Mathematical models and decomposition methods for the two-bar charts packing problem

Mathijs Barkel[(1)], Maxence Delorme[(1)]

(1) *Department of Econometrics and Operations Research, Tilburg University, 5037 AB Tilburg, The Netherlands*

Corresponding author `m.delorme@tilburguniversity.edu`

Abstract

We consider the two-bar charts packing (2-BCPP), a recent combinatorial optimization problem whose aim is to pack a set of one-dimensional items into the minimum number of bins. As opposed to the well-known bin packing problem, pairs of items are grouped to form bar charts, and a solution is only feasible if the first and second items of every bar chart are packed in consecutive bins. After providing a complete picture of the connections between the 2-BCPP and other relevant packing problems, we show how we can use these connections to derive valid lower and upper bounds for the problem. We then introduce two new integer linear programming (ILP) models to solve the 2-BCPP based on a non-trivial extension of the arcflow formulation. Even though both models involve an exponential number of constraints, we show that they can be solved within a constraint generation framework. We then empirically evaluate the performance of our bounds and exact approaches against an ILP model from the literature and demonstrate the effectiveness of our techniques, both on benchmarks inspired by the literature and on new classes of instances that are specifically designed to be hard to solve. The outcomes of our experiments are important for the packing community because they indicate that arcflow formulations can be used to solve targeted packing problems with precedence constraints and also that some of these formulations can be solved with constraint generation.

Keywords: Bin packing, Two-bar charts, Integer programming, Arcflow formulation, Constraint generation.

1 Introduction

Given a set of weighted items and an unlimited number of identical capacitated bins, the *bin packing problem* (BPP) requires to pack all the items into the minimum number of bins. The BPP was introduced several decades ago by Kantorovich (1960) and quickly became one of the most studied combinatorial optimization problems. Indeed, its relatively simple structure made it the perfect playground for the research community to develop and test the most effective solution methods, including branch-and-bound algorithms (Martello and Toth 1990), metaheuristics (Falkenauer 1996), branch-and-price algorithms (Vance 1998), pseudo-polynomial formulations

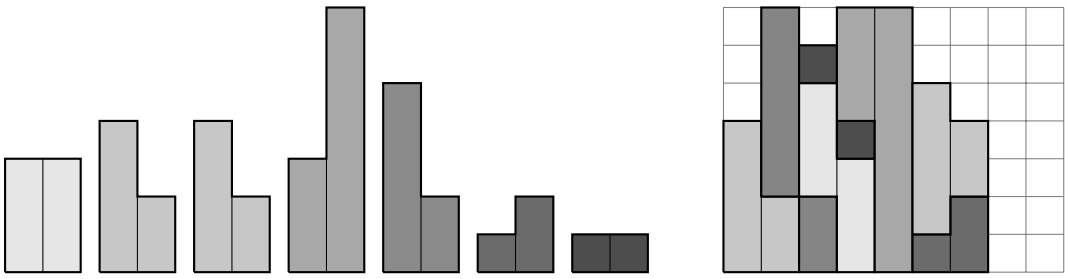
(Valério de Carvalho 1999), constraint programming (Shaw 2004), and reinforcement learning (Cai et al. 2019). Besides theoretical motivations, the BPP is also studied for its applications in the cutting industry (Stadtler 1990) and for its key role when solving more complex combinatorial optimization problems such as the vehicle routing problem (Lysgaard et al. 2004) and the two-dimensional BPP (Côté et al. 2021).

Along the years, many extensions of the BPP were introduced. We mention in particular (i) the BPP with conflicts (Muritiba et al. 2010) where pairs of items cannot be packed in the same bin, (ii) the contiguous BPP (Martello et al. 2003) where pairs of identical items must be packed in consecutive bins, and (iii) the vector packing problem (Brandão and Pedroso 2016) where the weights (resp. the capacities) of the items (resp. the bins) are multidimensional vectors and where a capacity constraint is imposed on every dimension.

Erzin et al. (2021b) introduced one of the latest BPP extensions which is known as the *bar charts packing problem* (BCPP). In the BCPP, we are given a set of ordered lists of weighted items (the bar charts) that must be packed using the minimum number of identical capacitated bins. Besides the usual capacity constraints, the BCPP also requires the items of a given bar chart to be packed in consecutive bins. The BCPP originates from a class of project scheduling problems where one must determine the starting date (the bin) of a set of projects (the bar charts) so that the total makespan (the number of bins used) is minimized. Every project runs for a certain number of periods (one item in the bar chart corresponds to one period) and consumes a given amount of resource that may vary from one period to the other (the weights of the items in the bar chart). The total amount of resource available per period (the bin capacity) is limited and once a project is started, it cannot be interrupted (contiguity constraints). In this paper, we are interested in the special case of the problem where every bar chart is composed of exactly two items: the 2-BCPP. We show in Figure 1 a 2-BCPP instance (on the left part) together with a feasible packing that uses seven bins (on the right part).

Even though the 2-BCPP has a limited set of practical applications, it is still an interesting problem to study because it lies at the intersection of several well-known BPP extensions. It is therefore reasonable to assume that some of the most effective techniques aimed at solving these extensions could be adapted to solve the 2-BCPP. However, the available literature on the topic is mostly dedicated to approximation algorithms and their worst-case performance ratio (Erzin et al. 2021a,b,c) while lower bounding procedures and exact approaches have not been thoroughly investigated.

Figure 1: A 2-BCPP instance and a feasible packing



1.1 Our contribution

We provide an overview of the most relevant packing problems and cluster these problems into three classes $\mathcal{C}_l$, $\mathcal{C}_u$, and $\mathcal{C}_e$. We show that:

- Every problem $p_l \in \mathcal{C}_l$ is a relaxation of the 2-BCPP and thus, algorithms aimed at solving p_l can also be used to derive valid lower bounds for our problem;
- The 2-BCPP is a relaxation of every problem $p_u \in \mathcal{C}_u$ and thus, algorithms aimed at solving p_u can also be used to derive valid upper bounds for our problem;
- The 2-BCPP is a special case of every member $p_e \in \mathcal{C}_e$ and thus, algorithms aimed at solving p_e could also be used to solve our problem, but the current state-of-the-art approaches are not well-suited for large-sized 2-BCPP instances.

We then introduce *Eulerian-flow* and *link-flow*, two new *integer linear programming* (ILP) models based on the well-known arcflow formulation introduced by Valério de Carvalho (1999) for the BPP. As both models use a pseudo-polynomial number of variables and an exponential number of constraints, we propose a constraint generation framework to solve each of these models to optimality.

Subsequently, we empirically evaluate the bounding procedures, the newly introduced decomposition approaches, and an ILP model proposed by Erzin et al. (2021b) on a large set of random instances and we identify instance features for which the tested approaches are particularly effective. For example, the model of Erzin et al. (2021b) works very well when the maximum item weight is below a tenth of the bin capacity while Eulerian-flow and link-flow are several orders of magnitude faster otherwise. We also design two new classes of hard instances for the 2-BCPP: *TRIPLETS*, where the main difficulty is to find an optimal solution (i.e., find the right upper bound), and *DONUTS*, where the main difficulty is to prove that the incumbent solution is optimal (i.e., find the right lower bound).

Besides making significant progresses in the 2-BCPP field, the ideas we introduce are also relevant for arcflow formulations in general. In particular, to the best of our knowledge, it is the first time that an arcflow model is proposed for a BPP with a form of precedence constraints, indicating that similar approaches could be used for a larger set of packing problems. It is also the first time that an arcflow model is solved through constraint generation. This could lead to promising developments in decomposition approaches where the master problem is a BPP (see e.g., the recent work of Côté et al. 2021 to solve the two-dimensional BPP). Indeed, arcflow formulations could now be considered to solve the master problem instead of the frequently used textbook (or "Kantorovich") ILP model.

1.2 Related work

The 2-BCPP was formally introduced by Erzin et al. (2021b). The authors proposed several approximation algorithms with proven worst-case performance ratios and compared the results obtained by their approaches to those obtained when solving a descriptive ILP model with the commercial solver CPLEX. They empirically showed that the solver was more effective for instances with up to 100 bar charts while the heuristics displayed better empirical performance

otherwise. The ILP model considered by Erzin et al. (2021b) uses one variable per combination of bar chart and bin and is a direct extension of the textbook BPP model. Later on, Erzin et al. (2021a) performed additional experiments where they used 2-BCPP instances with known optimal solutions to compute more accurate optimality gaps for the tested approaches. In a subsequent work, Erzin et al. (2021c) proposed two additional approximation algorithms for special versions of the 2-BCPP and derived the associated worst-case performance ratios. They studied the case where every bar chart contains at least one item with weight above half of the bin capacity and the case where, in addition to that condition, the first item of every bar chart is at least as large as the second item. Even though the 2-BCPP literature is relatively scarce, there are a few papers focusing on closely related packing problems that are particularly relevant for our study. In the following, we provide an overview of related problems together with a brief literature review, and group them into three classes: (i) $\mathcal{C}_l$, the problems that are relaxations of the 2-BCPP, (ii) $\mathcal{C}_u$, the problems for which the 2-BCPP is a relaxation, and (iii) $\mathcal{C}_e$, the problems for which the 2-BCPP is a special case.

1.2.1 Problem class $\mathcal{C}_l$

Every packing problem that is a valid relaxation of the 2-BCPP belongs to $\mathcal{C}_l$. In other words, transforming a 2-BCPP instance I into an instance I' of problem p ($p \in \mathcal{C}_l$) and solving I' to optimality provides a valid lower bound for I. If we remove the constraints forcing the items of a given bar chart to be packed in consecutive bins, we obtain the BPP. The version of the BPP where duplicated items are merged into a unique item with integer demand is called the *cutting stock problem* (CSP). A survey on heuristics and approximation algorithms for the BPP and the CSP was proposed by Coffman et al. (2013) while a survey focusing on exact approaches and mathematical models was presented by Delorme et al. (2016). Among the most effective approaches proposed in the literature, we mention branch-and-price algorithms (see Pessoa et al. 2020, Wei et al. 2020b) and pseudo-polynomial formulations (see Valério de Carvalho 1999, Brandão and Pedroso 2016, Delorme and Iori 2020). If we additionally forbid the two items of any bar chart to be packed in the same bin, we obtain the BPP with conflicts. Branch-and-price algorithms (see Sadykov and Vanderbeck 2013) and pseudo-polynomial formulations (see Brandão and Pedroso 2016) were also shown to be effective approaches for solving the BPP with conflicts. Finally, if we force the first item of a given bar chart to always be packed "before" the second item (i.e., in a bin with a lower index), we obtain the BPP with precedence constraints. So far, this problem was mostly tackled by branch-and-bound algorithms (see Dell'Amico et al. 2012, Pereira 2016) and branch-and-price algorithms (see Letelier et al. 2022).

1.2.2 Problem class $\mathcal{C}_u$

Every packing problem for which the 2-BCPP is a valid relaxation belongs to $\mathcal{C}_u$. In other words, transforming a 2-BCPP instance I into an instance I' of problem p ($p \in \mathcal{C}_u$) and solving I' to optimality provides a valid upper bound for I. If we force the first item of every bar chart to be packed in a bin with odd index (i.e., if the first item of every bar chart can only be packed with other first items), we obtain the *two-dimensional vector packing problem* (2D-VPP). This problem has been extensively studied in the literature and can be efficiently solved

through tailored branch-and-price (see Wei et al. 2020a) and pseudo-polynomial formulations (see Brandão and Pedroso 2016). If instead, we merge the two items of every bar chart into a two-dimensional item (i.e., if we force the bottom left corner of the two items in a given bar chart to have the same y-coordinate), we obtain a version of the strip-packing problem where the items can either be rectangles or "L-shaped" hexagons. Algorithms aimed at solving the classical strip packing problem (e.g., the Benders' decomposition approaches from Côté et al. 2014 and Delorme et al. 2017) could be extended to also take into account "L-shaped" hexagons.

1.2.3 Problem class $\mathcal{C}_e$

Every packing problem for which the 2-BCPP is a special case belongs to $\mathcal{C}_e$. In other words, transforming a 2-BCPP instance I into an instance I' of problem p ($p \in \mathcal{C}_e$) and solving I' to optimality provides an optimal solution for I. The 2-BCPP can be seen as a special case of the contiguous BPP which was introduced by Martello et al. (2003) as a relaxation of the *strip packing problem* (SPP). In that relaxation, every two-dimensional item is cut into unit-height slices and the objective is to find a packing that uses the minimum number of bins such that the slices originating from the same item are packed into contiguous bins. Even though the items involved in a contiguity constraint have the same weight in the SPP relaxation, one could also consider a version of the contiguous BPP where this is not the case. In the literature, the contiguous BPP has been tackled with ad-hoc branch-and-bound algorithms (see Martello et al. 2003) and ILP models solved by commercial solvers (see Côté et al. 2014, Delorme et al. 2017). These ILP models can easily be extended to the 2-BCPP, but Erzin et al. (2021b) showed that the resulting approaches are only able to solve small-sized instances to optimality.

The 2-BCPP is also a special case of the BPP with time lags recently introduced by Letelier et al. (2022). In that extension, two items involved in a time lag constraint have a minimum and a maximum "time lag", which is defined as the difference between the index of the bins where the two items are packed. In the version studied by Letelier et al. (2022), the minimum and maximum time lags can vary from one pair of items to the other and are not necessarily bounded. For example, one pair of items could be constrained to have a time lag in the range $[-\infty; -3]$ while another pair could be constrained to have a time lag in the range $[-1; 15]$. The 2-BCPP is thus a special case of the BPP with time lags where every pair of items in a given bar chart is forced to have a time lag in the range $[1; 1]$. Letelier et al. (2022) proposed two ILP models for the BPP with time lags: one that can be seen as an extension of the model proposed by Erzin et al. (2021b) and one that uses an exponential number of variables where each variable decides whether or not a given pattern is used in a bin with a given index. The latter model was solved with a branch-and-price algorithm and could handle instances with up to a few hundred items. Even though these approaches could also be used to solve the 2-BCPP, we do not expect them to work very well in practice because they need one set of variables per bin index, which quickly becomes impractical for large-sized instances. In this paper, we introduce two new ILP models that make use of the special structure of the 2-BCPP to avoid the necessity to take the bin index into account.

1.3 Layout

The remainder of this paper is structured as follows. In Section 2, we formally define the 2-BCPP and we discuss the ILP model proposed by Erzin et al. (2021b). In Section 3, we briefly introduce existing arcflow formulations for the CSP and the 2D-VPP and outline how valid lower and upper bounds for the 2-BCPP can be obtained by solving these models. In Section 4, we introduce two new ILP models for our problem. Both of these models are based on the arcflow formulation, have an exponential number of constraints, and can be solved within a constraint generation framework. In Section 5, we present two innovative ways to construct difficult instances for the problem and we report the findings of a large number of computational experiments. Finally, conclusions are drawn in Section 6.

2 Problem definition

In the 2-BCPP we are given an unlimited number of bins with capacity c and a set of n bar charts, each of them composed of two bars (or items, the terms are interchangeable) that must be packed in consecutive bins: the first bar whose weight is denoted by w_{i1} and the second bar whose weight is denoted by w_{i2}. We call $w_i = (w_{i1}, w_{i2})$ the weight vector of bar chart i. For feasibility purposes, we assume that $w_{i1}, w_{i2} \leq c$ for every bar chart $i = 1, \ldots, n$. We also assume that w_{i1}, w_{i2} and c are strictly positive integers and that every bar chart $i = 1, \ldots, n$ has an integer demand q_i. Note that in the original 2-BCPP definition from Erzin et al. (2021b), every bar chart had demand 1 and bar weights were positive real numbers. However, an instance of such problem can be transformed to fit our definition by merging duplicated bar charts, computing the associated demands, and multiplying the bar weights and the capacity by a suitable coefficient.

Given a 2-BCPP solution, we say that bar chart i is *packed* into bin j if the first item of bar chart i is packed into bin j (and thus, if the second item of bar chart i is packed into bin $j+1$). In addition, we say that a solution (or a packing) is *feasible* if the demand, the contiguity, and the capacity constraints are satisfied. We call z the *length* of a feasible packing, which is defined as the number of bins in the solution containing at least one item. For a feasible solution of length z, it is assumed that at least one item is packed in every bin $j = 1, 2, \ldots, z$. If this is not the case, one can simply delete the empty intermediate bins and shift the solution. Finally, we say that a feasible packing with length $\bar{z}$ is *optimal* if there exists no solution with length z such that $z < \bar{z}$.

Example 1 *Let us consider the following 2-BCPP instance with $n = 3$, $w_1 = (4, 1)$, $w_2 = (3, 2)$, $w_3 = (1, 4)$, $q_1 = q_2 = q_3 = 1$, and $c = 5$. An optimal packing is shown in Figure 2 where the first bar chart is packed in bin 1 (i.e., its first and second items are packed in bins 1 and 2, respectively), the second bar chart is packed in bin 2, and the third bar chart is packed in bin 3. This solution has length $\bar{z} = 4$ as it uses 4 bins.* □

Let us consider a valid upper bound K on the minimum number of bins required to pack all the bar charts. A trivial value for K is $2\sum_{i=1}^{n} q_i$. By introducing binary decision variables y_j that take value 1 if bin j is opened, and 0 otherwise ($j = 1, \ldots, K$) and integer decision variables

Figure 2: Optimal solution for the 2-BCPP instance introduced in Example 1

x_{ij} that indicate the number of times bar chart i is packed into bin j ($i = 1, \ldots, n,\ j = 1, \ldots, K$), the 2-BCPP can be modelled as follows (Erzin et al. 2021b):

$$\min \quad \sum_{j=1}^{K} y_j \tag{1}$$

$$\text{s.t.} \quad \sum_{j=1}^{K} x_{ij} = q_j \qquad i = 1, \ldots, n \tag{2}$$

$$\sum_{i=1}^{n} w_{i1} x_{i1} \leq c y_1 \tag{3}$$

$$\sum_{i=1}^{n} (w_{i1} x_{ij} + w_{i2} x_{i,j-1}) \leq c y_j \qquad j = 2, \ldots K \tag{4}$$

$$y_j \geq y_{j+1} \qquad j = 1, \ldots K - 1 \tag{5}$$

$$y_j \in \{0, 1\} \qquad j = 1, \ldots, K \tag{6}$$

$$x_{ij} \in \mathbb{N}_0 \qquad i = 1, \ldots, n,\ j = 1, \ldots, K \tag{7}$$

The objective function (1) minimizes the number of bins used while constraints (2) make sure that the bar chart demands are satisfied. Constraints (3) and (4) ensure that the sum of the item weights contained in a given bin never exceeds the bin capacity while also preventing any item to be packed in a closed bin (i.e., a bin whose corresponding y_j variable is equal to 0). Finally, symmetry breaking constraints (5) make sure that the bins are opened from lowest to highest index (i.e. without intermediate empty bins) and constraints (6) and (7) limit the variable domains. Model (1) - (7) uses $O(nK)$ variables and $O(n+K)$ constraints, which means that for reasonably small n and K values, the model size should not be a problem. Yet, we will show in our computational experiments that instances with as few as 50 bar charts cannot be solved after one hour of computing time by a commercial ILP solver, which motivates the introduction of better formulations.

3 Arcflow-based lower and upper bounding procedures

For many packing problems, researchers have shown that arcflow formulations provided much better results on average than "Kantorovich-like" models (see Delorme et al. 2016, Martinovic and Scheithauer 2016, Dell'Amico et al. 2019). The name "arcflow" originates from the fact that the variables in the model represent flows on individual arcs in a directed graph (see de Lima et al. 2022 for a recent survey on the topic). Arcflow formulations are known to have a very tight LP-relaxation, but this comes at the cost of not being polynomial in the input size. Indeed, both the number of variables and constraints involved in such models are usually polynomial in the capacity while the latter is coded as $O(\log_2 c)$ bits in the input, which is why arcflow formulations are said to be "pseudo-polynomial". There is a growing interest in the research community for arcflow formulations as: (i) they are usually easy to implement as they do not involve advanced optimization techniques, and (ii) they result in models that can often be solved to optimality with a state-of-the-art ILP solver, especially for instances with a reasonable capacity. In summary, arcflow models are seen as a good compromise between simplicity and efficiency.

Typically, an approach using an arcflow model is composed of three phases: (i) the graph construction phase, where the instance I of the original problem is transformed into an instance I' of a graph problem, (ii) the solving phase, where the instance I' is solved (e.g., with an ILP solver), and (iii) the mapping phase, where the solution of instance I' is mapped into a solution of instance I. In the following, we briefly introduce arcflow formulations for the CSP and for the 2D-VPP, and explain how they can be used to derive valid lower and upper bounds for the 2-BCPP.

3.1 Arcflow-based lower bounding procedure

The idea behind the arcflow formulation for the CSP was introduced by Wolsey (1977) and popularized by Valério de Carvalho (1999). The CSP is transformed into a graph problem where the objective is to minimize the flow going from a source node to a target node under demand and flow conservation constraints. In this subsection, n denotes the number of items in a CSP instance, w_i is the weight of item i ($i = 1, \ldots, n$), and q_i is the demand of the item.

Graph construction phase: we consider a graph $G = (V, A)$ where vertex set $V = \{0, 1, \ldots, c\}$ and where arc set $A = \big\{(d, e) : 0 \leq d < e \leq c \text{ and } e - d = w_i \text{ for } i = 1, \ldots, n\big\} \cup \big\{(d, d+1) : 0 \leq d < c\big\}$. Note that symmetry reduction techniques can be used to reduce the size of V and A (see Valério de Carvalho 1999, Côté and Iori 2018).

Solving phase: by introducing integer decision variables f_{de} that indicate the number of times arc $(d, e) \in A$ is selected, the arcflow model for the CSP can be defined as follows:

$$\min \quad z \tag{8}$$

$$\text{s.t.} \quad \sum_{\substack{(d,e)\in A \\ d=v}} f_{de} - \sum_{\substack{(d,e)\in A \\ e=v}} f_{de} = \begin{cases} z & \text{if } v = 0 \\ 0 & \text{if } v = 1, \ldots, c-1 \\ -z & \text{if } v = c \end{cases} \qquad v \in V \tag{9}$$

$$\sum_{\substack{(d,e)\in A \\ e-d=w_i}} f_{de} = q_i \qquad i = 1,\ldots,n \tag{10}$$

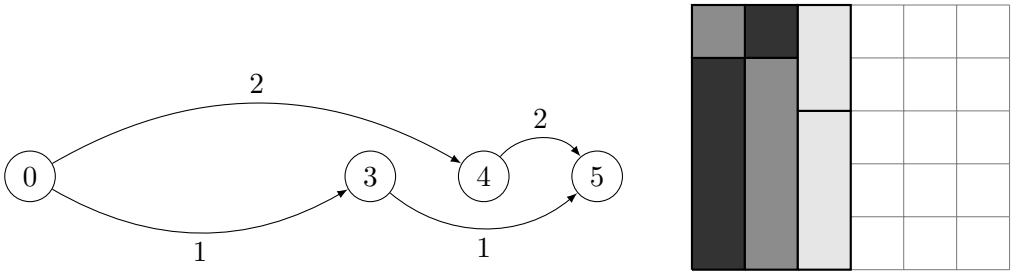
$$f_{de} \in \mathbb{N}_0 \qquad (d,e) \in A \tag{11}$$

The objective function (8) minimizes the flow going from 0 to c. Constraints (9) ensure that z units of flow both leave node 0 and enter node c, while also enforcing flow conservation in the other nodes. Constraints (10) make sure that the demand is satisfied while constraints (11) limit the variable domains.

Mapping phase: a flow decomposition algorithm is used on the optimal solution of model (8) - (11) to obtain a set of $0-c$ paths with unitary flow corresponding to an optimal CSP solution.

Example 1 *(resumed) By splitting every bar chart in the 2-BCPP instance, we obtain a CSP instance where* $w = (4,3,2,1)$*,* $q = (2,1,1,2)$*, and* $c = 5$*. The optimal arcflow solution is depicted on the left part of Figure 3 and uses three bins: two bins containing an item with weight 4 and an item with weight 1 each, and one bin containing an item with weight 3 and an item with weight 2. Therefore, a valid lower bound for the original 2-BCPP instance is 3. A graphical representation of the solution is shown on the right part of Figure 3.* □

Figure 3: Optimal arcflow solution for the CSP relaxation of Example 1 and its graphical representation



3.2 Arcflow-based upper bounding procedure

The arcflow formulation for the 2D-VPP can be attributed to Brandão and Pedroso (2016) who extended the CSP version of the model to be able to take into account additional dimensions. The 2D-VPP is again transformed into a graph problem where the objective is to minimize the flow leaving a source node under demand and (a form of) flow conservation constraints. In this subsection, n designs the number of items in a 2D-VPP instance, (c_1, c_2) is the capacity of the bin, $w_i = (w_{i1}, w_{i2})$ is the weight vector of two-dimensional item i $(i = 1,\ldots,n)$, and q_i is the demand of the item.

Graph construction phase: we consider a graph $G = (V, A)$ where vertex set $V = \Big\{(v_1, v_2) : 0 \le v_1 \le c_1, 0 \le v_2 \le c_2\Big\}$ and where arc set $A = \Big\{((d_1, d_2), (e_1, e_2)) : 0 \le d_1 < e_1 \le c_1$ and $0 \le$

$d_2 < e_2 \leq c_2$ and $(e_1 - d_1, e_2 - d_2) = w_i$ for $i = 1, \ldots, n\Big\}$. Note that symmetry reduction techniques can also be used to reduce the size of V and A (see Brandão and Pedroso 2016). As it is relevant for the models introduced in the next section, we provide in Algorithm 1 of the appendix a pseudo-code to construct graph G with a reduced vertex set V and a reduced arc set A.

Solving phase: by introducing integer decision variables $f_{(d_1,d_2),(e_1,e_2)}$ that indicate the number of times arc $\big((d_1, d_2), (e_1, e_2)\big) \in A$ is selected, the arcflow model for the 2D-VPP can be defined as follows:

$$\min \sum_{\substack{\big((d_1,d_2),(e_1,e_2)\big)\in A \\ (d_1,d_2)=(0,0)}} f_{(d_1,d_2),(e_1,e_2)} \tag{12}$$

$$\text{s.t.} \sum_{\substack{\big((d_1,d_2),(e_1,e_2)\big)\in A \\ (e_1,e_2)=(v_1,v_2)}} f_{(d_1,d_2),(e_1,e_2)} \geq \sum_{\substack{\big((d_1,d_2),(e_1,e_2)\big)\in A \\ (d_1,d_2)=(v_1,v_2)}} f_{(d_1,d_2),(e_1,e_2)} \qquad (v_1, v_2) \in V \setminus \{(0,0)\} \tag{13}$$

$$\sum_{\substack{\big((d_1,d_2),(e_1,e_2)\big)\in A \\ (e_1-d_1,e_2-d_2)=(w_{i1},w_{i2})}} f_{(d_1,d_2),(e_1,e_2)} = q_i \qquad i = 1, \ldots, n \tag{14}$$

$$f_{(d_1,d_2),(e_1,e_2)} \in \mathbb{N}_0 \qquad \big((d_1, d_2), (e_1, e_2)\big) \in A \tag{15}$$

The objective function (12) minimizes the flow going out of node $(0, 0)$. Constraints (13) ensure that the flow entering any node (v_1, v_2) that is not $(0, 0)$ is never below the flow leaving that node. Constraints (14) make sure that the demand is satisfied while constraints (15) limit the variable domains.

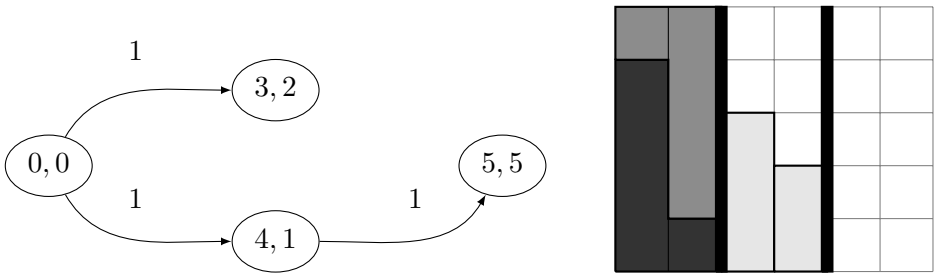
Mapping phase: a flow decomposition algorithm is used on the optimal solution of model (12) - (15) to obtain a set of $(0, 0) - (v_1, v_2)$ paths $\big((v_1, v_2) \in V\big)$ with unitary flow corresponding to an optimal 2D-VPP solution.

Example 1 *(resumed) After converting the 2-BCPP instance, we obtain a 2D-VPP instance where* $n = 3$, $w_1 = (4, 1)$, $w_2 = (3, 2)$, $w_3 = (1, 4)$, $q_1 = q_2 = q_3 = 1$, *and* $c_1 = c_2 = 5$. *The optimal arcflow solution is depicted on the left part of Figure 4 and uses two bins: the first one containing items 1 and 3 and the second one containing item 2. Therefore, a valid upper bound for the original 2-BCPP instance is 4 as one bin in the 2D-VPP corresponds to two bins in the 2-BCPP. A graphical representation of the solution is shown on the right part of Figure 4.* □

4 New arcflow formulations for the 2-BCPP

Even if the arcflow formulation was originally proposed for the CSP, it has been successfully extended to many other relevant packing problems in the last two decades (see Macedo et al. 2010, Nesello et al. 2018, Delorme et al. 2021). In this section, we first introduce Eulerian-flow, a non-trivial arcflow extension for the 2-BCPP where a feasible solution is represented by a single cycle in a graph (an Eulerian cycle). We will observe that while forcing the arcs selected in a

Figure 4: Optimal arcflow solution for the 2D-VPP transformation of Example 1 and its graphical representation



solution to form cycles is relatively easy, making sure that these cycles are all connected (i.e., that there are no subtours) is much harder and can only be achieved by using an exponential number of constraints. As the model size quickly becomes too large to be solved by an ILP solver, we propose a constraint generation framework that only adds the subtour elimination constraints that are necessary to find an optimal solution. We then introduce link-flow, a model inspired by Eulerian-flow where the number of variables is dramatically reduced at the cost of an extra set of constraints.

4.1 Eulerian-flow

Our first model builds upon the arcflow formulation for the 2D-VPP described in the previous section where a feasible packing was represented as a set of paths starting from node $(0, 0)$. In that formulation, one can easily modify the arc set definition to incorporate *feedback arcs* $\big((v_1, v_2), (0, 0)\big)$ for every $(v_1, v_2) \in V$ to represent the end of a bin and updating flow conservation constraints (13) to:

$$\sum_{\substack{\big((d_1,d_2),(e_1,e_2)\big)\in A \\ (e_1,e_2)=(v_1,v_2)}} f_{(d_1,d_2),(e_1,e_2)} = \sum_{\substack{\big((d_1,d_2),(e_1,e_2)\big)\in A \\ (d_1,d_2)=(v_1,v_2)}} f_{(d_1,d_2),(e_1,e_2)} \qquad (v_1, v_2) \in V \qquad (16)$$

A feasible solution f now consists of a set of arcs that can be split into a number of cycles where each cycle contains node $(0, 0)$. Let us define the directed multigraph $G_f = (V_f, A_f)$, also called the *solution graph* thereafter, where V_f contains the set of vertices visited in solution f and where A_f contains $f_{(d_1,d_2),(e_1,e_2)}$ copies of arc $\big((d_1, d_2), (e_1, e_2)\big)$ for every $\big((d_1, d_2), (e_1, e_2)\big) \in A$. According to Euler's theorem (developed to solve the Königsberg bridge problem in 1736), a directed graph contains an Eulerian cycle if and only if (i) the in-degree of any vertex is equal to its out-degree, and (ii) the graph is strongly connected. Solution graph G_f contains an Eulerian cycle as condition (i) is satisfied thanks to the updated flow conservation constraints (16) and condition (ii) is satisfied because the arcs in A_f can be split into a set of cycles such that node $(0, 0)$ is included in every cycle.

This updated formulation is not yet suitable for the 2-BCPP as it fills the bins by pairs, which means that it does not allow the first item of a bar chart to be packed with the second

item of another bar chart. To correct this issue, we introduce *transition arcs* $((v_1, v_2), (v_2, 0))$ for every $(v_1, v_2) \in V$ to represent the end of a 2-BCPP bin. Intuitively, one could see a transition arc as half of a feedback arc. In fact, using two successive transition arcs $((v_1, v_2), (v_2, 0))$ and $((v_2, 0), (0, 0))$ is the same as using one feedback arc $((v_1, v_2), (0, 0))$. We point out that after introducing transition arcs, the arcs in A_f can still be split into a set of cycles, but node $(0, 0)$ is not necessarily included in each of these cycles anymore. This means that an additional set of constraints is required to make sure that the solution graph G_f is strongly connected (and thus, that it contains an Eulerian cycle).

Graph construction phase: we consider a graph $G = (V, A)$ where vertex set $V = \Big\{(v_1, v_2) : 0 \leq v_1 \leq c, 0 \leq v_2 \leq c\Big\}$ and where arc set $A = \Big\{((d_1, d_2), (e_1, e_2)) : 0 \leq d_1 < e_1 \leq c \text{ and } 0 \leq d_2 < e_2 \leq c \text{ and } (e_1 - d_1, e_2 - d_2) = (w_{i1}, w_{i2}) \text{ for } i = 1, \ldots, n\Big\} \cup \Big\{((d_1, d_2), (d_2, 0)) : 0 \leq d_1 \leq c \text{ and } 0 \leq d_2 \leq c\Big\}$. As the number of vertices and arcs grows quickly with c, we detail in the following how graph G can be constructed with a reduced vertex set V and a reduced arc set A. Note that, to avoid any confusion, we temporarily use different notation for the initial graph G^α, the intermediary graph G^β, and the final graph G^γ.

1. Order the bar charts (e.g., from the largest bar chart to the smallest)
2. Construct graph $G^\alpha = (V^\alpha, A^\alpha)$ following Algorithm 1 (available in the appendix);
3. Save in set V^h every distinct v_2 value such that $(v_1, v_2) \in V^\alpha$ and $v_2 \neq 0$;
4. Construct graph $G^\beta = (V^\beta, A^\beta)$ following a modified version of Algorithm 1 where set V is initialized to V^h instead of $\{(0, 0)\}$ and create $V^\gamma = V^\alpha \cup V^\beta$ and $A^\gamma = A^\alpha \cup A^\beta$;
5. For every vertex $(v_1, v_2) \in V^\gamma \setminus \{(0, 0)\}$, add transition arc $\big((v_1, v_2), (v_2, 0)\big)$ to A^γ;

We point out that it is not necessary to update set V^h after the fourth step because every arc $((d_1, d_2), (e_1, e_2))$ in A^β has, by construction, its counterpart $\big((d_1 - v, d_2), (e_1 - v, e_2)\big)$ in A^α where $v \in V^h$. In fact, we can use this fact to generate V^γ and A^γ more efficiently by duplicating A^α and "shifting" the first dimension of both the tail and the head of every arc by the values in V^h as described in Algorithm 2 of the appendix. In the next phases, we refer to graph $G = (V, A)$ as the final graph G^γ built in this phase.

Solving phase: for the sake of compactness, we now sometimes refer to an arc as a instead of $\big((d_1, d_2), (e_1, e_2)\big)$. For every vertex $(v_1, v_2) \in V$, we also define index sets $\delta^+(v_1, v_2)$ (resp. $\delta^-(v_1, v_2)$) as the set of arcs leaving (resp. entering) vertex (v_1, v_2). For every $i = 1, \ldots, n$, we denote by A_i the set of arcs representing the packing of a bar chart of type i (i.e., the set of arcs $\big((d_1, d_2), (e_1, e_2)\big) \in A$ such that $(e_1 - d_1, e_2 - d_2) = (w_{i1}, w_{i2})$). We call $A_1 \cup A_2 \cup \cdots \cup A_n$ the set of *bar chart arcs*. We also denote by A_0 the set of transition arcs. By introducing integer decision variables f_a that indicate the number of times arc $a \in A$ is selected, the Eulerian-flow model can then be defined as follows:

$$\min \quad \sum_{a \in A_0} f_a \tag{17}$$

$$\text{s.t.} \quad \sum_{a \in \delta^+(v_1,v_2)} f_a = \sum_{a \in \delta^-(v_1,v_2)} f_a \qquad (v_1,v_2) \in V \qquad (18)$$

$$\sum_{a \in \delta^+(0,0)} f_a \geq 1 \qquad (19)$$

$$\sum_{a \in A_i} f_a = q_i \qquad i = 1,\ldots,n \qquad (20)$$

$$\sum_{\substack{a \in \delta^+(d_1,d_2) \cap \delta^-(e_1,e_2) \\ (d_1,d_2) \in \widetilde{V},\, (e_1,e_2) \in \widetilde{V}}} f_a \leq M \cdot \sum_{\substack{a \in \delta^+(d_1,d_2) \cap \delta^-(e_1,e_2) \\ (d_1,d_2) \in \widetilde{V},\, (e_1,e_2) \notin \widetilde{V}}} f_a \qquad \widetilde{V} \subseteq V \setminus \{(0,0)\} \qquad (21)$$

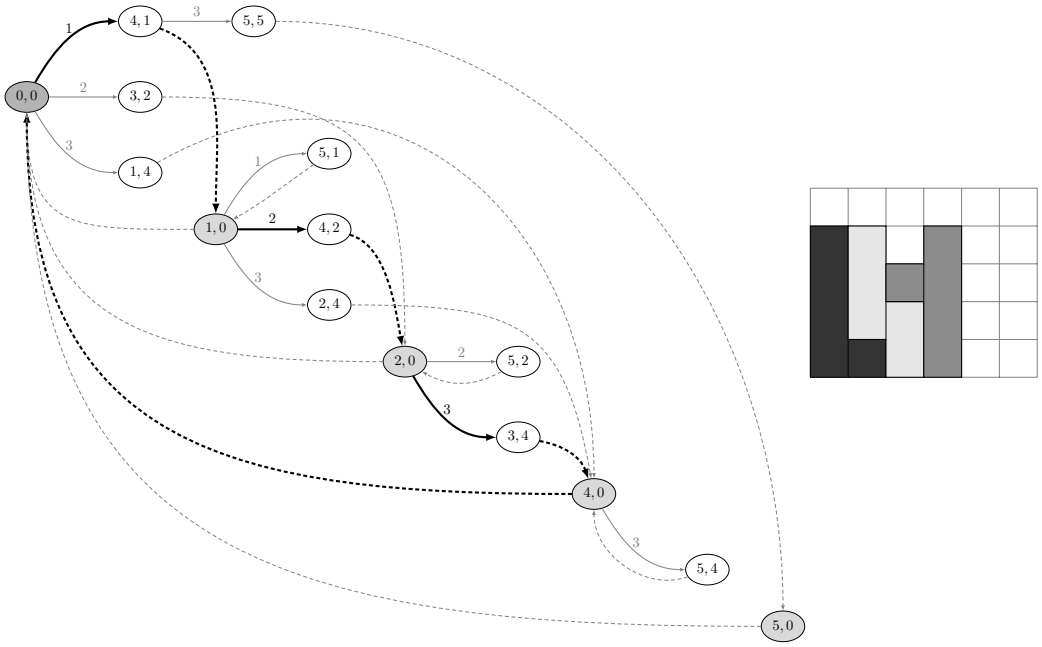
$$f_a \in \mathbb{N}_0 \qquad a \in A \qquad (22)$$

The objective function (17) minimizes the number of transition arcs used (i.e., the length of the packing). Constraints (18) ensure flow conservation while constraint (19) makes sure that there is at least one unit of flow leaving node $(0,0)$. Furthermore, constraints (20) enforce demand satisfaction and constraints (21) eliminate subtours so that the solution graph G_f is strongly connected. More specifically, big-M constraints (21) ensure that, for every potential subtour (i.e., every subset of nodes $\widetilde{V}$ that does not contain (0,0)), there can only be a flow circulating in the subtour (i.e., in an arc whose tail and head are both in $\widetilde{V}$) if there is also a flow leaving the subtour (i.e., passing through an arc whose tail is in $\widetilde{V}$ and whose head is not in $\widetilde{V}$). A suitable big-M value for constraints (21) is $\sum_{i=1}^{n} q_i + K$ as at most $\sum_{i=1}^{n} q_i$ units of flow circulate in bar chart arcs and at most K units of flow circulate in transition arcs in total (we recall that K is an upper bound on the minimum number of bins). Note that we could decrease big-M values by making them constraint-specific. In that case, for the constraint associated with a given subset of nodes $\widetilde{V}$, we could remove the q_i values of the bar charts for which there is no arc $a \in A_i$ with both tail and head in $\widetilde{V}$. Model (17) - (22) uses $O(nc^2)$ variables and $O(2^{(c^2)})$ constraints.

Mapping phase: after an optimal solution f has been found for model (17)-(22), we create the solution graph G_f and we find an Eulerian cycle in the graph starting in node $(0,0)$. Once such a cycle is obtained, we reconstruct a 2-BCPP solution by looping over every arc a in the Eulerian cycle and by either inserting the bar chart associated with a into bin j if a is an item arc or incrementing j if a is a transition arc (j is initialized to 1).

Example 1 *(resumed) We solve the same 2-BCPP instance with* $n = 3$, $w_1 = (4,1)$, $w_2 = (3,2)$, $w_3 = (1,4)$, $q_1 = q_2 = q_3 = 1$, *and* $c = 5$. *The graph obtained after the graph construction phase is depicted on the left part of Figure 5. Solid lines are bar chart arcs (the number on top of the arc is the bar chart index) and dashed lines are transition arcs. The solution obtained after the solving phase (e.g., given by an ILP solver) results in the flow* f *that is printed in bold and black on the left part of Figure 5 (every arc in* f *carries one unit of flow). To transform* f *into a valid 2-BCPP solution through the mapping phase, we identify Eulerian cycle* $(0,0) \xrightarrow{1} (4,1) \rightarrow (1,0) \xrightarrow{2} (4,2) \rightarrow (2,0) \xrightarrow{3} (3,4) \rightarrow (4,0) \rightarrow (0,0)$, *we initialize the bin index to 1, and we start looping over every arc in the cycle. We pack bar chart* 1 *in bin 1 (i.e., its first item is packed in bin 1 and its second item is packed in bin 2), then we increment the bin index because the second arc is a transition arc, then we pack bar chart* 2 *in bin 2, we increment again the bin index,*

Figure 5: Eulerian-flow graph for Example 1, optimal solution, and graphical representation



we pack bar chart 3 *in bin 3, and we finish by incrementing the bin index twice. A graphical representation of the 2-BCPP solution is shown on the right part of Figure 5 and uses 4 bins.*

Note that for this instance, an optimal solution to Eulerian-flow without subtour elimination constraints has objective value 3 and is composed of two cycles: $(0,0) \xrightarrow{1} (4,1) \xrightarrow{3} (5,5) \rightarrow (5,0) \rightarrow (0,0)$ *and* $(2,0) \xrightarrow{2} (5,2) \rightarrow (2,0)$*. This solution satisfies flow conservation, demand constraints, and has at least one unit of flow leaving node* $(0,0)$*. However, it cannot be mapped into a valid 2-BCPP solution as its associated solution graph does not contain an Eulerian cycle.* □

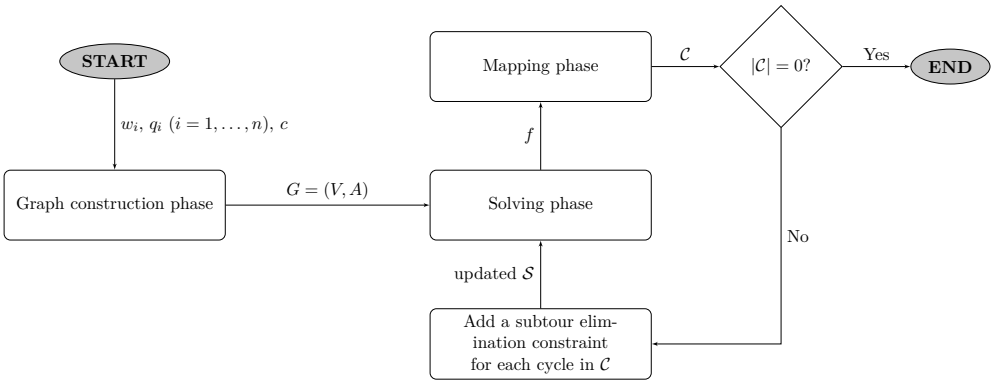
4.2 Constraint generation framework

A common way to solve ILP models that involve an exponential number of constraints is to use a constraint generation framework. The key idea behind constraint generation is to solve the formulation with a subset of constraints $\mathcal{S}$ (e.g., with an ILP solver) and check if the returned solution is feasible for the initial problem (i.e., if the solution found also satisfies the constraints that were not included in $\mathcal{S}$). If this is the case, then the solution is optimal. Otherwise, it means that at least one of the constraints not considered in the reduced model was important and needs to be added to $\mathcal{S}$. The process is iterated until an optimal solution is found. Note that constraint generation is very well-known in the routing community (see the work of Pferschy and Staněk 2017 on constraint generation for the travelling salesman problem).

A flowchart of our constraint generation framework is presented in Figure 6. After the graph construction phase where graph $G = (V, A)$ is generated, we solve the Eulerian-flow model

without subtour elimination constraints to obtain a solution f. We then create the solution graph G_f, split the arcs of A_f into the minimum number of disjoint cycles, and store the ones that do not contain node $(0,0)$ in $\mathcal{C}$. If $|\mathcal{C}| > 0$, then f contains one or more subtours, and thus, cannot be mapped into a feasible 2-BCPP solution. As a result, a subtour elimination constraint is added to $\mathcal{S}$ forbidding each of the $|\mathcal{C}|$ subtours and Eulerian-flow is solved once more with the updated subset $\mathcal{S}$. If $|\mathcal{C}| = 0$, the algorithm stops as f can be mapped into a feasible 2-BCPP solution.

Figure 6: Flowchart for constraint generation framework



Example 1 *(resumed) After solving the Eulerian-flow model without subtour elimination constraints, one could obtain a solution f containing subtour $(2,0) \xrightarrow{2} (5,2) \rightarrow (2,0)$. Such a solution can be prevented from being generated again during the solving phase by adding subtour elimination constraint $f_{(2,0),(5,2)} + f_{(5,2),(2,0)} \leq M \cdot (f_{(2,0),(3,4)} + f_{(2,0),(0,0)})$ to* (21). □

We point out that, in state-of-the-art ILP solvers, constraint generation frameworks are usually implemented within *callbacks* (a set of functions created by the user that is called during the enumeration tree). In our callback implementation, every integer solution found by the solver is tested for subtours. If at least one subtour is found, relevant constraints are added to the solver and the solution is therefore rejected. If no subtour is found, the solution becomes the new *incumbent* and the upper bound is therefore updated. In both cases, the solver continues its enumeration tree until the lower and upper bounds match.

4.3 Link-flow

Even though reduction techniques were applied in the graph construction phase of Eulerian-flow, preliminary tests showed that the model size grows quickly as n and c increase, even when the constraint generation framework is used. This is mainly due to the fourth step of the graph construction phase where the initial arc set A^α is duplicated up to $c-1$ times to allow any (ordered) combination of bar chart arcs to follow a transition arc. The key idea behind link-flow is to move the transition arc from the beginning of a bin to the end so that every bin now always starts from node $(0,0)$ and ends with a transition arc. While the idea appears to be relatively

simple and effective (as the duplication phase is no longer needed), its implementation is not necessarily trivial as we now need to make sure that the transition arcs "connect" to each other so that the bins form a feasible 2-BCPP solution. Intuitively, transition arcs can be seen as a set of pieces of a puzzle and the subset we select needs to form a circle.

For the sake of clarity, we describe link-flow model using two distinct graphs G^1 and G^2 (even though we will observe in the solving phase that building G^2 is not strictly necessary). The *primary graph* G^1 contains item arcs and *link arcs* (the counterpart of transition arcs in link-flow) while the *secondary graph* G^2 contains link arcs and *loss arcs*. Constraints in G^1 make sure that the bins created are feasible and that item requirements are satisfied. Constraints in G^2 make sure that link arcs (and thus, bins) can be combined together to form a feasible 2-BCPP solution (possibly with the help of loss arcs). This is done by ensuring that the secondary solution graph G^2_f contains an Eulerian cycle. Supplementary constraints ensure that the link arcs selected in G^1 and in G^2 are the same.

Graph construction phase: the primary graph $G^1 = (V^1, A^1)$ is initially constructed as the arcflow graph for the 2D-VPP (see Section 3.2 and Algorithm 1 in the appendix). Link arcs $((v_1, v_2), (0, 0))$ are then added to A^1 for every node $(v_1, v_2) \in V^1$. The secondary graph $G^2 = (V^2, A^2)$ has vertex set $V^2 = \{0, 1, \dots, c\}$ and arc set $A^2 = \{(c - v_1, v_2, lk) : (v_1, v_2) \in V^1\} \cup \{(d, d + 1, ls) : 0 \leq d < c\}$ where the third index separates the link arcs (identified by "lk") from the loss arcs (identified by "ls").

Solving phase: for every primary vertex $(v_1, v_2) \in V^1$, we still use $\delta^+(v_1, v_2)$ as the set of primary arcs leaving (v_1, v_2) and $\delta^-(v_1, v_2)$ as the set of primary arcs entering (v_1, v_2). Analogously, we define index sets $\delta^+(v)$ and $\delta^-(v)$ for every secondary vertex $v \in V^2$ as the set of secondary arcs leaving and entering v. For every $i = 1, \dots, n$, we denote by A^1_i the set of primary arcs representing the packing of a bar chart of type i. We call $A^1_1 \cup A^1_2 \cup \cdots \cup A^1_n$ the set of bar chart arcs and we denote by A^1_0 the set of link arcs in G^1. By introducing integer decision variables f^1_a (resp. f^2_a) that indicate the number of times primary (resp. secondary) arc $a \in A^1$ (resp. $a \in A^2$) is selected, the link-flow model can then be defined as follows:

$$\min \quad \sum_{a \in A^1_0} f^1_a \tag{23}$$

$$\text{s.t.} \quad \sum_{a \in \delta^+(v_1, v_2)} f^1_a = \sum_{a \in \delta^-(v_1, v_2)} f^1_a \qquad (v_1, v_2) \in V^1 \tag{24}$$

$$\sum_{a \in \delta^+(v)} f^2_a = \sum_{a \in \delta^-(v)} f^2_a \qquad v \in V^2 \tag{25}$$

$$\sum_{a \in \delta^+(0)} f^2_a \geq 1 \tag{26}$$

$$f^1_{(c-d,e),(0,0)} = f^2_{d,e,lk} \qquad (d, e, lk) \in A^2 \tag{27}$$

$$\sum_{a \in A^1_i} f^1_a = q_i \qquad i = 1, \dots, n \tag{28}$$

$$\sum_{\substack{a \in \delta^+(d) \cap \delta^-(e) \\ d \in \widetilde{V}^2,\, e \in \widetilde{V}^2}} f^2_a \leq M \cdot \sum_{\substack{a \in \delta^+(d) \cap \delta^-(e) \\ d \in \widetilde{V}^2,\, e \notin \widetilde{V}^2}} f^2_a \qquad \widetilde{V}^2 \subseteq V^2 \setminus \{0\} \tag{29}$$

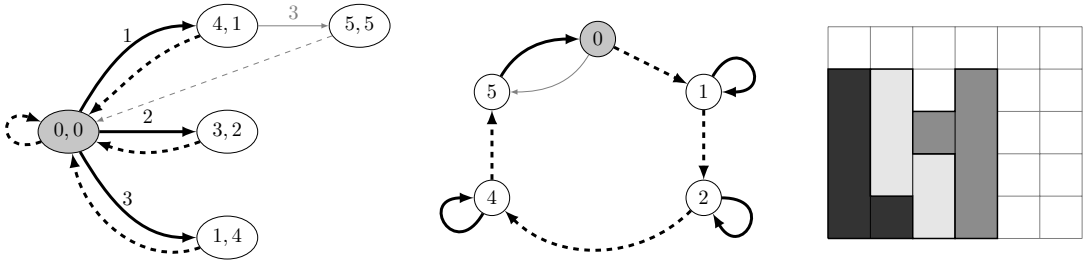
$$f_a^1 \in \mathbb{N}_0 \qquad a \in A^1 \qquad (30)$$
$$f_a^2 \in \mathbb{N}_0 \qquad a \in A^2 \qquad (31)$$

The objective function (23) minimizes the length of the packing. Constraints (24) and (25) enforce flow conservation in the primary and secondary graphs while constraint (26) makes sure that at least one unit of flow leaves secondary vertex 0. Furthermore, constraints (27) ensure that the link arcs selected in G^1 are the same as the link arcs selected in G^2, constraints (28) enforce demand satisfaction, and constraints (29) eliminate subtours so that the solution graph G_f^2 is strongly connected. A suitable big-M value for constraints (29) is $K + Kc$ as at most K link arcs are selected in G^1 (and thus, in G^2) and at most c loss arcs are required in G^2 per link arc. Note that we could decrease big-M values by making them constraint-specific. In that case, for the constraint associated with a given subset of nodes $\widetilde{V}^2$, we could use the actual number of loss arcs having both tail and head in $\widetilde{V}^2$ instead of the upper bound c. Model (23) - (31) uses $O(nc^2)$ variables and $O(2^c)$ constraints. Note that we can reduce the model size even further by replacing variables $f^2_{d,e,lk}$ associated with secondary link arcs $(d, e, lk) \in A^2$ by their counterpart $f^1_{(c-d,e),(0,0)}$ in the primary graph and by removing constraints (27). We can also use the constraint generation framework introduced previously to handle subtour elimination constraints (29).

Mapping phase: after an optimal solution $f = (f^1, f^2)$ has been found for model (23) - (31), we use a flow decomposition algorithm to obtain a set of $(0,0) - (0,0)$ cycles with unitary flow in the primary graph. Each of these cycles corresponds to a bin and we now need to order these bins. To do so, in the secondary graph, we create solution graph G_f^2 and we find an Eulerian cycle starting from node 0. Note that G_f^2 is constructed as described in Eulerian-flow: $G_f^2 = (V_f^2, A_f^2)$, where V_f^2 contains the set of secondary vertices visited in solution f and where A_f^2 contains $f^2_{d,e,l}$ copies of arc (d, e, l) for every $(d, e, l) \in A^2$. After initialising j to 1, we reconstruct a 2-BCPP solution by looping over every arc a in the Eulerian cycle and by inserting in bin j either one unit of loss space if a is a loss arc or the bar charts of the $(0,0) - (0,0)$ cycle that includes the counterpart of a in the primary graph otherwise. Index j is incremented every time a is a link arc.

Example 1 *(resumed) Consider again the running example. The primary and secondary graphs obtained after the graph construction phase are depicted on the left and middle parts of Figure 7, respectively. In the primary graph, solid lines are bar chart arcs (the number on top of the arc is the bar chart index) and dashed lines are link arcs. In the secondary graph, solid lines are link arcs and dashed lines are loss arcs. Note that as no secondary link arc starts or ends in 3, loss arcs* $(2, 3, ls)$ *and* $(3, 4, ls)$ *were merged into* $(2, 4, ls)$*. The solution obtained after the solving phase (e.g., given by an ILP solver) results in the flow* $f = (f^1, f^2)$ *that is printed in bold and black in the two graphs (every arc in* f^1 *and* f^2 *carries one unit of flow). To transform* f *into a valid 2-BCPP through the mapping phase, we first determine the* $(0,0) - (0,0)$ *cycles in the primary graph:* $(0,0) \xrightarrow{1} (4,1) \rightarrow (0,0)$*,* $(0,0) \xrightarrow{2} (3,2) \rightarrow (0,0)$*,* $(0,0) \xrightarrow{3} (1,4) \rightarrow (0,0)$*, and* $(0,0) \rightarrow (0,0)$*. We then identify Eulerian cycle* $0 \xrightarrow{ls} 1 \xrightarrow{lk} 1 \xrightarrow{ls} 2 \xrightarrow{lk} 2 \xrightarrow{ls} 4 \xrightarrow{lk} 4 \xrightarrow{ls} 5 \xrightarrow{lk} 0$ *in the secondary graph. We pack one unit of loss space in bin 1 together with bar chart* 1 *as secondary link arc* $(1, 1)$ *is associated with primary link arc* $\big((4,1),(0,0)\big)$*. We increment the bin*

Figure 7: Link-flow primary and secondary graphs for Example 1, optimal solution, and graphical representation



index, pack one unit of loss space in bin 2 together with bar chart 2 *as secondary link arc* $(2, 2)$ *is associated with primary link arc* $\big((3, 2), (0, 0)\big)$*. We increment the bin index, pack two units of loss space in bin 3 together with bar chart* 3 *as secondary link arc* $(4, 4)$ *is associated with primary link arc* $\big((1, 4), (0, 0)\big)$*. We increment the bin index and pack one unit of loss space in bin 4. Finally, we increment the bin index and complete the Eulerian cycle with secondary link arc* $(5, 0)$ *which is associated with primary link arc* $\big((0, 0), (0, 0)\big)$*. A graphical representation of the 2-BCPP solution is shown on the right part of Figure 7 and uses 4 bins.*

Note that for this instance, an optimal solution to link-flow without subtour elimination constraints has objective value 3 and is composed of cycles $(0, 0) \xrightarrow{1} (4, 1) \xrightarrow{3} (5, 5) \rightarrow (0, 0)$*,* $(0, 0) \xrightarrow{2} (3, 2) \rightarrow (0, 0)$*, and* $(0, 0) \rightarrow (0, 0)$ *in the primary graph and* $0 \xrightarrow{lk} 5 \xrightarrow{lk} 0$ *and* $2 \xrightarrow{lk} 2$ *in the secondary graph. This solution satisfies flow conservation, demand constraints, and has at least one unit of flow leaving secondary node* 0*. However, it cannot be mapped into a valid 2-BCPP solution as its associated solution graph* G_f^2 *does not contain an Eulerian cycle. By using the constraint generation framework, one could prevent such a solution from being generated again by adding subtour elimination constraint* $f^2_{2,2,lk} \leq M \cdot f^2_{2,4,ls}$ *to* (29)*.* □

5 Computational experiments

We empirically evaluated the performance of each of the proposed approaches, namely:

- **CSP**, the arcflow model (8)-(11) described in Section 3.1 to obtain a lower bound;
- **EUL-REL**, the Eulerian-flow model (17)-(22) without subtour elimination constraints (21) described in Section 4.1 to obtain a lower bound;
- **LINK-REL**, the link-flow model (23)-(31) without subtour elimination constraints (29) described in Section 4.3 to obtain a lower bound;
- **2DV**, the arcflow model (12)-(15) described in Section 3.2 to obtain an upper bound;
- **ERZIN**, model (1)-(7) described in Section 2 introduced by Erzin et al. (2021b);

- **EUL-FLOW**, the Eulerian-flow model (17)-(22) where subtour elimination constraints are added through the constraint generation framework described in Section 4.2;
- **LINK-FLOW**, the link-flow model (23)-(31) where subtour elimination constraints are added through the constraint generation framework described in Section 4.2.

Our algorithms were all coded in `C++` and can be downloaded from `https://github.com/mdelorme2/Two_Bar_Chart_Problem_Codes`. The experiments were run on an Intel(R) Core(TM) i7-4700MQ, 2.40GHz with 8GB of memory, running under Ubuntu 20.04.4 LTS, and Gurobi 9.5.0 was used to solve the ILP models. A single core was used for the tests (i.e., parameter `Threads` was set to 1), the barrier algorithm was used to solve the root nodes of the ILP models (i.e., parameter `Method` was set to 2), and callbacks were used for the constraint generation (and thus, parameter `LazyConstraints` was set to 1). In every instance, an upper bound K on the minimum number of bins required to pack the bar charts was obtained with a trivial first-fit decreasing approach. For every run, a time limit of 3600 seconds was imposed. We tested our approaches on six newly generated data sets with integer item weights and bin capacities, namely:

- **U-GEN**, where bar charts contain (a combination of) small, medium, and big items;
- **U-SMA**, where bar charts only contain small items;
- **U-MED**, where bar charts contain at least one medium item;
- **U-BIG**, where bar charts contain at least one big item;
- **TRIPLETS**, where an integer solution in which every bin is entirely filled exists;
- **DONUTS**, where the bound derived from EUL-REL and LINK-REL is exactly one bin away from the optimal solution value.

Our data sets can be downloaded from `https://github.com/mdelorme2/Two_Bar_Chart_Problem_Instances`. For each data set, we first describe the instance generation process and its motivation. We then empirically evaluate the performance of each of the aforementioned approaches, including an overview of the model size and quality of the LP-relaxation.

5.1 Data set generation and experiments on U-GEN, U-SMA, U-MED, and U-BIG

5.1.1 Data set generation.

In data sets U-GEN and U-SMA, the item weights of every bar chart are uniformly distributed in specific ranges: $[1; c]$ in U-GEN and $[1; \frac{c}{10}]$ in U-SMA. In data sets U-MED and U-BIG, the weight of one item of every bar chart (randomly selected between the first and the second item) is uniformly distributed in $[1; c]$ while the weight of the other item is uniformly distributed in $[\frac{c}{4}; 1]$ in U-MED and $[\frac{c}{2}; 1]$ in U-BIG. For each data set, we considered three distinct bin capacities $c \in \{50, 100, 500\}$ and various numbers of bar charts $\sum_{i=1}^{n} q_i$ (named Ω thereafter).

- For $c = 50$, we tried $\Omega \in \{10, 50, 100, 500, 1000, 5000, 10\,000, 50\,000, 100\,000\}$;
- For $c = 100$, we tried $\Omega \in \{10, 50, 100, 1000, 10\,000\}$;
- For $c = 500$, we tried $\Omega \in \{10, 50, 100\}$.

For each of the 17 combinations, we generated 10 instances resulting in 170 instances in total per data set. To create an instance, we simply generated the desired Ω bar charts according to the specified distribution and we merged the identical ones to obtain values n and q_i $(i = 1, \ldots, n)$.

5.1.2 Data set motivation.

Data sets similar to U-GEN and U-BIG were used in the experiments of Erzin et al. (2021a) on a standardized version of the 2-BCPP where the bin capacity was always equal to 1 and the item weights were real numbers with up to eight digits after the decimal point. We decided to recreate these two data sets in order to have more flexibility over the capacity and the total number of bar charts. We also mention that there were some inconsistencies in the instance files we downloaded from the web link shared in Erzin et al. (2021a)[1], which is another motivation for generating new data sets. As far as U-MED and U-SMA are concerned, the former was created to be a more difficult version of U-BIG while the latter was designed to outline the main weaknesses of our new flow formulation approaches.

5.1.3 Experiments.

Our first set of experiments aims at empirically evaluating three different rules for ordering the bar charts while creating the graphs in Eulerian-flow and link-flow. Rule "FID" sorts the bar charts according to the first item weight (the second item weight is used to break the ties), rule "SID" sorts the bar charts according to the second item weight (the first item weight is used to break the ties), and "MID" sorts the bar charts according to the sum of the first and second item weights (ties are broken randomly). In all three cases, we sorted the values in non-increasing order as it is the most common choice in arcflow formulations to reduce the model size (see Valério de Carvalho 1999). We report in Table 1 the results obtained by EUL-FLOW and LINK-FLOW with each of the three ordering rules. The first two columns identify the approach and the ordering rule used. The three following columns give some indicators of the performance of each method: the number of optimal solutions found, average CPU time over all runs (including the ones terminated by the time limit), and the average number of cuts added in the constraint generation part. Note that instances terminated because of memory issues were counted as unsolved within the time limit. The three last columns report some details about the model size: average number of variables, constraints, and non-zero elements).

From the table, we observe that shorter average computing times are obtained when using ordering rule MID. This can be explained by the fact that the models generated with MID are slightly smaller compared to those generated with FID and SID. We point out that even if the research community tends to agree that there is a strong correlation between smaller models and

[1]Indeed, link `https://disk.yandex.ru/d/U-EK0n4Z_HEAeQ` states that the solution found by CPLEX for the first instance of "class $N = 10$" is 10 while all our models indicate that the optimal solution is 12.

Table 1: Comparison of the ordering rules for Eulerian-flow and link-flow on U-GEN instances

Approach	Ordering rule	Values			Model size		
		#opt	time	number of cuts	number of variables	number of constraints	number of non-zeros
EUL-FLOW	FID	162	405.6	1.6	339 657	12 752	1 007 726
	SID	164	425.1	1.9	347 832	12 752	1 032 252
	MID	163	390.7	1.8	336 686	12 752	998 812
LINK-FLOW	FID	170	102.4	0.1	239 274	3967	721 545
	SID	169	123.1	0.1	240 376	3967	724 852
	MID	170	99.9	0.1	226 136	3967	682 131

better computing performance (all other things being equal), there are some random components within ILP solvers that could explain why EUL-FLOW solves more instances to optimality with SID than it does with MID. Nevertheless, as the model size is the main drawback of arcflow formulations, we decided to use ordering rule MID in the rest of our experiments.

We show in Table 2 the results obtained by each of the tested approaches on the 17 (c, Ω) combinations for data set U-GEN. The first two columns identify the combination while the third column reports the average optimal solution value computed over the 10 instances of the combination. The remaining columns give some indicators of the performance of each method. For the bounding procedures (i.e., CSP, EUL-REL, LINK-REL, and 2DV), column "diff" indicates the average difference between the bound obtained by the approach and the optimal solution value. Columns "#opt", "time", and "#cuts" report the number of optimal solutions found, the average CPU time over all runs (including the ones terminated by the time limit), and the average number of cuts added in the constraint generation part, respectively. For EUL-REL and LINK-REL, an instance was considered to be solved to optimality if the solution found by the solver was optimal and contained no subtours. Finally, column "abs gap" shows the average absolute gap obtained by the exact approaches (i.e., the average difference between the upper bound and the lower bound found by the solver). When an approach obtained particularly bad results for intermediate (c, Ω) combinations (e.g., ERZIN for combination (50,100)), we decided to not try larger combinations and filled the associated rows with sign "-". When Gurobi was not able to compute the continuous relaxation before the time limit (e.g., EUR-REL for combination (500,100)), near-zero lower bounds values were reported by the solver. In order to avoid abnormally large and hardly interpretable average absolute gaps in the table, we replaced those by "NI" (standing for "not interpretable"). The results in the table show that:

- The lower bound derived from CSP is very good, especially when the total number of bar charts increases. We mention that the average time required to solve CSP was below 5 seconds for every combination;
- The lower bounds derived from EUL-REL and LINK-REL are almost always equal to the optimal solution value, but the associated models take a long time to solve (sometimes even longer than their counterpart EUL-FLOW and LINK-FLOW). The solutions found by both approaches are often free of subtours, especially for large Ω values. We also

Table 2: Results of the tested approaches for U-GEN instances

c	Ω	avg. opt. value	CSP	EUL-REL			LINK-REL			ERZIN			EUL-FLOW				LINK-FLOW				2DV
			diff	# opt	time	diff	# opt	time	diff	# opt	time	abs gap	# opt	time	abs gap	# cuts	# opt	time	abs gap	# cuts	diff
50	10	11.1	0.2	2	0	0	8	0	0	10	0	11.1	10	0	0	3.7	10	0	0	0.2	3
	50	53.5	0.6	9	0	0	10	0	0	1	3244	51.6	10	1	0	1.6	10	0	0	0.1	8
	100	107.3	0.4	8	2	0	10	1	0	0	3600	103.5	10	2	0	0.4	10	1	0	0	16
	500	518.6	0	10	33	0	10	10	0	-	-	-	10	36	0	0	10	11	0	0	46
	1000	1038.2	0	10	88	0	10	21	0	-	-	-	10	88	0	0	10	22	0	0	77
	5000	5127.0	0	10	426	0	10	234	0	-	-	-	10	287	0	0	10	243	0	0	217
	10 000	10 235.1	0	10	452	0	10	240	0	-	-	-	10	421	0	0	10	263	0	0	367
	50 000	51 063.9	0	10	577	0	10	240	0	-	-	-	10	314	0	0	10	373	0	0	1378
	100 000	102 104.9	0	10	344	0	10	184	0	-	-	-	10	294	0	0	10	164	0	0	2473
100	10	11.6	0.2	5	0	0	9	0	0	10	0	11.6	10	0	0	3.8	10	0	0	0	3
	50	53.7	0.3	2	4	0	9	0	0	0	3600	51.7	10	5	0	1.5	10	0	0	0	12
	100	105.6	0.3	4	13	0	7	1	0	1	3281	102.8	10	21	0	2.2	10	1	0	0.1	15
	500	510.8	0.1	10	535	0	10	128	0	-	-	-	10	493	0	0	10	117	0	0	44
	1000	1018.2	0	10	1963	0	10	601	0	-	-	-	10	2100	0	0	10	502	0	0	66
500	10	12.4	0.6	4	0	0.1	9	0	0.1	10	0	0	10	0	0	3	10	0	0	0.3	3
	50	53.3	0	1	13	0	8	0	0	1	3241	3	10	13	0	9.3	10	0	0	0.3	9
	100	104.2	0.5	0	2744	NI	10	3	0	0	3600	6.2	3	2823	NI	1.9	10	2	0	0.1	17

note that LINK-REL is faster than EUL-REL on average and solves more instances to optimality;

- Exact approach ERZIN is not able to solve instances with as few as 50 bar charts, confirming the results reported by Erzin et al. (2021b). Further analysis (see Table 8 in the appendix) shows that such bad performance can be attributed to the model size (as K increases with Ω) and to the poor quality of the LP-relaxation bound (as large bar charts can conveniently be broken in an LP solution to reduce the required number of bins);

- Exact approach EUL-FLOW is particularly effective as it can solve all but 7 instances in the data set. LINK-FLOW is even better as it requires less computing time on average and can solve every instance. Further analysis (see Table 8 in the appendix) shows that these better results mostly come from smaller model sizes (e.g., EUL-FLOW required 1 022 159 variables on average for instances with 1000 bar charts and capacity 100 versus 515 538 for LINK-FLOW). We also note that more cuts are required when there are less bar charts for both approaches. This makes sense as subtours are more likely to happen when fewer arcs are selected in the solution. It is also interesting to notice that LINK-FLOW requires less cuts on average than EUL-FLOW. This also makes sense as subtours are more likely to happen in the graph of EUL-FLOW than in the secondary graph of LINK-FLOW because the former has up to $(c+1)^2$ nodes while the latter has $c+1$ nodes at most;

- Besides being relatively slow to compute (up to 100s on average), the upper bounds derived from 2DV are never close to the optimal solution values.

We show in Table 3 the results obtained by each of the tested approaches on data set U-SMA and report the most noticeable changes with respect to U-GEN thereafter. We observe

Table 3: Results of the tested approaches for U-SMA instances

c	Ω	avg. opt. value	CSP	EUL-REL			LINK-REL			ERZIN			EUL-FLOW				LINK-FLOW				2DV
			diff	# opt	time	diff	# opt	time	diff	# opt	time	abs gap	# opt	time	abs gap	# cuts	# opt	time	abs gap	# cuts	diff
	10	2.0	0.2	10	0	0	10	0	0	10	0	0	10	0	0	0.6	10	0	0	0.1	0
	50	6.4	0	0	14	0	0	13	0	10	0	0	10	959	0	183.0	10	508	0	42.0	1
	100	12.2	0	2	13	0	3	12	0	10	0	0	10	224	0	61.0	10	154	0	9.6	1
	500	61.1	0	3	11	0	3	12	0	10	1	0	10	41	0	11.9	10	19	0	2.4	2
50	1000	120.7	0	2	13	0	7	17	0	10	9	0	10	69	0	11.7	10	58	0	2.3	2
	5000	601.1	0	2	32	0	9	6	0	10	500	0	10	64	0	8.7	10	7	0	0.4	2
	10 000	1199.7	0	2	52	0	9	5	0	6	2695	0.7	10	30	0	8.3	10	5	0	0.1	3
	50 000	5999.2	0	2	19	0	9	9	0	0	3600	91.9	10	55	0	9.1	10	70	0	0	7
	100 000	11 995.3	0	4	104	0	5	20	0	0	3600	227.0	10	215	0	8.3	10	54	0	0.7	6
	10	2.0	0.3	10	1	0	10	0	0	10	0	0	10	1	0	0.4	10	0	0	0	0
	50	6.0	0	0	2898	0	0	1895	0	10	0	0	1	3462	2.9	15.0	1	3455	2.5	13.5	0
100	100	11.4	0	2	3474	0	0	2869	0	10	0	0	0	3600	3.4	4.3	0	3600	5.4	7.9	1
	500	55.3	0	7	1605	0	0	3600	0	10	8	0	4	2401	1.1	2.0	0	3600	2	4.3	3
	1000	110.3	0	7	2606	0	0	3600	0	10	74	0	7	2728	0.3	0.4	0	3600	2.8	0.9	5
	10	2.0	0.5	10	44	0	10	0	0	10	0	0	10	46	0	0.9	10	0	0	0	0
500	50	5.6	0	0	3600	NI	0	3600	NI	10	0	0	0	3600	NI	0	0	3600	NI	0	NI
	100	10.7	0	0	3600	NI	0	3600	NI	10	0	0	0	3600	NI	0	0	3600	NI	0	NI

that ERZIN is now able to solve instances with up to 5000 bar charts and is barely impacted by the bin capacity. Further analysis (see Table 9 in the appendix) shows that this change in performance can be attributed to the decrease in the model size (as bar charts are smaller, K now increases at a lower pace with Ω and the number of distinct bar charts n decreases) and to the increase in the quality of the LP-relaxation bound (as breaking small bar charts is less likely to reduce the required number of bins). We also notice that the flow formulations now struggle to solve instances with more than 50 bar charts when the bin capacity is above 100. Further analysis (see Table 9 in the appendix) shows that such decrease in performance can be attributed to the increase in the model size driven by smaller bar charts. We also observe that for $c = 100$, it is easier for EUL-FLOW to solve instances with 500 or 1000 bar charts than it is to solve instances with 50 or 100 bar charts. This can be explained by the fact that integer solutions found for instances containing many bar charts are less likely to contain subtours (all other things being equal). For example, 15 cuts on average were added by EUL-FLOW for instances with 50 bar charts while only 0.4 cuts on average were added for instances with 1000 bar charts. It is also interesting to see that EUL-REL now solves more instances than EUL-FLOW. This could be explained by the fact that certain Gurobi-specific reductions and transformations are incompatible with lazy constraints and are therefore activated in EUL-REL but not in EUL-FLOW. Another observation is that EUL-FLOW now solves more instances than LINK-FLOW. A possible explanation could be that, even though EUL-FLOW requires more variables on average than LINK-FLOW (see Table 9 in the appendix), its structure makes

it easier for Gurobi heuristics to find good quality solutions. Finally, it is worth noticing that the upper bounds derived from 2DV are now closer to the optimal solution values, but they still take a significant amount of time to compute (e.g., more than 1200s on average for combination (100,100)).

Finally, we report in Table 4 the results obtained by each of the tested approaches on data sets U-MED and U-BIG. We observe that the comments made for U-GEN are also valid for U-

Table 4: Results of the tested approaches for U-MED (top) and U-BIG (bottom) instances

c	Ω	avg. opt. value	CSP	EUL-REL			LINK-REL			ERZIN			EUL-FLOW				LINK-FLOW				2DV
			diff	# opt	time	diff	# opt	time	diff	# opt	time	abs gap	# opt	time	abs gap	# cuts	# opt	time	abs gap	# cuts	diff
	10	13.4	0.3	6	0	0	9	0	0	10	0	0	10	0	0	2.0	10	0	0	0.1	4
	50	61.7	0.6	5	0	0	8	0	0	0	3600	4.3	10	0	0	3.0	10	0	0	0.1	14
	100	120.7	0.1	5	1	0	10	0	0	0	3600	9.3	10	0	0	1.3	10	0	0	0	19
	500	605.5	0.1	10	7	0	10	2	0	0	3600	73.7	10	5	0	0	10	2	0	0	76
50	1000	1201.0	0	10	24	0	10	6	0	0	3600	184.0	10	23	0	0	10	5	0	0	144
	5000	5928.2	0	10	125	0	10	44	0	-	-	-	10	104	0	0	10	38	0	0	605
	10 000	11 841.4	0	10	119	0	10	95	0	-	-	-	10	70	0	0	10	78	0	0	1123
	50 000	59 094.4	0	10	149	0	10	209	0	-	-	-	10	81	0	0	10	475	0	0	5200
	100 000	118 176.5	0	10	100	0	10	58	0	-	-	-	10	74	0	0	10	49	0	0	10 221
	10	13.4	0.5	6	0	0	10	0	0	10	0	0	10	0	0	3.7	10	0	0	0	2
	50	59.7	0.5	2	0	0	9	0	0	0	3600	4.4	10	0	0	6.0	10	0	0	0.1	12
100	100	118.5	0.1	5	1	0	10	0	0	0	3600	9.2	10	2	0	3.5	10	0	0	0	19
	500	595.2	0.2	9	88	0	10	8	0	0	3600	69.5	10	78	0	0.1	10	6	0	0	77
	1000	1190.4	0	10	357	0	10	41	0	0	3600	122.3	10	290	0	0	10	36	0	0	140
	10	13.0	0.7	2	0	0.3	7	0	0.3	10	0	0	10	0	0	2.0	10	0	0	0.3	3
500	50	61.8	0.6	3	0	0	10	0	0	0	3600	3.8	10	0	0	0	10	0	0	0.1	13
	100	124.0	0.4	6	2	0	9	0	0	0	3600	11.5	10	2	0	0	10	0	0	0	21
	10	15.7	0.1	7	0	0	10	0	0	10	0	0	10	0	0	1.6	10	0	0	0	3
	50	76.2	0.1	3	0	0	10	0	0	8	722	0.2	10	0	0	4.4	10	0	0	0	11
	100	151.6	0.5	6	0	0	10	0	0	3	2652	15.9	10	0	0	0.8	10	0	0	0	17
	500	758.4	1.6	10	1	0	10	0	0	0	3600	126.9	10	1	0	0	10	0	0	0	54
50	1000	1514.1	1.2	10	1	0	10	0	0	0	3600	307.5	10	1	0	0	10	0	0	0	90
	5000	7544.3	4.2	10	6	0	10	4	0	-	-	-	10	5	0	0	10	3	0	0	266
	10 000	15 135.5	6.8	10	7	0	10	5	0	-	-	-	10	6	0	0	10	5	0	0	489
	50 000	75 520.5	12.2	10	6	0	10	5	0	-	-	-	10	5	0	0	10	4	0	0	1880
	100 000	150 971.3	17.8	10	5	0	10	5	0	-	-	-	10	5	0	0	10	5	0	0	3534
	10	14.9	0.1	6	0	0	10	0	0	10	0	0	10	0	0	1.5	10	0	0	0	4
	50	78.1	0.6	3	0	0	10	0	0	5	1801	0.7	10	0	0	5.9	10	0	0	0	10
100	100	146.5	0.2	3	0	0	10	0	0	6	1656	8.1	10	0	0	4.4	10	0	0	0	21
	500	754.2	0.9	9	2	0	10	0	0	0	3600	131.4	10	2	0	0	10	0	0	0	51
	1000	1510.9	1.4	10	6	0	10	1	0	0	3600	288.8	10	5	0	0	10	1	0	0	75
	10	15.7	0.2	4	0	0.1	9	0	0.1	10	0	0	10	0	0	1.7	10	0	0	0.1	3
500	50	75.3	0.3	2	0	0	10	0	0	8	723	0.3	10	0	0	0	10	0	0	0	10
	100	152.3	0.6	0	0	0	10	0	0	5	2015	10.4	10	0	0	0	10	0	0	0	16

MED and U-BIG: (i) the lower bounds derived from CSP are very good for U-MED (less than one bin away from the optimal solution on average) and slightly less good for U-BIG (between 0 and 35 bins away from the optimal solution), (ii) the lower bounds derived from EUL-REL and

LINK-REL are almost always equal to the optimal solution value and are often free of subtours but they can be long to compute, (iii) ERZIN struggles to solve instances with more than 50 bar charts because of its model size and the poor quality of its LP-relaxation bound (see Table 10 in the appendix), (iv) EUL-FLOW and LINK-FLOW can solve every tested instance, with LINK-FLOW being slightly faster on average than EUL-FLOW due to its reduced model size, and (v) the upper bounds derived from 2DV are not competitive.

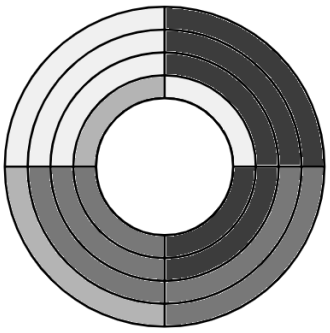
5.2 Data set generation and experiments on TRIPLETS and DONUTS

5.2.1 Data set generation.

In data set TRIPLETS, the item weights of every bar chart are distributed in the range $[\frac{c}{4}; \frac{c}{2}]$ in such a way that every bin contains exactly three items in an optimal solution, except the first and last bins that contain two items instead. To create a TRIPLETS instance with optimal value $\bar{z}$ (where $\bar{z}$ is an even number), we loop over $j = 1, \ldots, \bar{z} - 1$ and we generate the bar charts that are packed in bin j in an optimal solution. Throughout the process, we update the remaining capacity of bins j and $j+1$ (denoted by κ_j and κ_{j+1}) to be equal to the bin capacity minus the sum of the item weights already assigned to bins j and $j+1$. If j is odd, we create two bar charts. For the first bar chart, the first item is in the range $[\frac{c}{4}; \kappa_j - \frac{c}{4}]$ and the second item is in the range $[\frac{c}{4}; \frac{c}{2}]$. For the second bar chart, the first item is equal to κ_j and the second item is in the range $[\frac{c}{4}; \kappa_{j+1} - \frac{c}{4}]$. If j is even, we create one bar chart. Its first item is equal to κ_j and its second item is in the range $[\frac{c}{4}; \frac{c}{2}]$. For the special case where $j = 1$, the first items of the two generated bar charts have weight $\frac{c}{2}$. For the special case where $j = \bar{z} - 1$, the second items of the two generated bar charts have weight $\frac{c}{2}$.

DONUTS instances are composed of two subsets of bar charts. The first subset S_1 requires $\bar{z_1}$ bins (an even number) to be filled and is generated as a TRIPLETS instance. The second subset S_2 requires $\bar{z_2}$ bins (an even number) to be filled if subtours are allowed and $\bar{z_2}+1$ bins otherwise, and is called a *donut*. We show in Figure 8 an illustrative example of a donut composed of $n = 4$ bar charts, with $q_1 = q_2 = q_3 = q_4 = 1$, $w_1 = (3, 2), w_2 = (2, 3), w_3 = (1, 1), w_4 = (3, 1)$ and $c = 4$. We observe that the optimal 2-BCPP solution for this instance requires 5 bins but that a solution using 4 bins can be reached if it is allowed to pack the first item of bar chart 4 in bin 4 (top left in the figure) and the second item in bin 1 (top right in the figure), or in other words, if Eulerian-flow subtour $(1,0) \xrightarrow{1} (4,2) \to (2,0) \xrightarrow{2} (4,3) \to (3,0) \xrightarrow{3} (4,1) \to (1,0) \xrightarrow{4} (4,1) \to (1,0)$ is not forbidden. In data set DONUTS, subset S_2 is generated as a TRIPLETS instance except for $j = 1$ and $j = \bar{z_2} - 1$. For $j = 1$, the first item of the first bar chart is in the range $[\frac{c}{4}; \frac{c}{2}]$ while the first item of the second bar chart is in the range $[\frac{c}{4}; \kappa_1 - \frac{c}{4}]$. For $j = \bar{z_2} - 1$, the second item of the first bar chart is in the range $[\frac{c}{4}; \frac{c}{2}]$ while the second item of the second bar chart is in the range $[\frac{c}{4}; \kappa_{\bar{z_2}} - \frac{c}{4}]$. An additional bar chart with weights $(\kappa_{\bar{z_2}}, \kappa_1)$ is added to the subset to complete the donut. Note that in order to make sure that a DONUTS instance requires an extra bin when subtours are forbidden, we need to generate the bar charts in S_2 so that they cannot be mixed with the bar charts in S_1 to build a feasible 2-BCPP solution using $\bar{z_1} + \bar{z_2}$ bins. We also point out that subset S_2 alone is not a valid DONUTS instance because the solution using $\bar{z_2}$ bins does not contain node (0,0) (resp. secondary node 0), while having a flow going out of (0,0) (resp. 0) is a constraint in EUL-FLOW (resp. LINK-FLOW).

Figure 8: Donut with $n = 4, w_1 = (3, 2), w_2 = (2, 3), w_3 = (1, 1), w_4 = (3, 1)$, $q_1 = q_2 = q_3 = q_4 = 1$, and $c = 4$



For data set TRIPLETS, we considered three bin capacities $c \in \{80, 240, 400\}$ and four values for the number of bins $\bar{z} \in \{20, 50, 100, 250\}$ (which corresponds to $\Omega \in \{29, 74, 149, 374\}$ bar charts, respectively). For each of the 12 combinations, we generated 10 instances resulting in 120 instances in total. For data set DONUTS, we considered the same three bin capacities $c \in \{80, 240, 400\}$ and four values for the number of bins in the first subset $\bar{z}_1 \in \{20, 50, 100, 250\}$. We considered two options for $\bar{z}_2$ (the number of bins in the second subset): 2 and $0.2 \cdot \bar{z}_1$. We also evaluated the impact of adding three donuts per instance instead of one (i.e., generating three distinct subsets S_2). For each of the $3 \times 4 \times 2 \times 2 = 48$ combinations, we generated 10 instances resulting in 480 instances in total.

5.2.2 Data set motivation.

Data set TRIPLETS originates from the well-known "triplets" data set introduced by Falkenauer (1996) for the BPP. These instance were shown to be difficult to solve in practice because there are very few optimal solutions, meaning that a lot of exploration is needed in order to find one of these optimal solutions. Data set DONUTS originates from the "ANI" instances introduced by Delorme et al. (2016) for the BPP. These instance were shown to be difficult to solve in practice because the optimal solution is one bin away from the lower bound derived from the LP-relaxation of the existing models, meaning that a lot of exploration is needed in order to prove that a solution with value equal to the lower bound does not exist.

5.2.3 Experiments.

We display in Table 5 the results obtained by each of the tested approaches on data set TRIPLETS.

We observe that the lower bounds derived from CSP, EUL-REL, and LINK-REL are always equal to the optimal solution value, which does not come as a surprise since every TRIPLETS instance has an optimal solution that does not contain any loss space. ERZIN is only able to solve instances with $\bar{z} = 20$ to optimality (i.e, with 29 bar charts). Further analysis available in Table 11 in the appendix shows that such bad performance can neither be attributed to

Table 5: Results of the tested approaches for TRIPLETS instances

c	$\bar{z}$	avg. opt. value	CSP	EUL-REL			LINK-REL			ERZIN			EUL-FLOW				LINK-FLOW				2DV
			diff	# opt	time	diff	# opt	time	diff	# opt	time	abs gap	# opt	time	abs gap	# cuts	# opt	time	abs gap	# cuts	diff
80	20	20	0	9	0	0	6	0	0	8	1804	0.2	10	0	0	2.3	10	0	0	0.9	3
	50	50	0	4	3	0	4	2	0	0	3600	2.0	9	366	0.1	382.6	10	3	0	3.6	4
	100	100	0	5	4	0	5	4	0	0	3600	4.0	10	13	0	7.8	10	12	0	3.3	8
	250	250	0	6	15	0	7	73	0	0	3600	9.4	10	13	0	3.2	10	77	0	0.6	19
240	20	20	0	10	0	0	10	0	0	10	235	0	10	0	0	0.9	10	0	0	0	3
	50	50	0	2	58	0	0	52	0	0	3600	2.0	10	102	0	6.4	10	240	0	7.7	5
	100	100	0	5	152	0	4	289	0	0	3600	3.8	9	516	0.1	101.9	8	1019	0.2	6.4	8
	250	250	0	5	731	0	3	734	0	0	3600	10.7	9	1775	0.1	10.6	9	1262	0.2	3.8	18
400	20	20	0	10	0	0	10	0	0	9	383	0.1	10	0	0	0.3	10	0	0	0.2	3
	50	50	0	9	21	0	10	15	0	0	3600	2.1	10	31	0	7.8	10	13	0	0.2	5
	100	100	0	0	3300	0	0	3600	0	0	3600	3.9	1	3475	0.9	4.7	0	3600	1.0	0.9	8
	250	250	0	1	3583	0	0	3600	0	0	3600	11.2	0	3600	1.3	2.8	1	3561	0.9	3.0	20

the model size (which is comparable to the size of EUL-FLOW and LINK-FLOW), nor to the quality of the LP-relaxation bound (which is always equal to the optimal solution value). A possible explanation could be that the solutions obtained after solving a node in the branch-and-bound tree are rarely integer and that these solutions do not allow Gurobi heuristics to find good quality upper bounds. EUL-FLOW and LINK-FLOW are able to solve most instances, except the largest ones (where $c = 400$ and $\bar{z} = 100$ or $\bar{z} = 250$). Neither approach is clearly better than the other. As previously mentioned, a reason could be that EUL-FLOW structure makes it easier for Gurobi heuristics to find good quality solutions even though LINK-FLOW models are smaller. Finally, we note that the number of cuts required by EUL-FLOW varies a lot (between 0 and 3702) compared to the number of cuts required by LINK-FLOW (between 0 and 38).

We show in Table 6 the results obtained by each of the tested approaches on data set DONUTS grouped by the bin capacity c and the number of bins in the first subset $\bar{z}_1$. The lower bounds derived from CSP, EUL-REL, and LINK-REL are now always one bin away from the optimal solution value, which is expected as DONUTS instances were made with that purpose. We note that ERZIN still displays poor performance compared to the other models. LINK-FLOW is able to solve all but 15 instances and is clearly better than EUL-FLOW. This is due to smaller models (see Table 12 in the appendix) and a reduced number of required cuts. Further analysis is reported in Table 7 where the results for the data set are grouped by number of donuts (column "# don") and number of bins per donut (column "$\bar{z}_2$").

We observe that instances with large donuts (where $\bar{z}_2 = 0.2 \cdot \bar{z}_1$) are much harder to solve than instances with small donuts (where $\bar{z}_2 = 2$). This can be explained by two reasons: (i) instances with large donuts contain more bar charts, and thus, result in larger models, and (ii) instances with large donuts require more cuts to be solved to optimality than instances with small donuts, especially for EUL-FLOW. We also notice that the number of donuts has a marginal effect as instances with three donuts require more time to be solved on average than

Table 6: Results of the tested approaches for DONUTS instances grouped by c and $\bar{z}_1$

c	$\bar{z}_1$	avg. opt. value	CSP	EUL-REL			LINK-REL			ERZIN			EUL-FLOW				LINK-FLOW				2DV
			diff	# opt	time	diff	# opt	time	diff	# opt	time	abs gap	# opt	time	abs gap	# cuts	# opt	time	abs gap	# cuts	diff
80	20	27	1	0	0	1	0	0	1	6	3259	0.9	40	0	0	15.2	40	0	0	3.4	1
	50	63	1	0	0	1	0	0	1	0	3600	2.1	37	374	0.1	3210.4	40	0	0	3.3	4
	100	123	1	0	0	1	0	0	1	0	3600	3.7	40	24	0	1236.3	40	1	0	1.9	7
	250	303	1	0	0	1	0	0	1	0	3600	8.4	39	149	0	3409.8	40	1	0	1.7	18
240	20	20	1	0	0	1	0	0	1	0	3600	1.1	40	2	0	7.1	40	0	0	2.8	2
	50	50	1	0	4	1	0	3	1	0	3600	2.6	31	817	0.2	1812.9	40	7	0	24.7	5
	100	100	1	0	13	1	0	9	1	0	3600	5	30	911	0.2	1635.5	40	34	0	5.1	8
	250	250	1	0	52	1	0	41	1	0	3600	13.5	20	1810	0.6	2914.2	39	337	0	3.5	20
400	20	20	1	0	1	1	0	0	1	2	3520	1.1	40	2	0	6.3	40	0	0	2.1	2
	50	50	1	0	11	1	0	7	1	0	3600	2.6	39	125	0	16.1	40	11	0	6.2	4
	100	100	1	0	93	1	0	47	1	0	3600	5.2	30	968	0.2	936.3	36	522	0.1	78.9	9
	250	250	1	0	391	1	0	267	1	0	3600	17	20	1849	0.6	1227.6	30	1511	0.3	114.5	21

Table 7: Results of the tested approaches for DONUTS instances grouped by the number of donuts and $\bar{z}_2$

# don	$\bar{z}_2$	avg. opt. value	CSP	EUL-REL			LINK-REL			ERZIN			EUL-FLOW				LINK-FLOW				2DV
			diff	# opt	time	diff	# opt	time	diff	# opt	time	abs gap	# opt	time	abs gap	# cuts	# opt	time	abs gap	# cuts	diff
1	2	127	1	0	5	1	0	10	1	8	3460	3.4	120	13	0	2379.3	120	52	0	38.8	7
1	$0.2{\cdot}\bar{z}_1$	108	1	0	46	1	0	32	1	0	3600	5	99	670	0.2	2.8	111	314	0.1	2.4	9
3	2	169	1	0	9	1	0	12	1	0	3600	3.9	120	19	0	3088.9	120	61	0	38.1	7
3	$0.2{\cdot}\bar{z}_1$	112	1	0	129	1	0	69	1	0	3600	8.8	67	1641	0.4	4.9	114	382	0	3.4	10

instances with one donut.

6 Conclusions and future research

We studied the two-bar charts packing problem (2-BCPP), a combinatorial optimization problem introduced recently by Erzin et al. (2021b) that is particularly interesting as it lies at the intersection of many well-known packing problems. We started by clustering relevant packing problems into three groups (those that are 2-BCPP relaxations, those for which the 2-BCPP is a relaxation, and those for which the 2-BCPP is a special case) and showed how solving these problems could either provide a lower bound, an upper bound, or an optimal solution to the 2-BCPP. We then introduced two new ILP models based on the arcflow formulation where a 2-BCPP solution is represented as one or several cycles in a graph where nodes are partial bin fillings: Eulerian-flow and link-flow. The former model represents a solution as a single Eulerian cycle while the latter model represents a solution as a set of cycles that complete each other.

Even though both models require an exponential number of subtour elimination constraints, we showed that these constraints could initially be omitted and added on the fly within a constraint generation framework. We created six data sets to test our approaches: four inspired by the 2-BCPP literature in which the bar chart weights are uniformly distributed, and two inspired by the bin packing literature that were designed to be very hard to solve, either because the upper bound is difficult to reach, or because the lower bound is difficult to raise. Even though we empirically showed that our new algorithms clearly obtained better performance than the only exact approach proposed in the literature, we also identified instance features for which our methods did not perform very well due to a very large number of variables and constraints involved. Such features are (i) a large bin capacity and (ii) a large number of distinct small bar charts. Overall, we showed that our exact approaches could easily solve instances with 1000 bar charts when the bin capacity is limited to 100 and instances with 100 bar charts when the bin capacity is limited to 500, which is a huge improvement with respect to the literature. We also showed that the performance of the existing 2-BCPP model did not depend on the same instance features than our new models (K and n for the former, c and n for the latter).

Even though our exact approaches obtained state-of-the-art results for the 2-BCPP, we should keep in mind that the problem does not have many practical applications because of the constraint limiting the number of items in a bar chart to two. Nevertheless, it is the first time (to the best of our knowledge) that an arcflow model is proposed for a bin packing problem with a form of precedence constraints, which indicates that our approaches could be extended to solve other packing problems with precedence constraints. It is also the first time that an arcflow model is solved within a constraint generation framework. This means that such formulations could be used in decomposition approaches where the master problem is a packing problem (e.g., the Benders' decomposition approach of Côté et al. 2021 for the two-dimensional bin packing problem) instead of "Kantorovich-like" models which are known to have a very poor LP-relaxation value. As far as the generalization of our approaches for the n-BCPP is concerned, it would be interesting to see the computational behavior of Eulerian-flow and link-flow extensions for the 3-BCPP. We expect them to work well for small bin capacities (up to 100 if the instance is composed of few bar charts, or up to 50 otherwise). It would also be interesting to study similar extensions for the 4-BCPP and 5-BCPP. There, we expect that the full graph cannot be generated anymore and that alternative strategies are required. A first option could be to only add the arcs that are likely to be selected in an optimal solution (e.g, by using rules of thumb or machine learning techniques), which would result in a matheuristic. A second option could be to generate the arcs with column generation and embed the approach in a branch-and-price algorithm.

References

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

Cai Q, Hang W, Mirhoseini A, Tucker G, Wang J, Wei W (2019) Reinforcement learning driven heuristic optimization. *The 1st Workshop on Deep Reinforcement Learning for Knowledge Discovery (DRL4KDD '19)*.

Coffman E, Csirik J, Galambos G, Martello S, Vigo D (2013) Bin packing approximation algorithms: Survey and classification. Pardalos P, Du DZ, Graham R, eds., *Handbook of Combinatorial Optimization* (Springer New York).

Côté JF, Dell'Amico M, Iori M (2014) Combinatorial Benders' cuts for the strip packing problem. *Operations Research* 62:643–661.

Côté JF, Haouari M, Iori M (2021) Combinatorial benders decomposition for the two-dimensional bin packing problem. *INFORMS Journal on Computing* 33:963–978.

Côté JF, Iori M (2018) The meet-in-the-middle principle for cutting and packing problems. *INFORMS Journal on Computing* 30:646–661.

de Lima V, Alves C, Clautiaux F, Iori M, Valério de Carvalho J (2022) Arc flow formulations based on dynamic programming: Theoretical foundations and applications. *European Journal of Operational Research* 296:3–21.

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

Dell'Amico M, Díaz J, Iori M (2012) The bin packing problem with precedence constraints. *Operations Research* 60:1491–1504.

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.

Delorme M, Iori M, Martello S (2017) Logic based benders' decomposition for orthogonal stock cutting problems. *Computers & Operations Research* 78:290–298.

Delorme M, Iori M, Mendes N (2021) Solution methods for scheduling problems with sequence-dependent deterioration and maintenance events. *European Journal of Operational Research* 295:823–837.

Erzin A, Melidi G, Nazarenko S, Plotnikov R (2021a) A posteriori analysis of the algorithms for two-bar charts packing problem. *International Conference on Optimization and Applications*, 201–216.

Erzin A, Melidi G, Nazarenko S, Plotnikov R (2021b) Two-bar charts packing problem. *Optimization Letters* 15:1955–1971.

Erzin A, Nazarenko S, Melidi G, Plotnikov R (2021c) A 3/2-approximation for big two-bar charts packing. *Journal of Combinatorial Optimization* 42:71–84.

Falkenauer E (1996) A hybrid grouping genetic algorithm for bin packing. *J. of Heuristics* 2:5–30.

Kantorovich L (1960) Mathematical methods of organizing and planning production. *Management Science, English translation of a 1939 paper written in Russian* 6:366–422.

Letelier O, Clautiaux F, Sadykov R (2022) Bin packing problem with time lags. *INFORMS Journal on Computing* .

Lysgaard J, Letchford A, Eglese R (2004) A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming* 100:423–445.

Macedo R, Alves C, Valério De Carvalho J (2010) Arc-flow model for the two-dimensional guillotine cutting stock problem. *Computers & Operations Research* 37:991–1001.

Martello S, Monaci M, Vigo D (2003) An exact approach to the strip-packing problem. *INFORMS journal on Computing* 15:310–319.

Martello S, Toth P (1990) *Knapsack Problems: Algorithms and Computer Implementations* (Chichester: John Wiley & Sons).

Martinovic J, Scheithauer G (2016) Integer linear programming models for the skiving stock problem. *European Journal of Operational Research* 251:356–368.

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

Nesello V, Delorme M, Iori M, Subramanian A (2018) Mathematical models and decomposition algorithms for makespan minimization in plastic rolls production. *Journal of the operational research society* 69:326–339.

Pereira J (2016) Procedures for the bin packing problem with precedence constraints. *European Journal of Operational Research* 250:794–806.

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

Pferschy U, Staněk R (2017) Generating subtour elimination constraints for the tsp from pure integer solutions. *Central European journal of operations research* 25:231–260.

Sadykov R, Vanderbeck F (2013) Bin packing with conflicts: a generic branch-and-price algorithm. *INFORMS Journal on Computing* 25:244–255.

Shaw P (2004) A constraint for bin packing. *Principles and Practice of Constraint Programming – CP 2004*, volume 3258 of *Lecture Notes in Computer Science*, 648–662 (Springer Berlin Heidelberg).

Stadtler H (1990) A one-dimensional cutting stock problem in the aluminium industry and its solution. *European Journal of Operational Research* 44:209–223.

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

Vance P (1998) Branch-and-price algorithms for the one-dimensional cutting stock problem. *Computational Optimization and Applications* 9:211–228.

Wei L, Lai M, Lim A, Hu Q (2020a) A branch-and-price algorithm for the two-dimensional vector packing problem. *European Journal of Operational Research* 281:25–35.

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

Wolsey L (1977) Valid inequalities, covering problems and discrete dynamic programs. *Annals of Discrete Mathematics* 1:527–538.

A Pseudo-codes for graph construction

Algorithm 1 Graph construction for the 2D-VPP arcflow model

Input: ordered w_i, q_i $(i = 1, 2, \ldots, n)$, c
$V \leftarrow \{(0,0)\}$; $V' \leftarrow V$; $A \leftarrow \emptyset$
for $i = 1, 2, \ldots, n$ **do** ▷ for every bar chart type
  for all $(v_1, v_2) \in V$ **do** ▷ for every potential tail (or active vertex)
    $(d_1, d_2) \leftarrow (v_1, v_2)$ ▷ define the arc tail
    for $k = 1, 2, \ldots, q_i$ **do** ▷ for every unit of demand
      $(e_1, e_2) \leftarrow (d_1, d_2) + (w_{i1}, w_{i2})$ ▷ define the arc head
      if $e_1 \leq c$ and $e_2 \leq c$ **then** ▷ if the head is feasible
        $V' \leftarrow V' \cup \{(e_1, e_2)\}$; ▷ save the new head
        $A \leftarrow A \cup \{((d_1, d_2), (e_1, e_2))\}$ ▷ create the arc
        $(d_1, d_2) \leftarrow (e_1, e_2)$ ▷ the new tail is now the old head
  $V \leftarrow V'$ ▷ add every head added for bar chart i to the set of active vertices
Output: $G = (V, A)$

Algorithm 1 gives a pseudo-code for the graph construction procedure for the arcflow model for the 2D-VPP. This algorithm incorporates some of the symmetry reduction techniques introduced by Brandão and Pedroso (2016). The same graph is also used for the first steps of Eulerian-flow and link-flow.

Algorithm 2 Graph construction for Eulerian-flow

Input: ordered w_i, q_i $(i = 1, 2, \ldots, n)$, c
apply Algorithm 1 to obtain graph $G^\alpha = (V^\alpha, A^\alpha)$; $V^\gamma \leftarrow V^\alpha$; $A^\gamma \leftarrow A^\alpha$; $V^h \leftarrow \emptyset$
for all $(v_1, v_2) \in V^\alpha : v_2 \neq 0$ **do** ▷ for every vertex with non-zero second dimension
  $V^h \leftarrow V^h \cup \{v_2\}$; $V^\gamma \leftarrow V^\gamma \cup \{(v_2, 0)\}$ ▷ save distinct v_2 values
for all $v \in V^h$ **do** ▷ for every v_2 value
  for all $((d_1, d_2), (e_1, e_2)) \in A^\alpha$ **do** ▷ for every existing arc
    if $e_1 + v \leq c$ **then** ▷ if the head of the "shifted" arc is feasible
      $V^\gamma \leftarrow V^\gamma \cup \{(d_1 + v, d_2)\} \cup \{(e_1 + v, e_2)\}$ ▷ add the new tail/head in the set of vertices
      $A^\gamma \leftarrow A^\gamma \cup \{((d_1 + v, d_2), (e_1 + v, e_2))\}$ ▷ add the new arc in the set of arcs
for all $(v_1, v_2) \in V^\gamma$ **do** ▷ for every existing vertex
  $A^\gamma \leftarrow A^\gamma \cup \{((v_1, v_2), (v_2, 0))\}$ ▷ create a transition arc
Output: $G^\gamma = (V^\gamma, A^\gamma)$

Algorithm 2 gives a pseudo-code for the graph construction procedure for Eulerian-flow. Instead of creating a graph from scratch for each vertex in V^h, it "shifts" the first dimension of both the tail and the head of every arc contained in A^α by v for every $v \in V^h$. Note that appropriate data structures (such as sets) should be used for vertex and arc sets to avoid duplicates.

B Size and continuous relaxation values of the tested ILP models on data sets U-GEN, U-SMA, U-MED, and U-BIG

We report in Table 8 some supplementary information about the tested models on data set U-GEN. Column "# opt" indicates the number of times the continuous relaxation value of the model rounded up was equal to the optimal solution value. Column "diff" reports the difference between the optimal solution value and the continuous relaxation value of the model (which is itself reported in column "LP"). Columns "# var" and "# cons" indicate the average number of variables and constraints in the models.

Table 8: Size and continuous relaxation values of the tested models on data set U-GEN

c	Ω	ERZIN						EUL-REL						LINK-REL					
		# opt	diff	time	LP	# var	# cons	# opt	diff	time	LP	# var	# cons	# opt	diff	time	LP	# var	# cons
	10	3	1.4	0	9.7	133	33	10	0.3	0	10.8	421	226	10	0.3	0	10.8	94	77
	50	0	3.0	0	50.5	2990	167	10	0.4	0.2	53.1	8002	1752	10	0.4	0	53.1	1657	746
50	100	0	4.3	0.1	103.0	11 752	334	10	0.4	0.8	106.9	19 644	2239	10	0.4	0.1	106.9	6301	1521
	500	1	6.6	6.3	512.0	254 592	1572	10	0.3	6.8	518.3	129 492	2915	10	0.3	3	518.3	68 258	2773
	1000	0	14.7	38.4	1023.5	905 410	3025	10	0.4	13.9	1037.8	243 906	3320	10	0.4	6.7	1037.8	149 645	3263
	10	2	1.5	0	10.1	141	35	9	0.3	0	11.3	371	205	9	0.3	0	11.3	84	74
	50	0	3.1	0	50.6	3114	171	10	0.1	1.3	53.6	19 833	5273	10	0.1	0	53.6	2500	1205
100	100	0	4.0	0.1	101.6	11 751	332	10	0.2	7.9	105.4	57 676	7462	10	0.2	0.5	105.4	12 782	3734
	500	0	7.3	7.1	503.5	267 597	1581	10	0.4	82	510.4	468 857	10 120	10	0.4	33.2	510.4	208 137	9179
	1000	0	10.2	48.2	1008.0	1 026 480	3109	10	0.4	215.9	1017.8	1 022 159	10 754	10	0.4	79.8	1017.8	515 538	10 275

We gather in Tables 9 ad 10 some supplementary information about the tested models on data setx U-SMA, U-MED, and U-BIG.

Table 9: Size and continuous relaxation values of the tested models on data set U-SMA

c	Ω	ERZIN						EUL-REL						LINK-REL					
		# opt	diff	time	LP	# var	# cons	# opt	diff	time	LP	# var	# cons	# opt	diff	time	LP	# var	# cons
	10	10	0.7	0	1.3	17	11	10	0	0.1	2.0	4550	1170	10	0	0	2.0	659	295
	50	10	0.4	0	6.0	164	35	10	0.4	1.6	6.0	32 474	2152	10	0.4	0.9	6.0	20 421	1786
50	100	10	0.4	0	11.8	347	51	10	0.4	2	11.8	41 377	2298	10	0.4	1.2	11.8	29 738	2051
	500	10	0.6	0	60.5	1719	156	10	0.6	2.2	60.5	44 170	2352	10	0.6	1.2	60.5	33 626	2129
	1000	10	0.5	0	120.2	3463	290	10	0.5	2	120.2	44 170	2352	10	0.5	1.2	120.2	33 626	2129
	10	9	0.8	0	1.2	21	13	10	0	0.5	2.0	15 384	3482	10	0	0	2.0	1429	647
	50	10	0.5	0	5.5	271	52	10	0.5	33.7	5.5	232 575	8733	10	0.5	15.0	5.5	148 691	7365
100	100	10	0.5	0	10.9	780	87	10	0.5	68.2	10.9	428 992	9360	10	0.5	34.0	10.9	319 982	8531
	500	10	0.4	0.1	54.9	6044	219	10	0.4	128.8	54.9	722 866	9734	10	0.4	64.6	54.9	592 475	9266
	1000	10	0.3	0.2	110.0	12 120	339	10	0.3	135.2	110.0	734 934	9751	10	0.3	51.9	110.0	609 910	9301

Table 10: Size and continuous relaxation values of the tested models on data sets U-MED (top) and U-BIG(bottom)

c	Ω	ERZIN						EUL-REL						LINK-REL					
		# opt	diff	time	LP	# var	# cons	# opt	diff	time	LP	# var	# cons	# opt	diff	time	LP	# var	# cons
50	10	0	2.1	0	11.3	162	38	10	0.1	0	13.3	135	85	10	0.1	0	13.3	53	51
	50	0	4.7	0	57.0	3508	189	10	0.2	0	61.5	2787	970	10	0.1	0	61.6	491	308
	100	0	7.6	0.1	113.1	13 571	369	10	0.2	0.2	120.6	9559	1641	10	0.2	0	120.6	2017	807
	500	0	30.4	8.8	575.1	302 301	1796	10	0.3	1.8	605.2	61 092	2593	10	0.3	0.6	605.2	21 053	2095
	1000	0	53.2	50.9	1147.8	1 071 532	3470	10	0.4	4.7	1200.6	130 781	3088	10	0.4	2.1	1200.6	61 558	2790
100	10	0	2.3	0	11.1	158	38	10	0.2	0	13.2	174	107	10	0.2	0	13.2	64	60
	50	0	4.2	0	55.5	3462	185	10	0.3	0.1	59.4	6451	2498	10	0.3	0	59.4	724	456
	100	0	7.1	0.1	111.4	13 465	367	10	0.3	0.3	118.2	19 557	4510	10	0.3	0	118.2	2387	1213
	500	0	29.5	10.5	565.7	322 551	1810	10	0.4	20.3	594.8	204 405	8401	10	0.4	3.8	594.8	47 117	5428
	1000	0	53.8	80.2	1136.6	1 243 855	3579	10	0.3	46.9	1190.1	475 345	9531	10	0.3	13.9	1190.1	141 764	7392
50	10	0	3.1	0	12.6	183	42	10	0	0	15.7	80	57	10	0	0	15.7	41	43
	50	0	12.8	0.1	63.4	4161	214	10	0	0	76.2	1065	524	10	0	0	76.2	215	183
	100	0	25.1	0.2	126.5	15 923	421	10	0	0	151.6	3182	1040	10	0	0	151.6	503	359
	500	0	120.2	9.3	638.2	347 154	2031	10	0	0.3	758.4	20 596	2114	10	0	0	758.4	4030	1238
	1000	0	243.9	67.1	1270.2	1 201 931	3923	10	0	0.7	1514.1	41 526	2577	10	0	0.2	1514.1	11 405	1818
100	10	0	2.6	0	12.3	182	42	10	0	0	14.9	93	64	10	0	0	14.9	45	47
	50	0	13.8	0.1	64.3	4311	218	10	0	0	78.1	1299	691	10	0	0	78.1	234	209
	100	0	22.9	0.2	123.6	16 019	417	10	0	0	146.5	5456	2216	10	0	0	146.5	640	466
	500	0	122.6	12.0	631.6	381 932	2066	10	0	1.0	754.2	56 728	6089	10	0	0.1	754.2	5908	2393
	1000	0	248.4	88.9	1262.5	1 453 501	4062	10	0	3.1	1510.9	138 577	7574	10	0	0.4	1510.9	16 726	3621

C Size and continuous relaxation values of the tested ILP models on data sets TRIPLETS and DONUTS

We gather in Tables 11 and 12 some supplementary information about the tested models on data set TRIPLETS and DONUTS, respectively.

Table 11: Size and continuous relaxation values of the tested models on data set TRIPLETS

c	$\bar{z}$	ERZIN						EUL-REL						LINK-REL					
		# opt	diff	time	LP	# var	# cons	# opt	diff	time	LP	# var	# cons	# opt	diff	time	LP	# var	# cons
80	20	10	0	0	20	727	77	10	0	0.1	20	5871	1690	10	0	0	20	1597	713
	50	10	0	0	50	3951	185	10	0	0.2	50	19 974	2438	10	0	0.1	50	7224	1427
	100	10	0	0.2	100	13 826	355	10	0	0.7	100	40 870	2731	10	0	0.4	100	15 569	1758
	250	10	0	1	250	62 973	809	10	0	1.7	250	81 472	2975	10	0	1.2	250	36 008	2075
240	20	10	0	0	20	736	77	10	0	0.1	20	12 044	5271	10	0	0	20	2161	1202
	50	10	0	0.1	50	4482	193	10	0	1.1	50	79 751	13 271	10	0	0.4	50	19 134	5366
	100	10	0	0.2	100	17 461	384	10	0	7.9	100	264 103	18 184	10	0	5.3	100	81 688	9204
	250	10	0	2	250	103 207	943	10	0	49.1	250	874 338	22 012	10	0	27.3	250	292 389	12 989
400	20	10	0	0	20	748	78	10	0	0.1	20	14 448	6834	10	0	0	20	2215	1310
	50	10	0	0.1	50	4559	195	10	0	2	50	114 947	27 394	10	0	0.3	50	24 055	9018
	100	10	0	0.2	100	17 925	388	10	0	17.3	100	475 069	40 289	10	0	11.3	100	120 840	17 747
	250	10	0	2.1	250	108 130	959	10	0	155.7	250	2 138 746	56 063	10	0	143.5	250	662 809	30 044

Table 12: Size and continuous relaxation values of the tested models on data set DONUTS

c	$\bar{z}$	ERZIN						EUL-REL						LINK-REL					
		# opt	diff	time	LP	# var	# cons	# opt	diff	time	LP	# var	# cons	# opt	diff	time	LP	# var	# cons
80	20	0	1	0	26	865	88	0	1	0	26	3309	978	0	1	0	26	1041	421
	50	0	1	0	62	2985	185	0	1	0.1	62	5308	1158	0	1	0	62	1797	535
	100	0	1	0.1	122	7235	335	0	1	0.1	122	6716	1251	0	1	0	122	2404	602
	250	0	1	0.3	302	20 850	767	0	1	0.1	302	7918	1306	0	1	0	302	3007	657
240	20	0	1	0	26	1162	97	0	1	0.1	26	12 723	4141	0	1	0	26	3018	1305
	50	0	1	0.1	62	5982	226	0	1	0.7	62	46 597	7095	0	1	0.3	62	12 313	2600
	100	0	1	0.2	122	19 164	417	0	1	1.9	122	91 755	8441	0	1	0.9	122	25 698	3399
	250	0	1	1.6	302	81 096	934	0	1	5	302	175 471	9449	0	1	2	302	53 536	4181
400	20	0	1	0	26	1248	100	0	1	0.2	26	19 897	7479	0	1	0	26	3977	1917
	50	0	1	0.1	62	6707	235	0	1	1.7	62	92 399	15 477	0	1	0.6	62	22 918	5107
	100	0	1	0.3	122	23 700	448	0	1	7.3	122	240 432	19 947	0	1	3.5	122	61 989	7356
	250	0	1	2.7	302	118 046	1034	0	1	31.1	302	618 861	23 808	0	1	11.8	302	169 696	9806