97	19.50	450.00
Crainic 3	✗	1.00	73.36	14.24	450.00
	✓	1.00	73.36	14.24	450.00
Hemmelmayr 1	✗	7.00	596.02	186.43	760.00
	✓	7.00	574.55	181.29	738.53
Hemmelmayr 2	✗	3.00	70.58	62.46	175.00
	✓	3.00	68.76	61.84	173.18
Belov 1	✗	1.12	1927.86	99.14	4995.12
	✓	1.12	1927.86	99.14	4995.12
Belov 2	✗	1.88	2058.42	99.10	5069.44
	✓	1.88	2058.42	99.10	5069.44
Belov 3	✗	2.76	2078.52	99.24	5049.90
	✓	2.76	2078.52	99.24	5049.90
Belov 4	✗	3.82	2068.10	99.10	5032.20
	✓	3.82	2068.10	99.10	5032.20

Table 3: Comparison between the raw instances (labeled '✗') and the reduced instances (labeled '✓').

We now study the computational results obtained for the benchmarks sets F1, F2, and F3 from [5] in Tab. 4. The following observations can be made:

	Model (I)			Model (II)			Model (III)		
Class	value	opt	t	value	opt	t	value	opt	t
F1	0.00006	(180)	49.6	0.00005	(184)	46.0	0.00000	**(200)**	**3.2**
F2	0.00002	(191)	33.6	0.00001	(197)	18.5	0.00000	**(200)**	**7.0**
F3	0.00488	(156)	75.5	0.00513	(152)	93.4	0.00481	**(200)**	**1.2**
Average (Sum)	0.00165	(527)	52.9	0.00173	(533)	52.6	0.00160	**(600)**	**3.8**

Table 4: Numerical results for the instances from [5]. Each subset consists of 200 instances.

- In contrast to the results tabulated in [5], Model (I) is able to solve much more instances to proven optimality in much shorter time. To some extent, this can be explained by using another programming language and another solver. However, the improvement is surprisingly large, given the fact that the hardware configurations used in [5] and in the

current article are more or less comparable. It is therefore possible that the data tabulated in [5] is not completely correct, so that re-coding Model (I) was indeed necessary.

- For the instances considered here, the modifications performed in Model (II) do not have a considerable impact. At first, we note that the average number of items per item type is close to 3, so that the reduction associated to integer (instead of binary) assignment variables is not that pronounced. However, due to successful preprocessing for F1 and F2, see Tab. 3, a few more instances could be solved to proven optimality. In contrast, applying Lemma 3 does not change any instance of F3 at all. Hence, these instances more or less stay identically difficult for the assignment model, especially since the LP bound is typically not exact.

- The new flow model (Model (III)) solves any benchmark instance to proven optimality with a significant speed-up factor (roughly 14 times faster on average). The improvement is particularly pronounced for set F3, where finding just one feasible solution (of the BPP) is not yet sufficient to solve the OBPP. A more detailed investigation of the results (for that specific subset), not tabulated here, reveals that the LP bound of the flow model is better than the LP bound of the assignment models in 51 (out of 200) cases, thus empirically confirming the validity of Theorem 12. In the majority of these cases, the LP bound of the arcflow model was even equal to the true optimal value. Only for 15 instances from F3, the flow model showed a positive additive integrality gap, ranging from 1 to 15 cost units. Except for a very few special cases (e.g., with optimal value 2 and LP bound 1), these absolute gaps refer to rather small percentage gaps typically below 10%.

- We highlight that the assignment models are more successful for F1 and F2 than for F3, because the LP bound is exact for any of former subsets. In addition, there are at most two bin types in F3, meaning that the level of symmetry is higher than for F1 and F2, where it is more likely that swapping the contents of two arbitrary bins cannot will also change the objective value.

- The objective values found by the assignment models are in the same range as those obtained by the arcflow formulation. Hence, the former mainly struggle with proving the optimality of the feasible solutions found. We attribute this to the main drawbacks associated with that type of formulation, i.e., a typically bad LP bound (only for F3) and a highly symmetric solution space. However, from an overall point of view, the indicator *value* is just about $3\% - 8\%$ larger for the two assignment models, meaning that the latter also lead to good-quality solutions is short time for the considered benchmark set.

Overall, the two assignment models were able to solve about 88% of the instances from [5] to proven optimality, meaning that the instances considered can be assumed to be rather easy. To reveal a few more significant differences between the ILP approaches, we therefore have to resort to the (on average) more difficult benchmark sets also introduced in Subsect. 4.1. Remember that one main reason for the increased level of difficulty (at least from an assignment-based perspective) is given by the larger total number of bins and/or items.

To this end, let us start with the instances originating from [10] and [27], the results of which can be found in Tab. 5 and Tab. 6, respectively.

Class	Model (I) value	opt	t	Model (II) value	opt	t	Model (III) value	opt	t
Crainic 1	0.02915	(172)	145.6	0.02615	(190)	132.3	0.02591	**(300)**	**0.1**
Crainic 2	0.06244	(60)	0.4	0.07327	**(60)**	**<0.1**	0.07327	(60)	0.2
Crainic 3	0.01572	(325)	123.1	0.01393	(438)	30.2	0.01386	**(480)**	**<0.1**
Average (Sum)	0.03577	(557)	89.7	0.03778	(688)	54.2	0.03768	**(840)**	**0.1**

Table 5: Numerical results for the instances originating from [10]. The subsets consist of 300, 60, and 480 instances, respectively.

Class	Model (I) value	opt	t	Model (II) value	opt	t	Model (III) value	opt	t
Hemmelmayr 1	0.06727	(11)	287.4	0.05001	(12)	285.5	0.03293	**(150)**	**0.6**
Hemmelmayr 2	0.02741	(37)	96.2	0.02659	(39)	86.0	0.02646	**(50)**	**0.2**
Average (Sum)	0.04734	(48)	191.8	0.03830	(51)	185.7	0.02969	**(200)**	**0.4**

Table 6: Numerical results for the instances originating from [27]. The subsets consist of 150 and 60 instances, respectively.

The main observations are given by:

- Apart from the set 'Crainic 2', the assignment models now struggle to solve the instances to proven optimality. In fact, only 66% (resp. 81%) of the instances considered in Tab. 5 are tackled successfully by Model (I) (resp. Model (II)). For the instances originating from [27], only about 25% were solved to proven optimality. Especially for the set 'Hemmelmayr 1', the quality of the feasible solutions obtained by the assignment models is not close to the optimum anymore. Note that for this specific subset, the indicators $\sum_{k\in\bar{K}} D_k$ and $\sum_{i\in I} d_i$ are by far the largest, see Tab. 3, leading to very large ILP models to be solved.
- Especially for 'Crainic 3', Model (II) performs much better than Model (I). According to Tab. 3, there is no reduction at all for any of these instances. So, the only reason for the improved performance is that the instances largely benefit from the CSP-like representation used in Sect. 2. In fact, having a look at the last two columns of Tab. 3, we see that the fraction $\left(\sum_{i\in I} d_i\right)/m$ is particularly large for the set 'Crainic 3', meaning that lots of binary variables can be saved by using integer assignment variables.
- The flow model is again able to solve all the 1040 instances to proven optimality. Remarkably, the times taken are much shorter than those associated with the instances from [5]. We attribute this to the fact that (on average) the number of different bin types is typically smaller here, see Tab. 3. The latter is one important ingredient for the number of bin arcs (i.e., the number of "additional" variables) appearing in the flow formulation. Due to a smaller number of bin arcs, the problems can be solved even faster compared to the results from Tab. 4.

Remark 14. *Interestingly, the situation described in Remark 13 can be observed for the subset 'Crainic 2'. In fact, all three ILP models are able to solve any instance to proven optimality. However, the indicator value for Model (I) is different. This is caused by a successful reduction in the sense of Lemma 3. Remember that this preprocessing technique does not change the optimal value since only exactly filled bins with zero costs are removed. Since all the instances of 'Crainic 2' were solved optimally, the numerator z_{best} does not change, but the denominator $\sum_{k\in\bar{K}} D_k C_k$ does. Consequently, the indicator value becomes larger.*

To clearly reveal the limitations of Model (I) and Model (II), we now consider the instances originating from [4]. Note that these instances have approximately 5000 items, roughly 2000 bins, and the set of bins is rather homogeneous. In addition, we already know from Tab. 3 that the reduction introduced in Lemma 3 is not successful for that specific benchmark set. The corresponding results are given in Tab. 7.

	Model (I)			Model (II)			Model (III)		
Class	value	opt	t	value	opt	t	value	opt	t
Belov 1	0.56385	(0)	300.0	0.03614	(0)	300.0	0.00225	**(50)**	**32.7**
Belov 2	0.55697	(0)	300.0	0.03723	(0)	300.0	0.00186	**(50)**	**35.4**
Belov 3	0.51309	(0)	300.0	0.03863	(0)	300.0	0.00144	**(50)**	**39.3**
Belov 4	0.45815	(0)	300.0	0.03447	(0)	300.0	0.00121	**(50)**	**41.4**
Average (Sum)	0.52302	(0)	300.0	0.03662	(0)	300.0	0.00169	**(200)**	**37.2**

Table 7: Numerical results for the instances originating from [4]. Each subset consists of 50 instances.

We highlight the following observations:

- The assignment-based formulations are not able to solve a single instance within the given time limit. For Model (I), even the feasible points obtained are of extremely poor quality, according to the column termed *value*. For Model (II), the feasible points found are much better than for Model (I). This is again a consequence of the CSP-like representation, because the fraction $\left(\sum_{i \in I} d_i\right) / m$ is always large. However, the results of Model (II) are still far from being optimal. Despite typically having a positive integrality gap as a result of the instance construction, the small number of different bin types leads to a high level of symmetry. These two observations, together with the significantly increased number of items and bins, prevent the assignment models from finding reasonable feasible points.

- For Model (I), we see that the quality of the feasible points obtained by the assignment model becomes (a bit) better when moving from 'Belov 1' to 'Belov 4'. The main reason for this is given by the fact that the average number of bin types increases from 1.12 ('Belov 1') to 3.82 ('Belov 4') while most of the remaining indicators are almost constant, see Tab. 3. With increasing number of bin types, the level of symmetry decreases, and hence the optimization problems become a bit more tractable.

- The flow model is again able to find optimal solutions for any considered instance in relatively short time. We note that the times are the highest among all instance sets tested, which is mainly due to the heavily increased number of items. A large number of items typically leads to many item arcs. Hence, the size of the ILP becomes larger.

- On the other hand, the computation time for the flow model increases slightly when moving from 'Belov 1' to 'Belov 4'. Again, the reason is given by having more bin types (i.e., more sinks in the network). Hence, the number of bin arcs, which is the smaller part influencing the number of variables here, grows and the ILP model to solve becomes larger.

Altogether, we saw that the flow-based formulation is able to solve any instance to proven optimality, thus being clearly superior to the previous state-of-the-art approach. We also worked out that the number of bin types is a crucial factor for the performance of the two ILP models. While many different bin types are (on average) favorable for the assignment model to reduce the symmetry of the solution space, a very heterogeneous set of bins leads to many additional arcs in the flow-based approach.

Given the observations made within the computations so far, from now on we only focus on the flow formulation in our further numerical experiments and discussions.

4.3. A New Challenging Benchmark Set

In this subsection, we first present a new benchmark set called 'MS' consisting of 1000 instances divided into two subsets 'MS 1' and 'MS 2', see also `https://github.com/wotzlaff/obpp-instances`. Note that any generated instance satisfies $\sum_{i\in I} d_i c_i = \sum_{k\in\bar{K}} D_k C_k$, following the guidelines of the instances tailored for the OBPP in [5]. The main difference between MS 1 and MS 2 is the following:

- For MS 1, all the bins can be filled exactly, so that the optimal value is always given by zero. Hence, the LP bound of any model is exact, and there is no additive integrality gap.
- For MS 2, it is not known in advance whether an optimal packing with zero costs exists. Hence, the flow model may sometimes possess a strictly positive additive integrality gap.

For constructing the sets MS 1 and MS 2, we applied the following procedure, which is partly inspired by the instance generation from [5]:

- Any instance is parametrized by a pair (m, p) with $m \in \{10, 20, 30, \ldots, 100\}$ denoting the number of item types, and $p \in \{3, 5, 10, 15, 20\}$ is the (average) number[8] of items per bin in an optimal solution.
- For any given pair (m, p), we constructed 10 instances. Hence, either set will consist of 500 instances. Note that the items of a specific instance number in that list will always be the same for MS 1 and MS 2. In other words, these sets only differ in terms of the bin sizes.
- For any item type i, the corresponding item size c_i is randomly drawn from a uniform distribution in $[m, 5m] \cap \mathbb{Z}_+$. In addition, the quantity d_i of this item type is drawn randomly from a uniform distribution in $[1, 10] \cap \mathbb{Z}_+$.
- The bin sizes were constructed as follows: At first, the given items are partitioned into sets consisting of p items each. For MS 1, the total size of the items per subset defines the size of precisely one bin. For MS 2, the sizes of these bins are postprocessed by a small integer perturbation averaging zero. In that perturbation step, we pay attention that the value $\sum_{k\in\bar{K}} D_k C_k$ does not change. As a consequence of the overall construction, the bin sizes of 'MS' are rather heterogeneous. More precisely, we have

  $$|\bar{K}| \approx \frac{5.5 \cdot m}{p}$$

  as a rule of thumb, because there are m item types, each of which being (on average) available $(10 - 1)/2 = 5.5$ times, and the bin types originate from assigning p items per bin.

Let us first have a brief look at the size of the ILP models to be dealt with. To this end, the average numbers of variables, i.e., the number of arcs, are listed in Tab. 8. We highlight the following observations:

- As a direct consequence of the instance construction, we see that the larger the parameters m or p, the larger is the number of variables. Overall, the instances considered cover a wide range of difficulty levels, from relatively easy instances having a few thousand variables to very difficult scenarios with almost a million variables.

[8] By construction, at least for MS 1 there is always an optimal solution having precisely p items in any bin.

	$p = 3$		$p = 5$		$p = 10$		$p = 15$		$p = 20$	
m	MS 1	MS 2	MS 1	MS 2	MS 1	MS 2	MS 1	MS 2	MS 1	MS 2
10	2.36	2.48	2.94	2.97	4.25	4.27	5.17	5.18	6.55	6.52
20	10.72	10.91	14.82	15.20	20.22	20.25	24.55	24.73	29.18	29.65
30	29.09	29.63	36.68	37.06	48.07	49.47	60.95	60.26	73.52	73.40
40	46.33	46.87	65.84	67.33	86.67	87.54	110.47	111.72	132.37	133.50
50	81.81	86.24	99.91	100.78	132.58	132.09	179.59	183.19	216.82	217.57
60	123.03	123.18	142.17	142.82	213.32	211.87	274.28	277.42	325.20	319.60
70	150.88	150.90	199.97	199.98	280.37	290.67	368.49	367.57	445.23	452.09
80	208.27	210.68	258.30	262.04	382.70	382.31	505.10	511.58	589.29	603.72
90	271.19	281.85	343.19	344.57	494.78	504.62	621.95	628.85	722.75	729.19
100	344.04	346.58	428.23	433.82	625.36	648.64	793.81	823.98	962.20	976.43
Avg	126.77	128.93	159.20	160.66	228.83	233.17	294.43	299.45	350.31	354.17

Table 8: Average number of variables in units of 10^3 for benchmark sets 'MS 1' and 'MS 2', depending on the construction parameters (m, p).

- In addition to the previous point, we note that the impact of m on the number of variables is much more pronounced. By construction, doubling the number of item types m in addition leads to larger bin sizes, i.e., more "possibilities" to place the items, implying a superlinear increase in the number of item arcs. Instead, doubling the parameter p on average reduces the number of bin types by 50% (fewer bin arcs), but also leads to a factor of two in terms of the bin sizes, i.e., to more item arcs and more interior vertices (entailing more bin arcs). As these two effects pull in different directions, the number of variables does not increase quite as much. Having a closer look at the detailed numerical results (not tabulated here), we see that the number of bin arcs very roughly stays the same, while the number of item arcs always increases by a factor of two. Hence, doubling the parameter p roughly leads to a 50% increase in the number of variables.
- For MS 2, the number of variables is almost always a bit larger than for MS 1. Note that, effectively, the instances of MS 2 arise from those of MS 1 by slightly perturbing the bin sizes. For this reason, larger bin sizes have to be modeled in the arcflow networks of MS 2, typically leading to some additional arcs.

Let us now consider the numerical performance of the flow-based approach for the new benchmark set MS more in detail. To this end, we list the average solution time and the number of instances solved to proven optimality in Tab. 9 (for MS 1) and Tab. 10 (for MS 2). Since these instances can be expected to be particularly hard, the time limit was doubled to $t_{\max} = 600$ seconds for all these computations. From an overall point of view, the data gathered in these two tables are consistent with the observations made for the size of the ILP models, see Tab. 8. By that, we mean that in most of the cases, the instances become harder whenever m or p increase. However, there are also a very few situations where this "monotonicity" does not hold. We attribute this to the randomized solution techniques applied by the Gurobi solver. It is interesting to see that, in many cases, the difference between MS 1 and MS 2 in terms of t and 'opt' is rather small. On the one hand, this reflects the fact that the number of variables were always similar. On the other hand, our internal calculations (not tabulated here) have also shown that positive integrality gaps in MS 2 mainly occur when m and p are small, i.e., for the rather easy instances. In all other cases, there are so many possible item combinations that even the perturbed bin sizes can typically be combined exactly.

Remark 15. *For completeness, we also implemented the reflect model discussed in the final paragraph of Subsect. 3.2. Remember that this network requires a separate graph for any bin type, leading to a large number of variables and constraints. For the instances from MS 1 with* $m \leq 50$ *item types, the latter facts can be seen in Tab. C.11 and Tab. C.12 in Appendix C.*

	$p=3$		$p=5$		$p=10$		$p=15$		$p=20$	
m	t	opt	t	opt	t	opt	t	opt	t	opt
10	<0.1	(10)	<0.1	(10)	0.1	(10)	0.1	(10)	0.2	(10)
20	0.2	(10)	0.6	(10)	1.4	(10)	2.3	(10)	3.2	(10)
30	1.3	(10)	2.9	(10)	5.9	(10)	17.6	(10)	19.2	(10)
40	2.9	(10)	6.6	(10)	19.2	(10)	47.0	(10)	75.1	(10)
50	10.4	(10)	21.5	(10)	135.1	(10)	283.7	(9)	396.9	(7)
60	23.1	(10)	84.1	(10)	344.3	(9)	430.8	(4)	373.4	(5)
70	46.6	(10)	129.7	(10)	437.4	(4)	538.2	(2)	474.4	(3)
80	142.7	(10)	243.6	(9)	487.6	(2)	437.1	(3)	546.2	(1)
90	256.1	(10)	505.1	(5)	600.0	(0)	549.2	(1)	547.5	(1)
100	411.9	(6)	581.8	(1)	546.2	(1)	600.0	(0)	508.4	(2)
Avg/Sum	89.5	(96)	157.6	(85)	257.7	(66)	290.6	(59)	294.4	(59)

Table 9: Numerical results for benchmark set MS 1.

	$p=3$		$p=5$		$p=10$		$p=15$		$p=20$	
m	t	opt	t	opt	t	opt	t	opt	t	opt
10	0.1	(10)	<0.1	(10)	0.1	(10)	0.1	(10)	0.2	(10)
20	0.5	(10)	0.7	(10)	1.4	(10)	2.3	(10)	4.2	(10)
30	2.5	(10)	2.7	(10)	6.5	(10)	11.9	(10)	22.8	(10)
40	3.8	(10)	10.5	(10)	20.0	(10)	95.5	(10)	143.7	(10)
50	14.5	(10)	24.4	(10)	84.5	(10)	422.4	(5)	375.9	(7)
60	48.6	(10)	100.0	(10)	274.6	(8)	529.7	(3)	528.7	(2)
70	62.5	(10)	120.4	(10)	454.2	(3)	447.0	(3)	600.0	(0)
80	206.3	(10)	362.5	(9)	544.1	(1)	491.6	(2)	547.4	(1)
90	342.4	(8)	434.4	(5)	590.5	(1)	478.3	(3)	502.7	(2)
100	484.6	(3)	586.9	(1)	547.4	(1)	452.4	(3)	554.2	(1)
Avg/Sum	116.6	(91)	164.3	(85)	252.3	(64)	293.1	(59)	328.0	(53)

Table 10: Numerical results for benchmark set MS 2.

Interestingly, in these tables, the difference in size becomes particularly evident for the number of constraints with a scaling factor of up to 10 *for* $(m, p) = (50, 3)$*, whereas we mostly have a factor between* 1 *and* 3.5 *for the number of variables. In fact, moving from arcflow to reflect leads to* $|\bar{K}| \approx \frac{5.5 \cdot m}{p}$ *separate graphs modeling one and the same interior vertex several times, thus causing much more flow conservation constraints. However, due to the different bin sizes, not every single vertex of arcflow will appear in any of these seperate graphs. In addition, the reduction in terms of active nodes coming from just considering the first half of the graph is typically much larger than* 50%*. This is because, the density of the nodes in arcflow is particularly high for the second half of a bin, which is not modeled explicitly anymore in a reflect network.*

Due to the considerably larger ILP models, reflect already struggles to solve instances with $m = 40$ *to proven optimality, see Tab. C.13 and Tab. C.14 in Appendix C. Again, in many cases, the performance is slightly better for MS 1 than for MS 2. While all the precise numerical data can be found in Appendix C, here we just present the performance profile displayed in Fig. 3 to enable a rough overview.*

5. Conclusions

In this article, we considered a variant of the bin packing problem, called the overflowing bin packing problem, which was recently introduced by the authors of [5]. In this scenario, the total amount of under- and underutilization of the given bins has to be minimized. At first, we

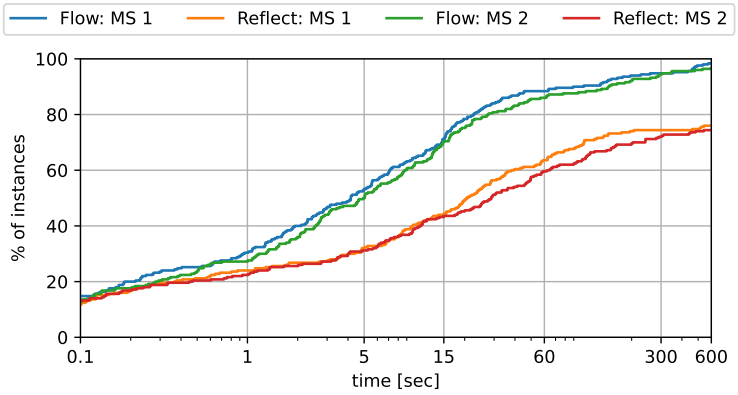

Figure 3: The percentage number of instances solved to proven optimality over time. Only the instances with $m \leq 50$, i.e., 250 instances, are considered. For any subset (MS 1 and MS 2), we see that the flow model from Subsect. 3.1 is clearly superior to a bin-specific reflect formulation. Moreover, we see that, for both ILP formulations, the instances of MS 1 are a bit easier, since the LP bound is always exact.

separated the problem under consideration from other neighboring fields in discrete optimization, establishing the former as an independent subject of research. We then reviewed an ILP model from the literature and stated a closed-form term for its LP value. The main contribution of this article was given by introducing a new flow-based ILP formulation, the LP bound of which was proven to dominate that of the assignment-based approach from the literature. Based on extensive numerical tests for differently characterized benchmark sets, the new ILP model was shown to be able to solve all existing benchmark instances to proven optimality, clearly beating the previous state of the art. Remarkably, the time taken to solve these instances was very small in all these scenarios.

Even if the flow model presented here is a very powerful tool for solving the OBPP in a short time, the limits of exact solvability can also be reached for this approach by constructing much larger instances. In addition to the ILP formulations discussed here, the investigation of heuristics is therefore of high relevance in future research. The authors of [5] have already done some empirical preliminary work on this, but many theoretical questions, such as the worst-case behavior of these approximation approaches, remain unsolved. Another important direction of research is to further improve the flow model introduced in this article, e.g., by identifying valid inequalities or using reduced state spaces. In particular, we would like to raise the so far open question of whether and, if so, how a reflect model can be set up for the OBPP as well. In this regard, we have done important preliminary work by clearly extracting and naming the main weak point of previous modeling approaches, related to skiving stock patterns of different threshold lengths.

As a last point, we also mention the investigation of the additive integrality gap of the arcflow model. For the ordinary BPP, this gap is conjectured to be bounded by a very small constant, see [39]. Can similar results be obtained for the OBPP as well?

CRediT authorship contribution statement

John Martinovic: Conceptualization; Methodology; Formal analysis; Writing - Original Draft

Nico Strasdat: Conceptualization; Methodology; Formal analysis; Software; Visualization

Declarations

Funding: Not applicable. **Conflicts of interest:** The authors declare that they do not have any conflicts of interest. **Availability of data and material:** For clarity, the instances used in this paper have been gathered

together at `https://github.com/wotzlaff/obpp-instances`. **Code availability:** The instances were solved by the commercial software Gurobi.

References

[1] Álves, C., Valério de Carvalho, J.M.: A stabilized branch-and-price-and-cut algorithm for the multiple length cutting stock problem. Computers & Operations Research 35(4), 1315–1328 (2008)

[2] Alvim, A.C.F., Ribeiro, C.C.: A hybrid bin-packing heuristic to multiprocessor scheduling. Lecture Notes in Computer Science, Vol. 3059. Springer-Verlag, Berlin, 1–13 (2004)

[3] Ball, M.O., Magazine, M.J.: Sequencing of Insertions in Printed Circuit Board Assembly. Operations Research 36(2), 192–201 (1988)

[4] Belov, G., Scheithauer, G.: A cutting plane algorithm for the one-dimensional cutting stock problem with multiple stock lengths. European Journal of Operational Research 141(2), 274–294 (2002)

[5] Cerrone, C., Dragone, R., Sciomachen, A.: Overflowing bin packing problem: the preservation of digital documents in archival information packages. Preprint, University of Genoa (2023) (available online: `http://dx.doi.org/10.2139/ssrn.4647288`[9])

[6] Chen, B.: Parallel Scheduling for Early Completion. Handbook of Scheduling – Algorithms, Models, and Performance Analysis. Chapman and Hall/CRC (2004)

[7] Cheng, T.C.E, Sin, C.C.S: A state-of-the-art review of parallel-machine scheduling research. European Journal of Operational Research 47(3), 271–292 (1990)

[8] Coffman, E.G., Garey, M.R., Johnson, D.S.: An application of bin-packing to multiprocessor scheduling. SIAM Journal on Computing 7, 1–17 (1978)

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

[10] Crainic, T., Perboli, G., Rei, W., Tadei, R.: Efficient lower bounds and heuristics for the variable cost and size bin packing problem. Computers & Operations Research 38(11), 1474–1482 (2011)

[11] Delorme, M., Iori, M., Martello, S.: Bin Packing and Cutting Stock Problems: Mathematical Models and Exact Algorithms. European Journal of Operational Research 255, 1–20 (2016)

[12] de Lima, V.L., Alves, C., Clautiaux, F., Iori, M., Valério de Carvalho, J.M.: Arc Flow Formulations Based on Dynamic Programming: Theoretical Foundations and Applications. European Journal of Operational Research 296(1), 3–21 (2022)

[13] de Lima V.L., Iori M., Miyazawa F.K.: Exact solution of network flow models with strong relaxations. Mathematical Programming 197, 813–846 (2023)

[14] Dell'Amico, M., Delorme, M., Iori, M., Martello, S.: Mathematical models and decomposition methods for the multiple knapsack problem. European Journal of Operational Research 274(3),886–899 (2019)

[15] Dell'Amico, M., Iori, M., Martello, S., Monaci, M.: Heuristic and Exact Algorithms for the Identical Parallel Machine Scheduling Problem. INFORMS Journal on Computing 20(3), 333–344 (2008)

[16] Dell'Amico, M., Martello, S.: Optimal scheduling of tasks on identical parallel processors. ORSA Journal on Computing 7, 191–200 (1995)

[17] Dell'Amico, M., Martello, S.: A note on exact algorithms for the identical parallel machine scheduling problem. European Journal of Operations Research 160, 576–578 (2005)

[18] Delorme, M., Iori, M.: Enhanced pseudo-polynomial formulations for bin packing and cutting stock problems. INFORMS Journal on Computing 32(1), 101–119 (2020)

[19] França, P.M., Gendreau, M., Laporte, G., Müller, F.M.: A composite heuristic for the identical parallel machine scheduling problem with minimum makespan objective. Computers & Operations Research 21, 205–210 (1994)

[9]The original PDF version is currently being revised by the authors and is therefore not available at the moment. However, a cached HTML version of the article can still be accessed at `https://scholar.googleusercontent.com/scholar?q=cache:_QyMLldSwaIJ:scholar.google.com`.

[20] Frangioni, A., Necciari, E., Scutellà, M.G.: A multi-exchange neighborhood for minimum makespan machine scheduling problems. Journal of combinatorial Optimization 8, 195–220 (2004)

[21] Frias Faria, A., Ricardo de Souza, S., Martins de Sá, E.: A mixed-integer linear programming model to solve the Multidimensional Multi-Way Number Partitioning Problem. Computers & Operations Research 127, Article Number 105133 (2021)

[22] Friesen, D.K., Deuermeyer, B.L.: Analysis of Greedy Solutions for a Replacement Part Sequencing Problem. Mathematics of Operations Research 6(1), 74–87 (1981)

[23] Friesen, D.K., Deuermeyer, B.L., Langston, M.A.: Scheduling to Maximize the Minimum Processor Finish Time in a Multiprocessor System. SIAM Journal on Algebraic Discrete Methods 3(2), 190–196 (1982)

[24] Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness. W. H. Freeman and Company (1979)

[25] Graham, R.L.: Bounds on Multiprocessing Timing Anomalies. SIAM Journal on Applied Mathematics 17 (2), 416–429 (1969)

[26] Hähnel, M., Martinovic, J., Scheithauer, G., Fischer, A., Schill, A., Dargie, W.: Extending the Cutting Stock Problem for Consolidating Services with Stochastic Workloads. IEEE Transactions on Parallel and Distributed Systems 29(11), 2478–2488 (2018)

[27] Hemmelmayr, V., Schmid, V., Blum, C.: Variable neighbourhood search for the variable sized bin packing problem. Computers & Operations Research 39(5), 1097–1108 (2012)

[28] Hochbaum, D.S., Shmoys, D.B.: Using Dual Approximation Algorithms for Scheduling Problems: Theoretical and Practical Results. Journal of the Association for Computing Machinery 34(1), 144–162 (1987)

[29] Kojić, J.: Integer linear programming model for multidimensional two-way number partitioning problem. Computers and Mathematics with Applications 60, 2302–2308 (2010)

[30] Korf, R.E.: Objective Functions for Multi-Way Number Partitioning. Proceedings of the Third Annual Symposium on Combinatorial Search (SOCS-10), 71–72 (2010)

[31] Markakis, E., Psomas, C.A.: On Worst-Case Allocations in the Presence of Indivisible Goods. Lecture Notes in Computer Science, vol 7090, 278–289 (2011)

[32] Mertens, S.: The Easiest Hard Problem: Number Partitioning. arXiv Preprint (available online: `https://arxiv.org/abs/cond-mat/0310317`) (2013)

[33] Martinovic, J., Delorme, M., Iori, M., Scheithauer, G., Strasdat, N.: Improved Flow-based Formulations for the Skiving Stock Problem. Computers & Operations Research 113, Article 104770 (2020)

[34] Martinovic, J., Scheithauer, G., Valério de Carvalho, J.M.: A Comparative Study of the Arcflow Model and the One-Cut Model for one-dimensional Cutting Stock Problems. European Journal of Operational Research 266(2), 458–471 (2018)

[35] Martinovic, J., Strasdat, N., Valério de Carvalho, J.M., Furini, F.: A Combinatorial Flow-based Formulation for Temporal Bin Packing Problems. European Journal of Operational Research 307(2), 554–574 (2023)

[36] Mokotoff, E.: Parallel machine scheduling problems: A survey. Asia-Pacific Journal of Operational Research 18, 193–242 (2001)

[37] Mokotoff, E.: An exact algorithm for the identical parallel machine scheduling problem. European Journal of Operations Research 152, 758–769 (2004)

[38] Nemhauser, G., Wolsey, L.: Integer and Combinatorial Optimization. Wiley, New York (1988)

[39] Scheithauer, G.: Introduction to Cutting and Packing Optimization – Problems, Modeling Approaches, Solution Methods. International Series in Operations Research & Management Science 263, Springer, 1.Edition (2018)

[40] Schreiber, E.L.: Optimal Multi-Way Number Partitioning. PhD dissertation, University of California (2014)

[41] Tsai, L.-H.: The Loading and Scheduling Problems in Flexible Manufacturing Systems. PhD Dissertation, University of California, Berkeley (1987)

[42] Valério de Carvalho, J.M.: LP models for bin packing and cutting stock problems. European Journal of Operations Research 141(2), 253–273 (2002)

[43] Walter, R.: Comparing the minimum completion times of two longest-first scheduling-heuristics. Central European Journal of Operations Research 21, 125–139 (2013)

[44] Wolsey, L.A.: Valid Inequalities, Covering Problems and Discrete Dynamic Programs. Annals of Discrete Mathematics 1, 527–538 (1977)

[45] Wu, Y.: Energy efficient virtual machine placement in data centers. Master thesis, Queensland University of Technology (2013)

Appendix A. Proof of Theorem 6

The proof consists of two parts. At first, we show that the term specified in (8) is indeed a lower bound to the LP value. In the second part, we discuss the tightness of this value.

(i) Let $(\boldsymbol{x}, \boldsymbol{s})$ denote any feasible solution to the LP problem. By combining the various constraints given, we obtain

$$\begin{aligned} \sum_{k\in K} s_k &\overset{(4),(5)}{\geq} \sum_{k\in K} \left| C_k - \sum_{i\in I} c_i \cdot x_{ik} \right| \geq \left| \sum_{k\in K} C_k - \sum_{k\in K}\sum_{i\in I} c_i \cdot x_{ik} \right| \\ &= \left| \sum_{k\in K} C_k - \sum_{i\in I} c_i \sum_{k\in K} x_{ik} \right| \overset{(3)}{=} \left| \sum_{k\in K} C_k - \sum_{i\in I} c_i d_i \right|. \end{aligned}$$

Since the solution considered was arbitrarily chosen, the same inequality applies for an optimal solution (and the respective optimal value on the left-hand side).

(ii) We fill the bins according to the *fractional greedy strategy (FGS)* displayed in Alg. 1.

Algorithm 1 Fractional Greedy Strategy (FGS)

Input: Instance E of the OBPP with items and bins, both sorted according to Remark 1.

1: Set $k := 1$ and $i := 1$.
2: **while** $i \leq m$ **do**
3: Compute the maximum fraction $x_{ik} \in [0, d_i]$ of item i that can be assigned to bin k.
4: **if** $x_{ik} = d_i$ **then**
5: Assign the complete item type i to bin k. Set $i := i + 1$.
6: **else**
7: Assign the portion $x_{ik} < d_i$ of item type i to bin k. Set $k := k + 1$ and update $d_i := d_i - x_{ik}$.
8: **end if**
9: **end while**

Output: objective value z_{LP}^{ass}

After having assigned all the items, we set $s_k := \left| C_k - \sum_{i\in I} c_i x_{ik} \right|$ for all $k \in K$. Obviously, FGS leads to an assignment with the following property: There is an index $k^\star \in K$, such that precisely the bins with $k \leq k^\star$ are balanced.

With this in mind, let us now consider two cases:

- **Case 1:** $\sum_{k\in K} C_k - \sum_{i\in I} c_i d_i \geq 0$
  In this scenario, the bin capacities are sufficiently large to accommodate all items without closing any bin. To be more precise, the bins $1, \ldots, k^\star$ will be balanced, whereas the remaining bins (if any) will be open. As a consequence of that, we have

$$C_k - \sum_{i\in I} c_i x_{ik} \geq 0 \tag{A.1}$$

  for all $k \in K$. Then, the objective value of the assignment constructed with Alg. 1 is given by

$$\begin{aligned} \sum_{k\in K} s_k &= \sum_{k\in K} \left| C_k - \sum_{i\in I} c_i x_{ik} \right| \overset{(A.1)}{=} \sum_{k\in K} \left(C_k - \sum_{i\in I} c_i x_{ik} \right) \\ &= \sum_{k\in K} C_k - \sum_{i\in I} c_i \sum_{k\in K} x_{ik} \overset{(1)}{=} \sum_{k\in K} C_k - \sum_{i\in I} c_i d_i = \left| \sum_{k\in K} C_k - \sum_{i\in I} c_i d_i \right|, \end{aligned}$$

where the last equation is true due to the assumption of the current case. Note that, in the very first step we made use of the definition of s_k, $k \in K$, that we used to introduce the linear assignment model.

- **Case 2:** $\sum_{k\in K} C_k - \sum_{i\in I} c_i d_i < 0$
  The discussion of this case can be done in an analogous way. The only thing to modify is that Alg. 1 now leads to an assignment in which the bins $1, \dots, k^\star$ are balanced, whereas the remaining bins are closed. Hence, we have
  $$C_k - \sum_{i\in I} c_i x_{ik} \leq 0 \tag{A.2}$$
  for all $k \in K$.

So, in both cases, the objective value matches the lower bound found in Part (i), and the proof is complete.

Appendix B. Proof of Theorem 12

Let $(\boldsymbol{\xi}, \boldsymbol{y})$ denote an arbitrary (but fixed) feasible LP solution. The network, the arcflow model is based on, is an acyclic directed graph. Hence, the $\boldsymbol{\xi}$-component of $(\boldsymbol{\xi}, \boldsymbol{y})$ decomposes into a set of paths each of which having constant flow along its arcs. To this end, we define the sets Γ_k, $k \in \bar{K}$, collecting the paths associated with the bins of type $k \in \bar{K}$. For any such path $\gamma \in \Gamma := \bigcup_{k\in\bar{K}} \Gamma_k$ we further introduce the following abbreviations:

- $\xi(\gamma)$: the constant flow[10] along γ,
- $v(\gamma)$: the last node along γ before the sink node.

We observe that the objective function can be rewritten as

$$\begin{aligned}
z^{af}(\boldsymbol{\xi}, \boldsymbol{y}) &= \sum_{i\in I} c_i y_i + \sum_{(v,t_k)\in\mathcal{A}_{bin}} \text{cost}(v, t_k) \cdot \xi_{v,t_k} \\
&\overset{(10)}{=} \sum_{i\in I} c_i \left(d_i - \sum_{(u,v)\in\mathcal{A}_{item}: v-u=c_i} \xi_{uv} \right) + \sum_{(v,t_k)\in\mathcal{A}_{bin}} \text{cost}(v, t_k) \cdot \xi_{v,t_k} \\
&= \sum_{i\in I} c_i d_i - \sum_{(u,v)\in\mathcal{A}_{item}} (v-u) \cdot \xi_{uv} + \sum_{(v,t_k)\in\mathcal{A}_{bin}} \text{cost}(v, t_k) \cdot \xi_{v,t_k} \\
&= \sum_{i\in I} c_i d_i + \sum_{e\in\mathcal{A}} \alpha_e \xi_e
\end{aligned}$$

with $\mathcal{A} = \mathcal{A}_{\text{bin}} \cup \mathcal{A}_{\text{item}}$ and

$$\alpha_e := \begin{cases} -(v-u), & \text{if } (u,v) \in \mathcal{A}_{item}, \\ \text{cost}(u, t_k), & \text{if } (u,v) = (u,t_k) \in \mathcal{A}_{bin}. \end{cases}$$

Using the flow decomposition property, we can rewrite the last term as

$$\sum_{e\in\mathcal{A}} \alpha_e \xi_e = \sum_{k\in\bar{K}} \sum_{\gamma\in\Gamma_k} \sum_{e\in\gamma} \alpha_e \xi_e = \sum_{k\in\bar{K}} \sum_{\gamma\in\Gamma_k} \xi(\gamma) \sum_{e\in\gamma} \alpha_e.$$

For any $k \in \bar{K}$ and $\gamma \in \Gamma_k$ we now obtain

$$\sum_{e\in\gamma} \alpha_e = -v(\gamma) + \text{cost}(v(\gamma), t_k) = -v(\gamma) + |C_k - v(\gamma)| = \max\{C_k - 2v(\gamma), -C_k\},$$

because, by definition, the sizes of the item arcs used in a path γ always sum up to $v(\gamma)$.

We now move on to case studies:

(I) Using always the first term of the maximum, we see that

$$\begin{aligned}
\sum_{e\in\mathcal{A}} \alpha_e \xi_e &\geq \sum_{k\in\bar{K}} \sum_{\gamma\in\Gamma_k} \xi(\gamma)(C_k - 2v(\gamma)) = \sum_{k\in\bar{K}} \sum_{\gamma\in\Gamma_k} \xi(\gamma) C_k - 2 \sum_{k\in\bar{K}} \sum_{\gamma\in\Gamma_k} \xi(\gamma) v(\gamma) \\
&= \sum_{k\in\bar{K}} C_k \sum_{\gamma\in\Gamma_k} \xi(\gamma) - 2 \sum_{\gamma\in\Gamma} \xi(\gamma) v(\gamma) \overset{(11)}{=} \sum_{k\in\bar{K}} C_k D_k - 2 \sum_{\gamma\in\Gamma} \xi(\gamma) v(\gamma) \\
&\overset{(10)}{\geq} \sum_{k\in\bar{K}} C_k D_k - 2 \sum_{i\in I} c_i d_i.
\end{aligned}$$

In that calculation, the following arguments were used:

[10] Strictly speaking, the values $\xi(\gamma)$, $\gamma \in \Gamma$, are not well-defined due to the non-uniqueness of the flow decomposition. However, the argumentation used in this proof is true for any such flow decomposition. So, no harm will arise from this slight inconsistency.

- At first, we note that any path $\gamma \in \Gamma_k$, $k \in \bar{K}$, has to terminate with some bin arc $(v, t_k) \in \mathcal{A}_{\text{bin}}$. Hence, Conditions (11) can be applied.
- Secondly, the arguments used in the very first equation chain in this proof can be applied backwards. By that, we mean the following:

$$\sum_{\gamma\in\Gamma} \xi(\gamma)v(\gamma) = \sum_{(u,v)\in\mathcal{A}_{\text{item}}} (v-u)\cdot\xi_{uv} = \sum_{i\in I} c_i \sum_{(u,v)\in\mathcal{A}_{\text{item}}: v-u=c_i} \xi_{uv} \overset{(10)}{\leq} \sum_{i\in I} c_i d_i.$$

  In other terms, we just made use of the fact that a weighted sum of the last "regular" nodes $v(\gamma)$, $\gamma \in \Gamma$, over all paths cannot exceed the total item size, since only item arcs were used to reach these nodes.

(II) On the other hand, using always the second term of the maximum leads to

$$\sum_{e\in\mathcal{A}} \alpha_e \xi_e \geq -\sum_{k\in\bar{K}} \sum_{\gamma\in\Gamma_k} \xi(\gamma) C_k = -\sum_{k\in\bar{K}} C_k \sum_{\gamma\in\Gamma_k} \xi(\gamma) \overset{(11)}{=} -\sum_{k\in\bar{K}} C_k D_k.$$

Taking both estimates together, we obtain 17 as a consequence of the very first equation chain appearing in the proof.

Appendix C. Further Numerical Results

	$p=3$		$p=5$		$p=10$		$p=15$		$p=20$	
m	Flow	Reflect	Flow	Reflect	Flow	Reflect	Flow	Reflect	Flow	Reflect
10	2.36	2.35	2.94	3.17	4.25	5.04	5.17	5.53	6.55	5.96
20	10.72	15.00	14.82	24.96	20.22	37.12	24.55	44.35	29.18	42.55
30	29.09	52.03	36.68	78.39	48.07	123.99	60.95	152.09	73.52	160.30
40	46.33	106.53	65.84	187.23	86.67	276.22	110.47	342.00	132.37	350.05
50	81.81	214.84	99.91	341.54	132.58	531.38	179.59	646.27	216.82	708.67

Table C.11: Average number of variables in units of 10^3 for the two flow-based approaches. Here, a subset of MS 1 is considered.

	$p=3$		$p=5$		$p=10$		$p=15$		$p=20$	
m	Flow	Reflect	Flow	Reflect	Flow	Reflect	Flow	Reflect	Flow	Reflect
10	0.21	0.51	0.28	0.56	0.44	0.70	0.58	0.72	0.79	0.74
20	0.48	2.11	0.67	2.57	1.00	2.92	1.26	3.02	1.63	2.80
30	0.85	5.48	1.15	5.76	1.56	6.47	2.07	7.18	2.59	7.17
40	1.04	8.60	1.48	10.38	2.14	10.98	2.77	11.79	3.51	11.64
50	1.46	14.69	1.83	15.59	2.63	17.06	3.66	18.25	4.53	18.96

Table C.12: Average number of constraints in units of 10^3 for the two flow-based approaches. Here, a subset of MS 1 is considered.

	$p=3$				$p=5$				$p=10$				$p=15$				$p=20$			
	Flow		Reflect		Flow		Reflect		Flow		Reflect		Flow		Reflect		Flow		Reflect	
m	t	opt	t	opt	t	opt	t	opt	t	opt	t	opt	t	opt	t	opt	t	opt	t	opt
10	0.0	(10)	0.0	(10)	0.0	(10)	0.0	(10)	0.1	(10)	0.1	(10)	0.1	(10)	0.2	(10)	0.2	(10)	0.2	(10)
20	0.2	(10)	0.6	(10)	0.6	(10)	2.1	(10)	1.4	(10)	6.1	(10)	2.3	(10)	11.6	(10)	3.2	(10)	8.2	(10)
30	1.3	(10)	6.8	(10)	2.9	(10)	19.7	(10)	5.9	(10)	57.9	(10)	17.6	(10)	93.2	(10)	19.2	(10)	170.9	(9)
40	2.9	(10)	33.3	(10)	6.6	(10)	50.9	(10)	19.2	(10)	586.2	(1)	47.0	(10)	542.6	(1)	75.1	(10)	600.0	(0)
50	10.4	(10)	25.3	(10)	21.5	(10)	254.6	(9)	135.1	(10)	600.0	(0)	283.7	(9)	600.0	(0)	396.9	(7)	600.0	(0)
Avg/Sum	**3.0**	**(50)**	13.2	(50)	**6.3**	**(50)**	65.5	(49)	**32.3**	**(50)**	250.1	(31)	**70.1**	**(49)**	249.5	(31)	**98.9**	**(47)**	275.9	(29)

Table C.13: Comparison between the two flow-based formulations for a subset of instances from MS 1.

	$p=3$				$p=5$				$p=10$				$p=15$				$p=20$			
	Flow		Reflect		Flow		Reflect		Flow		Reflect		Flow		Reflect		Flow		Reflect	
m	t	opt	t	opt	t	opt	t	opt	t	opt	t	opt	t	opt	t	opt	t	opt	t	opt
10	0.1	(10)	0.1	(10)	0.0	(10)	0.1	(10)	0.1	(10)	0.1	(10)	0.1	(10)	0.1	(10)	0.2	(10)	0.2	(10)
20	0.5	(10)	1.6	(10)	0.7	(10)	2.1	(10)	1.4	(10)	4.7	(10)	2.3	(10)	8.7	(10)	4.2	(10)	9.8	(10)
30	2.5	(10)	11.9	(10)	2.7	(10)	24.6	(10)	6.5	(10)	62.7	(10)	11.9	(10)	109.4	(10)	22.8	(10)	214.7	(9)
40	3.8	(10)	39.6	(10)	10.5	(10)	138.1	(9)	20.0	(10)	459.9	(4)	95.5	(10)	600.0	(0)	143.7	(10)	600.0	(0)
50	14.5	(10)	265.4	(7)	24.4	(10)	341.8	(7)	84.5	(10)	600.0	(0)	422.4	(5)	600.0	(0)	375.9	(7)	600.0	(0)
Avg/Sum	**4.3**	**(50)**	63.7	(47)	**7.7**	**(50)**	101.3	(46)	**22.5**	**(50)**	225.5	(34)	**106.4**	**(45)**	263.6	(30)	**109.4**	**(47)**	284.9	(29)

Table C.14: Comparison between the two flow-based formulations for a subset of instances from MS 2.