

A Branch-and-Cut-and-Price Algorithm for Cutting Stock and Related Problems*

Renan F.F. da Silva¹, Rafael C.S. Schouery¹

¹Institute of Computing – University of Campinas (UNICAMP)
Campinas – SP – Brazil

{renan.silva@students.ic, rafael@ic}.unicamp.br

***Abstract.** In this project, we introduce a branch-and-cut-and-price framework to solve the Cutting Stock Problems with strong relaxations using the Set Covering (Packing) Formulations, which are solved through column generation. We propose an extended Ryan-Foster branching scheme tailored to non-binary models, a pricing algorithm that produces convergence in a few iterations, and a variable selection technique based on branching history. These strategies are combined with subset-row cuts and custom primal heuristics to create a framework that overcomes the current state-of-the-art of Cutting Stock Problem, Skiving Stock Problem, and other related problems, being at least twice faster in the first problem and at least 60% faster in the second one.*

1. Introduction

Consider a factory that receives orders for paper rolls of different sizes from various customers. To fulfill these orders, the factory cuts large rolls of paper. However, this cutting process may result in waste material, thus the goal is to minimize this waste. This optimization challenge is known as the Cutting Stock Problem (CSP), which is classified as NP-hard, meaning there is no polynomial algorithm to solve it optimally unless $P = NP$.

Given its significant practical relevance in Industry and its inherent computational complexity, the CSP has been extensively researched in the literature since the 1960s. In recent decades, advancements in CSP algorithms have largely relied on Integer Programming (IP), a technique capable of solving NP-hard problems within a reasonable runtime in practice. Two prominent IP formulations are the Set Covering and Set Packing Formulations (SCF/SPF), which typically involve an exponential number of columns and are tackled using branch-and-price frameworks.

This report presents a branch-and-cut-and-price framework to solve problems with strong linear relaxations using SCF/SPF, and the preprint paper version of this work is available online¹. We select the CSP as the main focus, highlighting specific properties that enhance the runtime of our algorithm for this problem. Additionally, we address similar problems like Skiving Stock Problem (SSP), Identical Parallel Machines Scheduling with Minimum Makespan problem (IPMS), Ordered Open-End Bin Packing Problem (OOEBPP), and Class-Constrained Bin Packing Problem (CCBPP). We overcome

*Supported by the São Paulo Research Foundation (FAPESP) grants #2015/11937-9; and the Brazilian National Council For Scientific and Technological Development (CNPq) grants #311039/2020-0 and #425340/2016-3.

¹The preprint paper version: <https://gitlab.com/renanfernandofranco/a-branch-and-cut-and-price-algorithm-for-cutting-stock-and-related-problems>

the state-of-the-art in all these studied problems. As presented later, our framework is at least twice and 60% faster than other CSP and SSP algorithms, respectively. Computational results for the last three problems are omitted from this report due to space constraints.

Formally, in the CSP, we have an unlimited number of stock rolls with length $W \in \mathbb{Z}_+$ and a set of items $I = \{1, \dots, n\}$, where each item $i \in I$ have size $w_i \in \mathbb{Z}_+$ and a demand $d_i \in \mathbb{Z}_+$. The objective is to cut the minimum number of stock rolls to satisfy all demands. A particular case of the CSP is the Bin Packing Problem (BPP), where $d_i = 1$ for all $i \in I$. The two main IP formulations for these problems are the SCF proposed by Gilmore and Gomory [Gilmore and Gomory 1961] and the arc-flow formulation (AFF) presented by Valério de Carvalho [Valério de Carvalho 1999].

In recent decades, many algorithms have been proposed for the CSP [Delorme and Iori 2020, Pessoa et al. 2020, Wei et al. 2020, Lima et al. 2023]. They all efficiently solve the benchmark instances from the BPP library [Delorme et al. 2018], except for two instance classes named AI and ANI. The only algorithm that successfully tackles these challenging classes is NF-F, proposed by Lima et al. [Lima et al. 2023], which uses a fast pricing algorithm combined with a sophisticated variable-fixing scheme for the AFF. This scheme reduces the formulation size drastically, allowing it to be solved by General IP Solvers.

Our framework uses Subset Row Cuts [Jepsen et al. 2008], custom primal heuristics, a strategy termed the *splay operation* to control the height of the branch-and-bound tree, an extended Ryan-Foster scheme [Foster and Ryan 1976], and insights to achieve a lean model and a column generation process with fast convergence, which is based on multiple pattern generation.

The remainder of this report is organized as follows. Sections 2 and 3 present the SCF for the CSP and an overview of our algorithm, respectively. Section 4 introduces the splay operation and a branching scheme that extends the Ryan-Foster scheme. Our pricing algorithm and a safe dual bound are introduced in Section 5. The cutting planes are presented in Section 6, and the waste and reduced cost optimizations to obtain a lean model are shown in Section 7. Moreover, we explain the implementation of two primal heuristics in Section 8. In Section 9, we briefly introduce related problems and present the computational experiments comparing our algorithm with other state-of-the-art algorithms. Finally, Section 10 concludes and suggests possible directions for future research.

2. Preliminaries

We tackle the CSP using the SCF, as presented by Gilmore and Gomory [Gilmore and Gomory 1961], which is a formulation with a strong linear relaxation for the CSP. Let p be a pattern, i.e., a subset of items such that $\sum_{i=1}^n a_i^p w_i \leq W$, where a_i^p is the number of copies of item i in p . Thus, the Master Problem of SCF is described as:

$$(M) \quad \text{minimize} \quad \sum_{p \in \mathcal{P}} \lambda_p \quad (1)$$

$$\text{subject to} \quad \sum_{p \in \mathcal{P}} a_i^p \lambda_p \geq d_i, \quad i = 1, \dots, n, \quad (2)$$

$$\lambda_p \in \mathbb{Z}_+, \quad \forall p \in \mathcal{P}. \quad (3)$$

where, and for each $p \in \mathcal{P}$, there is an integer variable λ^p , representing how many times pattern p is used. The objective is to minimize the number of patterns used. As the linear relaxation of this formulation can have an exponential number of patterns, it is commonly solved by column generation using a Restricted Linear Master (RLM) Problem, which works just with a subset of variables.

The formulation used for CSP, OOEBPP, and CCBPP is exactly the SCF presented above, while we employ the Set Packing Formulation for the SSP. For the IPMS, we leverage the relationship between this problem and the CSP, solving the IPMS by employing our CSP solver embedded in a binary search. Moreover, at the root node, all pricing sub-problems required by the aforementioned formulations can be solved in pseudo-polynomial time using dynamic programming.

3. Overview

Next, we present an overview of our framework, a branch-and-cut-and-price (B&C&P) algorithm based on SCF. Our framework's success hinges on primal heuristics, fast-increasing lower-bound techniques, and a fast relaxation solver. We present the pseudocode of our framework in Algorithm 1.

The algorithm begins by computing the volume bound ($\lceil (\sum_{i \in I} d_i w_i) / W \rceil$). Then, we set the root node as the current node, and each current node is solved by column generation, using a multiple pattern generation approach with a diversification strategy that converges in a few iterations.

In each node of the B&C&P, the RLM solution is strengthened using Subset-Row Cuts. We use the custom heuristics Relax-and-Fix (RF) and its variation, proposed by us, Constrained Relax-and-Fix (CRF), which are employed to tackle challenging instances like the AI class, proving competitive with general heuristics like those in the commercial solver Gurobi. Our B&C&P follows a depth-first order, prioritizing the left branch, with an adaptive branching scheme based on previous choices and enhanced with conflict propagation to break symmetries.

Algorithm 1: Our framework.

```

1 Compute volume bound
2 Run the initial heuristic to obtain the initial incumbent and columns
3 current node  $\leftarrow$  root node
4 while solution is not optimal do
5     Run CRF heuristic if its counter is reached
6     Run RF heuristic if its counter is reached
7     current node  $\leftarrow$  left child from the current node
8     while current node can be pruned by bound do
9         if it is possible to remove nodes using splay operation then
10            Remove nodes using the splay operation
11         else if there is an unexplored right node then
12            current node  $\leftarrow$  deepest unexplored right node
13         else
14            break

```

A pruning strategy called the “splay operation” is applied to eliminate potentially unfruitful nodes, thereby increasing efficiency. The algorithm halts upon detecting an optimal solution, which may occur early if the initial heuristic finds an optimal solution matching the volume bound. A lean restricted model, incorporating waste limitations and reduced costs, further accelerates computation, which is particularly beneficial for solving the class of instances ANI.

4. Branching Scheme

Branch-and-Bound (B&B) algorithms require a branching scheme, which is used to divide the search space, allowing the enumeration of all possibilities. Non-binary problems are typically addressed through branching on variables, which often leads to highly asymmetric branches. To mitigate this issue, we propose an extension of the Ryan-Foster scheme, previously employed solely for binary problems. Moreover, we also introduce the splay operation, which is a technique used to remove unfruitful nodes in the B&B tree.

4.1. Extended Ryan-Foster Scheme

The Ryan-Foster Scheme is a branching strategy for binary problems, i.e., that ones with unitary demands. Given an optimal solution $\bar{\lambda}$ of RLM, let δ_{ij} be the affinity between items i and j , where $\delta_{ij} = \sum_{p \in \mathcal{P}: \{i,j\} \in p} a_i^p \bar{a}_j^p \bar{\lambda}_p$, where $\bar{a}_j^p = a_j^p$ if $i \neq j$, and $\bar{a}_j^p = a_j^p - 1$ otherwise. For binary problems, Vance et al. [Vance et al. 1994] show a property that says that if $\bar{\lambda}$ is fractional, then there is an item pair (i, j) with $\delta_{ij} \notin \mathbb{Z}$. Then we can branch in this decision, fixing that items i and j are in the same pattern on the left branch and different patterns on the right branch.

In our work, we extend this scheme to non-binary problems. Firstly, even though the previous property is not true in general, it holds for all considered instances. Therefore, given (i, j) with $\delta_{ij} \notin \mathbb{Z}$, we branch as follows. The left branch assumes that at least one more pair of items i and j are together, i.e., the item demands of i and j decrease by one, and the demand of item k increases by one, where $w_k = w_i + w_j$. In contrast, the right branch assumes that any pair of items i and j are not together, adding a conflict between i and j . This conflict includes possible copies of i and j generated in the left branches of descendants of this node. The correctness of this branching scheme is supported by a lemma regarding conflict propagation, which we present in our paper.

Lastly, there may be multiple pairs of items we can branch; thus, we also propose a technique of selection based on the history of choices. Sometimes, we choose a pair of items (i, j) where neither left nor right branches can produce a solution better than the incumbent solution. Observe that (i, j) is a good candidate to improve the lower bound. So, our selection strategy is based on keeping all pairs of items with this property found in the previous branches on a priority list and branching in the best-ranked pair in this list.

4.2. Splay Operation

We explore the B&B Tree by performing a depth-first search (DFS). Sometimes, the incumbent solution is already optimal, and we just need to increase the lower bound to prove its optimality. A way to do it quickly is by finding good candidates (i, j) to branch, as mentioned in the previous section. But, in some cases, we perform a sequence of left branches before finding a good candidate (i, j) . One question that arises is that perhaps

(i, j) could prune both children even if it is processed before the left branches, which would avoid all the right branches required by these nodes to improve the lower bound.

With this in mind, we propose a strategy called *splay operation* to control the height of the B&B tree and prioritize these good candidates. This strategy is always used when we find a good candidate (i, j) to branch, and it consists of checking if we can move up (i, j) in the B&B tree by removing a non-empty sequence S of left branches. A detail is that S cannot induce negative item demands in any node of the resulting tree. If it is possible to find S with these properties, then we perform this operation and reprocess the node (i, j) . Otherwise, we proceed to the next unexplored node.

5. Column Generation

As our branching scheme adds conflicts to items, the pricing problem is the *Knapsack Problem with Conflicts*. Consider that the set of items I is a vector with d_i copies of each item i , where I is indexed from 1 to $N = |I|$, and $v(p)$ be the value of pattern p , that is, $v(p) = \sum_{i=1}^n a_i^p \bar{\pi}_i$. The aim is to find a pattern p with the smallest reduced cost $\bar{c}_p = 1 - v(p)$. One lower bound for this is given by the classic Dynamic Programming (DP) for the *Knapsack Problem*, described by the following recurrence:

$$f(i, r) = \begin{cases} 1, & \text{if } i = 0. \\ f(i - 1, r) & \text{if } i \geq 1 \text{ and } r < w_{I[i]}. \\ \min(f(i - 1, r), f(i - 1, r - w_{I[i]}) - \bar{\pi}_{I[i]}) & \text{if } i \geq 1 \text{ and } r \geq w_{I[i]}, \end{cases} \quad (4)$$

where i is the current item, and r is the wasted capacity in the pattern. The recurrence can be computed in $O(W \sum_{i \in I} d_i)$. If $f(N, W) \geq 0$, then $\bar{\pi}$ is dual feasible and, therefore, dual optimal. Otherwise, the result is inconclusive since we ignore the conflicts.

To arrive at a conclusive decision, we employ a branch-and-bound approach over the DP table, starting at $f(N, W)$, and constructing a partial pattern \bar{p} , initially set as \emptyset . If the pattern with the smallest reduced cost yields a negative cost, we return it. Otherwise, $\bar{\pi}$ is optimal. However, returning just one pattern produces a slow convergence, so we use an enumeration over the DP table that returns a set of patterns with negative reduced costs. The key idea to improve convergence looks to be the diversification of items. This strategy has already been used by Lima et al. [Lima et al. 2023], which finds, for each item i , the pattern with the smallest reduced cost that contains i .

We opted, instead, to focus on generating patterns using large items and a degree of diversification, based on the intuition that the slow convergence is due to the hardness of finding good patterns with these items. For this, we partition I into sets I_1 (items without conflicts) and I_2 (the remaining items), which are sorted in a non-decreasing order of size and concatenated in the presented order. We note that $|I_2|$ is often small, and if we are in a state (i, r) with a depth greater than $|I_2|$, then $f(i, r)$ is the exact solution for the unprocessed suffix and no branching is needed (since the set of items I_1 does not have conflicts). Therefore, with this ordering, the bottleneck of finding any pattern with negative reduced cost is to compute the dynamic programming table.

Thus, our strategy is to perform an enumeration beginning at $f(N, W)$, exploring all possible paths that can produce patterns with a negative reduced cost and storing the found patterns in a pool \mathcal{B} . To add diversification, we explore a path only if each item

$i \in \bar{p}$ (the partial pattern) appears fewer than ζ times in \mathcal{B} , where ζ is a constant. However, even with the last constraint, we could explore many paths. With this in mind, we also stop when $\mathcal{B} \neq \emptyset$ and the number of recursive calls N^R is greater than a constant N_{\max}^R .

Additionally, linear programming solvers typically employ floating-point precision and a tolerance parameter ϵ . Consequently, an infeasible solution is deemed feasible by the solver if each pattern exhibits a reduced cost \bar{c}_p greater than $-\epsilon$. Thus, column generation is halted upon this occurrence. In the preprint version of our paper, we illustrate how to convert such an infeasible dual solution into a feasible one in a numerically safe way, utilizing fixed-point precision and a technique presented by Lima et al. [Lima et al. 2023]. Finally, we show in our paper that the minimum tolerance ϵ permitted by commercial solvers proves inadequate for solving all instances studied. We propose a scaling method that preserves algorithmic correctness while reducing this tolerance.

6. Cutting Planes

In this work, we employ the weak Subset-Row (SR) Cuts of size 3. These cuts consist of inequalities in the form $\sum_{p \in \mathcal{P}} \lfloor \frac{1}{2} \sum_{i \in S} a_i^p \rfloor \lambda_p \leq \lfloor \frac{\sum_{i \in S} b_i}{2} \rfloor$, where S is a subset of three items with unitary demand. Derived from less-than inequalities of the Set Partition Formulation, these cuts can be added to SCF while retaining at least one optimal integer solution. Moreover, these cuts and their generalizations have seen widespread use in recent decades [Pessoa et al. 2020, Wei et al. 2020], especially in vehicle routing problems.

Given that the SR separation problem was proven NP-hard by Jepsen et al. [Jepsen et al. 2008], our separation algorithm relies on enumeration. Additionally, these cuts are not robust, i.e., they alter the pricing structure. However, this is not problematic, as even after adding cutting planes, the branch-and-bound strategy of our pricing algorithm remains effective. We observe that cutting planes can weaken the lower bound (4) more than adding conflicts. Hence, we prioritize processing items belonging to the cuts followed by those that do not belong but have conflicts.

7. Model Optimization

In the branch-and-price algorithm, it is essential to keep the RLM as small as possible since it needs re-optimization several times. Here, we introduce two strategies aimed at achieving this objective.

Waste Optimization: This strategy, utilized by Lima et al. [Lima et al. 2023], ensures that any improvement solution utilizes patterns with waste at most $(z(\bar{\lambda}_{\text{inc}}) - 1) \cdot W - \sum_{i \in I} d_i w_i$, where $z(\bar{\lambda}_{\text{inc}})$ denotes the incumbent solution value. Consequently, we generate and retain in the RLM only those patterns satisfying this constraint.

Model Cleaning by Reduced Cost: Reduced Cost Variable Fixing (RCVF) is a technique widely employed in literature, particularly effective in arc-flow models [Delorme and Iori 2020, Lima et al. 2023]. Let $\bar{\pi}$ be a feasible dual solution and $\bar{\lambda}_{\mathbb{Z}}$ be a feasible integer solution. Irnich et al. [Irnich et al. 2010] demonstrated that any pattern p such that $z(\bar{\pi}) + \bar{c}_p > z(\bar{\lambda}_{\mathbb{Z}}) - 1$ (where \bar{c}_p represents the reduced cost of p in $\bar{\pi}$) cannot belong to a primal integer solution with a value less than $\bar{\lambda}_{\mathbb{Z}}$. While prior works in the

literature apply this strategy to fix variables to zero in the root node, we extend it to any B&B node, ensuring that a variable fixed in a node remains valid only in its subtrees. Given that the master model lacks a polynomial size, instead of fixing these variables to zero, we simply remove these patterns from RLM, resulting in a leaner model that re-optimizes faster. This strategy serves as a pre-routine, alongside Waste Optimization, executed before optimizing any B&B node.

8. Primal Heuristics

Previous works have already employed custom heuristics for CSP in the literature, such as the heuristics Sequential Value Correction [Belov and Scheithauer 2006] and Limited Discrepancy Search [Pessoa et al. 2020, Wei et al. 2020]. However, these heuristics are not competitive with the current state-of-the-art for the problem [Lima et al. 2023]. Thus, we present below two heuristics called Relax-and-Fix and Constrained Relax-and-Fix.

8.1. Relax-and-Fix Heuristic

The *Relax-and-Fix* (RF) Heuristic [Belvaux and Wolsey 2000] is a diving heuristic that finds an integer solution using an interactive process that relaxes and fixes variables. In each step, it solves the relaxed model optimally and selects a percentage of variables $F \subseteq \mathcal{P}$ and fixes their values to an integer value. Moreover, as the gap between the lower bound $\bar{\pi}$ and the incumbent solution $\bar{\lambda}_{\text{inc}}$ is usually very tight, we propose a fixing strategy that is not too aggressive in order to achieve good results.

First, we consider that our RLM is a relaxation of a binary model. Thus, given a (fractional) solution $\bar{\lambda} \geq 0$, we split each variable $\bar{\lambda}_p > 0$ into $\lfloor \bar{\lambda}_p \rfloor$ patterns with value $\bar{\lambda}'_p = 1$ and one pattern with value $\bar{\lambda}'_p = \bar{\lambda}_p - \lfloor \bar{\lambda}_p \rfloor$. After selecting F , instead of fixing these variables by adding constraints, we remove them from RLM and add them to an initially empty partial solution S , adjusting the demands as necessary.

In each iteration, we generate patterns to solve RLM optimally and check if there are unfixed variables equal to 1. If so, we take F equal to these variables. Otherwise, we build F incrementally. Let $g = (z(\bar{\lambda}_{\text{inc}}) - 1) - z(\bar{\pi})$ be the gap of an improvement solution. We iterate over variables $p \in \bar{\lambda}'$ in non-increasing order of value, adding p to F only if $\bar{\lambda}'_p > 0.5$, $(1 - \bar{\lambda}'_p) \leq g$, and the number of over-covered items does not increase if p is added to S . If we add p to F , we do $g \leftarrow g - (1 - \bar{\lambda}'_p)$, and we always accept the first pattern to avoid an infinite loop. We repeat this process until S becomes a feasible solution. This heuristic runs every ten nodes.

8.2. Constrained Relax-and-Fix Heuristic

A heuristic that consistently employs a linear relaxation solution as a starting point, such as the RF, may prove ineffective when dealing with polyhedrons containing many fractional vertices with many fractional variables. In such instances, the need to fix numerous variables arises, increasing drastically the chance of making wrong choices. Therefore, we propose an enhanced version of the RF heuristic. Among all runs of the RF, let S_{inc} be the largest set of fixed variables such that the resulting relaxation allows improving the incumbent solution. The *Constrained Relax-and-Fix* (CRF) Heuristic involves executing the RF while adding the following constraint: $\sum_{p \in S_{\text{inc}}} \lambda_p \geq |S_{\text{inc}}| - k$, where k is a small integer. We temporarily remove all conflicts and unmerge all items to run this heuristic.

9. Computational Experiments

Next, we compare our framework with the state-of-the-art algorithms for the problems CSP and SSP. All results were obtained with a computer with processor Intel® Xeon® CPU E5-2630 v4 @ 2.20GHz with 64 GB of RAM, operational system Ubuntu 22.04.1 LTS (64 bits), and using the language C++17 with compiler GCC 11.3.0. We use the general solver Gurobi 9.5.2 executed in a *single thread* to solve the linear relaxation and use the time limit presented in the captions. In the next tables, the columns “total” and “opt” are the number of instances and the number of solved instances, respectively. The columns “time”, “cols”, and “cuts” give the averages of time (in seconds), generated columns, and generated cuts between all instances in the instance class, respectively.

9.1. Cutting Stock Problem

Table 1 presents a comparison of our algorithm with the state-of-the-art CSP algorithms NF-F [Lima et al. 2023], VRP Solver [Pessoa et al. 2020] and EXM [Wei et al. 2020], utilizing instances from the BPP Lib [Delorme et al. 2018]. Our algorithm demonstrates superior performance across all instances, excelling in both the number of instances solved and execution time. Particularly, it is at least two times faster than NF-F, in addition to solving 10 and 5 additional instances in AI and ANI classes, respectively. Moreover, our algorithm generates fewer patterns than EXM, indicating not only convergence in fewer iterations but also a more stable column generation process.

Table 1. Comparison with other algorithms for the CSP (time limit of 1h).

Class	Total	EXM			VRP Solver		NF-F		Our			
		Opt	Time	Cols	Opt	Time	Opt	Time	Opt	Time	Cols	Cuts
AI 202	50	50	4.2	2965.4	50	52.3	50	2.0	50	0.4	975.9	10.0
AI 403	50	46	398.1	9538.8	47	491.4	50	25.2	50	5.9	2516.9	21.9
AI 601	50	27	1759.6	16858.3	35	1454.1	49	192.4	50	84.0	4699.4	55.9
AI 802	50	15	2766.3	20008.1	28	2804.7	46	566.5	49	129.9	6628.5	62.4
AI 1003	50	2	3546.1	23664.5	-	-	36	1577.1	42	737.7	9357.6	110.7
ANI 201	50	50	13.9	3521.8	50	16.7	50	3.0	50	0.4	1005.8	7.6
ANI 402	50	47	436.2	9523.6	50	96.0	50	24.9	50	2.7	2396.2	9.1
ANI 600	50	0	3602.7	22213.7	3	3512.5	50	140.7	50	13.6	4078.2	18.1
ANI 801	50	0	3605.9	22188.8	0	3600.0	49	393.2	50	88.3	6039.3	38.0
ANI 1002	50	0	3637.7	23596.6	-	-	43	1302.5	47	460.5	8762.2	87.7
Falkenauer T	80	80	1.9	1087.7	80	16.0	80	0.3	80	0.09	511.4	1.0
Falkenauer U	80	80	3.8	1353.6	-	-	80	0.1	80	0.02	127.7	0.0
Hard	28	28	41.5	1909.8	28	17.0	28	23.6	28	11.8	861.8	31.4
Random	3840	3840	6.2	494.7	-	-	3840	0.9	3840	0.06	250.4	0.2
Scholl	1210	1210	5.0	572.0	-	-	1210	1.4	1210	0.05	130.2	1.9
Schwerin	200	200	0.3	429.9	-	-	200	0.2	200	0.03	150.5	0.5
Waescher	17	17	8.7	1979.3	-	-	17	161.2	17	0.2	387.4	2.5

9.2. Skiving Stock Problem

In the Skiving Stock Problem (SSP), we are given an integer W and a set of items $I = \{1, \dots, n\}$, where each item $i \in I$ has a frequency $f_i \in \mathbb{Z}_+$, and a size $w_i \in \mathbb{Z}_+$, where $w_i < W$. The objective is concatenating items from I to build the maximum number of stock rolls with a size of at least W , utilizing at most f_i of each item I . The SSP can be formulated using the Set Packing Formulation, and we can adapt all properties previously presented for the CSP to this problem. However, we did not implement the cutting planes and the CRF heuristic because our algorithm easily solved the studied benchmarks.

For the computational experiments, we used the benchmarks A1, A2, A3, and B [Martinovic et al. 2020], and the benchmarks GI, FalkenauerU/T, Hard, Scholl, Schwerin, Waescher from BPP Lib, which were extended by Korbacher et al. [Korbacher et al. 2023] generating new instances for GI class with 750 and 1000 items of different sizes. We compare our algorithm with state-of-the-art algorithms MDISS [Martinovic et al. 2020], NF-F [Lima et al. 2023], and KIMS [Korbacher et al. 2023], all of which utilize models based on AFF.

Table 2 showcases average times and the number of instances optimally solved for each algorithm. Our algorithm outperforms all previous algorithms, being between 60% up to dozens of times faster. Additionally, it is the first time that one algorithm solves all instances of classes A2 and A3.

Table 2. Comparison with other algorithms for the SSP (time limit of 1h).

Class	Total	MDISS		NF-F		KIMS		Our		
		Opt	Time	Opt	Time	Opt	Time	Opt	Time	Cols
A1	1260	1260	0.1	1260	0.1	–	–	1260	0.009	43.9
A2	1050	1011	250.9	1024	148.7	1023	137.3	1050	1.1	970.7
A3	600	–	–	–	–	575	171.2	600	1.2	567.1
B	160	126	970.7	160	61.4	160	38.7	160	2.7	1596.7
GI 125	80	40	1802.1	–	–	80	19.4	80	12.1	416.9
GI 250	80	40	1854.0	–	–	80	109.6	80	26.1	1076.2
GI 500	80	2	3584.2	–	–	79	596.9	80	68.4	2719.6
GI 750	40	–	–	–	–	39	664.2	40	183.9	5529.8
GI 1000	40	–	–	–	–	38	1570.3	39	346.0	8571.5
FalkenauerT	80	80	0.9	–	–	–	–	80	0.5	1771.7
FalkenauerU	80	80	0.1	–	–	–	–	80	0.03	171.2
Hard	28	28	3.0	–	–	–	–	28	0.3	854.1
Scholl	1210	1210	8.8	–	–	–	–	1210	0.2	256.5
Schwerin	200	200	0.2	–	–	–	–	200	0.07	168.1
Waescher	17	17	253.3	–	–	–	–	17	0.002	4.4

10. Conclusions

This study presents a framework for solving problems using SCF / SPF with strong relaxations. We demonstrate how to achieve convergence in column generation within a few iterations for the problems we investigate. This leads to significant performance improvements compared to other state-of-the-art algorithms, particularly in the case of SSP. Our primal heuristics are highly effective, often solving many instances at the root node or exploring only a few nodes. Additionally, we exploit the symmetry of the CSP and implement strategies to shrink the model, allowing us to successfully solve challenging instance classes such as AI and ANI.

In future research, these insights could yield promising results for addressing other problems that involve SCF / SPF, strong relaxations, and pseudo-polynomial pricing algorithms. Furthermore, the success of our algorithm in handling AI and ANI instances is strongly tied to the proximity of the relaxation polyhedron to the convex hull of integer solutions. Thus, instances lacking this property could serve as challenging benchmarks.

References

- Belov, G. and Scheithauer, G. (2006). A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. *European Journal of Operational Research*, 171(1):85–106.
- Belvaux, G. and Wolsey, L. A. (2000). bc — prod: A specialized branch-and-cut system for lot-sizing problems. *Management Science*, 46(5):724–738.
- 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. (2018). BPPLIB: a library for bin packing and cutting stock problems. *Optimization Letters*, 12(2):235–250.
- Foster, B. A. and Ryan, D. M. (1976). An integer programming approach to the vehicle scheduling problem. *Operational Research Quarterly (1970-1977)*, 27(2):367–384.
- Gilmore, P. C. and Gomory, R. E. (1961). A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859.
- 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.
- Jepsen, M., Petersen, B., Spoorendonk, S., and Pisinger, D. (2008). Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, 56(2):497–511.
- Korbacher, L., Irnich, S., Martinovic, J., and Strasdat, N. (2023). Solving the skiving stock problem by a combination of stabilized column generation and the reflect arc-flow model. *Discrete Applied Mathematics*, 334:145–162.
- Lima, V. L. d., Iori, M., and Miyazawa, F. K. (2023). Exact solution of network flow models with strong relaxations. *Mathematical Programming*, 197:813–846.
- 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.
- Pessoa, A., Sadykov, R., Uchoa, E., and Vanderbeck, F. (2020). A generic exact solver for vehicle routing and related problems. *Mathematical Programming*, 183(1):483–523.
- Valério de Carvalho, J. M. (1999). Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, 86:629–659.
- Vance, P. H., Barnhart, C., Johnson, E. L., and Nemhauser, G. L. (1994). Solving binary cutting stock problems by column generation and branch-and-bound. *Computational Optimization and Applications*, 3(2):111–130.
- Wei, L., Luo, Z., Baldacci, R., and Lim, A. (2020). A new branch-and-price-and-cut algorithm for one-dimensional bin-packing problems. *INFORMS Journal on Computing*, 32(2):428–443.