Bounds and Heuristic Algorithms for the Bin Packing Problem with Minimum Color Fragmentation

Mathijs Barkel[a], Maxence Delorme[a], Enrico Malaguti[b], Michele Monaci[b,*]

[a] *Tilburg University, Department of Econometrics and Operations Research, 5037 AB Tilburg, The Netherlands,*

[b] *Università di Bologna, Dipartimento di Ingegneria dell'Energia Elettrica e dell'Informazione "Guglielmo Marconi", Viale del Risorgimento, 2, 40136, Bologna, Italy*

Abstract

In this paper, we consider a recently introduced packing problem in which a given set of weighted items with colors has to be packed into a set of identical bins, while respecting capacity constraints and minimizing the total number of times that colors appear in the bins. We review exact methods from the literature and present a fast lower bounding procedure that, in some cases, can also provide an optimal solution. We theoretically study the worst-case performance of the lower bound and computationally test our solution method on a large benchmark of instances from the literature: quite surprisingly, all of them are optimally solved by our procedure in a few seconds, including those for which the optimal solution value was still unknown. Thus, we introduce additional harder instances, which are used to evaluate the performance of a constructive heuristic method and of a tabu search algorithm. Results on the new instances show that the tabu search produces considerable improvements over the heuristic solution, with a limited computational effort.

Keywords: packing, lower bounds, tabu search, computational experiments

1. Introduction

In the *Bin Packing Problem* (BPP) we are given a set of *items*, each having a positive *weight*, and an unlimited number of identical *bins*, each having a positive *capacity*. The problem asks to pack all the items into the minimum number of bins so that the total weight of the items packed in each bin does not exceed the capacity. This problem is known to be strongly $\mathcal{NP}$-hard and has applications in different areas, including e.g., logistics, cutting and packing, and telecommunications.

In this paper, we consider a packing problem that is strictly related to the BPP. We are given a limited set $\mathcal{B}$ of bins, containing exactly B identical bins, each with capacity W, and a set $\mathcal{I}$ containing I items. Besides the weight w_i, each item $i \in \mathcal{I}$ is characterized by a *color* $c_i \in \mathcal{C}$,

*Corresponding author

Email addresses: m.j.barkel@tilburguniversity.edu (Mathijs Barkel), m.delorme@tilburguniversity.edu (Maxence Delorme), enrico.malaguti@unibo.it (Enrico Malaguti), michele.monaci@unibo.it (Michele Monaci)

where $\mathcal{C} = \{1, 2, \ldots, C\}$ is a given set of colors which induces a partition of the item set into subsets $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_c$. As is common in bin packing problems, we make the natural assumption that both the bin capacity and the items weights are positive integers. The problem requires to pack all items into the available bins without exceeding their capacity while minimizing the color fragmentation of the solution. More precisely, the color fragmentation for a color is given by the number of different bins that contain at least one item of that color, and the objective is to minimize the overall value of this figure. The resulting problem, denoted as the *Bin Packing Problem with Minimum Color Fragmentation* (BPPMCF), has been introduced in the literature in Bergman et al. (2019). Recently, the BPPMCF has been addressed by the same authors (Mehrani et al. 2022), where possible applications in surgical scheduling and group event seating were considered. In the latter paper, several mathematical formulations and solution algorithms were proposed, and strong $\mathcal{NP}$–hardness of the problem was reported for the case $B \geq 3$. We observe that, in this case, even finding a feasible solution for a BPPMCF instance is a strongly $\mathcal{NP}$–complete problem, as it reduces to solving the BPP instance induced by ignoring colors.

For the sake of completeness, we now review the descriptive mathematical model introduced in Bergman et al. (2019) and Mehrani et al. (2022). The formulation involves binary variables x_{bi} representing the assignment of item i to bin b and binary variables y_{bc} that indicate whether bin b includes some items of color c or not ($i \in \mathcal{I}$, $b \in \mathcal{B}$, $c \in \mathcal{C}$). The formulation is thus as follows

$$\min \quad \sum_{b \in \mathcal{B}} \sum_{c \in \mathcal{C}} y_{bc} \tag{1}$$

$$\text{s.t.} \quad \sum_{b \in \mathcal{B}} x_{bi} = 1 \qquad i \in \mathcal{I} \tag{2}$$

$$\sum_{i \in \mathcal{I}} w_i \, x_{bi} \leq W \qquad b \in \mathcal{B} \tag{3}$$

$$\sum_{i \in \mathcal{I}_c} w_i \, x_{bi} \leq W y_{bc} \qquad b \in \mathcal{B},\, c \in \mathcal{C} \tag{4}$$

$$x_{bi} \in \{0, 1\} \qquad b \in \mathcal{B},\, i \in \mathcal{I} \tag{5}$$

$$y_{bc} \in \{0, 1\} \qquad b \in \mathcal{B},\, c \in \mathcal{C}. \tag{6}$$

This model is then strengthened by adding the following inequalities

$$x_{bi} \leq y_{bc} \qquad b \in \mathcal{B},\, c \in \mathcal{C},\, i \in \mathcal{I}_c \tag{7}$$

$$\sum_{b \in \mathcal{B}} y_{bc} \geq L_c \qquad c \in \mathcal{C}, \tag{8}$$

where L_c denotes any lower bound on the fragmentation of each color c, i.e., on the number of bins that contain at least one item in set $\mathcal{I}_c$. In Mehrani et al. (2022), L_c is set to the value of the well-known lower bound L_2 for the BPP instance induced by item set $\mathcal{I}_c$ (see, Martello and Toth

1990).

Formulation (1)–(8) is denoted as `IP1` in Mehrani et al. (2022), where also a similar model is proposed that exploits the characteristic of most real instances to include many identical items (i.e., with the same weight and same color). This more effective model is referred to as `IP2`, and is one of the methods that will be computationally considered in our experiments (see Section 4). In addition, we will consider two further exact approaches that turned out to be very effective in Mehrani et al. (2022), denoted as `RM2-GIFF` and `EM`, respectively. The former is a decision diagram combined with a general integer flow formulation, whereas the latter is based on a set-partitioning formulation of the problem and can be interpreted as a Dantzig-Wolfe reformulation of model `IP2`.

Some different variants of the bin packing problem in which items are associated with colors have been proposed in the literature; however, in these problems, colors typically induce constraints on the feasibility of a solution. For example, in the co-printing problem, items have multiple colors but no weight and bin capacity limits the total number of different colors for its items. This problem, which finds applications in beverage package printing, has been considered in Peeters and Degraeve (2004) and in Kochetov and Kondakov (2017). General incompatibility between items of different colors is addressed in the bin packing problem with conflicts (see, Gendreau et al. 2004, Fernandes Muritiba et al. 2010).

The rest of the paper is organized as follows. In Section 2 we consider bounding procedures that are based on the solution of a BPP problem. We introduce both a lower bound, for which we analyze the worst-case performance, and two alternative heuristic algorithms. Section 3 presents a metaheuristic scheme that is based on the tabu search paradigm. In Section 4 the proposed algorithms are computationally evaluated and compared with state-of-the-art methods on a large benchmark of instances from the literature. In addition, a new set of challenging instances is introduced, allowing for a more accurate analysis of the performance of the methods. Finally, Section 5 draws some conclusions and indicates future research directions.

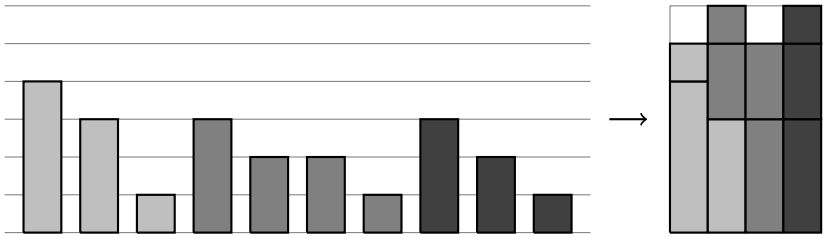
Throughout the paper, we will make use of the following example for describing solution methods.

Example 1. *Let us consider the following BPPMCF instance with* $B = 4$*,* $W = 6$*,* $C = 3$*,* $I = 10$*, items* $\mathcal{I} = \{i_1, i_2, \ldots, i_{10}\}$ *with weights equal to* $\{4, 3, 1, 3, 2, 2, 1, 3, 2, 1\}$*. Item colors are equal to* $\{1, 1, 1, 2, 2, 2, 2, 3, 3, 3\}$*, i.e.,* $\mathcal{I}_1 = \{i_1, i_2, i_3\}$*,* $\mathcal{I}_2 = \{i_4, i_5, i_6, i_7\}$ *and* $\mathcal{I}_3 = \{i_8, i_9, i_{10}\}$*. This instance, together with a corresponding optimal solution, is shown in Figure 1, where color* 1 *is light gray, color* 2 *is medium gray, and color* 3 *is dark gray. This solution uses all* 4 *bins. The items with color* 1 *are spread over two bins, as are the items with color* 2*, whereas the items with color* 3 *are all in the same bin. Therefore, the overall color fragmentation of this solution is equal to* $2 + 2 + 1 = 5$*.*

2. BPP-based bounding procedures

In this section, we consider lower and upper bounds that can be computed by solving BPP instances. We point out that, even though the BPP is strongly $\mathcal{NP}$-hard, several approaches have

Figure 1: Example of a BPPMCF instance and a corresponding optimal solution.



been proposed in the literature for solving BPP instances effectively. The first effective proposals were combinatorial lower bounding procedures and branch-and-bound algorithms described in Martello and Toth (1990) and pseudo-polynomial arcflow formulations by Valério de Carvalho (1999). These methods are reviewed in the survey of Delorme et al. (2016). More recent methods include enhancements of arcflow formulations, allowing effective solution by means of state-of-the-art ILP solvers (see Delorme and Iori 2020), and tailored branch-and-price algorithms (see Pessoa et al. 2020; Wei et al. 2020).

Section 2.1 briefly reports the arcflow formulation that is used in our algorithms; Section 2.2 reviews a simple lower bound introduced in Mehrani et al. (2022) and analyzes its worst-case performance; finally, Section 2.3 describes two heuristic algorithms for computing a feasible solution for a BPPMCF instance. Note that, even though our algorithms use the arcflow formulation to solve the BPP, they could also use any black-box procedure that returns an optimal solution for a given BPP instance.

2.1. The arcflow formulation

Introduced by Valério de Carvalho (1999) and based on the earlier works of Shapiro (1968) and Wolsey (1977), the standard arcflow formulation considers a directed multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ where vertex set $\mathcal{V} = \{0, 1, \ldots, W\}$ includes a vertex for each capacity value, and where arc set $\mathcal{A}$ includes for each item $i \in \mathcal{I}$, $W - w_i + 1$ arcs associated with packing that item, namely all arcs originating from a vertex $d \in \{0, 1, \ldots, W - w_i\}$ and terminating in vertex $e = d + w_i$. For notational convenience, each arc is identified by a triplet (d, e, i). We point out that the resulting graph has pseudo-polynomial size, which can be hard to manage in practice. For this reason, procedures for avoiding to construct redundant vertices and arcs have been proposed in Valério de Carvalho (1999). In brief, items are sorted according to non-increasing weight, and for a given item i only the arcs $(v, v + w_i, i)$ with $v \leq W - w_i$ are considered for which v is either 0 or the head of another arc associated with a previously considered item. Further improvements based on effective reduction strategies and efficient procedures for the definition of the graph have been extensively studied in recent years (see Brandão and Pedroso 2016; Côté and Iori 2018).

The formulation uses, for each arc $(d, e, i) \in \mathcal{A}$, a binary decision variable f_{dei} taking value 1 if the arc is selected in the solution, and value 0 otherwise. In other words, a variable $f_{dei} = 1$ indicates that item $i \in \mathcal{I}$ is packed in a bin from position d to position e, though not defining the *actual* bin

in which the item is packed. The resulting arcflow model for the BPP is as follows

$$\min \quad \sum_{(0,e,i)\in\mathcal{A}} f_{0ei} \tag{9}$$

$$\text{s.t.} \quad \sum_{(v,e,i)\in\mathcal{A}} f_{vei} \leq \sum_{(d,v,i)\in A} f_{dvi} \qquad v \in \mathcal{V}\setminus\{0,W\} \tag{10}$$

$$\sum_{(d,e,i)\in\mathcal{A}} f_{dei} = 1 \qquad i \in \mathcal{I} \tag{11}$$

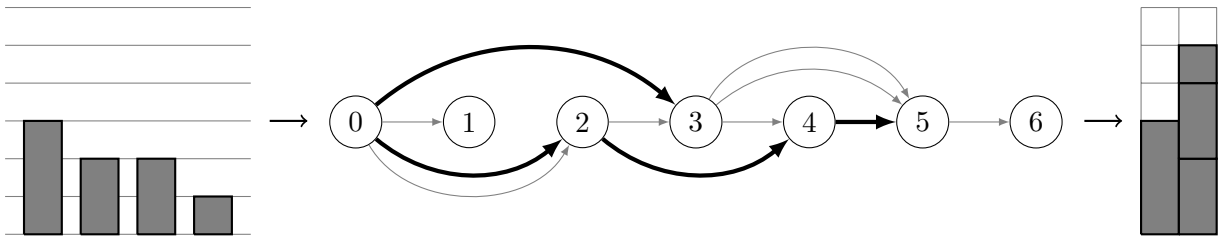
$$f_{dei} \in \{0,1\} \qquad (d,e,i) \in \mathcal{A}. \tag{12}$$

The objective function (9) minimizes the amount of flow leaving node 0, i.e., the number of bins used. Constraints (10) ensure that, for every node $v \in \mathcal{V}\setminus\{0,W\}$, the amount of flow leaving v cannot exceed the amount of flow entering v. Finally, constraints (11) impose that, for each item i, exactly one arc associated with that item is selected, i.e., the item is packed.

In order to retrieve a possible packing of items into bins, one can use a flow decomposition algorithm (see chapter 3.5 in Ahuja et al. 1993) on the solution provided by model (9)-(12).

Example 1. *(resumed) Let us consider the BPP instance induced by item set $\mathcal{I}_2$ with weights $\{3,2,2,1\}$ and $W = 6$. This instance is depicted in the left part of Figure 2. The graph $\mathcal{G}$ obtained after applying the reduction procedures suggested by Valério de Carvalho (1999) is depicted in the center of Figure 2. Here, arcs in bold denote an optimal solution of model* (9)-(12)*, requiring two bins. A flow decomposition algorithm results in the first item packed alone in the first bin and the other three items packed together in the second bin. A graphical representation of the corresponding solution is shown in the right part of Figure 2.*

Figure 2: The BPP instance induced by item set $\mathcal{I}_2$, its arcflow graph, and solution.



2.2. Lower bounds

For every color $c \in \mathcal{C}$, let $\mathcal{P}_c$ be the BPP instance induced by item set $\mathcal{I}_c$ and bin capacity W. Also, let L_c be any lower bound on the optimal solution value of this instance. Then, it is straightforward that $L = \sum_{c\in\mathcal{C}} L_c$ is a valid lower bound for the BPPMCF. Formally, this relaxation can be derived from model (1)–(6) by (i) relaxing constraints (3); (ii) observing that the resulting problem can be decomposed into C independent subproblems $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_C$; and (iii) computing a lower bound for each such subproblem. In the following we will denote by L^* the lower bound obtained by computing

the optimal solution value L_c^* for each instance $\mathcal{P}_c$, i.e., by solving a series of strongly $\mathcal{NP}$-hard subproblems. Conversely, the procedure used in Mehrani et al. (2022) is based on computing, for each color c, lower bound L_2 for the associated BPP instance, and hence the corresponding lower bound L can be computed in polynomial time.

Given an instance I of a minimization problem P, and a lower bounding procedure L, let $L(I)$ and $z(I)$ be the values of the lower bound and of the optimal solution, respectively. The *absolute worst-case performance ratio* of L is defined as

$$r(L) = \inf_{I \in P} \left\{ \frac{L(I)}{z(I)} \right\}.$$

Although each subproblem is solved to optimality when computing lower bound L^*, the next theorem shows that the worst-case performance of lower bound L^* is arbitrarily bad for the BPPMCF.

Theorem 1. *The absolute worst-case performance ratio for lower bound L^* is arbitrarily close to zero.*

Proof. Consider a family of instances having B bins of capacity $W = 1$, C colors and $B \cdot C$ items, where there are B items of weight $\frac{1}{2^c} + \varepsilon$ for each color $c \in \mathcal{C}$, with ε a small positive constant. For convenience we temporarily depart here from the assumption that all input data is integer, but one can easily obtain integer input by scaling the bin capacity and all item weights by an appropriate constant.

First, we show that every feasible solution must have exactly one item of each color in each bin, meaning that $z(I) = B \cdot C$. Indeed, the number of items of each color is equal to the number of bins and no two items of color 1 fit together. Moreover, after packing the items of color 1, no two items of color 2 fit together, the residual space being $\frac{1}{2} - \epsilon$. Similarly, after packing the items of color 1 and 2, no two items of color 3 fit together, and so on.

For every $c \in \mathcal{C}$, we have that $L_c^* = \left\lceil \frac{B}{2^c - 1} \right\rceil$, as we can pack at most $2^c - 1$ items of weight $\frac{1}{2^c} + \varepsilon$ per bin (for small enough ε). Therefore we have

$$L^* = \sum_{c=1}^{C} L_c^* = \sum_{c=1}^{C} \left\lceil \frac{B}{2^c - 1} \right\rceil \leq \sum_{c=1}^{C} \left(\frac{B}{2^c - 1} + 1 \right) \leq \sum_{c=1}^{C} \left(\frac{1}{2} \right)^{c-1} B + C \leq 2B + C.$$

Hence, if we set $C = B$, we get

$$\frac{L^*(I)}{z(I)} \leq \frac{2B + C}{B \cdot C} = \frac{3B}{B^2} = 3/B \to 0$$

as $B \to \infty$. □

Despite the negative result in terms of absolute worst-case performance, our computational experiments (see Section 4) show that, for all instances introduced in the literature, lower bound L^* is very

effective: indeed, it yields a very tight approximation of the optimal solution value while requiring a very short computing time.

2.3. Upper bounds

2.3.1. `BPP-LB`

Our first heuristic algorithm is based on lower bound L^* introduced in Section 2.2. The procedure operates as follows:

1. First, the problem is decomposed into C subproblems $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_C$, which are solved by means of an exact algorithm for the BPP. For each color c, let L_c^* denote the optimal solution value for subproblem c. Then, we replace item set $\mathcal{I}_c$ by L_c^* "super-items", one for each bin in the subproblem solution, where the weight of each super-item is set to be equal to the overall weight of the items in the corresponding bin.
2. Then, a BPP instance, say $\mathcal{P}^*$ is defined by all the $L^*(= \sum_{c=1}^{C} L_c^*)$ super-items produced in the previous step by all subproblems.
3. Finally, we solve the new BPP instance $\mathcal{P}^*$ to optimality, and let B^* denote the optimal solution value.

Proposition 1. *If $B^* \leq B$, the solution returned by* `BPP-LB` *is optimal.*

Proof. Clearly if $B^* \leq B$, the solution returned by the algorithm is feasible for the original BPPMCF instance. In addition, the total color fragmentation of this solution is equal to L^*, which proves optimality. □

Note that the same argument applies if, in step 3 of the algorithm, we stop as soon as a BPP solution to instance $\mathcal{P}^*$ is found that uses at most B bins.

In case $B^* > B$, the method does not produce any feasible solution. To reduce the likelihood of this outcome, we slightly modify step 1 of the algorithm as follows. For each subproblem $\mathcal{P}_c$, instead of computing *any* optimal BPP solution (requiring L_c^* bins), we look for an optimal solution such that the content of a given (say, the first) bin is minimized. The resulting problem is denoted as the *BPP with loss concentration*. Solving a sequence of C BPP with loss concentration subproblems produces a BPP instance $\mathcal{P}^{**}$ in which many super-items have a small weight compared with the bin capacity, thus increasing the likelihood of packing them together with other super-items into a bin.

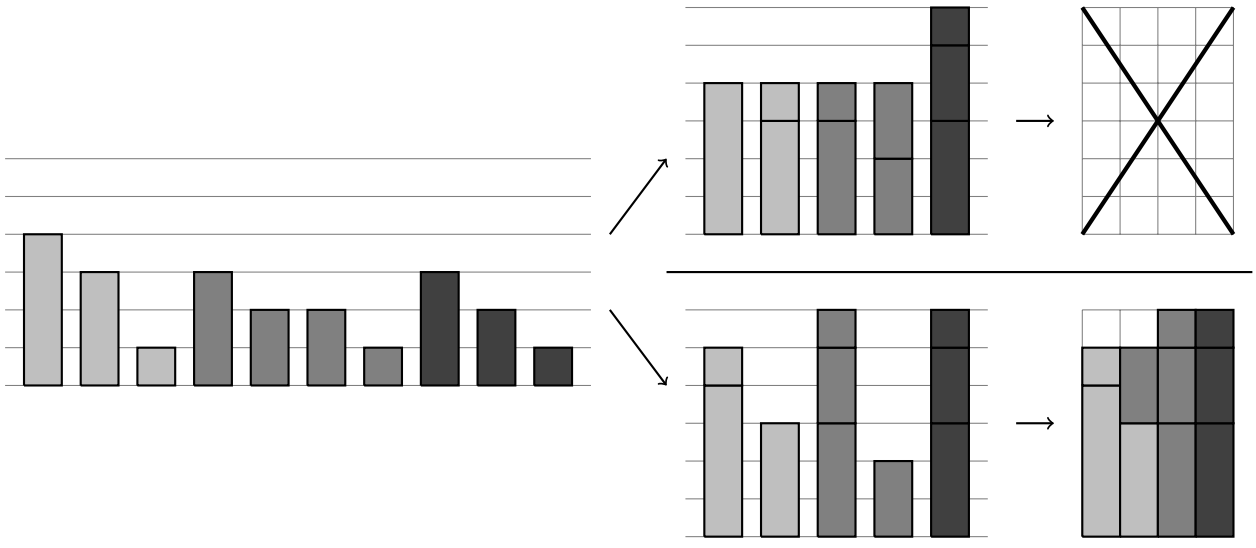
The BPP with loss concentration can easily be solved by guessing the amount Δ of free space in the first bin, adding a dummy item with weight Δ to item set $\mathcal{I}_c$, and using any existing BPP approach as a black box. As an initial guess, one can use an upper bound, setting $\Delta = \min\left\{W - \min_{i \in \mathcal{I}_c}\{w_i\}, W \cdot L_c^* - \sum_{i \in \mathcal{I}_c} w_i\right\}$. Any BPP solution that uses L_c^* bins for the resulting instance is optimal in terms of loss fragmentation. If no such solution exists, the guess is wrong and one can try with a smaller value for Δ, iterating the method until a BPP solution using L_c^* bins is found.

Observe that one can limit the search to Δ values such that $W - \Delta$ is a combination of item weights. Preliminary experiments showed that using binary search when searching the optimal Δ value did not yield computational improvements.

From the implementation perspective, ad hoc strategies aimed at reducing the computation time required to solve the BPP with loss concentration can be devised when using the arcflow formulation. To this aim, for a given guess Δ, it is enough to add to the arc set of graph $\mathcal{G}$ a dummy arc $(W - \Delta, W, 0)$ corresponding to a fictitious item with index 0 and weight Δ. Note that, in this case, only non-isolated nodes are candidates for being the tail of the dummy arc at any iteration, allowing for an efficient identification of the next Δ value.

Example 1. *(resumed) Consider the previously introduced BPPMCF instance, as depicted in the left part of Figure 3. Solving the BPP subproblems $\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{P}_3$ gives the new BPP instance $\mathcal{P}^*$ consisting of the 5 super-items presented top-left in the right part of Figure 3. Note that there is no solution to instance $\mathcal{P}^*$ that uses at most $B = 4$ bins, so* `BPP-LB` *just returns the lower bound $L^* = 5$.*
Instead, if we solve the BPP with loss concentration for each of the subproblems $\mathcal{P}_1, \mathcal{P}_2$ and $\mathcal{P}_3$, we obtain a BPP instance $\mathcal{P}^{**}$ consisting of 5 different super-items presented bottom-left in the right part of Figure 3. This latter instance admits a solution with $B = 4$ bins, as presented bottom-right in the right part of Figure 3, i.e., a feasible BPPMCF solution of value 5, hence an optimal solution according to Proposition 1.

Figure 3: `BPP-LB` applied to Example 1, without (top) and with (bottom) loss concentration.



Note that solving the BPP with loss concentration could work counterproductively. Indeed, it is easy to come up with instances for which solutions to the BPP subproblems that do not maximize loss concentration result in an instance $\mathcal{P}^*$ admitting a packing into B bins, while the instance $\mathcal{P}^{**}$ does not.

2.3.2. BPP-UB

Our second heuristic, denoted as BPP-UB, is a two-stage approach based on the solution of a BPP instance.

- In the first phase, we "delete" all item colors and obtain a BPP instance $\mathcal{P}_0$. The algorithm terminates if it can prove that the instance does not admit a solution using B bins or less, as this implies infeasibility of the original BPPMCF instance.
- Otherwise, we compute a BPP solution using at most B bins. Then, we execute a second phase in which this solution is used to produce a feasible BPPMCF solution minimizing color fragmentation by recoloring the items, possibly recombining the content of some bins.

We now detail the second phase of the algorithm, which is based on a representation of the BPP solution provided by the arcflow formulation. Let $\widetilde{\mathcal{G}} = (\widetilde{\mathcal{V}}, \widetilde{\mathcal{A}})$ be a multigraph having the same structure as the one described in Section 2.1 but containing only the arcs selected in the solution. Note that graph $\widetilde{\mathcal{G}}$ can also be retrieved from a BPP solution consisting of a bin-item allocation: for each bin b, one should simply consider the set $\mathcal{I}^b$ of items packed in the bin and add to $\widetilde{\mathcal{A}}$ the set of arcs induced by these items. More in detail, we let the initial node be $u = 0$ and consider items in $\mathcal{I}^b$ one at a time: when evaluating item i, we add to $\widetilde{\mathcal{A}}$ an arc (u, v) where $v = u + w_i$, and set $u = v$. When all items in $\mathcal{I}^b$ have been considered, we set $u = 0$ and then perform the same operations for the next bin, until all bins have been considered.

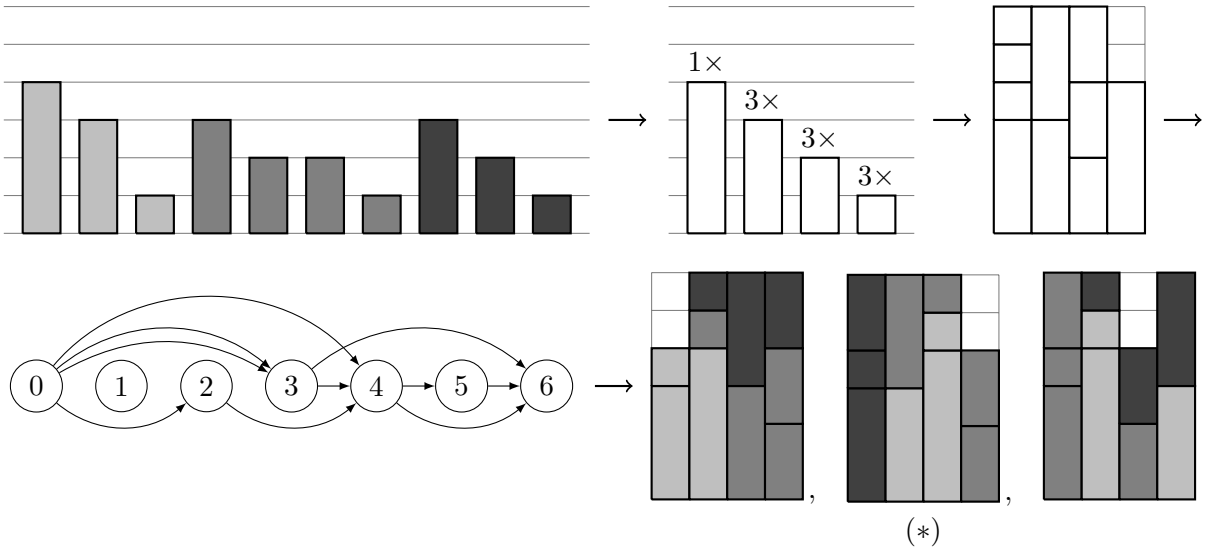
Given graph $\widetilde{\mathcal{G}}$, we define a BPPMCF solution by a flow decomposition algorithm which simultaneously selects an arc and assigns it the most suitable (colored) item of the BPPMCF instance. Every partition (and item assignment) of the arcs in $\widetilde{\mathcal{A}}$ into at most B paths starting from node 0 and ending in a node for which the in-degree is strictly larger than the out-degree corresponds to a feasible BPPMCF solution. We construct a set of arc-disjoint paths, one at a time, as follows. Initially, all items are available. Each path starts from node 0 with a randomly selected arc of length, say, u. To determine the associated item, we randomly select one item of the BPPMCF instance that is still available and has weight u. We then consider all arcs leaving u, which identify a set of candidate items to be selected. A candidate item whose color is already present in the path, if any, is selected (random choice if more than one). If there are no candidate items, we terminate the current path if the in-degree of u is strictly larger than its out-degree. Otherwise, a random available item of another color is selected. When an arc is added to the path and an item is assigned to it, we update u accordingly, remove the arc from $\widetilde{\mathcal{A}}$, mark the associated item as unavailable, and iterate.

As graph $\widetilde{\mathcal{G}}$ may include an exponential number of paths whose arcs can be assigned to items in different ways, and random choices are taken, one can apply the algorithm several times and return the best solution found. In our implementation, we construct $\widetilde{\mathcal{G}}$ only once and apply the second phase Θ times, where Θ is an input parameter.

Example 1. *(resumed) Consider the previously introduced BPPMCF instance depicted top-left in*

Figure 4 and let $\Theta = 3$. By disregarding colors, we obtain the BPP instance $\mathcal{P}_0$ presented top-center in the figure, for which an optimal solution is depicted top-right. This solution uses $B = 4$ bins, implying that the original BPPMCF instance is feasible. Transforming this solution into its arcflow representation, we obtain the multigraph $\widetilde{\mathcal{G}}$ presented bottom-left in the figure. By applying the second stage $\Theta = 3$ times, we may obtain the three solutions presented bottom-right, whose objective values are 8, 6 and 7, respectively. Eventually, algorithm `BPP-UB` *returns the second solution.*

Figure 4: `BPP-UB` applied to Example 1 for $\Theta = 3$.



3. A tabu search algorithm

Algorithm `BPP-UB` is very fast in practice and can be used as the starting point for a metaheuristic approach. This class of algorithms embed local search and escape local optima by means of different strategies. In tabu search algorithms (see, Glover and Laguna (1997)), one always moves to the best solution in the neighborhood of the current solution, possibly accepting worsening moves. To prevent cycling, a tabu list is used that stores some features of recently visited solutions and forbids, for some iterations, the algorithm to return to solutions with these features again. We developed a tabu search algorithm for the BPPMCF, denoted as `TS`, in which each solution is represented as a partition of the items into bins and a neighborhood is defined by means of the following moves:

- **Single transfer:** We move an item i from bin b to another bin $\bar{b}$.
- **Super transfer:** We move all items of the same color c from a bin b to another bin $\bar{b}$, provided at least two items of color c are packed in bin b.
- **Single swap:** We swap an item i with an item $\bar{i} > i$ that is packed in a different bin. We do not consider swaps between identical items.
- **Super swap:** We swap all items of the same color c from a bin b with all items of the same

color $\bar{c}$ from another bin $\bar{b} > b$. We do not allow for swaps involving complete bins or less than 3 items.

While transfer moves involve one color at a time, swap moves may involve two different colors. Thus, moves of the former (resp. latter) type can potentially increase or decrease the solution value by 1 (resp. 2) unit(s). Our tabu search algorithm performs a number of iterations defining a (current) solution at each iteration. Whenever the current solution includes an empty bin, a random super transfer is executed. Otherwise, all moves that can be derived from the current solution are evaluated and the move leading to the best solution in terms of color fragmentation is selected. Under the tabu search paradigm, when the current solution is a local optimum, a possibly worsening solution is selected to allow the algorithm to continue the search without getting stuck. Ties are broken by considering a secondary goal based on color concentration, a figure defined for each color and bin which measures the total weight of the items of that color in that bin. Intuitively, when most of the weight of a color is concentrated in a bin (i.e., when a bin allocates a large fraction of the overall weight of items of that color) and only a small weight of that color is packed into another bin, it will be easier later on to fully get rid of that color in the latter bin. More precisely, given a color c and a bin b, let $l(c, b)$ denote the total weight of items of color c in bin b. Now assume that a move involving bins b and $\bar{b}$ is executed and let $l'(\cdot, \cdot)$ be the counterpart of $l(\cdot, \cdot)$ in the resulting solution. Then, we can compute the relative variation of color concentration for color c with respect to bins b and $\bar{b}$ as

$$\delta(c) = \frac{|l'(c, b) - l'(c, \bar{b})| - |l(c, b) - l(c, \bar{b})|}{W}. \tag{13}$$

Subsequently, moves involving a single color, say c, are ranked according to non-increasing values of $\delta(c)$, whereas moves involving two colors, c and $\bar{c}$, are ranked according to $\delta(c) + \delta(\bar{c})$.

To prevent cycling, we make use of a tabu rule forbidding an item to re-enter a bin that it had left in the previous τ iterations, where τ is the tabu tenure. Items identical to item i (i.e., having the same c and w) also cannot be moved to bin b. In case of a super swap or super transfer, we randomly select only one of the items affected by the move and store it in the tabu list. Aspiration criteria allow to perform a tabu move if it leads to a solution improving the incumbent.

The algorithm includes a diversification scheme in which loss concentration replaces color concentration. Loss concentration is based on the observation that when one bin has a large residual capacity (loss space), then more moves involving that bin are likely to yield a feasible packing, thus increasing the probability to decrease the solution value later in the algorithm. Loss concentration can be computed according to equation (13) by replacing the weight of color c with the empty space in the bin. In our algorithm, we execute diversification after a certain number, say ND, of iterations with no updating of the incumbent. Color concentration is restored as soon as a new incumbent is found or after ND iterations of diversification.

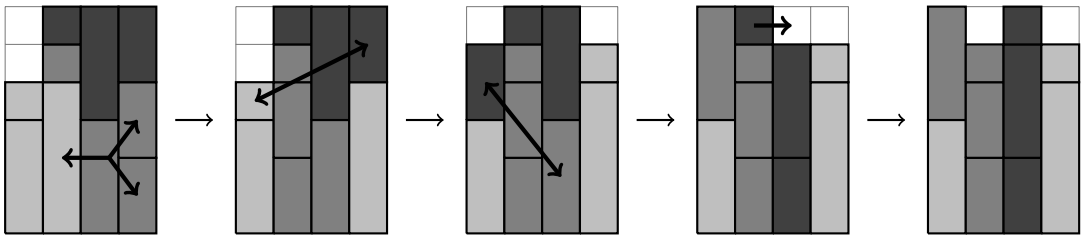
Finally, the algorithm includes a reset option that can be used when all non-tabu moves are worsen-

ing. In this case, we erase the tabu list and apply diversification. The reset option is available only once, and is restored each time the incumbent solution is updated.

The algorithm terminates after NT iterations with no improvements (where NT is an input parameter), if the value of the incumbent solution equals lower bound L^*, or if all moves are tabu.

Example 1. *(resumed) Consider the previously introduced BPPMCF instance and the first solution found by* `BPP-UB` *presented in Figure 4, and assume that this solution is used as a starting point for* `TS`*. In Figure 5, we depict how algorithm* `TS` *improves this initial solution to an (optimal) solution in* 4 *iterations, reducing color fragmentation from* 8 *to* 5*. In the first iteration, we perform a super swap of the light gray item from bin* 2 *with the two medium gray items from bin* 4*, thus reducing color fragmentation by* 1 *unit. In the second iteration, we perform a single swap of the light gray item of weight* 1 *from bin* 1 *with the dark gray item from bin* 4*. This does not change color fragmentation, but it does improve the secondary score by* 1/3*. In the third iteration, we perform a single swap of the dark gray item from bin* 1 *with the medium gray item from bin* 3*, reducing color fragmentation by* 1 *unit. Note that this move was chosen rather than the single transfer of the dark gray item from bin* 2 *to bin* 1*. While both moves reduce color fragmentation by 1 unit, the selected one improves the secondary score by* 2/3 *whereas the latter improves the secondary score by only* 1/3*. Finally, in the fourth iteration we execute a single transfer of the dark gray item from bin* 2 *to bin* 3*, which further reduces color fragmentation by* 1*, thus yielding a solution with value* 5*.*

Figure 5: `TS` applied to Example 1.



4. Computational experiments

In this section, we computationally evaluate the performance of the proposed methods on different benchmarks of instances.

The algorithms of the previous sections were implemented in C++ and can be downloaded from `https://github.com/MathijsBarkel/BPPMCF`. All experiments were run on an Intel(R) Core(TM) i7-6900K, 3.20GHz with 64GB of memory and performed in single thread with a time limit equal to 1800 seconds. ILP models were solved by means of Gurobi 10.0.1 in its default setting. The only exception arises for flow formulations, where the continuous relaxation at the root node of the branch-and-bound tree was solved by means of the barrier algorithm. This is consistent with the observation in Brandão and Pedroso (2016) that "the use of interior-point methods at the root node considerably improves the time for solving the linear relaxation, compared to using the simplex

algorithm". This behavior was confirmed in other experiments (see Delorme and Iori 2020) and in preliminary tests on our algorithms.

4.1. Instances from the literature

We first consider the four data sets (D1, D2, D3 and D4) that were proposed by Mehrani et al. (2022)[1]. In Table 1 we summarize the main features of each data set, namely, the number of instances *inst* and averages of the following features: number of bins B, number of colors C, number of items I, and bin capacity W. Finally, column $\%B$ gives the average percentage of the number of available bins that are needed for packing all items in $\mathcal{I}$.

Table 1: Average instance features per data set.

	inst	B	C	I	W	$\%B$
D1	120	85.0	54.6	236.2	10	85.5
D2	50	215.9	3.0	434.0	150	80.7
D3	60	12.5	6.0	177.1	100	88.3
D4	180	75.0	5.0	364.7	450	97.4

The table shows that this benchmark includes instances with very different characteristics. Instances in class D1 are characterized by a very large number of colors, whereas instances in class D3 require to pack many items per bin (this figure being roughly expressed by I/B). Finally, we observe that instances in class D2 have a large number of available bins that are not strictly necessary for a feasible packing, whereas almost all bins are required for a feasible solution in the instances in class D4.

In Table 2 we present the results of `BPP-LB`. In our implementation we solve BPP subproblems by means of the Gurobi MIP solver applied to the arcflow model, enhanced as discussed in Delorme and Iori (2020), where possible. For comparison purposes, we also present the results of the methods `IP2`, `RM2-GIFF` and `EM` proposed by Mehrani et al. (2022). We used the codes available at the authors' Github page `https://github.com/saharnazmehrani/BPPMCF-IJOC`, and run them on our machine. For each algorithm, we give the number of instances solved to proven optimality (*opt*), the number of instances for which a feasible solution was found ($feas$), and the average computing time over all instances (t). For algorithm `BPP-LB` we do not distinguish between feasible and optimal solutions, as whenever the algorithm computes a feasible solution it also proves its optimality. The best values of *opt* in each line are marked in bold.

Although `BPP-LB` has no guarantee of returning a feasible solution with value matching the lower bound L^*, even in case such a solution exists, we observe that this is the case for all instances in

[1]These instances were taken from `https://github.com/saharnazmehrani/BPPMCF-IJOC`. However, there are 20 instances for which Mehrani et al. (2022) did present results, but not the instances themselves. It concerns instance sets "10-100-2" and "15-100-2" of data set 3. We did not consider these instances.

Table 2: Results of exact methods from the literature and BPP-LB on the original instances.

		IP2			RM2-GIFF			EM			BPP-LB	
	inst	*opt*	*feas*	*t*	*opt*	*feas*	*t*	*opt*	*feas*	*t*	*opt*/*feas*	*t*
D1	120	**120**	120	2.8	**120**	120	0.4	77	109	865.7	**120**	0.2
D2	50	6	50	1591.6	**50**	50	10.8	37	41	801.5	**50**	1.0
D3	60	**60**	60	0.0	**60**	60	169.6	36	50	757.0	**60**	0.2
D4	180	95	180	1065.5	53	120	1545.9	155	176	276.0	**180**	7.7
All	410	281	410	662.7	283	350	704.9	305	376	583.1	**410**	3.6

this benchmark set. The computational effort required by BPP-LB is extremely limited, the average computing time being equal to 3.6 seconds on average, thereby significantly outperforming the exact methods proposed by Mehrani et al. (2022). The largest computing times appear for instances of data set D4, that are characterized by large bin capacity and are therefore more time-consuming for solution approaches based on a flow formulation. Finally, despite IP2 being able to compute a feasible solution for each instance, there are still 6 instances (all from data set D4) that none of the three methods by Mehrani et al. (2022) solves to proven optimality. Actually, these instances as well are solved by BPP-LB.

As BPP-LB already solves all instances of these data sets within a few seconds, we need more challenging instances for evaluating the performances of our more sophisticated algorithms. A new benchmark of hard instances will be introduced in the next section.

4.2. New harder instances

The previous section shows that the instances from the literature are easy as the number of bins that is actually required for packing all items is relatively small compared with the number of available bins, i.e., there is room to avoid color fragmentation. Therefore, in order to obtain harder instances, we modify each one of the original instances by setting B equal to the minimum number of required bins to pack all items. The resulting data sets are denoted by D1*, D2*, D3* and D4*. These new instances are relevant also from a practical point of view, as they can arise in applications where the goal is to minimize color fragmentation subject to the constraint that the number of bins is minimized.

In addition, we introduce a further data set D5*, consisting of 80 instances obtained by adding colors to the "triplet" instances proposed by Falkenauer (1996) for the BPP. These instances have the property that any optimal packing requires 3 items per bin for which the weights sum up to exactly W. Starting from an optimal solution to an original triplet instance for the BPP, we construct a corresponding BPPMCF instance with $C = 3$, where we assign to each items of a triplet a different color, and we set the number of available bins to $I/3$. These instances can be downloaded from https://github.com/MathijsBarkel/BPPMCF.

In order to evaluate the hardness of the new benchmark with exact methods from the literature,

we tested the same algorithms considered in Table 2, namely methods IP2, RM2-GIFF and EM, and BPP-LB. Results are reported in Table 3, and confirm that these new instances are more challenging than their original counterparts. On these instances, RM2-GIFF is the best exact method, though it is able to compute a feasible solution in 335 cases only out of 490 and a provably optimal solution in only around 60% of the instances (293 out of 490). Also the performance of BPP-LB is less satisfactory than for the original instances: this method is now able to solve only 188 cases, i.e., 38.4% of the instances (45.9% when ignoring D5*, compared to 100% on the original instances). Nevertheless, BPP-LB is the most effective method for class D4*, for which it finds 105 feasible and optimal solutions out of 180 instances; this is coherent with the observation that the instances in this data set are not so different with respect to their original counterparts in terms of number of available bins. At the same time, bad performance on instances of data set D5* may be expected by construction of those instances, where grouping items by colors does not preserve triplets and is very unlikely to result in a feasible solution. To evaluate the quality of lower bound L^* returned by BPP-LB, we computed its average performance ratio L^*/z, where z denotes the optimal solution value. This was possible only for instances in data sets D1*, D3* and D5*, that are all solved to optimality by at least one solution method. The resulting values are 0.92, 0.98 and 0.59, confirming that instances in class D5* are by far the most challenging ones for this solution approach. In addition, we point out that, for 5 instances in data set D2* and 75 instances in data set D4*, the optimal solution value is still unknown. Finally, there are 45 instances, all from data set D4*, for which no method is able to even compute a feasible solution. Summarizing, the results clearly show the computational hardness of the instances of the new benchmark, motivating the need for alternative approaches, at least for deriving a feasible solution.

Table 3: Results of exact methods from the literature and BPP-LB on the new instances.

		IP2			RM2-GIFF			EM			BPP-LB	
	inst	*opt*	*feas*	*t*	*opt*	*feas*	*t*	*opt*	*feas*	*t*	*opt*&*feas*	*t*
D1*	120	0	120	1800.0	**120**	120	0.9	0	29	1800.0	0	0.2
D2*	50	1	5	1769.5	**45**	50	397.3	14	26	1432.2	36	1.0
D3*	60	**60**	60	38.4	39	59	802.7	2	8	1740.0	47	0.2
D4*	180	0	8	1800.0	9	26	1766.1	23	85	1588.0	**105**	7.6
D5*	80	0	1	1800.0	**80**	80	25.1	19	32	1518.1	0	10.7
All	490	61	194	1581.2	**293**	335	791.9	58	180	1631.2	188	4.7

In Table 4 we present summarized results of the performance of the heuristic methods BPP-UB and TS on all five new data sets. In the former algorithm, BPP subproblems are solved by means of the standard arcflow formulation, and the second stage (flow decomposition) is executed $\Theta = 100$ times. As to the latter algorithm, preliminary experiments suggested to set the tabu tenure $\tau = 50$, the maximum number of non-improving iterations before diversification $ND = 40$, and the maximum number of non-improving iterations before termination $NT = 5000$. A detailed discussion on the sensitivity of the algorithm with respect to these parameters is given in Section 4.3.

Both algorithms are always able to compute a feasible solution, which is why we do not explicitly report the number of feasible solutions. For a heuristic solution with value U, we compute the optimality gap with respect to a lower bound (say) L as $100\frac{U-L}{L}$. The table reports, for each method, the average value of the percentage gap with respect to the best known lower bound (either L^* or the best lower bound returned by any of the exact methods, column gap_{best}), the number of instances for which the solution value matches the best known lower bound (column opt_{best}), and the average computing time (column t). In addition, we report into brackets the number of optimal solutions that would be found in case the algorithms were executed right after `BPP-LB`.

Table 4: Results of heuristic methods on the new instances.

		`BPP-UB`				`TS`			
	inst	gap_{best}	opt_{best}		t	gap_{best}	opt_{best}		t
D1*	120	40.8	0	(0)	0.0	0.5	85	(85)	64.1
D2*	50	38.9	0	(36)	0.3	1.3	21	(37)	14.0
D3*	60	93.2	0	(47)	0.1	0.1	59	(60)	0.5
D4*	180	123.9	0	(105)	11.2	5.6	98	(105)	17.4
D5*	80	15.2	8	(8)	0.9	13.7	8	(8)	3.5
All	490	73.4	8	(196)	4.3	4.6	271	(295)	24.2

Even though `BPP-UB` always finds a feasible solution for a given instance (if one exists), the results show that these solutions are typically weak, as shown by the associated percentage gap of 73.4% with respect to the best known lower bound. In addition, `BPP-UB` never finds a solution matching the best known LB for the instances in data sets D1*, D2*, D3* and D4*, and only 8 optimal solutions are found for those in data set D5*. On the other hand, algorithm `TS` finds considerably better solutions, whose associated percentage gap is reduced to 4.6%. For more than 55% of the instances, a solution is found whose value matches the best known lower bound, showing the effectiveness of the tabu search approach in improving over the solution provided by `BPP-UB`.

By combining `TS` with `BPP-LB`, one would compute an optimal solution for a large number of instances (more than 60% of the entire benchmark), the most notable exception being data set D5*. For these instances, even though the initial solution is quite good, it is rarely improved by `TS`; this happens because in these instances all feasible solutions have a very strong structure which limits the possibility of swap moves and makes transfer moves inapplicable.

Finally, we observe that, for all data sets, the average computing time of `TS` is typically less than a minute, and is below 30 seconds on the entire benchmark.

4.3. `TS` parameters

We now discuss the sensitivity of algorithm `TS` with respect to its main parameters τ, ND and NT. To this aim, we show the performance of the algorithm while modifying one of the parameters at a time with respect to the default values (equal to 50, 40 and 5000, respectively). In our analysis, we

always report the average percentage gap (gap_{best}), the number of optimal solutions (opt_{best}) (both figures being computed with respect to the best known lower bound for a given instance), and the average computing time (t).

In Table 5 we examine the results obtained with different values of the tabu tenure, namely $\tau = 25, 50, 75$.

Table 5: Sensitivity of TS with respect to the tabu tenure τ.

		$\tau = 25$			$\tau = 50$			$\tau = 75$		
	$inst$	gap_{best}	opt_{best}	t	gap_{best}	opt_{best}	t	gap_{best}	opt_{best}	t
D1*	120	0.6	78	68.0	0.5	85	64.1	0.6	71	63.6
D2*	50	2.2	14	13.4	1.3	21	14.0	1.0	25	13.4
D3*	60	0.0	60	0.5	0.1	59	0.5	0.0	60	0.5
D4*	180	3.8	98	19.8	5.6	98	17.4	7.4	99	16.7
D5*	80	13.6	8	4.1	13.7	8	3.5	13.7	8	2.2
All	490	4.0	258	26.0	4.6	271	24.2	5.2	263	23.5

These results show that $\tau = 50$ results in the largest number of optimal solutions, and a smaller average gap than $\tau = 75$. However, $\tau = 25$ results in a smaller average gap than $\tau = 50$, especially on D4*, meaning that there is a trade-off between the number of optimal solutions and the average gap.

In Table 6 we compare TS for different values of the number of non-improving iterations before diversification, namely $ND = 20, 40, 60$. In addition, in the final columns we present the results of TS without the diversification feature.

Table 6: Sensitivity of TS with respect to the diversification parameter ND.

		$ND = 20$			$ND = 40$			$ND = 60$			No diversification		
	$inst$	gap_{best}	opt_{best}	t	gap_{best}	opt_{best}	t	gap_{best}	opt_{best}	t	gap_{best}	opt_{best}	t
D1*	120	0.5	79	66.3	0.5	85	64.1	0.5	77	63.1	0.6	79	66.6
D2*	50	1.4	22	15.1	1.3	21	14.0	1.7	18	13.4	2.9	15	11.1
D3*	60	0.0	60	0.5	0.1	59	0.5	0.0	60	0.5	0.2	58	0.5
D4*	180	5.7	98	17.5	5.6	98	17.4	5.8	99	17.1	4.7	93	18.8
D5*	80	13.7	8	3.5	13.7	8	3.5	13.7	8	3.5	13.6	8	3.5
All	490	4.6	267	24.8	4.6	271	24.2	4.7	262	23.7	4.4	253	25.0

This table indicates that $ND = 40$ results in a larger number of optimal solutions and a lower average gap than $ND = 20$ and $ND = 60$. Moreover, using diversification with $ND = 40$ is clearly better than not using any diversification, resulting in significantly more optimal solutions while only having a slightly larger average gap.

Finally, Table 7 gives the results of TS for different values of the number of non-improving iterations before termination, namely $NT = 2500, 5000, 7500$.

Table 7: Sensitivity of TS with respect to the termination parameter NT.

		$NT = 2500$			$NT = 5000$			$NT = 7500$		
	inst	gap_{best}	opt_{best}	t	gap_{best}	opt_{best}	t	gap_{best}	opt_{best}	t
D1*	120	0.6	73	37.4	0.5	85	64.1	0.4	88	88.6
D2*	50	1.7	19	8.6	1.3	21	14.0	1.3	21	18.0
D3*	60	0.1	59	0.3	0.1	59	0.5	0.0	60	0.7
D4*	180	6.5	97	14.6	5.6	98	17.4	5.5	98	19.6
D5*	80	13.7	8	2.2	13.7	8	3.5	13.7	8	4.8
All	490	4.9	256	15.8	4.6	271	24.2	4.5	275	31.6

As expected, the results show that increasing NT results in a smaller average gap and in a larger number of optimal solutions, but also in a larger average computing time. Note that the improvement from $NT = 2500$ to $NT = 5000$ is more relevant than the improvement from $NT = 5000$ to $NT = 7500$.

5. Conclusions

Given a set of items with a weight and a color and a finite set of identical bins, we considered the problem of packing all items in the available bins while minimizing the total number of times that colors appear in the bins. The close relation of this problem with the well-known bin packing problem inspired the design of fast lower and upper bounding procedures. We analyzed the performances of these methods both from a theoretical and from a computational perspective. Experiments on a large set of instances from the literature have been carried out to evaluate the performance of these methods, showing that they are able to solve to proven optimality all instances in a very short time. We thus introduced a set of more challenging instances and developed a metaheuristic algorithm based on the tabu search paradigm to solve them. This second set of experiments showed that the algorithm has good performance, considerably improving, within a limited computational effort, an initial solution provided by the fast bounding procedure. Nevertheless, some of the new instances remain unsolved, which we hope would stimulate further research on exact methods for the problem.

Acknowledgements

Maxence Delorme and Mathijs Barkel thank the Dutch ministry of education for financing their research through the starter grant framework.
Enrico Malaguti and Michele Monaci were supported by the PNRR National Recovery and Resilience Plan, National Research Center in High Performance Computing, Big Data and Quantum Computing.

References

Ahuja, R.K., Magnanti, T.L., Orlin, J.B., 1993. Network flows: theory, algorithms, and applications. Prentice-Hall, Upper Saddle River, NJ, USA.

Bergman, D., Cardonha, C., Mehrani, S., 2019. Binary decision diagrams for bin packing with minimum color fragmentation, in: Rousseau, L.M., Stergiou, K. (Eds.), Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Springer International Publishing, Cham. pp. 57–66.

Brandão, F., Pedroso, J.P., 2016. Bin packing and related problems: General arc-flow formulation with graph compression. Computers & Operations Research 69, 56–67.

Valério de Carvalho, J.M., 1999. Exact solution of bin-packing problems using column generation and branch-and-bound. Annals of Operations Research 86, 629–659.

Côté, J.F., Iori, M., 2018. The meet-in-the-middle principle for cutting and packing problems. INFORMS Journal on Computing 30, 646–661.

Delorme, M., Iori, M., 2020. Enhanced pseudo-polynomial formulations for bin packing and cutting stock problems. INFORMS Journal on Computing 32, 101–119.

Delorme, M., Iori, M., Martello, S., 2016. Bin packing and cutting stock problems: Mathematical models and exact algorithms. European Journal of Operational Research 255, 1–20.

Falkenauer, E., 1996. A hybrid grouping genetic algorithm for bin packing. Journal of Heuristics 2, 5–30.

Fernandes Muritiba, A.E., Iori, M., Malaguti, E., Toth, P., 2010. Algorithms for the bin packing problem with conflicts. INFORMS Journal on Computing 22, 401–415.

Gendreau, M., Laporte, G., Semet, F., 2004. Heuristics and lower bounds for the bin packing problem with conflicts. Computers & Operations Research 31, 347–358.

Glover, F., Laguna, M., 1997. Tabu Search. Kluwer Academic Publishers, Boston.

Kochetov, Y., Kondakov, A., 2017. VNS matheuristic for a bin packing problem with a color constraint. Electronic Notes in Discrete Mathematics 58, 39–46.

Martello, S., Toth, P., 1990. Lower bounds and reduction procedures for the bin packing problem. Discrete Applied Mathematics 28, 59–70.

Mehrani, S., Cardonha, C., Bergman, D., 2022. Models and algorithms for the bin-packing problem with minimum color fragmentation. INFORMS Journal on Computing 34, 1070–1085.

Peeters, M., Degraeve, Z., 2004. The co-printing problem: A packing problem with a color constraint. Operations Research 52, 623–638.

Pessoa, A., Sadykov, R., Uchoa, E., Vanderbeck, F., 2020. A generic exact solver for vehicle routing and related problems. Mathematical Programming 183, 483–523.

Shapiro, J.F., 1968. Dynamic programming algorithms for the integer programming problem-I: The integer programming problem viewed as a knapsack type problem. Operations Research 16, 103–121.

Wei, L., Luo, Z., Baldacci, R., Lim, A., 2020. A new branch-and-price-and-cut algorithm for one-dimensional bin-packing problems. INFORMS Journal on Computing 32, 428–443.

Wolsey, L.A., 1977. Valid inequalities, covering problems and discrete dynamic programs, in: Hammer, P., Johnson, E., Korte, B., Nemhauser, G. (Eds.), Studies in Integer Programming. Elsevier. volume 1 of *Annals of Discrete Mathematics*, pp. 527–538.