Solving the Skiving Stock Problem by a Combination of Stabilized Column Generation and the Reflect Arc-Flow Model

Laura Korbacher[*,a], Stefan Irnich[a], John Martinovic[b], Nico Strasdat[b]

[a]*Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Jakob-Welder-Weg 9, D-55128 Mainz, Germany.*
[b]*Institute of Numerical Mathematics, Technische Universität Dresden, Zellescher Weg 12-14, D-01069 Dresden, Germany.*

Abstract

The skiving stock problem represents a natural counterpart of the extensively studied cutting stock problem and requires the construction of as many large units as possible from a given set of small items. Although it owes its scientific beginnings to the better-known "original", in recent years it has been able to develop into an independent branch of research within the OR community. Due to the continuous development of modern optimization software, the focus was not only on concrete applications, such as in wireless communications, but also on the design of efficient integer models and their improvement by suitable reductions. This process has recently reached its preliminary peak with the introduction of a powerful graph-theoretic approach, the reflect arc-flow model. Even though this model significantly improved the state of the art, there are still many benchmark instances that even the current formulations cannot yet contribute to solving. We present a new approach that is based on the observation that solutions of very good quality can already be determined on rather sparse (arc-flow) graphs, in general. More precisely, the arc sets of these graphs are defined by appropriate patterns obtained from a stabilized column-generation approach, so that a consideration of the complete integer reflect arc-flow model is only necessary in a few cases. Compared to the reflect+ approach originally introduced for the cutting stock problem, we apply several modifications and show their numerical advantages by extensive computational tests. We succeed in optimally solving many very challenging benchmark instances for the first time.

Key words: Packing, Cutting, Skiving Stock Problem, Column Generation, Flow Formulation

1. Introduction

In this article, we study the one-dimensional *skiving stock problem* (SSP), one of the classic representatives in cutting and packing. We are given a set of *item types*, characterized by their *item sizes* and *frequency* of appearance (also called *availability*). The SSP requires the combination of these items to a maximum number of larger units each of which exhibiting at least a predefined *threshold* length. Even though the concrete notation will be introduced explicitly in the following section, we would like to refer to Figure 1 for the sake of illustration. Already this introductory description clearly reveals some parallels to the extensively studied one-dimensional *cutting stock problem* (CSP) – and, indeed, the scientific framing of the SSP cannot be done without having established a thorough overview of the CSP history first. In the cutting stock problem, a given demand of small items has to be satisfied by using as little stock material as possible. Before moving forward into a more detailed literature review, we point out that, although forming an obviously related pair of minimization and maximization problems and sharing the same kind of input data, the CSP and the SSP are not dual to each other from the perspective of mathematical optimization. Consequently, despite their obvious common features, the two problems are to be regarded as independent.

[*]Corresponding author
Email addresses: lkorbac@uni-mainz.de (Laura Korbacher), irnich@uni-mainz.de (Stefan Irnich), john.martinovic@tu-dresden.de (John Martinovic), nico.strasdat@tu-dresden.de (Nico Strasdat)

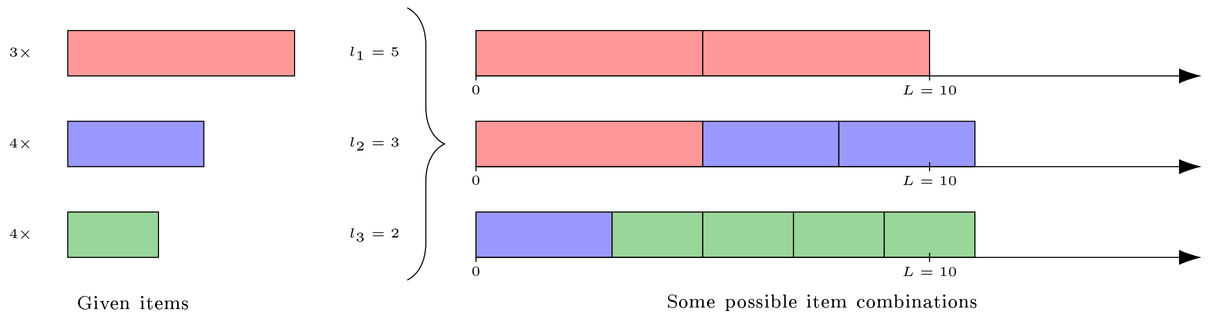


Figure 1: A simple instance of the skiving stock problem with three item types and threshold length $L = 10$.

1.1. Literature Review

The scientific history of the CSP dates back to 1939, when Kantorovich described this problem for the first time in a Russian publication and formulated an assignment model, which from today's point of view has two decisive disadvantages: (i) a highly symmetric solution space, and (ii) a poor *linear programming* (LP) relaxation. As a result of the turmoil of war and post-war time, Kantorovich's contributions did not achieve international visibility within the Operations Research community until his research results were translated into English in the early 1960s, see (Kantorovich, 1960). From this moment on, an increased interest in and an intensified discussion of the topics of cutting and packing optimization could be perceived in the academic world. In the first place, the articles (Gilmore and Gomory, 1961, 1963) have to be mentioned, through whose contributions the still today often favored pattern-based (or set-covering) view of such optimization problems came into focus. More precisely, the authors based their considerations on so-called *cutting patterns* (i.e., feasible combinations of items that can be cut from one piece of stock material) and were thus able to present a novel description of the CSP as an integer linear program. However, the only linear number of constraints was opposed by an exponential number of variables, and therefore also by a formulation that was difficult to handle at that time. While the integer problem could still not be addressed satisfactorily, the general approach chosen by Gilmore and Gomory had the decisive advantage that the continuous relaxation of the model led to very good approximations[1] for the actual optimal value. Moreover, the solution of the LP relaxation could be obtained effectively with the help of column generation (Desaulniers *et al.*, 2005), which also allowed to generate feasible integer solutions (e.g., by rounding procedures) of reasonably good quality. In particular, it is also due to these convincing properties of the relaxation that it still serves as an important cornerstone of some of the most powerful solution methods, such as branch-and-price algorithms, for the CSP or the closely related *bin packing problem* (BPP), see (Belov and Scheithauer, 2002) or (Valério de Carvalho, 1999) to mention only some (by far not exhaustive) references.

Because of the exponential number of variables in the pattern-based model already alluded to, a parallel branch of research has emerged in the literature dealing with *pseudo-polynomial formulations*. The theoretical foundations of such approaches mainly include the relations between integer and dynamic programming (viewed from a perspective of *network flows*) and were already introduced a little later by Shapiro (1968) and Wolsey (1977). Another modeling idea, based largely on publications by Rao (1976) and Dyckhoff (1981), on the other hand, focuses on the individual cuts necessary to define a pattern (which is, more or less, a collection of cuts) and is therefore also referred to as the *one-cut model*. Even though these approaches, as we know today, combined the advantages of a relatively manageable size with a good LP bound at the same time, they were rather of theoretical use at the moment of their conception, since even moderate instances

[1]Although this will not be covered intensively in our article, we would like to note that this observation has developed into one of the most important (open) theoretical research questions in the field of cutting and packing to date, the so-called *MIRUP conjecture* (for the CSP) or *MIRDP conjecture* (for the SSP). For the purpose of further reading, we recommend the articles by Caprara *et al.* (2015) and Kartak *et al.* (2015) (for the CSP case) as well as Martinovic and Scheithauer (2016b, 2018) (for the SSP case) to the inclined reader, but not without mentioning that the references used therein represent the historical developments within this field much better than is possible by only citing a few contributions as examples.

were not solvable with the just developing computer technology. For the sake of exposition, we would like to quote here a passage from (Stadtler, 1988) that summarizes well the attitude towards the alternative models at that time:

> "This leads to the conclusion, that all cutting stock problems that can be solved by one-cut models could also be solved by the column generation approach for the complete-cut model (but not vice versa). [...] The set of real world cutting stock problems solvable by the one-cut model is only a subset of those which could be tackled by the column generation approach."

Even though the importance of numerical aspects was manifested in discrete optimization at least since the well-known book (Nemhauser and Wolsey, 1988), it was not until around the turn of the millennium that these pseudo-polynomial models could be profitably used to solve the CSP. One of the first contributions is due to Valério de Carvalho (1999), who succeeded to solve exactly larger instance sizes, including some open instances of the BPP, by implementing a column (and row) generation algorithm (for a flow-based approach) that reduced the computational efforts considerably. Thanks to these first numerical successes, a very detailed theoretical overview article (Valério de Carvalho, 2002), and the steadily progressing development of suitable hardware components and (commercial) optimization software, the scientific discussion of the alternative models was significantly revived. The advantages already hinted at in the early contributions mentioned above were finally given the opportunity to prove themselves in meaningful practical experiments. From that moment on, any of the approaches (i.e., the pattern-based model and the pseudo-polynomial formulations) has seen numerous improvements over the years, all of which fostering contributions to further increase the size of problems that can be solved in a reasonable amount of time. We would like to highlight the following contributions in particular as significant milestones:

- stabilization techniques for column generation, limiting the fluctuations of dual prices so that a faster convergence is obtained (Valério de Carvalho, 2005; Ben Amor *et al.*, 2006a; Gschwind and Irnich, 2016),
- reduction methods for the one-cut model (Martinovic *et al.*, 2018), and for the flow-based formulations (Brandão and Pedroso, 2016), the latter of which culminating in the very powerful *reflect arc-flow* approach proposed by Delorme and Iori (2020).

While both these major techniques will be explained in detail in the corresponding main parts of the article, here we just mention the survey articles (Delorme *et al.*, 2016) and (de Lima *et al.*, 2022a) for a good overview of further contributions in terms of the CSP and corresponding modeling frameworks.

The SSP has found its way into the scientific discussion much later than the CSP, but it has nevertheless followed a strikingly similar trajectory. More precisely, the SSP was first described in the Ph.D. thesis of Assmann (1983) and the follow-up article of Assmann *et al.* (1984), namely, conceptually slightly misleading, as a *dual bin packing problem* (DBPP). As already indicated at the beginning, this does not at all mean a duality in the sense of mathematical optimization, but merely an interaction of the two problems in terms of content. While the DBPP primarily addressed the processing of a very heterogeneous list of items (i.e., all of which have a frequency of one), the term SSP was first used to describe a concrete practical application in the paper industry (Johnson *et al.*, 1997; Zak, 2003), but henceforth has stood for the actual counterpart to the CSP, i.e., a problem with a very homogeneous list of items whose elements can be grouped into item types. Although being similar at first view, different solution methods have been developed for any of the two variants to effectively tackle the few (but existing) problem-specific challenges. We refer the reader to the research of Labbé *et al.* (1995) and Peeters and Degraeve (2006), introducing early branch-and-bound based approaches for the DBPP, as well as the article by Zak (2003) focussing on a pattern-based approach, the so-called *standard model*, for the SSP case. Despite its still young age, a certain range of application fields for the SSP has opened up in the last two decades, especially, where responsible and sustainable handling of raw materials or offcuts is desirable. In addition to the examples from the paper industry, here we further mention manufacturing and inventory planning (Arbib *et al.*, 2002), multiprocessor scheduling problems (Alvim *et al.*, 2004), further practical case studies (Agoston, 2019), and wireless communications (Martinovic *et al.*, 2017; Tragos *et al.*, 2013). Especially in the last mentioned area, in which the optimal reconstruction of unused portions of the frequency spectrum is concerned, approaches based on the SSP have contributed to a substantial improvement of the previous state of research. Moreover, the advancing scientific discussion of the SSP proved useful even when it itself appeared only as a subproblem within combined cutting-and-packing applications (Chen *et al.*, 2019; Kłosowski *et al.*, 2018; Wang *et al.*, 2020).

In recent years, this development of an increasingly broadening range of relevant application fields has been favored primarily by the fact that significant progress has also been made with respect to the theory of alternative and more sophisticated modeling techniques. Most notably, the first holistic research in terms of pseudo-polynomial formulations for the SSP has been conducted in Martinovic and Scheithauer (2016a): The authors not only proposed an *arc-flow model* and a *one-stick model* for the problem under investigation, but also constructively proved the equivalence of their LP relaxations among one another and to that of the pattern-based standard model. This not only provided additional modeling options, but also highlighted their general and computational strengths at the same time, bringing together observations that had remained separate for several decades in the CSP case. However, motivated by the results in Delorme and Iori (2020), some serious improvements of the original arc-flow model have been proposed in Martinovic *et al.* (2020). To be more precise, the authors of that article made use of reversed loss arcs and the idea of the reflect arc-flow model, that (roughly speaking) only requires to deal with the first half of the vertex set to obtain *integer LP* (ILP) formulations of less challenging size. Moreover, the aforementioned piece of research contained the first systematic approach to collect a plethora of differently characterized benchmark instances for the SSP, which then served as a solid basis for extensive computational experiments clearly showing the convincing performance of the new formulations. De Lima *et al.* (2021) present a more generic approach for arc-flow models of pseudo-polynomial size which combines column generation, variable-fixing based on reduced costs, and an asymmetric branching scheme. Experiments on the SSP show clear improvements in terms of average computation times for smaller benchmark instances in comparison to the just mentioned arc-flow formulation and the reflect arc-flow formulation. However, they did not present results of their new approach for the very large, challenging SSP instances that we will consider.

1.2. Overview and Contribution

Despite all the progress on solving the SSP, there is still a large number of unresolved instances within the various classes of test sets. This justifies further research on several possible algorithmic refinements. In this sense, we adapt and refine the reflect+ algorithm of Delorme and Iori (2020) that was originally designed to solve CSP and BPP instances. Reflect+ is an exact algorithm trying to reduce the computational effort in the average case. To this end, a sequence of reflect arc-flow models with only a subset of the arcs/variables are solved by a MIP solver, each providing a heuristic primal solution. The linear relaxation of a pattern-based formulation is solved beforehand to compute a dual bound and to identify promising patterns and (implicitly) also promising arcs for the integer reflect arc-flow formulation. When primal and dual bound coincide, optimality is proven. Otherwise, the next larger reflect arc-flow model of the sequence is considered. The last model is the complete reflect arc-flow model, in which however provably non-optimal arcs are eliminated using reduced-cost arguments.

Algorithm 1 summarizes the adapted and refined reflect+ algorithm that we suggest for solving the SSP. Steps 2–4 are related to the solution of the linear relaxation of the pattern-based formulation. Compared to the reflect+ algorithm of Delorme and Iori, we differ in the following aspects:

- For pricing, we use the smaller arc-flow network. This improves the run time (the impact is however minor) and makes sure that all generated patterns can be later transformed into arcs of the reflect arc-flow model. In contrast, Delorme and Iori (2020) use COMBO from (Martello *et al.*, 1999) that solves a binary knapsack problem first heuristically and exactly only if the heuristic fails.
- We stabilize the column-generation process with the help of dual inequalities, which is particularly beneficial for very large-scale SPP instances where otherwise the column-generation process is too slow so that reflect+ already fails at this stage. Solutions to stabilized pattern-based models consist of a mix of pattern columns and columns representing the dual inequalities. A retransformation into a pure pattern-based solution is then necessary (see Step 4).

The result are two sets P_B and P_{LP} of those patterns present in the final basic solution of the linear relaxation and all generated patterns, respectively.

As a preparatory step (Step 9), reduced costs of all variables of the corresponding arc-flow formulation are computed. In contrast to the suggestion of Delorme and Iori (2020), we obtain them directly from the

Algorithm 1 Reflect+ for the Skiving Stock Problem

```
1: Greedy heuristic $H^1$                                                          ▷ $LB_0$
2: Build initial patterns $P_{init}$ for the column generation (CG) procedure with subset-sum procedure $H^2$
3: Use stabilized CG to solve linear relaxation of pattern-based model     ▷ $z_{LP}$ and $UB = \lfloor z_{LP} \rfloor$
4: Transformation of the CG solution into a pure pattern-based solution             ▷ $P_B$ and $P_{LP}$
5: Level 1: Solve reflect arc-flow model with arc set $\mathcal{A}_1$ constructed with $P_B$            ▷ $LB_1$
6: Level 2: Solve reflect arc-flow model with arc set $\mathcal{A}_2$ constructed with $P_{LP}$         ▷ $LB_2$
7: Level 3: Solve reflect arc-flow model with arc set $\mathcal{A}_3$ where unpromising arcs fixed to zero   ▷ $LB_3$
8: $LB \leftarrow max(LB_0, LB_1, LB_2, LB_3)$
9: Compute reduced costs $\tilde{c}_{ij}$ of arcs $(i, j)$ using path-reduced costs from the CG solution
10: while ($LB < UB$) do
11:     Level 4: Fix arc variables $x_{ij}$ to zero if $\tilde{c}_{ij} > z_{LP} - UB$ to obtain arc set $\mathcal{A}_4$   ▷ $\mathcal{A}_4$
12:     Solve reflect arc-flow model with arc set $\mathcal{A}_4$                             ▷ $LB_4$
13:     $LB \leftarrow max(LB, LB_4)$ and $UB \leftarrow UB - 1$
14: end while
```

linear relaxation of the pattern-based formulation exploiting again that pricing uses the arc-flow network. Transferring these reduced costs to the reflect arc-flow formulation is possible.

The Steps 5–13 are describing how the sequence of reflect arc-flow models is used. At the Levels 1 to 3 (Steps 5, 6, and 7), the solution approach is heuristic because the restricted models are built with heuristic rules to thin out the arc/variable set. In contrast, the loop (Steps 9–13) iteratively solves exact models based on the assumption that the computed lower bound LB is the optimal objective. As long as the new computed solution (if any) is worse than UB, then UB is decreased by one and the process repeats. Note that if the MIRDP conjecture holds, then not more than two traversals of the loop of Level 4 result.

We describe the details of all steps of the reflect+ algorithm in the following sections with a focus on the algorithmic refinements (also underlined in the pseudo code of Algorithm 1) compared to the algorithm of Delorme and Iori (2020) for the CSP.

1.3. Structure of the Paper

The remainder of the paper is structured as follows. Section 2 introduces notation and presents the greedy heuristic and subset-sum procedure. Pattern-based formulations and the solution of their linear relaxation with the help of a stabilized column-generation algorithm are discussed in Section 3. Section 4 then elaborates the construction of restricted reflect arc-flow models and their solution with a MIP solver. Computational results are presented in Section 5. The paper closes with final conclusions drawn in Section 6.

2. Preliminaries and Heuristics

Let us start by formally introducing the SSP. Let $\mathbb{N}$ denote the natural numbers $\mathbb{N} = \{1, 2, 3, \dots\}$ and $\mathbb{Z}_+ = \{0\} \cup \mathbb{N}$ the non-negative numbers.

Definition 1. *An* instance *E of the SSP is given by a tuple $(m, \mathbf{l}, L, \mathbf{b}) \in \mathbb{N} \times \mathbb{N}^m \times \mathbb{N} \times \mathbb{N}^m$ with the interpretation that m is the number of different items types, $\mathbf{l} = (l_1, \dots, l_m)$ the vector of item sizes (lengths), L the threshold length, and $\mathbf{b} = (b_1, \dots, b_m)$ the item frequencies. For convenience, we refer to $I = \{1, \dots, m\}$ as the* item (type) set.

Without loss of generality, we assume the item types to be ordered decreasingly with respect to size, i.e., $l_1 > l_2 > \dots > l_m$ has to be satisfied. Moreover, $l_1 < L$ can be assumed, because any item equal to or larger than L does not have to be considered within the optimization.

Definition 2. *Let $E = (m, \mathbf{l}, L, \mathbf{b})$ denote an SPP instance.*

(i) Any vector $\mathbf{p} \in \mathbb{Z}_+^m$ satisfying $\mathbf{l}^\top \mathbf{p} \geq L$ is called a pattern *(a.k.a. packing pattern).*

(ii) A pattern $\mathbf{p} \in \mathbb{Z}_+^m$ *is* minimal *if there is no other pattern* $\tilde{\mathbf{p}} \in \mathbb{Z}_+^m$ *with* $\tilde{\mathbf{p}} \leq \mathbf{p}$ *(in a componentwise sense).*

(iii) A pattern $\mathbf{p}$ *is* proper *if* $\mathbf{p} \leq \mathbf{b}$ *(in a componentwise sense).*

(iv) The set of all patterns *is denoted by* $P(E)$*, the set of all minimal patterns by* $P^\star(E)$*, and the set of all proper patterns by* $P_p(E)$*.*

Before we are going to deal with exact methods for solving the SSP in the following sections, two heuristic approaches will be discussed here first: the greedy heuristic $H^{(1)}$ and the subset-sum procedure $H^{(2)}$. Both methods have already been briefly mentioned in Algorithm 1, but without explaining their concrete mechanisms. $H^{(1)}$ provides a lower bound for the optimal value and a feasible solution for a given SSP instance (together with an associated set of patterns used), whereas from $H^{(2)}$ we obtain only an expedient set of patterns. While the former represents purely numerical information that can be provided by several SSP heuristics with similar or even identical quality, the selection of, in a sense, "promising" patterns is crucial for the further progress of the reflect+ algorithm. We therefore distribute the two aforementioned purposes of such an approximate solution over two different methods, using:

- the very simple greedy heuristic $H^{(1)}$ to generate a first lower bound,
- the somewhat more sophisticated subset-sum procedure $H^{(2)}$ to provide a reasonable set of, in a sense "low-waste", initializing patterns for the column-generation process.

The greedy heuristic $H^{(1)}$ follows the procedure already presented in (Peeters and Degraeve, 2006) and (Martinovic *et al.*, 2020). By that, we mean that a bin is first filled with as many copies as possible of the currently largest available item type $i^\star \in I$ (so that the threshold is not yet exceeded). Among all items then still available, one is searched for that completes the previous configuration to a feasible pattern while wasting as little material as possible. If such an item does not exist, the bin is first further filled with as many items of the type $i^\star + 1$ (again, without exceeding the capacity), and then it is checked again whether a completing item (in the previously defined sense) exists.

In contrast, the subset-sum procedure $H^{(2)}$ builds patterns based on the subset-sum problem. More precisely, we iteratively fix an item type $i \in I$ and apply the following steps:

S1: Define a new threshold length $L_i := L - l_i$ and a new set of items: $I_i := I \setminus \{i\}$.

S2: Solve a subset-sum problem to find a subset of items (of I_i), each of which appearing at most once, whose item lengths add up to the new threshold length L_i. If there is no such combination, find a subset of items with the shortest pattern length larger than L_i.

S3: Add item i to this subset and store the set of items as a new pattern.

We implemented the algorithm of Cormen *et al.* (2009) to tackle the subset-sum problems. Typically, the main advantage of the subset-sum procedure is that most of the resulting patterns have a pattern length equal to or only slightly larger than the threshold length, whereas patterns obtained from the greedy heuristic are often strictly longer than L. In other words, one could say that the patterns constructed using procedure $H^{(2)}$ are generally more promising in the sense that they exhibit better material utilization. As we will see later, this idea is particularly helpful for those benchmark instances whose integer solution consists of low-waste or even waste-free patterns (like AI/ANI), see Section 5.

3. Pattern-based Formulations and Solution with Stabilized Column Generation

With the definition of patterns, it is possible to formulate pattern-based models of the SSP as proposed by Zak (2003). Such a model can be formulated for any set of patterns P, e.g., the complete set $P = P(E)$ or a subset of it. Let $\lambda_{\mathbf{p}} \in \mathbb{Z}_+$ for each $\mathbf{p} = (p_i)_{i \in I} \in P$ count how often the respective pattern is built, then we obtain the pattern-based formulation:

$$(F_P) \qquad z(F_P) = \max \sum_{\mathbf{p} \in P} \lambda_{\mathbf{p}} \tag{1a}$$

$$\text{subject to} \quad \sum_{\mathbf{p} \in P} p_i \lambda_{\mathbf{p}} \leq b_i, \qquad i \in I, \tag{1b}$$

$$\lambda_{\mathbf{p}} \in \mathbb{Z}_+, \qquad \mathbf{p} \in P. \tag{1c}$$

The objective function (1a) maximizes the number of constructed patterns, while constraints (1b) make sure that at most b_i items of type $i \in I$ are used. The domain of the decision variables is stated in (1c).

Any pattern set $P \supseteq P^\star(E) \cap P_p(E)$ guarantees that the corresponding solution is optimal for the instance E. In particular, the integer models for $P(E)$, $P^\star(E)$, and $P_p(E)$ are equivalent, i.e.,

$$z(F_{P(E)}) = z(F_{P^\star(E)}) = z(F_{P_p(E)}).$$

The linear relaxation of the pattern based model (1) is denoted by $LP(F_P)$, its objective value by $z_{LP}(F_P)$. For the linear relaxations, we have

$$z_{LP}(F_{P^\star(E)}) = z_{LP}(F_{P(E)}) \qquad \text{and} \qquad z_{LP}(F_{P_p(E)}) \leq z_{LP}(F_{P(E)}),$$

where, in the latter case, $<$ is possible for some SSP instances E. Consequently, using proper patterns may lead to a better linear-relaxation bound, as observed in Martinovic and Scheithauer (2016c). In contrast, the first relation has to be an equation as any non-minimal pattern can be reduced to an element of $P^\star(E)$ even in the LP relaxation.

3.1. Stabilized Column Generation

Given the generally exponentially large number of patterns and variables in the aforementioned models, they cannot be solved directly in their integer version. However, the continuous relaxation of such a pattern-based formulation can be solved effectively using the column-generation method. Here, one starts with a small set of patterns and iteratively checks whether more patterns need to be added to this set by solving a pricing problem. In our case, we use the arc-flow network to handle the pricing subproblems and add up to 500 new patterns with negative reduced cost to the pattern pool per iteration (recall that the reduced or opportunity cost of a pattern $\mathbf{p}$ is $-1 + \boldsymbol{\pi}^\top \mathbf{p}$ for a dual-prices vector $\boldsymbol{\pi}$ in the linear maximization problem). Of course, those patterns (which result from their associated paths in the graph) with particularly small reduced cost are selected preferentially.

The column-generation process for solving (1) produces new patterns complementing those of the initial pattern set P_{init}, i.e., Step 3 of Algorithm 1. For the quality of the linear-relaxation bound, we are interested to only generate patterns from the set $P^*(E) \cap P_p(E)$. However, other patterns $\mathbf{p} \in P(E) \setminus (P^*(E) \cap P_p(E))$ may be generated also because of the following two algorithmic components.

First, pricing problems for CSP and SPP are solved as a type of shortest-path problem over a digraph (V, A) in which vertices $v \in V$ correspond the possible lengths of partial patterns, i.e., values $0, 1, 2, \dots, L$ for the CSP and likewise $0, 1, 2, \dots, L, L+1, \dots, L + l_1 - 1$ for the SSP, and arcs $(j, j + l_i) \in A$ represent the item types $i \in I$ (Gilmore and Gomory, 1963; Zak, 2003). This can lead to patterns containing a number of items larger than their demands in the CSP and larger than their frequencies in the SSP. In contrast, by solving a binary knapsack problem only binary patterns which are proper and maximal are generated for the BPP. Our pricing algorithm uses the SSP arc-flow network for pricing out patterns and solves a shortest-path problem over it (a detailed definition of the network can be found in Martinovic *et al.*, 2020, Section 2). The use of the arc-flow network reduces the occurrence of patterns that are not proper. Furthermore, it guarantees that patterns are minimal. Neither the arc-flow nor the reflect arc-flow model can however guarantee that only proper patterns occur (see again Martinovic *et al.*, 2020, Remark 4 and Example used in Theorem 8). As a result, we generally have the relationship

$$P_p(E) \cap P^\star(E) \subseteq P_{CG}(E) \subset P(E),$$

and therefore

$$z_{LP}(F_{P^\star(E) \cap P_p(E)}) \leq z_{LP}(F_{P_{CG}(E)}) \leq z_{LP}(F_{P(E)})$$

where $P_{CG}(E)$ refers to the pattern set computed with the column-generation procedure. Note that strict inequalities are possible.

Second, we use a stabilization approach with dual inequalities for the column-generation algorithms. Let $\boldsymbol{\pi} = (\pi_i)_{i \in I}$ be the dual variables to the constraints (1b). Any inequality $\mathbf{a}^\top \boldsymbol{\pi} \geq c$ with $(\mathbf{a}, c) \in \mathbb{R}^{m+1}$ is a dual inequality corresponding with a column $\mathbf{a}$ with cost $c \in \mathbb{R}$ in the primal formulation (1). Ben Amor *et al.* (2006a) classify dual inequalities into

- *dual-optimal inequalities* (DOIs) are dual inequalities that do not cut off any dual optimal solution $\boldsymbol{\pi}^\star \in \Pi^\star$ (we denote the set of dual optimal solutions by $\Pi^\star$ in the following)
- *deep dual-optimal inequalities* (DDOIs). The latter term refers to a set $\{\mathbf{a}_k^\top \boldsymbol{\pi} \geq c_k\}$ of dual inequalities and means that the inequalities together do not cut off the entire dual optimal space, i.e., $\Pi^\star \cap \{\boldsymbol{\pi} \in \mathbb{R}_+^m : \mathbf{a}_k^\top \pi \geq c_k \text{ for all } k\} \neq \varnothing$.

Ben Amor *et al.* (2006a) and Gschwind and Irnich (2016) prove several equivalent properties of DDOIs, among others, that the primal formulation extended by columns corresponding to a set of dual inequalities and the primal formulation itself provide the same linear-relaxation bound if and only if the dual inequalities are DDOIs. Moreover, this is equivalent to the existence of a constructive procedure that transforms any optimal mixed solution (patterns and columns resulting from dual inequalities) into a pure pattern solution of the same cost. The point is now that this transformation may produce patterns that are not proper. The type of produced patterns depends on both the initial pattern set P_{init}, the pattern generation process in column generation, and the type of dual inequalities added to the RMP. We therefore describe families of dual inequalities more precisely.

Following the taxonomy of Gschwind and Irnich (2016) and Heßler *et al.* (2018), the most general family of dual inequalities that they consider is defined as follows: For any item $j \in I$, subset $S \subset I \setminus \{j\}$, and weights $(w_i)_{i\in S} \in \mathbb{N}^{|S|}$ with $\sum_{i\in S} w_i l_i \geq l_j$, the inequality

$$\sum_{s\in S} w_i \pi_i - \pi_j \geq 0$$

is called *weighted subset inequality* (WSI) for $(j, S, (w_i)_{i\in S})$. Several WSIs defined by $(j_k, S_k, (w_i^k)_{i\in S_k})$ indexed by $k \in K$ lead to an extended primal model in which (1b) is replaced by

$$\sum_{\mathbf{p}\in P} p_i \lambda_{\mathbf{p}} - \sum_{k\in K: j_k = i} y_k + \sum_{k\in K: i\in S_k} w_k y_k \leq b_i, \qquad i \in I, \tag{1b'}$$

with additional non-negative continuous variables

$$y_k \geq 0, \qquad k \in K. \tag{2}$$

If all weights are equal to one, i.e., $w_i = 1$ for all $i \in S$, the dual inequality is a *subset inequality* (SI) for the pair (j, S). The interpretation of an SI defined by (j, S) for the primal model (1) is that one can replace an item of type j by the items of type $i \in S$ in every pattern without making the pattern infeasible. If $S = \{i\}$ is a singleton set (assuming $l_i > l_j$), the inequality becomes $\pi_i - \pi_j \geq 0$ and is denoted as a *pair inequality* (PI) for the item pair (j, i). The dual interpretation of the PI for (j, i) is that one unit of supply of item type i is at least as useful as one unit of supply of item type j, i.e., $\pi_i \geq \pi_j$. The set of all PIs (with $\mathcal{O}(m^2)$ elements) is equivalent to the set of *ranking inequalities* (RIs) $\pi_i - \pi_{i+1} \geq 0$ for $i \in I \setminus \{m\}$ (recall that we assume $l_1 > l_2 > \cdots > l_m$). The set of RIs comprises only $\mathcal{O}(m)$ elements.

		RIs $\subset$ PIs	$\subset$	SIs	$\subset$	WSIs
Pattern-based formulation (1a), (1b'), $\lambda_{\mathbf{p}} \geq 0$, and (2) with pattern set P	$P = P_p(E)$	DDOIs		—		—
	$\subset$					
	$P = P_{CG}(E)$	DDOIs		—		—
	$\subset$					
	$P = P(E)$	DOIs		DOIs		DOIs

Table 1: Families of dual inequalities (DIs) and their relationship to pattern-based formulations defined over different pattern sets; entries "—" indicate that the family of dual inequalities is not necessarily a set of DOIs or DDOIs; some inequalities may however be DOIs or DDOIs depending on the SSP instance E.

Table 1 summarizes what is known about the relationship between different families of dual inequalities and the linear-relaxation of the extended linear pattern-based formulations.

3.2. Transformation into a Pure-Pattern Solution

A well-known issue related to the patterns generated during the stabilized column-generation procedure is that they can obtain items in a larger quantity than allowed by the input data of the given instance (Ben Amor *et al.*, 2006b). Although one could simply delete such patterns without compensation in the construction of thinned out graphs that we are aiming at, this would cause us to lose valuable information from the previously determined solution of the LP relaxation. Thus, it is strongly recommended to transform these configurations, which are invalid for our networks, into actual patterns.

Remark 1. *More precisely, if the patterns generated during stabilized column generation are not transformed, the resulting arc-flow digraph of Level 1 and Level 2 of reflect+ will include (more) paths that represent invalid patterns regarding the given items of the instance. The effects of a missing transformation can later be observed in Table 4 for the large GI instances.*

In general, the following steps are required to obtain a pure-pattern solution:

1. Select a DOI that is part of the LP solution and find all patterns of the solution containing the item that was transformed into a longer item by this DOI. Store these patterns in a list.
2. Sort this list of patterns in increasing order by the value of the item that will be transformed.
3. Take the first element of the list: Retransform the item that was exchanged by the DOI. In case the frequency of occurence of this pattern in the LP solution is high enough, eliminate the DOI completely and update frequency of the pattern (also in the list). Otherwise, convert the pattern completely into an original pattern, delete the element from the list, and reduce the solution value attached to the DOI accordingly.
4. If the solution value of the DOI is still positive, go to Step 3 again. In the opposite case (that is, the DOI is eliminated), take the next DOI (if available) and go back to Step 1.

Typically, especially for larger instance sizes, there will be multiple options to choose a pattern to be transformed for the considered DOI. In our internal precalculations, we obtained slightly better results whenever as many patterns as possible have been transformed. Hence, in Step 2 of the above list, the patterns are sorted in increasing order by the value of the item that will be transformed. By that, the solution value of the artificial DOI column is distributed over the maximum number of patterns in the final pure-pattern solution.

4. Restricted Reflect Arc-Flow Models

Flow formulations form a powerful tool in cutting and packing, as they combine important structural properties (e.g., a good LP relaxation) with a large illustrativeness and a generally manageable model size, and so they can typically be handled well by ILP solvers. The reflect arc-flow model has recently emerged as the most important representative within this group (Delorme and Iori, 2020; Martinovic *et al.*, 2020). In contrast to classical graph-based approaches, this formulation is particularly convincing in that, roughly speaking, only half the bin size has to be modeled and, thus, numerous variables can be saved. This is significant because in conventional networks the number of active vertices and arcs is very large, especially in the second half of the graph, due to the ever increasing combination possibilities of the items. In return for these savings, there is no longer a classical representation of patterns by paths from the source to a sink, but a pattern is modeled as a composition of two sub-paths that must be suitably combined. Without going into all conceivable details, it can be stated that a reflect graph consists of different classes of arcs. It can be obtained from a classical arc-flow network with arc set $\mathcal{A}$ according to the following rules:

- Any arc $(u, v) \in \mathcal{A}$ with $u < v \leq R$ is kept as a *standard arc* (u, v, s),
- Any arc $(u, v) \in \mathcal{A}$ with $u < R < v$ is replaced by a *reflected arc* $(u, L - v, r)$,
- Any arc $(u, v) \in \mathcal{A}$ with $R \leq u < v$ is omitted.

In this construction, $R := L/2$ serves as the reflection point, and fractional coordinates can be avoided by scaling any item length by a factor of 2, if required. Moreover, we introduce *loss arcs* (d, e, ℓ) with $e = succ_{\mathcal{U}}(d)$, specifying vertex e as the direct successor of vertex d. Loss arcs are added to the network, starting at the lowest-indexed vertex that is the head of a reflected arc and continuing step-by-step until

R is reached. Finally, an artificial arc $(L/2, L/2, r)$ has to be added to be able to combine two sub-paths representing exactly half of the bin size. Let us refer to the obtained vertex and arc sets by $\mathcal{U}$ and $\mathcal{A}$, respectively. Moreover, we define $\mathcal{U}^\star := \mathcal{U} \setminus \{0\}$ and $\mathcal{A}^\star := \mathcal{A} \setminus \{(R, R, r)\}$ for the sake of an easier notation in the following ILP model. Introducing $\xi_{de\kappa} \in \mathbb{Z}_+$ to indicate the units of flow carried by arc $(d, e, \kappa) \in \mathcal{A}$, where κ is a generic label for the arc type, we obtain the reflect arc-flow model for the SSP proposed in Martinovic *et al.* (2020):

$$(F_{\mathcal{A}}) \qquad z_{\mathcal{A}} = \max \sum_{(d,e,r)\in\mathcal{A}} \xi_{der} \tag{3a}$$

$$\text{subject to} \sum_{\substack{(d,e,s)\in\mathcal{A}:\\ e-d=l_i}} \xi_{des} + \sum_{\substack{(d,e,r)\in\mathcal{A}:\\ e=2R-d-l_i}} \xi_{der} \le b_i \qquad i \in I \tag{3b}$$

$$\sum_{(0,e,s)\in\mathcal{A}} \xi_{0es} + \sum_{(0,e,r)\in\mathcal{A}} \xi_{0er} = 2 \sum_{(d,e,r)\in\mathcal{A}} \xi_{der} \tag{3c}$$

$$\sum_{(d,e,s)\in\mathcal{A}} \xi_{des} + \sum_{(e,f,\ell)\in\mathcal{A}} \xi_{ef\ell} = \sum_{(d,e,r)\in\mathcal{A}} \xi_{der} + \sum_{(d,e,\ell)\in\mathcal{A}} \xi_{de\ell} + \sum_{\substack{\kappa\in\{r,s\},\\ (e,f,\kappa)\in\mathcal{A}}} \xi_{ef\kappa} \qquad e \in \mathcal{U}^* \tag{3d}$$

$$\sum_{(d,e,\ell)\in\mathcal{A}} \xi_{de\ell} + \sum_{(d,e,r)\in\mathcal{A}} \xi_{der} \ge \sum_{(e,f,\ell)\in\mathcal{A}} \xi_{ef\ell} \qquad e \in \mathcal{U}^* \tag{3e}$$

$$\xi_{de\kappa} \in \mathbb{Z}_+ \qquad (d, e, \kappa) \in \mathcal{A}^* \tag{3f}$$

$$\xi_{R,R,r} \in \mathbb{Z} \tag{3g}$$

For reflect+, we extend the formulation of Martinovic *et al.* (2020) by adding additional variables that follow the same logic as the RIs in the pattern-based model. More precisely, let $t_i \in \mathbb{Z}_+$ for $i > 1$ mimic what an RI for items $i-1$ and i is doing: replace item i by longer item $i-1$ (knowing that $l_{i-1} > l_i$ holds). Therefore, constraints (3b) are replaced by the following constraints:

$$\sum_{\substack{(d,e,s)\in\mathcal{A}:\\ e-d=l_i}} \xi_{des} + \sum_{\substack{(d,e,r)\in\mathcal{A}:\\ e=2R-d-l_i}} \xi_{der} - \begin{cases} t_i, & i > 1 \\ 0, & \text{otherwise} \end{cases} + \begin{cases} t_{i+1}, & i < m \\ 0, & \text{otherwise} \end{cases} \le b_i \tag{3b'}$$

Remark 2. *Similar conditions could also be added to describe the exchange of two or more items by one object with a larger total length. However, it was shown in Delorme and Iori (2020) that this generally does not lead to any improvement. The drawback is the relatively large number of additional constraints so that here we just focus on the RI-like exchanges.*

In our algorithm, we do not use the complete set of arcs in many places, but only subsets $\mathcal{A}' \subset \mathcal{A}$ of them in order to be able to work on thinned out graphs. It is therefore indispensable to describe how exactly a subset P of patterns is integrated into the network, since this largely determines the final combination possibilities. To this end, we run through the given subset of patterns and repeat the following procedure for any fixed pattern:

- We start with the largest item i appearing in the pattern $\mathbf{p}$ and add its corresponding arc(s) (starting at vertex 0; as often as the coefficient p_i indicates) to the network. We then place the arc of the second largest item next to it and process the remaining items equivalently until the reflection point is reached or exceeded.
- We then sort the remaining items of the given pattern with respect to increasing item lengths and include them in the graph (again starting at vertex 0) in the same way as before. Obviously, this time all items will be used before exceeding the reflection point.
- Combine the two paths by adding loss arcs or the artificial reflected arc, if required.

Before we deal with the numerical test calculations in the next section, we would now like to conclude by going into more detail about the individual stages of our algorithm:

Level 1: At first we solve $F_{\mathcal{A}_1}$, i.e., the reflect arc-flow model with just the pattern set P_B. By that we obtain a feasible solution and a lower bound LB_1 for the unknown optimal value. In case the gap between LB_1 and the upper bound UB is less than one, we have already found an optimal solution and can stop the algorithm.

Level 2: We proceed in the same way as in Level 1, but this time solve $F_{\mathcal{A}_2}$ which originates from the typically much larger pattern pool P_{LP}.

Level 3: As an auxiliary tool, we first solve the linear relaxation of $F_{\mathcal{A}}$ and obtain the solution $\bar{\xi}$. We use the information provided by that specific solution to build the set $\mathcal{U}_n := \{u \in \mathcal{U} \colon \bar{\xi}_{de\kappa} < \epsilon, \forall (d,e,\kappa) \in \mathcal{A}$ incident to $u\}$ for $\epsilon = 10^{-6}$, which contains all vertices through which effectively no flow units pass. We then solve Solve $F_{\mathcal{A}}$ with subsequent additional constraints

$$\xi_{de\kappa} = 0 \qquad d \in \mathcal{U}_n, (d,e,\kappa) \in \mathcal{A}, \tag{4}$$

which is equivalent to dealing with $F_{\mathcal{A}_3}$, a formulation that does not contain the unpromising arcs at all. On the one hand, the calculation effort gets a lot smaller with these constraints, but on the other hand there might be some arcs fixed to zero now that are part of an optimal integer so that the solution obtained in this level is just a lower bound with a gap of at least one.

Level 4: This level contains an iterative mechanism. At first, we assume that $UB := \lfloor z_{LP} \rfloor$ is the true optimal value. In case the reduced cost of an arc (from $\mathcal{A}$) is larger than the difference between z_{LP} and UB, this arc is not required for an optimal solution and its flow can be fixed to zero. By that, we obtain an arc set $\mathcal{A}_4$. The solution of $F_{\mathcal{A}_4}$ provides the lower bound LB_4. In case an optimal integer solution has not been found during an iteration, the *tentative upper bound* UB is decreased by one before the next iteration starts.
Whereas Delorme and Iori (2020) use reduced costs from the LP relaxation of the reflect arc-flow model, we found out that it is slightly advantageous to use path-reduced costs from the pattern-based formulation of the column-generation algorithm instead (see Section 5). Path-reduced costs can be computed via a bidirectional search technique as proposed by Irnich *et al.* (2010). Given the reduced costs of a path, the corresponding arc-variables in the arc-flow model can be fixed to zero in case the path-reduced costs are too high. Irnich *et al.* (2010) also show how the reduced costs of the arc-variables can be calculated from the reduced costs of paths.

5. Computational Results

The entire reflect+ algorithm is implemented in C++ and compiled with Microsoft Visual Studio 2017 into 64-bit single thread code. We use CPLEX 12.10 for (re)optimizing the RMPs and whenever LP relaxations of the reflect arc-flow model need to be solved. Gurobi 9.1.0 is utilized when solving ILP formulations. Default values are kept for all parameters of the two solvers, except from setting the number of threads to one. In addition, Gurobi exerts the barrier algorithm. The computational study was executed on a 64-bit Microsoft Windows 10 computer with Intel® Core™ i7-5930k CPU clocked at 3.5 GHz and 64 GB of RAM. The time limit is set to 3600 seconds per instance.

5.1. Results on SSP Benchmark Instances

We let the refined reflect+ algorithm run with all new components (underlined in green in Alg 1). Subsequently, we will then systematically analyze the impact of each main algorithmic component in Section 5.2.

In our numerical investigation, we focus in particular on those classes of SSP benchmark instances that could only be solved by the ILP solver using the traditional reflect arc-flow model from Martinovic *et al.* (2020) to a limited extent or not at all within the time limit of one hour. These are the classes

- AI/ANI of Delorme *et al.* (2016), which are characterized by particularly many feasible patterns and the fact that any optimal solution is composed of waste-free item combinations only. Furthermore, it should be noted that all ANI instances have an integrality gap of one and are, thus, even more challenging.

- GI of Gschwind and Irnich (2016), which are mainly characterized by a huge bin size of up to 1.5 million and thus generally lead to very large networks (and also very large ILPs).

We run reflect+ and compare the state-of-the-art results obtained with the compact formulation (3) both with our implementation and that in (Martinovic *et al.*, 2020). Table 2 provides aggregated results. For each subclass of instances (the number of instances is given in column #), we list the average computation times (*time*) in seconds and the number (#opt) of instances solved to proven optimality. Moreover, we indicate the best values by using bold font.

		Refined reflect+		ILP solver with reflect arc-flow model			
				Our implementation		Martinovic *et al.*	
Instances	#	*time*	#opt	*time*	#opt	*time*[†]	#opt
AI201	50	22.5	**50**	**20.3**	**50**	26.3	**50**
AI402	50	2024.4	28	**1801.1**	**32**	2016.9	31
ANI201	50	274.5	**50**	**206.0**	**50**	238.5	**50**
ANI402	50	3586.0	**1**	**3548.2**	**1**	3600.0	0
GI125	80	**29.6**	**80**	1794.8	40	1802.1	40
GI250	80	**153.2**	**80**	1836.1	40	1854.0	40
GI500	80	**707.2**	**79**	3004.6	32	3584.2	2
GI750	40	**1105.9**	**39**	3600	0	–	–
GI1000	40	**2351.6**	**33**	3600	0	–	–

Table 2: Reflect+ vs. ILP solver using the reflect arc-flow formulation on large benchmark instances of Gschwind and Irnich (2016); Delorme *et al.* (2016); †: Martinovic *et al.* (2020) use Gurobi 8.0.1 on a PC with a AMD A10-5800K CPU with 16GB of RAM.

The main observations can be summarized as follows:

- For the ANI instances, it is clear for theoretical reasons that reflect+ can in principle lead to no improvement, since all levels of the algorithm must be executed due to the positive integrality gap. In particular, reflect+ must solve the complete reflect arc-flow ILP at the end (as well as handle the additional levels before in vain).
- For the AI instances, reflect+ does not lead to any noticeable improvements here either. Presumably, this is due to the fact that an optimal solution consists of nothing but waste-free patterns. Is seems that the required representatives of these patterns, however, are not generated to a sufficient extent in the column-generation process and are therefore not present in the thinned-out graphs constructed from them.
- Whereas the ILP solver using the reflect arc-flow formulation of Martinovic *et al.* (2020) was only able to deal with some easier subclasses of the GI set (not involving the maximum bin size of 1.5 million units), reflect+ is able to successfully cope with almost all the instances, even if the number of item types is particularly high. Altogether, only nine GI instances (seven of which coming from the most challenging parameter setting with $m = 1000$) could not be solved by reflect+ in reasonable time. This is a clear improvement compared to the traditional reflect arc-flow approach.

Table 3 makes the same comparison between the reflect+ algorithm and the ILP solver-based approach for the smaller benchmark instances of Scholl *et al.* (1997). The meaning of the table entries is identical as for Table 3. Clearly, the refined reflect+ algorithm can keep up with the ILP solver applied to the reflect arc-flow model. It even outperforms it for the class Scholl 3. Thus, we like to point out that the multi-level reflect+ algorithm is not disadvantageous even for smaller instances that can be easily solved directly with a compact formulation. Based on this finding, these smaller instances will not be considered in all subsequent experiments.

		Refined reflect+		ILP solver with reflect arc-flow model			
Instances	#	*time*	#opt	*time*	#opt	*time*†	#opt
Scholl 1	720	0.2	**720**	0.1	**720**	**0.0**	**720**
Scholl 2	480	13.4	**480**	**13.1**	**480**	20.8	**480**
Scholl 3	10	**6.7**	**10**	34.9	**10**	65.1	**10**

Table 3: Reflect+ vs. ILP solver using the reflect arc-flow formulation on smaller benchmark instances of Scholl *et al.* (1997).

5.2. Evaluation of Algorithmic Components

Now we compare the fully-fledged reflect+ (referred to as Setting (1) in the following) with versions of reflect+ in which always one of main algorithmic components is switched off. Specifically, this means without the component:

(2): subset-sum procedure (only greedy heuristic),
(3): stabilized column generation (unstabilized),
(4): retransformation into a pure pattern-based solution (pattern columns only), and
(5): reduced costs from the LP relaxation of the reflect arc-flow model instead of path-reduced costs from the column generation.

Settings (2)–(5) represent the novelties in which our reflect+ algorithm differs from the primary reflect+ algorithm for the CSP presented in (Delorme and Iori, 2020).

Table 4 summarizes the comparison between the settings (1)–(4) and Table 5 the comparison with setting (5) in a more detailed fashion.
We highlight the following facts:

		Setting (1) refined reflect+		Setting (2) only greedy heuristic		Setting (3) non-stabilized column generation		Setting (4) no pattern retransformation	
Instances	#	*time*	#opt	*time*	#opt	*time*	#opt	*time*	#opt
AI201	50	22.5	50	31.5	50	22.4	50	23.3	50
AI402	50	2024.4	28	2571.4	19	2193.8	25	2034.9	28
ANI201	50	274.5	50	305.4	50	293.4	50	278.8	50
ANI402	50	3586.0	1	3600.0	0	3582.6	1	3575.9	1
Subtotal	200	1476.9	129	1627.1	119	1523.1	126	1478.2	129
GI125	80	29.6	80	18.6	80	29.5	80	26.8	80
GI250	80	153.2	80	105.2	80	165.2	80	139.9	80
GI500	80	707.2	79	592.0	79	857.0	78	707.3	79
GI750	40	1105.9	39	639.7	39	1249.4	39	1169.0	38
GI1000	40	2332.7	34	1430.5	38	2557.3	23	2332.0	34
Subtotal	320	638.4	312	437.7	316	677.3	300	656.1	311

Table 4: Effects of adapted settings in the refined reflect+ algorithm

- Using AI/ANI instances as an example, we see that the stabilization of column-generation process (Setting 3) and the transformation of artificial patterns (Setting 4) have no significant effect on the computations. Although this may seem surprising at first, we recall that an optimal solution of these instances must consist exclusively of exact patterns $\mathbf{p}$, i.e., patterns with $\mathbf{l}^\top\mathbf{p} = l$. Since this is also true for the continuous relaxation, no DOIs can be used in the LP solution, so no transformation of patterns occurs either. It is therefore understandable that, except for possible random effects, Settings (1), (3), and (4) basically come to the same results.

- In contrast, however, we see impressively in the example of class AI402 in Table 4 that employing the subset-sum procedure can lead to a significant improvement in overall performance, since about 50% more instances can be solved optimally. It can be concluded that, unlike the greedy heuristic, the subset-sum method produces more "useful" patterns with little or no waste. Therefore, noticeably better feasible SSP solutions can be constructed in the thinned-out networks considered at the various stages of our overall algorithm. Even if these improvements do not quite reach the level of the reflect arc-flow model from Martinovic *et al.* (2020), the findings make a significant contribution to improving the classical reflect+ algorithm presented in Delorme and Iori (2020) in an instance-specific way by clever preprocessing.
- The effects for the GI instances are best seen by looking at the most challenging subclass GI1000 in Table 4. For these instances (with a very large number of different item types), executing column generation (before entering Level 1 of our algorithm) is already very difficult and, consequently, omitting stabilization techniques leads to large performance degradations. More precisely, 11 to 15 instances less were solved than in all other settings.
- Comparing the number of optimally solved instances with Setting (1) and Setting (4) in Table 4, only one instance from GI750 stands out that can be solved with our refined reflect+ algorithm, but without transforming the DDOI columns it is not solved exactly within the time limit. Regarding the average computation times of these two settings, no significant advantage for the refined reflect+ is apparent at first glance. Except for random effects, the computation times hardly differ. Although here it seems that the transformation brings only a minimal benefit, later we will demonstrate a systematic advantage of this action regarding the levels of reflect+ that need to be passed.
- It is noticeable that in the GI instances the use of the subset-sum procedure is rather harmful. In the classes GI125 to GI750 in Table 4, this can be seen above all in the sometimes significantly larger computation times compared to the mere application of the greedy heuristic. For the most difficult class GI1000, one even misses the optimal solution of four instances. To explain this, we give two reasons: On the one hand, running the subset-sum procedure itself takes significantly more time than an ordinary greedy procedure, especially when such auxiliary problems have to be solved in advance for up to $m = 1000$ different item types. However, from our point of view, the crucial factor is that, on the other hand, GI instances, in general, do not rely on low-waste patterns in the optimal solution. Substantial effort is invested by the subset-sum procedure to finally generate patterns that are not necessarily needed to successfully tackle the given instance. This obviously has a negative overall impact on the numerical results.
- Last but not least, it is slightly beneficial to use the path-reduced costs from the pattern-based formulation of the column-generation approach instead of the reduced costs from the reflect arc-flow model in Level 4 of reflect+. This positive effect can be observed in Table 5 which contains only instances solved exactly in Level 4. The choice of instances is due to the fact that the reduced costs are only relevant in the last level of reflect+. Except for GI125 und ANI402, which each contain only one instance in this case, the average computation time is faster when path-reduced costs are used.

To examine the effects just described in more detail, we look at three different ways to design column generation before entering Level 1 in Table 6. In that table, #tl denotes the number of instances that ran into the time limit. Moreover, we use #bv and #v to refer to the average number of variables (=patterns) appearing in a basis solution and the entire column-generation process, respectively. We note that #bv may be larger than the number of items whenever DDOI columns had to be transformed into true patterns.

None of the AI/ANI instances comes even close to the time limit of one hour, i.e., column generation can be executed without any problems in any case. Nevertheless, we recognize a positive effect of the subset-sum procedure, since here only about half as large computation times occur and much fewer feasible patterns have to be dealt with (which is, at the same time, an indication of a smaller number of iterations in column generation). Once again, there is no significant difference between stabilized and unstabilized methods, since, as mentioned before, the optimal solution cannot contain DDOI columns. In the case of the GI instances, it

		Setting (1) refined reflect+			Setting (5) reduced cost from reflect arc-flow model		
Instances	#	overall time	time Level 4	#opt	overall time	time Level 4	#opt
AI201	50	54.3	23.5	6	62.7	31.5	6
AI402	50	1005.4	538.2	9	1217.7	745.6	9
ANI201	50	274.5	244.8	50	287.8	257.8	50
ANI402	50	2902.3	2655.7	1	2801.2	2568.7	1
GI125	80	44.6	31.5	1	21.5	8.6	1

Table 5: The two different ways to calculate the reduced costs in Level 4 of reflect+: comparison of instances solved optimal in Level 4

		Setting (1) refined reflect+				Setting (2) only greedy heuristic				Setting (3) non-stabilized column generation			
Instances	#	*time*	#tl	#bv	#v	*time*	#tl	#bv	#v	*time*	#tl	#bv	#v
AI201	50	7.9	0	170.5	9624.7	17.4	0	170.4	15188.6	8.5	0	170.5	9653.0
AI402	50	75.1	0	352.1	31993.5	143.5	0	352.2	38004.3	84.0	0	352.2	32092.5
ANI201	50	8.3	0	169.6	9506.7	18.3	0	169.6	15260.4	9.1	0	169.7	9574.5
ANI402	50	77.5	0	351.4	32077.6	152.0	0	351.5	39250.5	85.6	0	351.6	32256.5
GI125	80	28.9	0	130.8	3565.2	17.9	0	128.3	2319.4	28.8	0	125.0	3657.3
GI250	80	152.3	0	260.3	7988.3	104.3	0	256.3	5484.5	164.4	0	250.0	8352.7
GI500	80	673.1	0	512.2	18956.4	547.1	0	508.2	12554.4	802.8	0	500.0	20244.6
GI750	40	1005.8	0	785.1	32171.3	547.7	0	770.9	23216.9	1116.4	0	750.0	35841.7
GI1000	40	2323.5	6	1033.9	41074.4	1354.7	0	1022.4	33492.8	2552.9	17	1000.0	45014.5

Table 6: Comparison of different column-generation settings

is easy to see that both the subset-sum procedure and the absence of stabilization techniques lead to many more generated patterns. Therefore, the overall algorithm often runs into the time limit already in this preparatory step. Nevertheless, in combination with Table 4 it can be said for the set GI1000 that with the application of the subset-sum method, every instance whose column-generation procedure does not fail at the time limit could also be solved by the reflect+ algorithm in the end.

Next, we analyze the relevance of the different levels for reflect+. For that purpose, Table 7a contains information up to which level the reflect+ algorithm was run through for the different instances. We use #term to specify the number of instances that terminate at this level (solved or timed out), supplementary #sol refers to the number of instances that were solved on this last level. Concerning the AI set, most of the instances were solved to optimality on Level 2. Levels 3 and 4 were necessary to solve a few more instances and only for one instance the optimal solution could already be found on Level 1. Looking at AI402 isolated, it is noticeable that about half of the instances reaching Level 2 could not be solved on this level within the time limit. Due to the already mentioned integrality gap of at least one of all ANI instances, all instances of this class can be solved only (if at all) on the last level of the reflect+ algorithm. In contrast, a different picture emerges for the GI set: Nearly all instances are solved to optimality already on the first level. Only one instance of the subclasses GI125, GI500, and GI750 is solved exactly on Level 2. Concluding, the patterns from the solution of the column-generation procedure are highly relevant for the GI set. While the patterns from the linear relaxation of the pattern-based formulation are useless for the AI instances, the larger set of all patterns generated during the column-generation process seems to be beneficial for the subsequent levels of the reflect+ algorithm.

			Level 1		Level 2		Level 3		Level 4	
Instances	#	#opt	#term	#sol	#term	#sol	#term	#sol	#term	#sol
AI201	50	50	1	1	38	38	5	5	6	6
AI402	50	28	–	–	33	17	2	2	15	9
ANI201	50	50	–	–	–	–	–	–	50	50
ANI402	50	1	–	–	30	–	1	–	19	1
GI125	80	80	78	78	1	1	–	–	1	1
GI250	80	80	80	80	–	–	–	–	–	–
GI500	80	79	79	78	1	1	–	–	–	–
GI750	40	39	39	38	1	1	–	–	–	–
GI1000	40	34	34	34	–	–	–	–	–	–

(a) Default Setting (1): with pattern retransformation after column-generation process

			Level 1		Level 2		Level 3		Level 4	
Instances	#	#opt	#term	#sol	#term	#sol	#term	#sol	#term	#sol
GI125	80	80	78	78	–	–	1	1	1	1
GI500	80	79	79	78	–	–	1	1	–	–
GI750	40	38	39	38	–	–	1	–	–	–

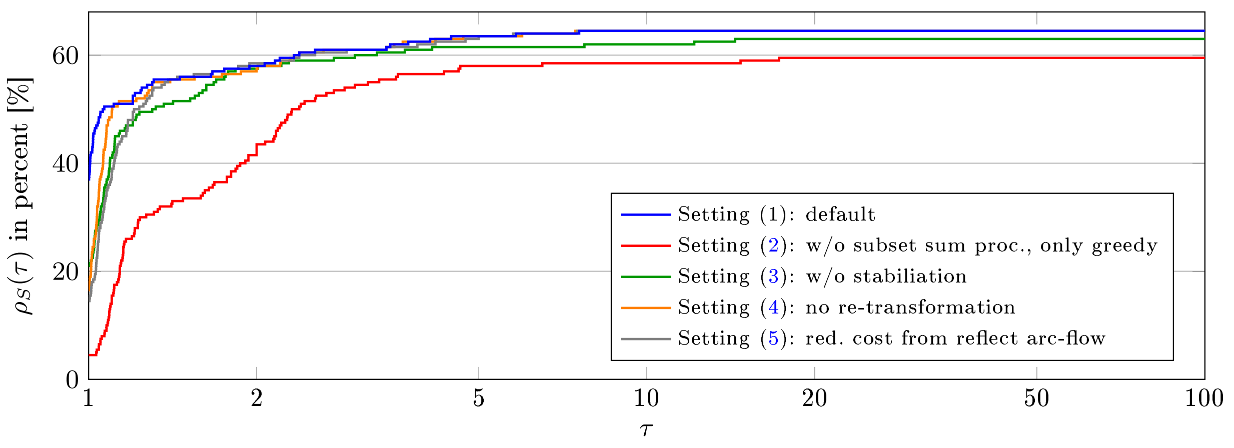
(b) Setting (4): without pattern retransformation after column-generation process (only rows with differing entries are displayed)

Table 7: Comparison of reached levels of refined reflect+ algorithm

As already mentioned before, it is important to transform a solution containing DDOI columns into a pure-pattern solution. A comparison of the data from Tables 7a and 7b provides a rationale for this: Table 7b is structured in the same way as the prior Table 7a and contains the data of Setting (4) from Table 4. For more clarity, there are only listed instance sets with instances that were solved to optimality on a different (later) level than in Table 7a with the refined reflect+. We can see the reason why reflect+ solved one instance more to optimality of subclass GI750 in comparison to Setting (4). While the transformed pure-patterns from the column-generation process suffice to build the optimal integer solution, the patterns of the column-generation solution including DDOI columns lost valuable information that is necessary for the optimal solution. Therefore this instance needs to run through the next Level 3 where the time limit is exceeded an no solution is found. In GI500 and GI750 are two additional instances with the same behaviour, but in these cases the instances can be solved successfully in Level 3. These examples do not lead to substantial improvements of the average computational time, but it is clearly a systematic benefit.

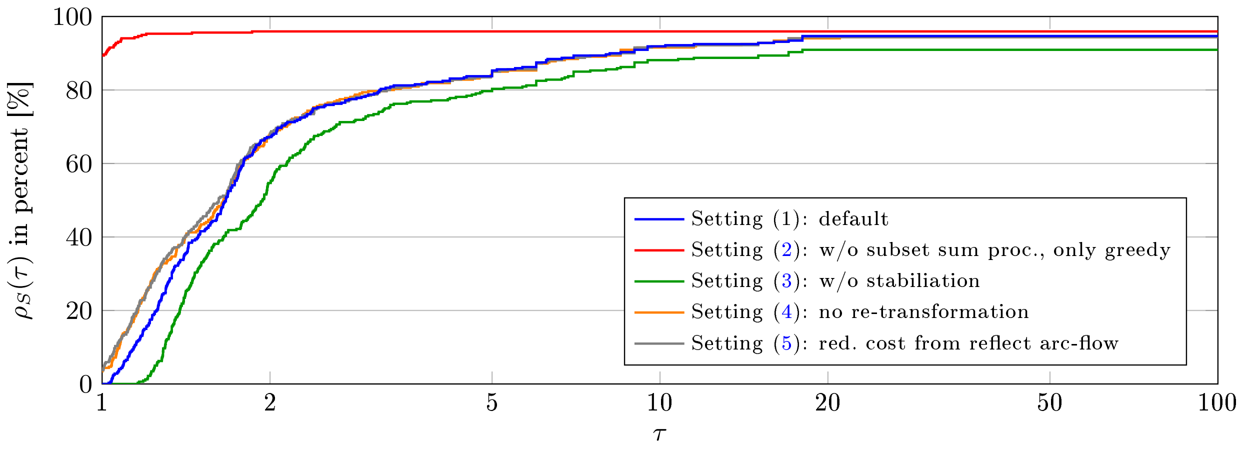
Finally, we compare the different settings with the help of performance profiles (Dolan and Moré, 2002). For a set of algorithms $\mathcal{S}$ (the five settings in our case), the performance profile $\rho_S(\tau)$ of an algorithm $S \in \mathcal{S}$ describes the ratio of instances that can be solved by S within a factor τ compared to the fastest algorithm, i.e. $\rho_S(\tau) = \left|\left\{I \in \mathcal{I} : t_I^S / t_I^* \leq \tau\right\}\right| / |\mathcal{I}|$ in which $\mathcal{I}$ is the set of instances, t_I^S is the computation time of algorithm S when applied to instance $I \in \mathcal{I}$, and t_I^* is the smallest computation time among all algorithms of set $\mathcal{S}$ for instance I. Unsolved instances are taken into account with $t_I^S = \infty$ (assuming also that $t_I^* = \infty$ gives $t_I^S / t_I^* = \infty$). In particular, the values of a profile at the two points $\tau = 1$ and $\tau = \infty$ can be interpreted well: $\rho_S(1)$ is the percentage of instances for which S is the fastest, and $\rho_S(\infty)$ is the percentage of instances that are solved by S within the time limit.

Figure 2 shows the performance profiles comparing the five settings separately for the AI/ANI and GI instances. Many of the above findings can be confirmed now. For the AI/ANI instances, see Figure 2a, the default setting of the reflect+ algorithm that integrates all four new algorithmic components is for 37% of



(a) For the AI/ANI instance

4



(b) For the GI instance

Figure 2: Performance profiles of the five different settings of the reflect+ algorithm; note the logarithmic scale of the τ-axis

the instances the fastest algorithm. Almost the entire profiles $\rho_{\text{Setting (1)}}(\tau)$ is above all other profiles of all other settings, characterizing the default reflect+ algorithm as the clear winner here. The subset-sum procedure is the most beneficial algorithmic component for the AI/ANI instances, as can be seen from the profile of Setting (2). Settings (3)–(5) are a little bit inferior to the default setting with only Setting (3) solving significantly fewer AI/ANI instances to proven optimality.

The relationship between the default setting and Setting (2) not using the subset-sum procedure is completely opposite for the GI instance, see Figure 2b. The reflect+ version without the subset-sum procedure shows a striking performance: It is the fastest algorithm for 89% of the GI instances. It is also superior regarding the instances solved exactly with the one-hour time limit, but the absolute number of optima is comparable to all other settings but Setting (3). This is what we already found in Table 4 and explained by the need of stabilizing the column-generation phase in particular for the large-scale GI instances.

6. Conclusions

In this article, we have introduced a refined reflect+ algorithm with algorithmic components tailored to the SSP. In general, this solution strategy is a multi-stage procedure, based largely on the fact that an optimal solution can often be found even if not all arcs of the ordinary reflect model are included, see (Delorme and Iori, 2020). Therefore, feasible points in thinned-out graphs are first determined, and then an attempt is made to prove their optimality using appropriate bounds. If this fails, an adapted (larger) network is considered in the next step. On the basis of extensive test calculations with different challenging benchmark classes, it can be shown that with this approach a large number of instances can be solved optimally for the first time. Within the context of these numerical investigations, we work out precisely the performance contributions of any of the refinements we have applied to the basic reflect+ algorithm. Thus, we succeed in obtaining valuable insights into the general numerical behavior of various instance classes and are able to derive concrete recommendations for their efficient practical treatment.

For future research, two main issues are particularly interesting:

(1) One effect observed during the computational evaluation remains unclear to us even after analyzing in detail the instance classes and their behaviour of reaching and passing the different levels of the reflect+ algorithm: While using the subset-sum procedure to equip the column-generation phase with columns of very good (exact) patterns, producing no or almost no loss, this component is clearly beneficial for the difficult AI/ANI benchmark, but rather harmful for the large-scale GI benchmark. Future research might find new ways to analyze and better understand this part of the column-generation process and its impact on subsequent phases of reflect+.

(2) The work of de Lima *et al.* (2022b) exploits pseudo-polynomial formulations in a innovative fashion presenting an approach using column generation and its reduced-cost information to define a kind of core subproblems solved separately with an ILP solver. This is possible because of a completely new branching scheme for the subproblems. Overall, the approach of de Lima *et al.* combines important techniques also used in reflect+ in a new way. It would be interesting to see results of this and other new approaches also for the challenging and AI/ANI and GI benchmarks.

References

Agoston, K. (2019). The effect of welding on the one-dimensional cutting-stock problem: The case of fixed firefighting systems in the construction industry. *Advances in Operations Research*. Article ID 6507054.

Alvim, A., Ribeiro, C., Glover, F., and Aloise, D. (2004). A hybrid improvement heuristic for the one-dimensional bin packing problem. *Journal of Heuristics*, **10**, 205–229.

Arbib, C., Di Iorio, C., Marinelli, F., and Rossi, F. (2002). Cutting and reuse: an application from automobile component manufacturing. *Operations Research*, **50**(6), 923–934.

Assmann, S. (1983). *Problems in discrete applied mathematics*. Ph.D. thesis, Mathematics Department, Massachusetts Institute of Technology.

Assmann, S., Johnson, D., Kleitman, D., and Leung, J. (1984). On a dual version of the one-dimensional bin packing problem. *Journal of Algorithms*, **5**, 502–525.

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

Ben Amor, H., Desrosiers, J., and Valério de Carvalho, J. (2006a). Dual-optimal inequalities for stabilized column generation. *Operations Research*, **54**(3), 454–463.

Ben Amor, H., Desrosiers, J., and Valério de Carvalho, J. M. (2006b). Dual-optimal inequalities for stabilized column generation. *Operations Research*, **54**(3), 454–463.

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

Caprara, A., Dell'Amico, M., Diaz Diaz, J., Iori, M., and Rizzi, R. (2015). Friendly bin packing instances without integer round-up property. *Mathematical Programming B*, **150**, 5–17.

Chen, Y., Song, X., Ouelhadj, D., and Cui, Y. (2019). A heuristic for the skiving and cutting stock problem in paper and plastic film industries. *International Transactions in Operational Research*, **26**(1), 157–179.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to algorithms*. The MIT Press, 3nd edition.

de Lima, V. L., Iori, M., and Miyazawa, F. K. (2021). Exact solution of network flow models with strong relaxations. Technical report, Institute of Computing, University of Campinas and DISMI, University of Modena and Reggio Emilia.

de Lima, V. L., Alves, C., Clautiaux, F., Iori, M., and de Carvalho, J. M. V. (2022a). Arc flow formulations based on dynamic programming: Theoretical foundations and applications. *European Journal of Operational Research*, **296**(1), 3–21.

de Lima, V. L., Iori, M., and Miyazawa, F. K. (2022b). Exact solution of network flow models with strong relaxations. *Mathematical Programming*. doi: https://doi.org/10.1007/s10107-022-01785-9.

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

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

Desaulniers, G., Desrosiers, J., and Solomon, M. M., editors (2005). *Column Generation*. Springer, New York, NY.

Dolan, E. D. and Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, **91**(2), 201–213.

Dyckhoff, H. (1981). A new linear programming approach to the cutting stock problem. *Operations Research*, **29**(6), 1092–1104.

Gilmore, P. C. and Gomory, R. E. (1961). A linear programming approach to the cutting-stock problem. *Operations Research*, **9**(6), 849–859.

Gilmore, P. C. and Gomory, R. E. (1963). A linear programming approach to the cutting stock problem - part ii. *Operations Research*, **11**(6), 863–888.

Gschwind, T. and Irnich, S. (2016). Dual inequalities for stabilized column generation revisited. *INFORMS Journal on Computing*, **28**(1), 175–194.

Heßler, K., Gschwind, T., and Irnich, S. (2018). Stabilized branch-and-price algorithms for vector packing problems. **271**(2), 401–419.

Irnich, S., Desaulniers, G., Desrosiers, J., and Hadjar, A. (2010). Path-reduced costs for eliminating arcs in routing and scheduling. *INFORMS Journal on Computing*, **22**(2), 297–313.

Johnson, M., Rennick, C., and Zak, E. (1997). Skiving addition to the cutting stock problem in the paper industry. *SIAM Review*, **39**(3), 472–483.

Kantorovich, L. V. (1960). Mathematical methods of organizing and planning production. *Management Science*, **6**(4), 366–422.

Kartak, V., Ripatti, A., Scheithauer, G., and Kurz, S. (2015). Minimal proper non-IRUP instances of the one-dimensional cutting stock problem. *Discrete Applied Mathematics*, **187**, 120–129.

Kłosowski, G., Kozłowski, E., and Gola, A. (2018). Integer linear programming in optimization of waste after cutting in the furniture manufacturing. In A. Burduk and D. Mazurkiewicz, editors, *Intelligent Systems in Production Engineering and Maintenance – ISPEM 2017*, pages 260–270, Cham. Springer International Publishing.

Labbé, M., Laporte, G., and Martello, S. (1995). An exact algorithm for the dual bin packing problem. *Operations Research Letters*, **17**, 9–18.

Martello, S., Pisinger, D., and Toth, P. (1999). Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management Science*, **45**(3), 414–424.

Martinovic, J. and Scheithauer, G. (2016a). Integer linear programming models for the skiving stock problem. *European Journal of Operational Research*, **251**(2), 356–368.

Martinovic, J. and Scheithauer, G. (2016b). Integer rounding and modified integer rounding for the skiving stock problem. *Discrete Optimization*, **21**, 118–130.

Martinovic, J. and Scheithauer, G. (2016c). The proper relaxation and the proper gap of the skiving stock problem. *Mathematical Methods of Operations Research*, **84**(3), 527–548.

Martinovic, J. and Scheithauer, G. (2018). Characterizing IRDP-instances of the skiving stock problem by means of polyhedral theory. *Optimization*, **67**(10), 1797–1817.

Martinovic, J., Jorswieck, E., Scheithauer, G., and Fischer, A. (2017). Integer linear programming formulations for cognitive radio resource allocation. *IEEE Wireless Communication Letters*, **6**(4), 494–497.

Martinovic, J., Scheithauer, G., and Valério de Carvalho, J. (2018). A comparative study of the arcflow model and the one-cut model for one-dimensional cutting stock problems. *European Journal of Operational Research*, **266**(2), 458–471.

Martinovic, J., Delorme, M., Iori, M., Scheithauer, G., and Strasdat, N. (2020). Improved flow-based formulations for the skiving stock problem. *Computers and Operations Research*, **113**, 104770.

Nemhauser, G. and Wolsey, L. (1988). *Integer and Combinatorial Optimization*. Wiley, New York.

Peeters, M. and Degraeve, Z. (2006). Branch-and-price algorithms for the dual bin packing and maximum cardinality bin packing problem. *European Journal of Operational Research*, **170**(2), 416–439.

Rao, M. (1976). On the cutting stock problem. *Journal of the Computer Society of India*, **7**, 35–39.

Scholl, A., Klein, R., and Jürgens, C. (1997). Bison: a fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers and Operations Research*, **24**(7), 627–645.

Shapiro, J. F. (1968). Dynamic programming algorithms for the integer programming problem—I: The integer programming problem viewed as a knapsack type problem. *Operations Research*, **16**(1), 103–121.

Stadtler, H. (1988). A comparison of two optimization procedures for 1- and 1 1/2-dimensional cutting stock problems. *Operations-Research-Spektrum*, **10**(2), 97–111.

Tragos, E., Zeadally, S., Fragkiadakis, A., and Siris, V. (2013). Spectrum assignment in cognitive radio networks: A comprehensive survey. *IEEE Communications Surveys and Tutorials*, **15**(3), 1108–1135.

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.

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

Valério de Carvalho, J. (2005). Using extra dual cuts to accelerate convergence in column generation. *INFORMS Journal on Computing*, **17**(2), 175–182.

Wang, D., Xiao, F., Zhou, L., and Liang, Z. (2020). Two-dimensional skiving and cutting stock problem with setup cost based on column-and-row generation. *European Journal of Operational Research*, **286**(2), 547–563.

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

Zak, E. J. (2003). The skiving stock problem as a counterpart of the cutting stock problem. *International Transactions in Operational Research*, **10**(6), 637–650.