

INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
<http://pubsonline.informs.org>

Machine Learning-Supported Prediction of Dual Variables for the Cutting Stock Problem with an Application in Stabilized Column Generation

Sebastian Kraul , Markus Seizinger , Jens O. Brunner

To cite this article:

Sebastian Kraul , Markus Seizinger , Jens O. Brunner (2023) Machine Learning-Supported Prediction of Dual Variables for the Cutting Stock Problem with an Application in Stabilized Column Generation. INFORMS Journal on Computing

Published online in Articles in Advance 29 Mar 2023

. <https://doi.org/10.1287/ijoc.2023.1277>

Full terms and conditions of use: <https://pubsonline.informs.org/Publications/Librarians-Portal/PubsOnLine-Terms-and-Conditions>

This article may be used only for the purposes of research, teaching, and/or private study. Commercial use or systematic downloading (by robots or other automatic processes) is prohibited without explicit Publisher approval, unless otherwise noted. For more information, contact permissions@informs.org.

The Publisher does not warrant or guarantee the article's accuracy, completeness, merchantability, fitness for a particular purpose, or non-infringement. Descriptions of, or references to, products or publications, or inclusion of an advertisement in this article, neither constitutes nor implies a guarantee, endorsement, or support of claims made of that product, publication, or service.

Copyright © 2023 The Author(s)

Please scroll down for article—it is on subsequent pages



With 12,500 members from nearly 90 countries, INFORMS is the largest international association of operations research (O.R.) and analytics professionals and students. INFORMS provides unique networking and learning opportunities for individual professionals, and organizations of all types and sizes, to better understand and use O.R. and analytics tools and methods to transform strategic visions and achieve better outcomes.




For more information on INFORMS, its publications, membership, or meetings visit <http://www.informs.org>

Machine Learning–Supported Prediction of Dual Variables for the Cutting Stock Problem with an Application in Stabilized Column Generation

 Sebastian Kraul,^{a,*} Markus Seizinger,^b Jens O. Brunner^{b,c}

^aDepartment of Operations Analytics, Vrije Universiteit Amsterdam, 1081 HV Amsterdam, Netherlands; ^bHealth Care Operations/Health Informations Management, University of Augsburg, 86159 Augsburg, Germany; ^cDepartment of Technology, Management, and Economics, Technical University of Denmark, DK-2800 Kongens Lyngby, Denmark

*Corresponding author

Contact: s.kraul@vu.nl,  <https://orcid.org/0000-0002-8779-555X> (SK); markus.seizinger@uni-a.de,  <https://orcid.org/0000-0002-2574-8969> (MS); jens.brunner@uni-a.de,  <https://orcid.org/0000-0002-2700-4795> (JOB)

Received: March 1, 2022

Revised: September 1, 2022; December 31, 2022; January 18, 2023

Accepted: January 19, 2023

Published Online in *Articles in Advance*: March 29, 2023

<https://doi.org/10.1287/ijoc.2023.1277>

Copyright: © 2023 The Author(s)

Abstract. This article presents a prediction model of the optimal dual variables for the cutting stock problem. For this purpose, we first analyze the influence of different attributes on the optimal dual variables within an instance for the cutting stock problem. We apply and compare our predictions in a stabilization technique for column generation. In most studies, the parameters for stabilized column generation are determined by numerical tests, that is, the same problem is solved several times with different settings. We develop two learning algorithms that predict the best algorithm configuration based on the predicted optimal dual variables and thus omit the numerical study. Our extensive computational study shows the tradeoff between the learning algorithms using full and sparse instance information. We show that both algorithms can efficiently predict the optimal dual variables and dominate the common update mechanism in a generic stabilized column generation approach. Although the learning algorithm with full instance information is applicable when one has to solve the problem mainly for a fixed set of items, the algorithm with sparse instance information is applicable when there is more variability in the number of items between the different instances.

History: Accepted by Andrea Lodi, Area Editor for Design & Analysis of Algorithms–Discrete.



Open Access Statement: This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. You are free to download this work and share with others for any purpose, except commercially, if you distribute your contributions under the same license as the original, and you must attribute this work as “*INFORMS Journal on Computing*.” Copyright © 2023 The Author(s). <https://doi.org/10.1287/ijoc.2023.1277>, used under a Creative Commons Attribution License: <https://creativecommons.org/licenses/by-nc-sa/4.0/>.

Supplemental Material: The online appendix is available at <https://doi.org/10.1287/ijoc.2023.1277>.

Keywords: cutting stock problem • machine learning • stabilized column generation • parameter optimization • duality

1. Introduction

The cutting stock problem (CSP) is a generalization of the bin packing problem and one of the earliest problems that have been studied in the area of operations research (OR) (Kantorovich 1960). There are many applications from CSP such as pallet loading, packing, and industrial production planning, as well as computer operations and telecommunications (Lirov 1992, Cheng et al. 1994). The CSP is defined by a set of *items* $i = 1, \dots, m$ and a sufficiently large number of stocks with a *standard length* L . Each item i has a *demand* d_i and a *length* l_i . The problem is to find the minimum number of stocks to satisfy total demand. Different formulations of the CSP exist (Dyckhoff 1991). Because of the context of this paper, we will focus on the multiple-cut model formulation by Gilmore and Gomory (1961). This formulation is based on the generation of cutting patterns j , which are identified by a vector $(a_{1j}, a_{2j}, \dots, a_{mj})$ that is restricted by

$$\sum_{i=1}^m l_i a_{ij} \leq L \tag{1a}$$

$$a_{ij} \in \mathbb{Z}^+ \quad \forall i = 1, \dots, m. \tag{1b}$$

In this context, a_{ij} represents the number of times item i appears in pattern j . The total number of all cutting patterns satisfying (1a) and (1b) is given by n . This number may be very large but finite. Decision variable λ_j states the number of times cutting pattern j is used. Herewith, the integer programming model of the CSP can be formulated as follows:

$$\min \sum_{j=1}^n \lambda_j \quad (2a)$$

s.t.

$$\sum_{j=1}^n a_{ij} \lambda_j \geq d_i \quad \forall i = 1, \dots, m, \quad (2b)$$

$$\lambda_j \in \mathbb{Z}^+ \quad \forall j = 1, \dots, n. \quad (2c)$$

This formulation can also be considered as a result of a Dantzig-Wolfe decomposition (Dantzig and Wolfe 1960). The problem is often solved by column generation (CG) to avoid the enumeration of all cutting patterns, that is, $j = 1, \dots, n'$ and $n' \ll n$. In this context, Model (2) is known as the master problem and Model (1) as the subproblem. The subproblem is necessary because the number of cutting patterns n (extreme points) in the master problem (2) is generally large and only implicitly known in advance. Instead, one starts with a subset of variables λ_j and solves the linear relaxation, also known as the restricted master problem (RMP). The Dantzig-Wolfe formulation of the CSP does not have extreme rays because the subproblem (1) is bounded. Additionally, the convexity constraint for the extreme points is omitted because the number of stocks is part of the objective function (2a) and not fixed by the input (Barnhart et al. 1998). A common way of generating the starting patterns is to assign exactly one item i to one pattern j so that the algorithm starts with $n' = m$ patterns. The dual information of the RMP is needed to determine the existence of an unknown column that can improve the objective function value, that is, a column with negative reduced cost. The dual of the RMP can be stated as follows:

$$\max \sum_{i=1}^m d_i \pi_i \quad (3a)$$

s.t.

$$\sum_{i=1}^m a_{ij} \pi_i \leq 1 \quad \forall j = 1, \dots, n', \quad (3b)$$

$$\pi_i \geq 0 \quad \forall i = 1, \dots, m. \quad (3c)$$

The dual variables π_i correspond to Constraints (2b) of the primal problem. Constraints (3b) force the reduced cost $\bar{c}_j = 1 - \sum_{i=1}^m a_{ij} \pi_i$ of all columns j in the RMP to be nonnegative. At LP optimality of RMP, we have to prove that no absent columns with negative reduced cost exist. Therefore, we define a generic reduced cost function

$$\min 1 - \sum_{i=1}^m \pi_i a_{ij}, \quad (4)$$

and use it as objective function in the subproblem (1) to generate the most negative reduced cost column. However, any negative reduced cost column might be sufficient as in the final step of the algorithm we show that no such column exists. With this objective function, the subproblem is an unbounded knapsack problem where the values of the dual variables π_i refer to the profit of an item i . As long as the objective function value of the subproblem is negative, a new column is added to the RMP and both problems are solved again. Different implementations of this standard algorithm exist, that is, adding the most negative reduced cost column or the first negative reduced cost column as extremes. Adding more than one column is another choice. When the CG terminates, a lower bound of the CSP is found; that is, the LP relaxation of the MP is solved. To find the optimal solution, the CG needs to be embedded in a branch-and-price algorithm (Vance 1998). However, the lower bound given by the LP relaxation of Model (2) is known to be very tight. It is therefore of particular interest to solve the CSP's root node within the CG. Most CSP instances have the integer round-up property, meaning they have an absolute gap smaller than one. Nevertheless, not all instances have this property (Kartak et al. 2015). Scheithauer and Terno (1995) conjectured that all CSP instances have an absolute gap smaller than two, a property denoted as the modified integer round-up property. To the best of our knowledge, this conjecture is still open for the CSP.

CG is a well-studied technique, originally used for linear programs, that has been shown over the last three decades to be efficient for particularly large integer problems, that is, real-world problems (Lübbecke and Desrosiers 2005). However, CG has to deal with various problems like any other solution method, that is, heading-in, yo-yo, plateau, and tailing-off effect (Vanderbeck 2005). A well-known and extensively studied problem of CG is that the algorithms have instability problems for different applications (Gschwind and Irnich 2017). The values of the dual variables oscillate strongly before converging to the optimal values. One reason is that the RMP is degenerated, that is, the solution of the RMP consists only of some variables $\lambda_j > 0$ so that the basis of the RMP includes several other variables with a value of zero. The result is that a new column from the subproblem has little or no improving effect on the objective of the RMP. This behavior is known as the tailing-off effect (Lübbecke and Desrosiers 2005). To counteract this effect, it is necessary to take measures to stabilize the dual variables.

Stabilization techniques for CG are widely studied in the literature, as discussed in the next section. In general, problem-specific and generic approaches can be identified. Although the problem-specific approaches usually perform better for the respective problem, they can often only be applied to special cases. In this paper, we will focus on generic stabilization techniques and specifically on the approach of Du Merle et al. (1999). The different steps that need to be performed within a stabilized CG algorithm, that is, update mechanisms and parameter settings, are generally evaluated in extensive numerical tests, that is, the same instance is solved several times to find the best setting. From a practical point of view, this approach is not applicable. Instead, predefined routines are used, which perform well but do not exploit the full potential of the techniques. The information or knowledge about the (optimal) dual values of an instance plays an essential role in stabilization, as we will explain in more detail in Section 3.

The purpose of our paper is to develop a learning algorithm that can estimate the optimal dual values of instances for the CSP. The contribution of our paper is as follows: First, we develop two learning algorithms for the CSP, estimating the optimal dual values using full and sparse instance information. The learning algorithm with full information can be used particularly well if almost always planning the same number of items (m). The learning algorithm with sparse information, on the other hand, can be used well when the number of items varies between the different instances. Second, we analyze the effects of different parts of the instance information on the optimal dual values. We show a correlation between optimal dual values and relative item sizes within instances. In addition, we analyze the distribution of dual values with respect to the relative (sorted) position of an item. Third, we evaluate our learning algorithms in an extensive computational study with more than 100,000 different CSP instances of varying sizes. We use the instance generator by Gau and Wäscher (1995) to generate the different instances. Finally, we apply our learning algorithms in a stabilized CG approach and compare the results with a standard approach.

The remainder of this article is organized as follows. In Section 2, we analyze the literature concerning the different methodological topics. The mathematical formulation for the stabilized CSP and the learning algorithms derived from it are described in Section 3. We evaluate the performance of the learning algorithms and their application for the stabilized CG in a computational study in Section 4. The paper closes with a conclusion in Section 5, along with future research avenues.

2. Literature Review

Our literature review will focus on two different research aspects. First, we will analyze learning algorithms for combinatorial optimization problems and group our approach into this context. Second, we will review the literature focusing on stabilized CG.

2.1. Learning for Combinatorial Optimization Problems

The application of learning algorithms to combinatorial optimization problems is currently very much in research interest, and new findings and results are published regularly (Bertsimas and Kallus 2020, Modaresi et al. 2020, Bengio et al. 2021, Mazyavkina et al. 2021). Nevertheless, this research area is not entirely new. First papers appeared already in the 1980s (Smith 1999). The current literature can be classified into mainly three research areas following Bengio et al. (2021).

End-to-end learning solves the optimization problem directly by a learning algorithm, that is, the OR method is replaced. Current applications mainly investigate the traveling salesperson problem and its respective extensions. Bello et al. (2017) develop a reinforcement learning algorithm using a recurrent neural network to predict the distribution over different city permutations. They use the negative tour length as the reward function for the algorithm. Using instances for problems up to 100 nodes, they achieve close to optimal solutions. Nazari et al.

(2018) solve the vehicle routing problem using a reinforcement algorithm with feasibility rules. Their model is applicable for different instances with the same problem size. In their computational study, they compare their results with different other heuristics and Google's OR-Tools. For medium-sized instances, they outperform the other approaches.

Instead of solving the entire optimization problem, *algorithm configuration* supports the OR method with additional information, that is, a subproblem is solved via a learning algorithm. A problem already studied in great detail in this area is parameter tuning, such as for metaheuristics (Hoos 2011). Kruber et al. (2017) consider another problem in the area of algorithm configuration. They test different classifier algorithms to estimate whether a mathematical model is faster to solve in its compact or decomposed formulation. In their extensive study, they show that the k -nearest-neighbor algorithm performs best. A similar approach has also been taken by Bonami et al. (2018). They use a classifier algorithm to investigate whether a quadratic model can be solved more efficiently in the quadratic or linearized formulation. They show that the random forest or support vector machine approach performs better depending on the feature selection.

Machine learning alongside optimization algorithms is the third research area defined by Bengio et al. (2021). In contrast to *algorithm configuration*, this research area uses the same learning algorithm for one optimization problem several times with updated information over the iterations of the OR method. A promising application in this area is the choice of branching strategy within a branch-and-bound tree. Alvarez et al. (2017) train a learning algorithm to imitate the decision of a strong branching with a fast approximation function. Their computational study evaluates the developed learning algorithm on different the Mixed Integer Programming Library problems. Although the results are promising, the authors qualify the significance of the results by stating that strong branching is not automatically the best strategy for every problem type. In addition to branching, primal heuristics are also a promising area of application. Khalil et al. (2017) train a predictive learning algorithm to decide whether using a primal heuristic is promising in a given node within the branch-and-bound tree. Their approach improves the runtime of a state-of-the-art solver (CPLEX) by 6% on average and even by up to 60% for a special problem class. Václavík et al. (2018) support the solution process of a branch-and-price algorithm by developing a learning algorithm for the subproblem. In each iteration, the learning algorithm decides whether a new column with negative reduced costs exists or not. Their computational study shows that their approach reduces solution time by up to 40% on average. Morabit et al. (2021) develop a supervised learning approach that decides at each iteration in a column generation framework which of the new identified columns should be used; that is, a binary classification problem is solved. They train a graph neural network to solve the problem. They test their approach in an experimental study for different instances of a vehicle routing problem and a crew scheduling problem. The new approach can reduce the solution time by 20%–30%.

Our approach can be classified to the research area *algorithm configuration*. We train a learning algorithm to predict the optimal dual values for a CSP instance and then use this information for a solution algorithm to speed up the solution time. Thus, our problem can be seen as a multioutput regression. To the best of our knowledge, there is no other approach considering the dual information of a combinatorial optimization problem. Additionally, we could not find any paper solving a multioutput regression concerning a combinatorial optimization problem. Although a variety of different learning algorithms are used to solve the problems, a tendency toward deep learning techniques can be identified, which we also use in our approach.

2.2. Stabilized CG

As already mentioned in Section 1, CG also has to deal with various issues. Stabilization is one broad approach to tackle some of these issues. The most conventional approaches of stabilizing the dual variables can be divided into four categories, that is, *limitation*, *penalization*, *smoothing*, and *centralization* (Table 1).

One of the first and very intuitive approaches is the Boxstep method (Marsten et al. 1975). They force the dual variables to remain in a box around a stability center by adding appropriate restrictions (forming RMP_1^*). If the optimum of this modified RMP_1^* is assumed inside the box, it is also optimal for the original problem, and the procedure is finished. If the optimum lies on an edge of the box, this serves as the new stability center; that is, the box is shifted, and the resulting RMP_2^* is solved. Instead of a large master problem, the Boxstep method thus solves a finite sequence of RMPs. However, the size of the box must be chosen appropriately. Although the effort to solve the problems increases with their size, the number of required boxes decreases, but so does the stabilization effect. An implicit way of stabilization is by adding cutting planes to the dual RMP (Gschwind and Irnich 2017). These cutting planes are known as dual-optimal inequalities (DOIs) or deep dual-optimal inequalities (DDOIs) and can be added directly at the beginning of the CG or in any following iteration. DOIs restrict the feasible set of the dual of the RMP but not the optimal solutions. However, DDOIs even exclude some optimal solutions. Herewith, the oscillations of the dual variables are stabilized in a natural way, whereby the more restrictive the (D)DOIs are, the stronger the effect. Valério de

Table 1. Publications Classified According to Different Stabilization Techniques

Authors (year)	Centralization	Limitation	Penalization	Smoothing
Marsten et al. (1975)		X		
Goffin et al. (1992)	X			
Gondzio and Sarkissian (1996)	X			
Wentges (1997)				X
Du Merle et al. (1999)			X	
Neame (1999)				X
Valério de Carvalho (2005)		X		
Ben Amor et al. (2006)		X		
Ben Amor and Desrosiers (2006)			X	
Rousseau et al. (2007)	X			
Ben Amor et al. (2009)			X	
Lee and Park (2011)	X			
Brunner and Stolletz (2014)				X
Gschwind and Irnich (2017)		X		
Pessoa et al. (2018)			X	X

Carvalho (2005) and Ben Amor et al. (2006) develop (D)DOIs for the CSP and show a reduction in the number of iterations.

Another widely used approach is to stabilize the dual variables by *smoothing*. Instead of the calculated optimal dual variables, this stabilization technique passes smoothed variables to the pricing problem, that is, a convex combination of the calculated optimum and a stability center (Wentges 1997, Neame 1999, Brunner and Stolletz 2014). The main differences are the choice of the stability center and the update of the smoothing parameters, for example, static or dynamic. Pessoa et al. (2018) use an auto-adaptive approach in which the smoothing parameter is increased or decreased after each iteration depending on the information generated by the subproblem. Gondzio and Sarkissian (1996) are among the first using an interior-point method for solving the RMP. However, they do not solve the problem to optimality by using an abortion criterion. Herewith, a strictly feasible primal-dual solution can be calculated, which implicitly stabilizes the dual variables.

Rousseau et al. (2007) define a center for the dual variables by determining several extreme points in the RMP using different objective functions. The center is then calculated by a convex combination of these extreme points. Another *centralization* approach is the analytic-center-method (Goffin et al. 1992). The RMP is modified in a way that the analytic center of the dual solution space is calculated in each iteration and provided to the subproblem. The main advantage is that the center is unique for a strictly concave maximization problem and can be calculated as a barrier problem, that is, using Newton’s method. Instead of the analytic center, Lee and Park (2011) transform the idea of Chebyshev centers to a stabilization approach. They test their approach in different variations on several problem classes.

The idea of the Boxstep method is extended by Du Merle et al. (1999). They allow the dual variables to leave the box for a certain price. They integrate linear *penalization* terms in the objective function of the dual RMP that prevents the dual variables from moving too far away from the chosen stability center. Ben Amor and Desrosiers (2006) extend this approach by assuming a piecewise linear penalization term. They show in their computational study that the piecewise linear function affects both the runtime and the number of iterations in CG. In a follow-up paper, Ben Amor et al. (2009) show the effect on the runtime with different piecewise linear functions, that is, three and five segments. Here it is shown that five segments were always superior to the function with only three segments.

Each of the stabilization methods offers different strengths and weaknesses. One potential weakness, especially for generic use, is that many stabilization methods require specified parameters that affect runtime, that is, penalization weights or adjusted centers. Because of the increasing need to use CG as a generic approach, it is also of interest to deal with the appropriate choice of parameters for stabilization. Although general operations of the algorithms are described in the respective papers, to our knowledge, no work optimizes the potential of these methods by parameter tuning with machine learning. Our paper fills this gap.

3. Problem Development

This section first describes stabilized CG for the CSP following Du Merle et al. (1999). We then describe our learning algorithms and their advantages and disadvantages.

3.1. Stabilized CG for the CSP

In Section 2, we briefly described the stabilization approach of Du Merle et al. (1999). Because stabilization takes place in the dual space, we extend Model (3) accordingly. The dual variable π_i should be restricted. The parameter δ_i^- and δ_i^+ define the box's position (center) of each dual variable. As it is allowed to leave the box, decision variables η_i^- and η_i^+ are needed to measure the violation of dual value π_i , that is, the distance between the box and the dual value. This violation will be penalized in the objective function with associated coefficients ϵ_i^- and ϵ_i^+ , respectively. The dual perspective of the stabilized RMP for the CSP can then be formulated as follows:

Parameters: d_i , demand of item i ; a_{ij} , number of times item i appears in pattern j ; $\epsilon_i^-, \epsilon_i^+$, penalty weight of leaving the box for item i ; δ_i^-, δ_i^+ , position of the box for item i .

Decision variables: π_i , dual variable of Constraint (2b) for item i ; η_i^-, η_i^+ , penalty; distance of the π_i to the interval $[\delta_i^-, \delta_i^+]$.

$$\max \sum_{i=1}^m d_i \pi_i - \sum_{i=1}^m \epsilon_i^+ \eta_i^+ - \sum_{i=1}^m \epsilon_i^- \eta_i^- \quad (5a)$$

s.t.

$$\sum_{i=1}^m a_{ij} \pi_i \leq 1 \quad \forall j = 1, \dots, n', \quad (5b)$$

$$\pi_i - \eta_i^+ \leq \delta_i^+ \quad \forall i = 1, \dots, m, \quad (5c)$$

$$-\pi_i - \eta_i^- \leq -\delta_i^- \quad \forall i = 1, \dots, m, \quad (5d)$$

$$\pi_i, \eta_i^+, \eta_i^- \geq 0 \quad \forall i = 1, \dots, m. \quad (5e)$$

Objective function (5a) maximizes the dual variables concerning the demand of the items. However, in contrast to Objective (3a), the violations of leaving the box are subtracted. The reduced costs of a cutting pattern in Constraints (5b) are the same as in Constraints (3b); that is, the subproblem does not change. The box spans the interval from δ_i^- to δ_i^+ and is considered in Constraints (5c) and (5d). As already mentioned, it is possible to leave the box. Therefore, the variables η_i^+ and η_i^- are necessary, which are penalized with the coefficients ϵ_i^+ and ϵ_i^- in the objective function (5a).

Formulations (3) and (5) are different. However, they can converge to the same solution if the CG algorithm is adapted; that is, ϵ_i^- and ϵ_i^+ must become zero after a finite number of iterations, and δ_i^- and δ_i^+ are only updated if the new column of the subproblem has nonnegative reduced costs (Du Merle et al. 1999). It is therefore important that the parameters, especially those of the box, are chosen sensibly. Du Merle et al. (1999) suggest that the parameters δ_i^- and δ_i^+ are set to the value $\tilde{\pi}_i^{t-1}$, that is, dual values of the previous iteration. Nevertheless, they note that an estimator may better adjust the parameters. The best estimator is the dual value π_i^* of the final iteration when the algorithm terminates, that is, the optimal dual value. The open question is whether such an estimator exists. We answer this question and develop such an estimator by using a learning algorithm.

3.2. Estimator for the Optimal Dual Variables of the CSP

We assume that such an estimator exists and it is possible to train a learning algorithm to find valid δ -values for the stabilized CSP; that is, there exists a function $f: \mathbb{R}^k \rightarrow \mathbb{R}^m$. Let k be the number of known inputs (features) necessary to train the algorithm and m be the number of outputs, that is, the optimal dual values. The relationship between the inputs and outputs is nontrivial, although the dual values can be naturally derived from solving Model (2).

3.2.1. Dual Feasible Functions. The idea of using a function f that allows conclusions about the dual optimal solution is not completely new. Closely related to our topic are dual feasible functions. A function $f: [0, 1] \rightarrow [0, 1]$ is dual feasible if for any finite set S of nonnegative real numbers $\sum_{s \in S} x \leq 1 \Rightarrow \sum_{s \in S} f(x) \leq 1$ is true (Fekete and Schepers 2001). Applying this concept to Model (3), a function f is dual feasible if there is a feasible solution π such that $f(l_i/L) = \pi_i$ for any value l_i in $[1, L]$. Dual feasible functions are used to generate valid inequalities for integer programs and to compute lower bounds. An overview of dual feasible functions is given in Clautiaux et al. (2010). In the following, we will introduce two intuitive and good performing dual feasible functions, which we consider in the analysis of our computational study. Both were originally formulated by Fekete and Schepers (2001) and tested in several studies (Fekete and Schepers 2004, Alves and Valério de Carvalho 2008, Clautiaux et al. 2010). The first function f_0^l formalizes the procedure of Martello and Toth (1990) to derive the bin packing lower bound L2. The idea is to neglect all items smaller than a given threshold $\gamma \in [0, 1/2]$. All items greater than

$1 - \gamma$ are then considered larger for compensation purposes. The function is defined as follows:

$$f_0^\gamma : [0, 1] \rightarrow [0, 1],$$

$$\frac{l_i}{L} \rightarrow \begin{cases} 0, & \text{for } \frac{l_i}{L} < \gamma, \\ \frac{l_i}{L}, & \text{for } \gamma \leq \frac{l_i}{L} \leq 1 - \gamma, \\ 1, & \text{for } \frac{l_i}{L} > 1 - \gamma. \end{cases}$$

The second function f_1^γ is based on a special rounding procedure with $\gamma \in \mathbb{Z}$ (Clautiaux et al. 2010). The function is defined as follows:

$$f_1^\gamma : [0, 1] \rightarrow [0, 1],$$

$$\frac{l_i}{L} \rightarrow \begin{cases} \frac{l_i}{L}, & \text{for } \frac{l_i}{L}(\gamma + 1) \in \mathbb{Z}, \\ \frac{\lfloor \frac{l_i}{L}(\gamma + 1) \rfloor}{\gamma}, & \text{otherwise.} \end{cases}$$

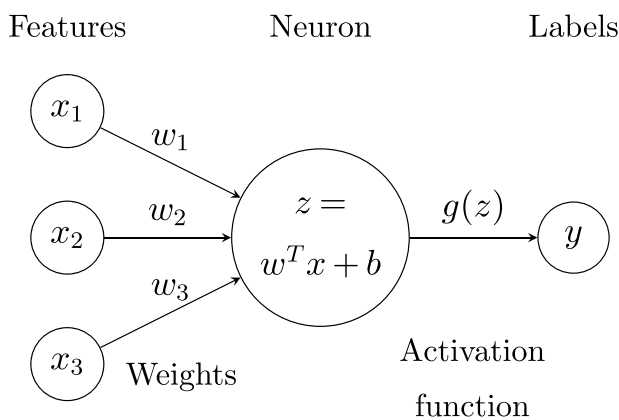
In contrast to dual feasible functions, we are not primarily interested in calculating a lower bound of Model (2). Of course, an estimate of the dual values can be used to determine an objective function value. However, it cannot be guaranteed that the estimated dual values are feasible, in the sense that they may not form a feasible dual solution, and so the objective function value may not be a valid lower bound. This happens, for example, when there is an overestimate of all the dual values of an instance.

3.2.2. Learning Algorithms. Estimating the dual values for the cutting stock problem can be seen as a multioutput regression (Borchani et al. 2015). Supervised learning is a well-suited method for understanding complex relationships based on provided data (Kotsiantis et al. 2007). A problem in multioutput regression is that for most methods the different outputs are predicted like using several single output predictors; that is, these methods do not leverage any possible relation between outputs. However, the field of deep learning has made significant progress in the last decades and offers completely new possibilities. Deep learning focuses on neural networks consisting of multiple layers between the input and output layers, called hidden layers. A comprehensive overview of deep learning can be found in Goodfellow et al. (2016) and LeCun et al. (2015). Before we introduce our two learning algorithms, we briefly describe the basic concepts of neural networks.

3.2.3. Neuron. The basic building block of any neural network is a neuron, which models the relationship between the inputs called *features* and the outputs called *labels*. Figure 1 shows a generic neuron with three features and one label. The input values x_1 , x_2 , and x_3 are mapped to the output value y . The number of inputs and outputs depends on the problem.

Two operations are performed to predict the label y . In the first step, the dot product of the feature vector x and the corresponding weight vector w is calculated, and a bias b is added. This follows the idea of a linear

Figure 1. Visualization of a Generic Neuron Mapping Three Inputs to One Output



regression. In a second step, an activation function $g(z)$ is applied to calculate the prediction of label y . Exactly this activation function $g(z)$ can introduce nonlinearity into the model. Nonlinearity is one of the main reasons for the better performance of neural networks compared with classical inference approaches like linear regression. Moreover, they can also be understood to map the results of computations to their natural ranges, for example, for probabilities to the interval between zero and one. Standard activation functions are sigmoid, ReLu, and tanh (Goodfellow et al. 2016).

3.2.4. Neural Network. The combination of several neurons can result in a neural network. Therefore, multiple neurons form an additional layer called the hidden layer. With one hidden layer, all neurons are connected to all features in the first layer (input layer) and all labels in the last layer (output layer). The labels in the output layer are fed with the outputs of the neurons in the previous layer. Thus, more possibilities for weighting the features are available, and more complex functions can be represented. Adding more hidden layers to a neural network can further increase the complexity of the represented function. However, adding additional layers to a neural network does not necessarily increase the prediction's performance. The *topology* of a neural network can then be described by the number of layers and neurons in each layer. Note that the problem already defines the input and output layers under consideration. Consequently, only the number of hidden layers and their number of neurons must be chosen. A common way to describe the topology of a neural network is to state the number of neurons in each hidden layer. For example, a neural network with three hidden layers and each with 10 neurons would be declared as 10 : 10 : 10.

The weights w (including the bias b) that the neural network uses on the different neurons to predict the labels are not known in advance, that is, the decision variables. Accordingly, the neural network must be trained to learn the relationship between inputs (features) and outputs (labels). During training, the neural network is shown many *samples*, that is, multiple feature instances and the corresponding labels. In our problem, the features are described by the instance information of the cutting stock problem, that is, stock length L , item length l_i , and item demand d_i . The labels correspond to the optimal dual values. To evaluate the performance of the prediction, a *loss function* $Loss(y, y^*, w)$ is used to measure the deviation between the predicted label y and the original labels y^* given weights w . Different loss functions can be used dependent on the type of problem, that is, classification or regression problems. The most common one for regression problems is the mean squared error (see Equation (6)). Independent of the type of loss function, the goal is to find weights w such that the value of the loss function is minimized, that is, $\min_w Loss(y, y^*, w)$.

$$Loss(y, y^*, w) = \frac{1}{N} \sum_{n=1}^N (y_n^* - y_n)^2 \quad (6)$$

The samples provided to the neural network need to be split to get an unbiased performance measure; that is, the neural network should not be evaluated on samples it has already seen. Therefore, the samples are split into three subsets. First, the *training set* is used to train or optimize the neural network. Second, the *validation set* evaluates the performance during the training of the neural networks and optimizes the hyperparameters. One important hyperparameter is the *learning rate* α , which influences the change of the weights w during the training iterations (see Equation (7)).

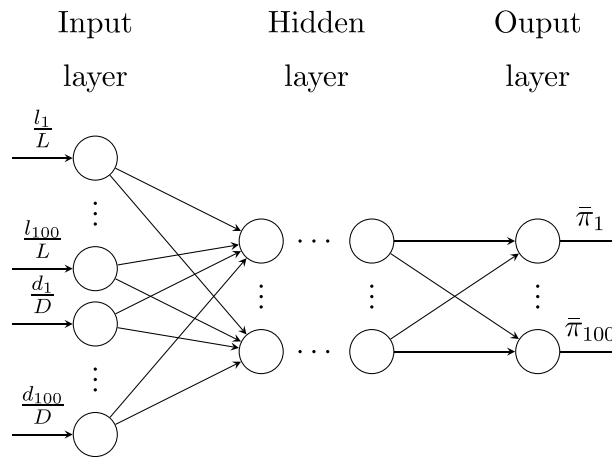
$$w := w - \alpha \frac{2}{N} \sum_{n=1}^N (y_n^* - y_n)x \quad (7)$$

Third, the *test set* is used after the optimization of the neural network to evaluate how well the network performs on unknown samples. By testing unknown data, we can make statements about the *generalizability* of the learning algorithm, that is, to what extent the algorithm can be used for data not yet considered. A common evaluation metric for a regression model is the R^2 (Glantz et al. 2016). The best possible score is one, meaning that the regression model explains 100% of the variability observed in the labels prediction. It is calculated by

$$R^2(y, y^*) = 1 - \frac{\sum_{n=1}^N (y_n^* - y_n)^2}{\sum_{n=1}^N (y_n^* - \bar{y})^2}, \quad (8)$$

where $\bar{y} = 1/N \sum_{n=1}^N y_n^*$. One problem that can occur during training is that the algorithm performs very well on the training data but poorly on the test data. This behavior is known as *overfitting* and can be mitigated by *regularization*. For this purpose, a penalty function is added to the loss function (6) using a *regularization term* ζ to control the penalization.

Figure 2. Visualization of the Neural Network with Full Information



As already mentioned, we estimate the optimal dual values based on the instance information, that is, stock length L , item length l_i , and item demand d_i . Nevertheless, this instance information can be used differently depending on the design of the learning algorithm. In the following, we assume that the instances are sorted in descending order of item size, that is, the largest item is at position 1. Next, we analyze a design with full and one with sparse instance information, to evaluate the instance’s value of information for predicting optimal dual values.

3.2.5. Full Information Learning Algorithm. We want to predict m values ($\bar{\pi}_i$) for the algorithm with full information, that is, the optimal dual value π_i^* for all items. Therefore, a multiregression model is needed. Full information in this context means that the learning algorithm receives the entire instance information as input. We want to use a network that can handle different instances. Therefore, we set a maximum instance size, which in our case is $m = 100$. Accordingly, the algorithm always predicts the values for the maximum of 100 items, although a smaller instance $m < 100$ is given as input. The remaining $100 - m$ items are modeled with size l_i and demand d_i equal to zero. Because instances that differ in length only by a constant factor have the same dual values, we use the relative item size l_i/L as input. The same applies to the demand for items, so we also consider the relative demand $\frac{d_i}{D}$ with $D = \sum_{i \in I} d_i$. Apart from these two dimensions, the network with full information does not need any further inputs. A visualization of our neural network with full information is given in Figure 2.

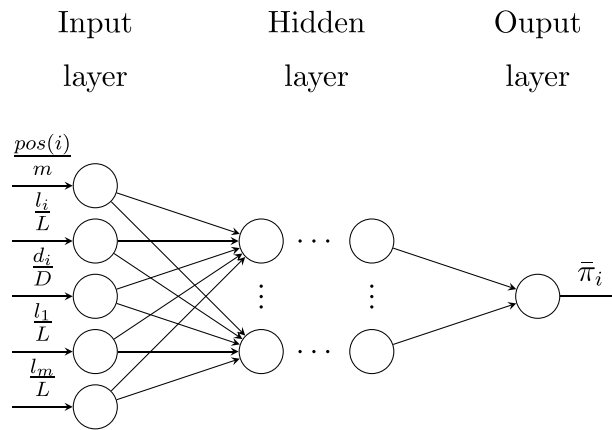
The advantage of this form of design is that the network can estimate the optimal dual values based on all the information from the instance. The most obvious disadvantage is that the network can only be used for instances up to the selected size. In our case, instances with more than 100 items could not be used on this network.

3.2.6. Sparse Information Learning Algorithm. The disadvantage of the network with full information should be overcome in the network with sparse information. However, the price for this is the loss of information. As with the previous network, we estimate the optimal dual values for all items of an instance. Though, this is not done in a single step. Instead, the network estimates the optimal dual value for a single item of an instance. For this purpose, we have to process the instance information differently. The input is the item’s relative position $pos(i)/m$ under consideration, relative size l_i/L , and relative demand d_i/D ; that is, the largest item of an instance will be on the relative position 1 with $pos(1) = m + 1 - 1 = m$. Here, $pos(i)$ can be seen as the index of item i if the items are sorted in ascending order of item size. In addition to these inputs, other instance information can be given to the network, such as the relative size of the largest l_1/L and smallest item l_m/L . A visualization of our neural network with sparse information is given in Figure 3.

The advantage of this design is that we can use any instance regardless of the number of items. The loss of information cannot be avoided entirely but can be compensated by additional inputs, that is, by including the nearest items as well. This design reflects the other extreme compared with the learning algorithm with full information.

4. Computational Results

In this section, the performance of the learning algorithms and the application in the stabilized CG is investigated. In the first step, we analyze the training and test set data to get a better understanding of the problem at hand. Second, we test the algorithms’ logic and their performance in comparison with a naïve approach, that is,

Figure 3. Visualization of the Neural Network with Sparse Information

a random generator. Afterward, we will test the application of our approach against the basic stabilization approach by Du Merle et al. (1999).

All computations are performed on a 1.9-GHz (Intel Core CPU i7-8650U) with 16 GB RAM running under the Windows 10 Enterprise operating system. The learning algorithms are coded in Python 3.9 using the scikit-learn package (Pedregosa et al. 2011). The technical setup for the algorithms in this study will be given in the following. We evaluated the network design in a prestudy with a parametric analysis to identify a reliable setting. The best results are as follows: For the full information network (FULL), the learning rate is set to $\alpha = 0.01$ and the regularization term to $\zeta = 10^{-6}$. The number of hidden layers is six with an equal size of 150, that is, the topology is 150 : 150 : 150 : 150 : 150. The setting for the sparse network (SPARSE) is quite different with a learning rate and a regularization term of 0.001. The number of hidden layers is two with an equal size of seven, that is, the topology is 7 : 7.

Gurobi 9.0 is used to solve all instances of the CSP with the CG algorithm, that is, the RMP and the subproblem (Gurobi Optimization, LLC 2021). The default settings of Gurobi are used. We generated and solved 100,000 different instances of the CSP with varying sizes to optimality using CG without stabilization to receive the optimal dual values for the training, validation, and testing of the learning algorithm. These instances are only a small fraction of all possible combinations for our setting. For the generation, we used the random instance generator developed by Gau and Wäscher (1995). The settings for the random instance generator are as follows and summarized in Table 2: The number of items m and the standard length of a stock L is uniformly distributed in $\mathcal{U}\{50, 100\}$ and $\mathcal{U}\{300, 1000\}$, respectively. The lower and upper bound for the relative size of length l_i in relation to L is uniformly distributed in $\mathcal{U}\{0.05, 0.45\}$ and $\mathcal{U}\{0.5, 0.85\}$. The average demand is set to 10 and uniformly distributed between the various demands. All our problem data are available on GitHub (Kraul 2023).

4.1. Analysis of the CSP Data

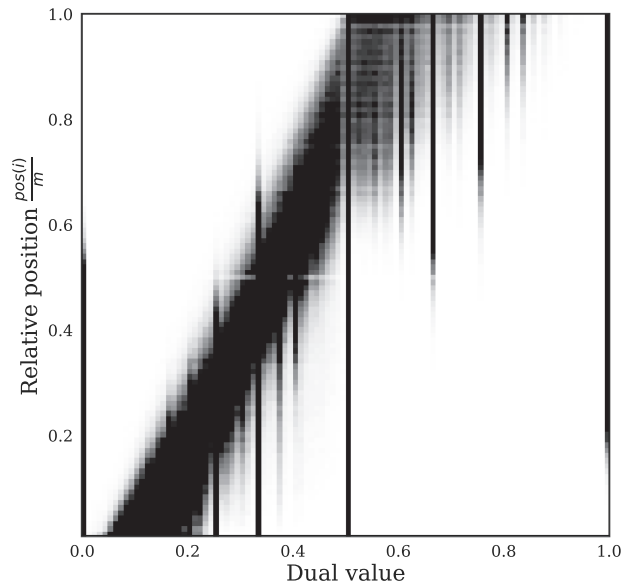
The CSP is already well studied, and some properties are known. From theory, we know that the optimal dual values in a CSP are decreasing by item size, that is, $\pi_i^* \geq \pi_h^*$ if $l_i \geq l_h$ for any item $i, h \in \{1, \dots, m\}$. Additionally, the dual values must range between zero and one. Although the optimal dual value is unknown, stronger bounds can be defined depending on the relative item size l_i/L . Caprara et al. (2005) proved that the following bounds are valid for any maximal dual solution of Model (3):

$$\begin{aligned}
 0 \leq \pi_i &\leq \frac{1}{\lfloor L/l_i \rfloor} && \text{for } 0 < l_i < \frac{L}{2}, \\
 \pi_i &= \frac{1}{2} && \text{for } l_i = \frac{L}{2}, \\
 1 - \frac{1}{\lfloor L/(L-l_i) \rfloor} &\leq \pi_i \leq 1 && \text{for } \frac{L}{2} < l_i < L, \\
 \pi_i &= 1 && \text{for } l_i = L.
 \end{aligned}$$

Table 2. Summary of the Inputs for the Instance Generator

Items (m)	Stock length (L)	Lower bound (l_i)	Upper bound (l_i)	Average demand \bar{d}
$\mathcal{U}\{50, 100\}$	$\mathcal{U}\{300, 1000\}$	$\mathcal{U}\{0.05, 0.45\}$	$\mathcal{U}\{0.5, 0.85\}$	10

Figure 4. Density Between the Optimal Dual Value and the Relative Position of Items



Interestingly, for items i with a length of $l_i = L/2$ the dual value is exactly $1/2$, and for items i with a length of the stock size $l_i = L$ the dual value is exactly 1 (Clautiaux et al. 2011).

For our analysis, we consider the information from the results of the 100,000 instances in detail. In Figure 4, we look at the distribution of the optimal dual values to the relative position of the items across all 100,000 instances. The more often items with the same relative position have the same dual value, the darker the region in Figure 4. The property mentioned previously of descending dual values with item size can also be seen across all instances, that is, the closer the relative position is to one (largest item), the stronger the expression toward high dual values. Furthermore, it can be seen that the dual value of 0.5 is special. Items of any relative position can have this value. The dual value of zero and one is also remarkable. Interestingly, the density has the same distribution for instances with 100 items only. The entire relationship only becomes apparent when Figure 5 is considered. The graph is constructed in the same way as Figure 4.

However, it analyzes the relative size instead of the relative position. The same graph is plotted twice, but the lower and upper bounds by Caprara et al. (2005) are highlighted in the right graph. The graph splits into two parts at $(0.5, 0.5)$ as stated by the lower and upper bounds of Caprara et al. (2005) at this section's beginning. All

Figure 5. (Color online) Density Between the Optimal Dual Value and the Relative Size of Items (Left), Including Lower and Upper Bounds by Caprara et al. (2005) (Right)

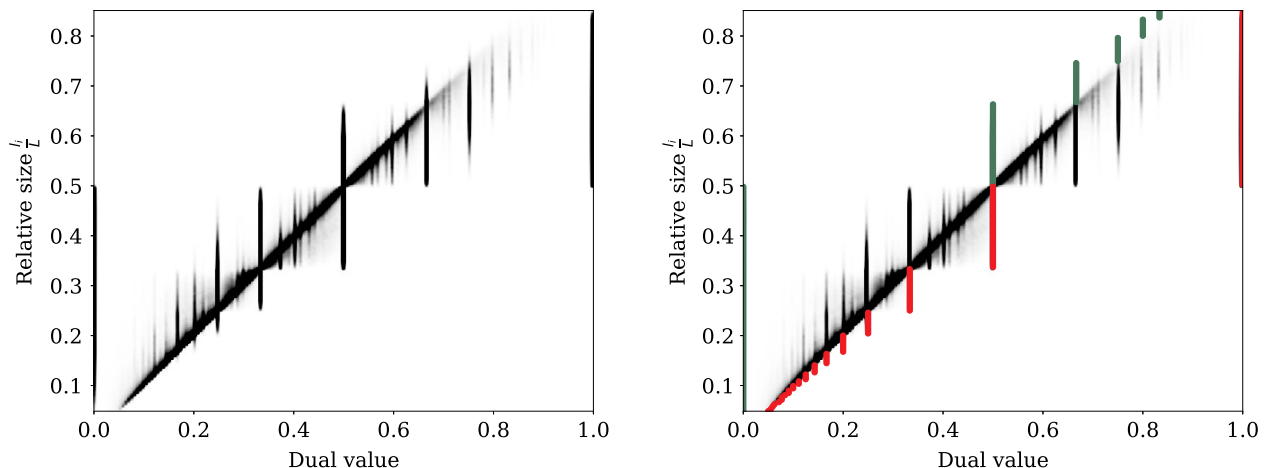
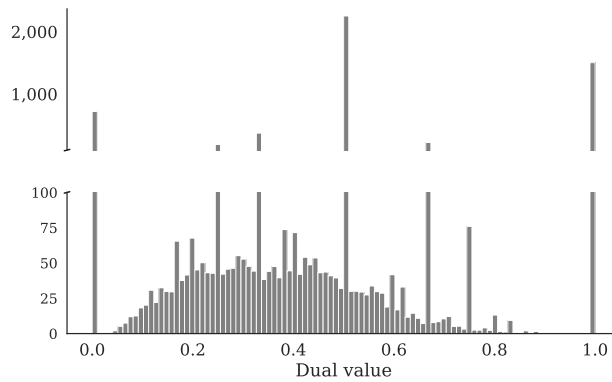


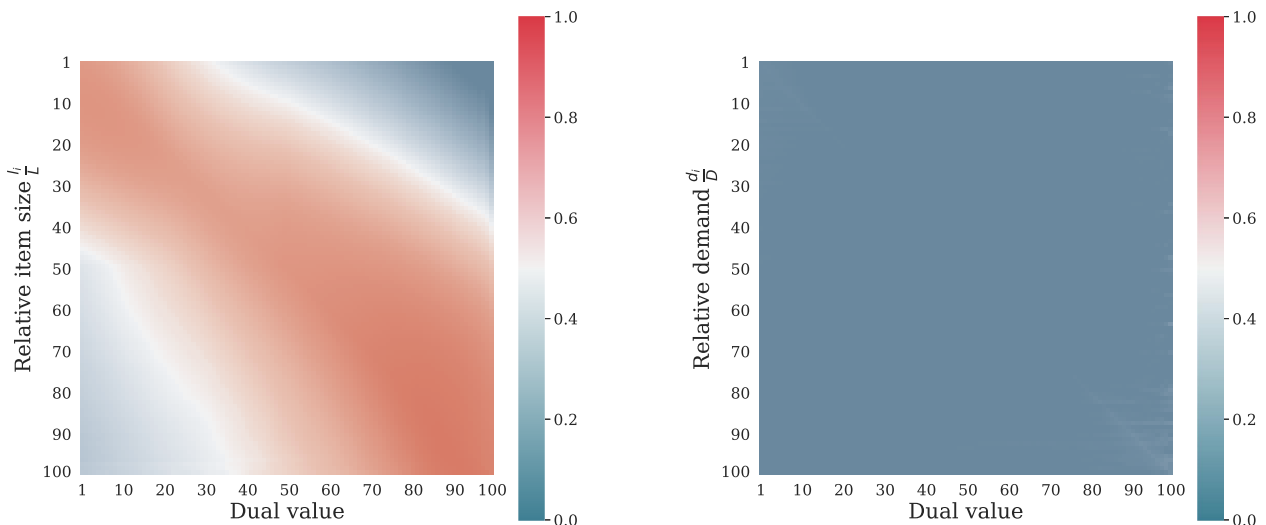
Figure 6. Frequency (in Thousands) of Optimal Dual Values in the Data Set (100,000 Instances)



items smaller than half the stock length L have an optimal dual value of at most 0.5. Likewise, all items larger than half the stock length L have an optimal dual value of at least 0.5. An optimal dual value of exactly 0.5 is only achieved by items between a length of one-third and two-thirds of the stock length. The reason for this can be deduced from the bounds of the dual values. In addition, the optimal dual value often takes a value close to its relative item size l_i/L . This behavior also corresponds to the concept of the Fekete and Schepers (2004) dual feasible function f_0^γ , which we introduced in Section 3.2. When γ is small enough, the function corresponds to the identity function. Looking at the right graph of Figure 5 in more detail, one can see that for items smaller than $L/2$ the dual values often take the values of the different upper bounds of smaller items. The same applies to items larger than $L/2$ but here for the lower bounds of larger items. The dual feasible function f_1^γ can also reconstruct this property. Here, depending on the selected γ , the steps can be set at similar positions but also considering items of larger size. This relationship becomes even more evident if we look exclusively at the distribution of dual values (Figure 6). There are peaks in decreasing order at $1/2, 1, 0, 1/3, 2/3, 1/4$, and $3/4$. All these peaks correspond to the bounds. Remember that for items with $l_i = L/2$ the dual value is $1/2$.

Because the learning algorithms are essentially driven by relative size and demand, we consider the correlation between these and the dual values in Figure 7. This graph is to be read differently than the previous ones. Each row/column describes items of the same order. For instance, the first row in the left part of the figure visualizes the correlation of the relative size of the largest item (l_1/L) to all 100 optimal dual values, that is, 100 items. Only linear correlation is visible based on the samples of the 100,000 instances of the data set. Interestingly, there is no visible correlation between the optimal dual values π_i^* and the relative demand d_i/D . However, there is a correlation between the respective dual value and the others for the relative item size. On average, the relative items

Figure 7. (Color online) Correlation Between Optimal Dual Values and Relative Item Size (Left)/Demand (Right)



size of one item has a correlation bigger than 0.7 for 47.94 dual values with a minimum of 16 and a maximum of 67. Therefore, we can deduce that the learning algorithm FULL can especially benefit from the different relative item sizes given as input.

4.2. Performance of the Learning Algorithms

The two learning algorithms are trained for the CSP with random samples between 50 and 100 items. The goal is that the algorithms can estimate the optimal dual values π_i^* using the information defined by the instance. The inputs for FULL are the relative item size (l_i/L) and the relative demand d_i/D for the maximum of 100 items (Figure 2). As a result, the size of the input layer is 200. The input layer's size will increase linearly with the number of items, that is, two neurons per item. Additionally, the size of the output layer is 100, that is, the dual value corresponding to Constraints (2b) or one neuron per item. The inputs for SPARSE are as described in Section 3. Consequently, the size of the input layer is five and of the output layer is one. In contrast to FULL, this network does not change for any instance size.

4.2.1. Explanatory Power. As a first step, we analyze the performance of the networks concerning smaller data sets. To train the learning algorithms, we randomly split our data set in a ratio of 4 : 1, that is, 80,000 instances for training/validation and 20,000 instances for testing. We then further divide the training set four times by 90% each; that is, we have a training set with 80,000, 8,000, 800, 80, and 8 instances. Especially for FULL, it can be beneficial if only a small amount of training is required because specialized networks could then be used for the respective instance size. Therefore, we want to measure how well our learning algorithm explains the observed data. As described in Section 3.2, the R^2 is a common way to measure the performance of a regression model. The results are given in Table 3. Both algorithms fit well for the complete training set with an $R^2 = 0.71$ for FULL and $R^2 = 0.82$ for SPARSE. However, the smaller the training sets become, the lower the explanatory power becomes. Although the R^2 value for FULL decreases very strongly, it remains high for SPARSE even with only eight instances in the training set. Nevertheless, one should not rely on the R^2 alone. We will analyze the learning algorithms trained with the entire training set for the rest of the study, that is, 80,000 instances.

4.2.2. Pattern Accuracy. In a second study, we investigate whether the predicted values of the networks follow the patterns described in Section 4.1, that is, $\pi_i^* \geq \pi_{i+1}^*$ if $l_i \geq l_{i+1}$, and $\pi_i^* \in [0, 1]$. Remember that the items in each instance are sorted by size, that is, item $i - 1$ is at least as large as item i . Additionally, we check if the predicted values are within the bounds derived by Caprara et al. (2005). We use this analysis to see further how well the algorithms can make a good prediction of the optimal dual variables by using problem-specific information to evaluate them. Table 4 shows the total number of violations of the above three conditions using different accuracies for the test set, that is, 20,000 instances with more than 1.5 million predictions. An accuracy of 10^{-2} is sufficient to achieve good results for stabilized CG. Obviously, the number of violations decreases with greater accuracy for both learning algorithms. However, the performance of the two characteristics is quite different. FULL can represent the relation between dual values ($\bar{\pi}_i \neq \bar{\pi}_{i-1}$) better than SPARSE, that is, 1% versus 10% of all predictions with an accuracy of 10^{-2} . This is also evident because SPARSE does not receive any additional knowledge about the other dual values of the instance. There is an equivalent behavior for the range of the dual values ($\bar{\pi}_i \notin [0, 1]$). However, although FULL leaves the range equally for both sides, SPARSE has almost exclusively outliers greater than one. Nevertheless, there is no value smaller than -0.1 or greater than 1.1 . Although we know that this violates the CSP, it can be tolerated as the consequences are very small or nonexistent for our application. One way to deal with this problem is by using a mask as postprocessing where the values are set to zero or one, and the order is restored as we did for the following study (Nazari et al. 2018). A different behavior can be seen when considering the bounds by Caprara et al. (2005). SPARSE is much better at keeping the predictions in bounds, that is, 5% versus 16% of all predictions with an accuracy of 10^{-2} . Adjusting the predictions for $\bar{\pi}_i \notin [0, 1]$, the result for SPARSE becomes even better.

Table 3. R^2 of the Learning Algorithms Using Different Training Set Sizes

No. of instances	FULL	SPARSE
80,000	0.71	0.82
8,000	0.67	0.81
800	0.53	0.80
80	0.29	0.78
8	-0.02	0.74

Table 4. Total Number of Violations in the Predicted Test Data with Different Accuracy

Accuracy	FULL			SPARSE		
	$\bar{\pi}_i \neq \bar{\pi}_{i-1}$	$\bar{\pi}_i \notin [0, 1]$	Bound	$\bar{\pi}_i \neq \bar{\pi}_{i-1}$	$\bar{\pi}_i \notin [0, 1]$	Bound
10^{-6}	171,846	28,567	331,492	426,968	35,047	118,605
10^{-4}	167,668	28,241	330,552	422,432	34,836	118,107
10^{-2}	12,269	8,336	236,939	144,799	18,295	76,370

4.2.3. Naïve Validation. The third study on the performance of the learning algorithms compares the prediction of the networks with the naïve approach of random draws (RDs) for the optimal dual values. For this purpose, we generate 100 new instances of each of the sizes 100, 75, 50, and 25. The problem size 25 is one on which both networks were not trained and allow additional statements about generalizability. The results are given in Table 5. We highlight the best result in bold. For each instance, we consider how often the optimal dual value is overestimated or underestimated on average. This shows that both networks tend to underestimate for the sizes 25, 50, and 75. FULL achieves a more balanced ratio for instances with 100 items. Remember that the network is explicitly designed for instances with 100 items. As expected, RD achieves the same ratio of over- and underestimations in all instances. In addition, we consider the average deviations of the estimated dual values from the optimal dual value for all instances, that is, $\sum[(\bar{\pi}_i - \pi_i^*)]/[100 | I |]$, where $\bar{\pi}_i$ is the predicted optimal dual value. Here, SPARSE and RD perform better for all problem sizes than FULL. However, what is ignored is that in the deviation across an instance, the overestimates and underestimates can cancel out. For this reason, we have also calculated the absolute deviation, that is, $\sum[|\bar{\pi}_i - \pi_i^*|]/[100 | I |]$. We can see that both networks perform significantly better than RD when estimating the optimal dual values, that is, RD misestimate dual values by 0.34 on average. However, there are differences when comparing the two networks. SPARSE performs consistently well for all problem sizes. On the other hand, FULL performs significantly worse in estimating the optimal dual values for the unknown problem size 25. Consequently, in terms of generalizability SPARSE is better. For the size of 100 items, however, FULL performs better. This is evident because the network overestimates and underestimates in equal proportions.

4.2.4. Problem-Specific Validation. In our fourth and final study of learning algorithm performance, we analyze how close the predictions of the two learning algorithms are to the optimal solution at the end of CG. We compare the results with the best dual values obtained from the dual feasible function f_0^* by Fekete and Schepers (2001); that is, we use these dual values in the CG in the same way as the predicted values. In detail, we compare the Lagrangian bound $\bar{z} + \kappa \bar{c}^*$ obtained in the first iteration of the CG using the dual values of the respective algorithm (Desrosiers and Lübbecke 2005). Here, \bar{z} is the objective value of the (dual) RMP in the first iteration, κ is the maximum number of stocks, and \bar{c}^* the minimum reduced cost. Additionally, we compare the distance between the optimal objective value z^* (when CG terminates) and the objective value derived from the (predicted) dual values (calculating Objective (3a)).

We analyze the same 400 instances with four different problem sizes. To evaluate the instances, we calculate the relative gap $[UB - LB]/UB$ of both bounds for each instance. The results are given in Table 6. We highlight the best average result in bold. The average value is given with each bound's standard deviation in brackets. The first column for each algorithm refers to the Lagrangian bound. Overall, SPARSE provides the best Lagrangian bound for all sizes except of problem size 100. However, the average value for FULL is slightly better for problems with 75 items. The standard deviation is quite high, with a value of 27.0. In general, we can see a very similar behavior for FULL and SPARSE as in Table 5. The Lagrangian bound for SPARSE is relatively stable even

Table 5. Performance of the Predictions by FULL, SPARSE, and Random Draw (RD) of the Optimal Dual Variables: All Values on Average

Items	FULL			SPARSE				RD				
	Underestimate	Overestimate	Deviation	Absolute deviation	Underestimate	Overestimate	Deviation	Absolute deviation	Underestimate	Overestimate	Deviation	Absolute deviation
25	19.04	5.92	-0.18	0.25	16.07	8.93	0.00	0.09	13.03	11.97	-0.02	0.35
50	33.11	16.87	-0.04	0.10	33.07	16.93	0.00	0.09	26.31	23.69	-0.02	0.35
75	45.72	29.26	-0.02	0.08	48.98	26.02	0.01	0.08	35.45	39.55	0.02	0.32
100	49.71	50.26	0.01	0.08	65.48	34.52	0.00	0.08	51.13	48.87	-0.01	0.34

Table 6. Performance of the Predictions by FULL, SPARSE, and the Dual Feasible Function f_0^y of the Optimal Objective Value: All Values on Average

Items	FULL		SPARSE		f_0^y	
	Lagrangian bound (%)	FULL bound (%)	Lagrangian bound (%)	SPARSE bound (%)	Lagrangian bound (%)	DFF bound (%)
25	89.0 (25.0)	56.2 (33.3)	25.8 (16.1)	4.3 (4.7)	70.1 (45.1)	2.5 (3.2)
50	30.0 (36.4)	8.9 (5.0)	25.4 (15.7)	3.5 (2.6)	58.6 (48.3)	2.0 (2.9)
75	19.9 (27.0)	5.6 (3.9)	25.4 (15.5)	3.6 (4.0)	44.7 (49.1)	2.3 (2.9)
100	25.9 (18.8)	3.0 (2.6)	26.6 (18.7)	2.8 (2.9)	55.9 (48.9)	1.6 (2.3)

between the different problem sizes, while the values for FULL vary widely. The Lagrangian bound derived from the dual values of the dual feasible function f_0^y performs significantly worse than the other two. One reason can be that the dual feasible functions are used to calculate the objective value and are not intended for stabilization, that is, the difference between the actual dual value and the one from the dual feasible function might be large. The good performance on the objective value can be seen in the second bound. Here, the dual feasible function performs best for all instances. However, SPARSE and FULL also achieve good results with an average gap between 2.8% and 8.9%. However, FULL has an average gap of 56.2% for instances with only 25 items. This supports the results from the previous study where we could see that FULL cannot predict the dual values for the unknown problem size 25.

4.3. Application of the Learning Algorithms

We motivated the prediction of the optimal dual values with stabilized CG and the utility of these as stability centers. In this section, we compare the performance of the predictions in the stabilized CG with the classical approach of Du Merle et al. (1999) as described in Section 3.

4.3.1. Study Instances. We use the same 400 instances as in the previous study, that is, 100 instances with 25, 50, 75, and 100 items each. Additionally, we evaluate the performance of SPARSE for 100 instances with 125 items. The penalty weights ϵ_i^+ and ϵ_i^- (see Model (5)) are updated in all tests in the same way. As soon as no column with negative reduced cost can be found, the values are reduced by half. In our approach, the penalty weights are always terminated toward zero only at the end of the CG, that is, the centers are active until the end. The average number of iterations, as well as the solution time, is given in Table 7. We highlight the best result in bold. The number of items for the instances is given in the first column. The next two columns show the average performance of the predicted dual values of FULL, that is, the average number of iterations and the average total solution time of the CG algorithm with stabilization. Columns 3 and 4, as well as 5 and 6, give the same information, but for the dual values of SPARSE and the classical approach by Du Merle et al. (1999), that is, updating the δ values after each iteration of CG. Overall, both the networks dominate the classical approach: not only on average but in all instances as can be seen in Table 8. The table shows how often one of the three approaches performs best for the 100 instances of different item sizes. The same differences appear between the two networks as in the previous study. SPARSE performs best in instances of sizes 25, 50, and 75. For the problem size of 100 items, FULL dominates in both runtime and number of iterations. In this case, the solution time is halved compared with the classical approach. To the question of why FULL outperforms SPARSE at 100 items as well, we have to look at Table 5 again. Although both networks, on average, misestimate the optimal dual value by 0.08 units, the ratio of overestimation and underestimation is more balanced for FULL. This leads to the fact that the algorithm can

Table 7. Evaluation of the Performance of the Three Update Mechanisms: All Values on Average

Items	FULL		SPARSE		Classical approach	
	Iterations	Time (s)	Iterations	Time (s)	Iterations	Time (s)
25	65.89	0.50	55.53	0.24	74.70	0.47
50	113.43	1.01	102.49	0.93	172.17	1.48
75	181.11	3.06	177.77	2.89	333.95	4.72
100	207.35	3.76	233.34	4.14	469.18	7.77
125	—	—	472.77	7.18	661.65	13.64
Average	—	—	208.38	3.08	342.33	5.62

Table 8. Evaluation of the Dominance of the Three Update Mechanisms: 100 Instances

Items	FULL	SPARSE	Classical approach
25	17	83	0
50	27	73	0
75	40	60	0
100	73	27	0
125	—	96	4
Σ	156	340	4

strive faster against the optimum. For the out-of-sample case with 125 items, we see that SPARSE dominates here as well. The solution time is half as long as for the classical approach. Nevertheless, there are losses with respect to the ratio of iterations, that is, only 28.5% in contrast to 50.3% for the instances with 100 items. From this, it can be assumed that the performance of the network is not unrestrictedly good depending on the number of items.

4.3.2. BPPLIB Instances. Because the previous analysis is relatively limited in how instances are generated, we now extend this aspect by performing the same analysis for the test instances of the BPPLIB (Delorme et al. 2018). Therefore, we use the two algorithms FULL and SPARSE in the same way as before; that is, we do not retrain the learning algorithms based on the new instances but leave them as in the previous study. Remember that we can use FULL only for instances with up to 100 items. Consequently, we analyze FULL only for a subset of instances, whereas we analyze SPARSE on all instances. We present the results in the same way as for the study instances; that is, the average number of iterations and the solution time are given in Table 9. The first three columns give information about the instances, that is, to which class they belong, how many instances the class contains, and how many of these instances have no more than 100 items. An interesting result is that our algorithms perform well on instances not generated uniformly from the stock length, such as Falkenauer T. Nevertheless, it must also be mentioned that the algorithm SPARSE failed for classes Augmented Integer Round-Up Property (AI) and Augmented Non-Integer Round-Up Property (ANI). One reason could be that the number of items and the stock size differed significantly from our training data. Table 10 shows the different classes in detail, that is, how often one of the three approaches performs best. FULL is the best choice for multiple instances, that is, for a total of 756 instances. Nevertheless, FULL also performs weaker than SPARSE considering the relative number of instances per class for the respective algorithm (Table 9). This relative proportion again underlines that FULL only performs well for instances with the maximum size of 100 items. The weak performance for the classes AI and ANI also show that there might still be room for improvement. However, it can also be interpreted that the selected learning algorithms should not be used for all instances of the CSP regardless of the selected training dimensions, that is, stock length and the number of items.

5. Conclusion

Learning algorithms are finding more and more applications in combinatorial optimization. They are used to solve the problem itself or support other solution methods in the optimization process. The appropriate choice of parameters for a stabilized CG is a largely untouched topic. Often suitable values are determined by numerical studies. However, this approach is not useful for the general use, for example, in generic solvers.

Table 9. Evaluation of the BPPLIB Instances of the Three Update Mechanisms: All Values on Average

Class	# ins.	# ins. ≤ 100	FULL		SPARSE		Classical approach	
			Iterations	Time (s)	Iterations	Time (s)	Iterations	Time (s)
AI	250	0	—	—	5,251.20	1,319.03	2,571.46	663.65
ANI	250	0	—	—	4,468.42	861.06	2,575.62	682.53
Falkenauer T	80	40	158.60	1.05	195.79	9.72	708.49	22.87
Falkenauer U	80	80	189.66	1.54	179.43	1.26	329.53	1.87
Hard	28	0	—	—	517.89	25.28	700.36	24.29
Irnich	240	0	—	—	1,258.70	873.62	1,751.14	877.38
Random	3,840	2,411	127.86	0.51	257.70	8.69	560.39	15.27
Scholl	1,210	1,032	131.56	1.11	166.34	5.01	264.94	6.18
Schwerin	200	200	115.15	0.76	93.46	0.80	136.75	0.95
Waescher	17	17	193.94	15.17	184.18	13.57	188.18	12.42
Average	—	—	—	—	643.93	128.51	696.16	99.48

Table 10. Evaluation of the Dominance of the Three Update Mechanisms: BPPLIB Instances

Class	FULL	SPARSE	Classical approach
AI	—	0	250
ANI	—	0	250
Falkenauer T	0	80	0
Falkenauer U	23	57	0
Hard	—	28	0
Irnich	—	240	0
Random	437	3,362	41
Scholl	290	856	64
Schwerin	3	197	0
Waescher	3	10	4
Σ	756	4, 830	609

This paper analyzed whether suitable parameters for a stabilized CG can be determined using a learning algorithm for the CSP. In this context, we developed two learning algorithms that can predict the optimal dual values for different instances of the CSP, that is, FULL and SPARSE. For this purpose, we generated and solved 100,000 instances of the CSP with varying numbers of items. The data analysis revealed exciting correlations between the relative size of the items and the (optimal) dual value. We evaluated our learning algorithms by training the networks with different data sizes. The analysis showed that SPARSE could achieve good results with smaller data sets. In contrast, FULL can better map the correlations of the different dual values of an instance. In comparison with a naïve random drawing, it was then shown that both networks are superior. In a final study, we investigated the application of the learning algorithms in stabilized CG. Here we also showed that the algorithm terminates twice as fast with the parameters determined by the learning algorithms than with the classical approach. Consequently, it can be said that FULL is applicable when one has to solve the problem mainly for a fixed set of items and only occasionally solves instances with fewer items. On the other hand, SPARSE is applicable when there is more significant variability in the number of items between instances.

This contribution can be further built upon in several ways. The generic stabilization approach of Du Merle et al. (1999) is one of the best known and most widely used in the literature. However, as the literature review has shown, there are numerous other approaches, each with its own strengths and weaknesses. It would be interesting to analyze which stabilization methods can also be supported by learning algorithms and to what extent the networks differ. Additionally, the focus of our investigation was the CSP. An adaptation of the method to other problem classes could reveal additional information about the behavior of the dual values and thus lead to a deeper understanding of the problems. The prediction algorithm might also be further improved by incorporating additional information as features, such as bounds on the dual values or dual feasible functions. Although we have chosen two extremes in the information representation, combinations can also be constructed.

References

- Alvarez AM, Louveaux Q, Wehenkel L (2017) A machine learning-based approximation of strong branching. *INFORMS J. Comput.* 29(1): 185–195.
- Alves C, Valério de Carvalho JM (2008) A branch-and-price-and-cut algorithm for the pattern minimization problem. *RAIRO Oper. Res.* 42(4):435–453.
- Barnhart C, Johnson EL, Nemhauser GL, Martin WP Savelsbergh, Vance PH (1998) Branch-and-price: Column generation for solving huge integer programs. *Oper. Res.* 46(3):316–329.
- Bello I, Pham H, Le VQ, Norouzi M, Bengio S (2017) Neural combinatorial optimization with reinforcement learning. Preprint, submitted November 29, 2016, <https://arxiv.org/abs/1611.09940>.
- Ben Amor H, Desrosiers J (2006) A proximal trust-region algorithm for column generation stabilization. *Comput. Oper. Res.* 33(4):910–927.
- Ben Amor H, Desrosiers JV, de Carvalho JM (2006) Dual-optimal inequalities for stabilized column generation. *Oper. Res.* 54(3):454–463.
- Ben Amor HM, Desrosiers J, Frangioni A (2009) On the choice of explicit stabilizing terms in column generation. *Discrete Appl. Math.* 157(6):1167–1184.
- Bengio Y, Lodi A, Prouvost A (2021) Machine learning for combinatorial optimization: A methodological tour d’horizon. *Eur. J. Oper. Res.* 290(2):405–421.
- Bertsimas D, Kallus N (2020) From predictive to prescriptive analytics. *Management Sci.* 66(3):1025–1044.
- Bonami P, Lodi A, Zarpellon G (2018) Learning a classification of mixed-integer quadratic programming problems. van Hoes WJ, ed. *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, vol. 10848 of SpringerLink Bücher (Springer, Cham, Switzerland), 595–604.
- Borchani H, Varando G, Bielza C, Larrañaga P (2015) A survey on multi-output regression. *Data Mining Knowledge Discovery* 5(5):216–233.
- Brunner JO, Stolletz R (2014) Stabilized branch and price with dynamic parameter updating for discontinuous tour scheduling. *Comput. Oper. Res.* 44:137–145.

- Caprara A, Locatelli M, Monaci M (2005) Bidimensional packing by bilinear programming. Jünger M, Kaibel V, eds. *Integer Programming and Combinatorial Optimization*, vol. 3509 of Lecture Notes in Computer Science (Springer, Berlin), 377–391.
- Cheng C, Feiring B, Cheng T (1994) The cutting stock problem—A survey. *Internat. J. Production Econom.* 36(3):291–305.
- Clautiaux F, Alves C, Valério de Carvalho J (2010) A survey of dual-feasible and superadditive functions. *Annals Oper. Res.* 179(1):317–342.
- Clautiaux F, Alves C, Valério de Carvalho J, Rietz J (2011) New stabilization procedures for the cutting stock problem. *INFORMS J. Comput.* 23(4):530–545.
- Dantzig GB, Wolfe P (1960) Decomposition principle for linear programs. *Oper. Res.* 8(1):101–111.
- Delorme M, Iori M, Martello S (2018) BPPLIB: A library for bin packing and cutting stock problems. *Optim. Lett.* 12(2):235–250.
- Desrosiers J, Lübbecke ME (2005) A primer in column generation. Desaulniers G, ed. *Column Generation* (Springer, New York), 1–32.
- Du Merle O, Villeneuve D, Desrosiers J, Hansen P (1999) Stabilized column generation. *Discrete Mathe.* 194(1–3):229–237.
- Dyckhoff H (1991) Approaches to cutting and packing problems. Pridham M, ed. *Production Research* (Taylor & Francis, London), 46–54.
- Fekete SP, Schepers J (2001) New classes of fast lower bounds for bin packing problems. *Math. Programming* 91(1):11–31.
- Fekete SP, Schepers J (2004) A general framework for bounds for higher-dimensional orthogonal packing problems. *Math. Methods Oper. Res.* 60(2):311–329.
- Gau T, Wäscher G (1995) CUTGEN1: A problem generator for the standard one-dimensional cutting stock problem. *Eur. J. Oper. Res.* 84(3):572–579.
- Gilmore PC, Gomory RE (1961) A linear programming approach to the cutting-stock problem. *Oper. Res.* 9(6):849–859.
- Glantz SA, Slinker BK, Neilands TB (2016) *Primer of Applied Regression & Analysis of Variance*, 3rd ed. (McGraw-Hill Education, New York).
- Goffin JL, Haurie A, Vial JP (1992) Decomposition and nondifferentiable optimization with the projective algorithm. *Management Sci.* 38(2):284–302.
- Gondzio J, Sarkissian R (1996) Column generation with a primal-dual method. Logilab technical report 96-6, Department of Management Studies, University of Geneva, Geneva.
- Goodfellow I, Bengio Y, Courville A (2016) *Deep Learning. Adaptive Computation and Machine Learning* (MIT Press, Cambridge, MA).
- Gschwind T, Irnich S (2017) Stabilized column generation for the temporal knapsack problem using dual-optimal inequalities. *OR Spectrum* 39(2):541–556.
- Gurobi Optimization LLC (2021) Gurobi optimizer reference manual. Accessed December 1, 2021, <https://www.gurobi.com>.
- Hoos HH (2011) Automated algorithm configuration and parameter tuning. Hamadi Y, ed. *Autonomous Search* (Springer, Berlin), 37–71.
- Kantorovich LV (1960) Mathematical methods of organizing and planning production. *Management Sci.* 6(4):366–422.
- Kartak VM, Ripatti AV, Scheithauer G, Kurz S (2015) Minimal proper non-IRUP instances of the one-dimensional cutting stock problem. *Discrete Appl. Math.* 187:120–129.
- Khalil EB, Dilkina B, Nemhauser GL, Ahmed S, Shao Y (2017) Learning to run heuristics in tree search. Sierra C, ed. *Proc. Internat. Joint Conf. on Artificial Intelligence* (Curran Associates, Red Hook, NY), 659–666.
- Kotsiantis SB, Zaharakis I, Pintelas P (2007) Supervised machine learning: A review of classification techniques. *Informatica (Slovenia)* 31(3):249–268.
- Kraul S (2023) Problem data of the study. <https://github.com/SebastianKraul/Machine-learning-supported-prediction-of-dual-variables-for-the-cutting-stock-problem>.
- Kruber M, Lübbecke ME, Parmentier A (2017) Learning when to use a decomposition. Salvagnin D, Lombardi M, eds. *Integration of AI and OR Techniques in Constraint Programming*. Lecture Notes in Computer Science (Springer, Cham, Switzerland), 202–210.
- LeCun Y, Bengio Y, Hinton G (2015) Deep learning. *Nature* 521(7553):436–444.
- Lee C, Park S (2011) Chebyshev center-based column generation. *Discrete Appl. Math.* 159(18):2251–2265.
- Lirov Y (1992) Knowledge based approach to the cutting stock problem. *Math. Comput. Modelling* 16(1):107–125.
- Lübbecke ME, Desrosiers J (2005) Selected topics in column generation. *Oper. Res.* 53(6):1007–1023.
- Marsten RE, Hogan WW, Blankenship JW (1975) The boxstep method for large-scale optimization. *Oper. Res.* 23(3):389–405.
- Martello S, Toth P (1990) Lower bounds and reduction procedures for the bin packing problem. *Discrete Appl. Math.* 28(1):59–70.
- Mazyavkina N, Sviridov S, Ivanov S, Burnaev E (2021) Reinforcement learning for combinatorial optimization: A survey. *Comput. Oper. Res.* 134:105400.
- Modaresi S, Sauré D, Vielma JP (2020) Learning in combinatorial optimization: What and how to explore. *Oper. Res.* 68(5):1585–1604.
- Morabit M, Desaulniers G, Lodi A (2021) Machine-learning-based column selection for column generation. *Transportation Sci.* 55(4):815–831.
- Nazari M, Oroojlooy A, Takáč M, Snyder LV (2018) Reinforcement learning for solving the vehicle routing problem. *Proc. 32nd Internat. Conf. on Neural Inform. Processing Systems* (Curran Associates, Red Hook, NY), 9861–9871.
- Neame PJ (1999) Nonsmooth dual methods in integer programming. Dissertation, University of Melbourne, Melbourne, Australia.
- Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, et al. (2011) Scikit-learn: Machine learning in Python. *J. Machine Learn. Res.* 12:2825–2830.
- Pessoa A, Sadykov R, Uchoa E, Vanderbeck F (2018) Automation and combination of linear-programming based stabilization techniques in column generation. *INFORMS J. Comput.* 30(2):339–360.
- Rousseau LM, Gendreau M, Feillet D (2007) Interior point stabilization for column generation. *Oper. Res. Lett.* 35(5):660–668.
- Scheithauer G, Terno J (1995) The modified integer round-up property of the one-dimensional cutting stock problem. *Eur. J. Oper. Res.* 84(3):562–571.
- Smith KA (1999) neural networks for combinatorial optimization: A review of more than a decade of research. *INFORMS J. Comput.* 11(1):15–34.
- Václavík R, Novák A, Šůcha P, Hanzálek Z (2018) Accelerating the branch-and-price algorithm using machine learning. *Eur. J. Oper. Res.* 271(3):1055–1069.
- Valério de Carvalho JM (2005) Using extra dual cuts to accelerate column generation. *INFORMS J. Comput.* 17(2):175–182.
- Vance PH (1998) Branch-and-price algorithms for the one-dimensional cutting stock problem. *Comput. Optim. Appl.* 9(3):211–228.
- Vanderbeck F (2005) Implementing mixed integer column generation. Desaulniers G, ed. *Column Generation* (Springer, New York), 331–358.
- Wentges P (1997) Weighted Dantzig-Wolfe decomposition for linear mixed-integer programming. *Internat. Trans. Oper. Res.* 4(2):151–162.