Les Cahiers du GERAD ISSN: 0711–2440

Learning and modeling implicit constraints in optimization models through decision trees

M. Bayani, Y. Adulyasak, L.-M. Rousseau

G–2024–78

December 2024

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Learning and modeling implicit constraints in optimization models through decision trees

Mahdis Bayani [a, b, c]

Yossiri Adulyasak [d, b]

Louis-Martin Rousseau [a, c]

[a] *Département de mathématiques et de génie industriel, Polytechnique Montréal, Montréal, (Qc), Canada, H3T 1J4*

[b] *GERAD, Montréal (Qc), Canada, H3T 1J4*

[c] *CIRRELT, Montréal (Qc), Canada, H3T 1J4*

[d] *Department of Logistics and Operations Management, HEC Montréal, Montréal (Qc), Canada, H3T 2A7*

mahdis.bayani@polymtl.ca
yossiri.adulyasak@hec.ca
louis-martin.rousseau@polymtl.net

December 2024
Les Cahiers du GERAD
G–2024–78

Abstract : Planners in different industries use optimization software for decision-making. In numerous practical applications, these optimization tools are often not readily adjustable or configurable by end users due to limited knowledge, resources, or the required investment to consistently make such customizations. As a result, planners frequently adjust the solutions obtained from software based on implicit internal operational rules and preferences to make them feasible in practice. These practices may differ across various business units and subsidiaries. One can leverage data-driven methods to learn and embed implicit side constraints in a mixed integer linear program (MILP) to ensure that such rules can be learned and systematically incorporated into optimization models. These implicit constraints can be created from machine learning models trained using previously implemented solutions. To this end, we extend a data-driven constraint customization framework in the literature developed to extract constraints in the form of a traditional linear regression model to the case when constraints take the form of decision trees. This allows us to learn and incorporate implicit constraints in a non-linear or logical form. To demonstrate the value of this framework, experiments were conducted on the knapsack and nurse rostering problems where various combinations of hidden operational rules were simulated. The solutions obtained by our proposed solution framework suggest that it can effectively adjust the solutions based on the constraints extracted from historical solutions. The implicit constraints that take the form of decision trees generally outperform those based on linear regression models, mainly when part of the decision model comprises discrete variables.

Keywords : Constraint customization, data-driven methods, mixed integer linear programming, decision tree

1 Introduction

Optimization models are employed in various industrial applications to help planners make better decisions. However, most common models, when embedded in software-based decision support systems, fail to capture all the subtleties or practical requirements of the specific operational settings they are deployed into. Hence, planners often need to manually modify the output solutions (even when they are optimal from the model's point of view) to meet informal operational rules and preferences (Pawlak and Krawiec 2017, Hewitt and Frejinger 2020). These post-optimization adjustments are mostly human-driven and susceptible to error. Furthermore, as the business rules have the potential to change over time and be influenced by the preferences of different decision-makers within the company, the process of customization becomes time-consuming and may need to be repeated. The main drawback of this type of customization is that the software transforms into a customized decision support system for each customer, which is not an ideal solution in today's information technology industry.

One possible approach involves employing data-driven methods to learn to incorporate the underlying implicit business rules and preferences specific to each decision-maker using predictive models Hewitt and Frejinger (2020). By doing so, the software provider can develop a single core model for all subsidiaries, allowing data-driven customization to be implemented automatically without human intervention. This allows for the expansion of the initial optimization model to systematically learn and adapt to the individual requirements of each decision-maker. This idea of providing a data-driven framework to extract business rules from historical decisions and integrate them into general optimization problems (GOP) was first investigated in Hewitt and Frejinger (2020). The first assumption of this framework is that the company already knows the original optimization problem that has to be solved (GOP) and has access to the historical data of both the solutions produced and the modifications that led to the executed plans. This is practical for organizations that have been using optimization-based tools for some time. In Hewitt and Frejinger (2020), the authors present a methodology to learn implicit side constraints for MILPs using a traditional linear regression model.

This paper contributes to further expanding the framework presented in Hewitt and Frejinger (2020) in the following directions. First, we showcase how regularized linear regression and decision trees can enhance the efficacy of the data-driven constraint customization framework. Second, we present a MILP model that integrates implicit constraints, which take the form of decision trees, enabling us to learn and incorporate non-linear or logical customization constraints. Third, in addition to linear programming problems, we empirically validate the framework's performance on the binary knapsack and nurse rostering problems where various combinations of concealed operational rules are simulated.

The rest of the paper is organized as follows: The next section provides an overview of the related literature. In Section 3, we present the proposed framework of Hewitt and Frejinger (2020) and give more precisions on our contributions in that context. Section 4 is dedicated to presenting illustrative examples where we implement the approach to two industrial problems and report the obtained results. Finally, in Section 5, we discuss this study's findings and possible future directions.

2 Related work

Lately, the field of analytics has captured the interest of different scientific communities. The fast-growing availability of data motivates utilizing data-driven methods to find the latent patterns of the data, estimate the parameters, and build optimization models (Nicosia et al. 2017). Moreover, more sophisticated data-driven methods must be developed to tackle the complexity of solving these optimization problems enhanced by data (Khaniyev 2018). Thus, recently, combining machine learning methods with optimization problems has drawn researchers' attention. On one side, many studies are leveraging machine learning methods directly to solve combinatorial optimization problems. For instance, consider solving patient appointment scheduling (Pham et al. 2023) and routing problem of a radiotherapy device (Kafaei et al. 2021) through machine learning methods. In addition, Junior Mele

et al. (2021) discusses several studies that aim to address the traveling salesperson problem (TSP) and its variations using machine learning techniques. Employing data-driven approaches to deal with the uncertainty in stochastic and robust optimization problems is also a highly active stream of research (Ning and You 2018, Augustin et al. 2022). We refer the reader to Bengio et al. (2021), Nguyen et al. (2022) and the references therein for more details.

In many other existing studies on the combination of prescriptive and predictive methods, predictive analytics are implemented beforehand, and their outputs are fixed. For example, the parameters of optimization models can be obtained from a machine learning method and incorporated as parameters in the optimization model. Ferreira et al. (2016) apply regression trees to predict the demand for initial exposure of the products with no historical sales data for an online retailer. Based on the estimated demand from the regression trees using various input features, they optimize the price of each item to maximize the overall revenue. In this paradigm, called *predict-then-optimize*, the first step maximizes the prediction power of the model, and the second step optimizes the decisions based on the predictions and associated costs (Vanderschueren et al. 2022). Recent studies focus on integrating predictive and prescriptive analytics by leveraging the optimization problem in the training step (Elmachtoub and Grigas 2022). By considering the cost of decision-making in the loss function of the prediction model, the decisions can become more cost-effective in the presence of uncertainty (Elmachtoub and Grigas 2022).

The proposed methodology also aims to combine predictive and prescriptive analytics. However, we distinguish this work from the studies mentioned above, which attempt to estimate the parameters of the optimization models or those replacing an optimization approach with a machine learning approach. Instead, This research presents a framework for embedding machine learning models in optimization models with incomplete information on the problem's constraints, not their parameters. From this perspective, the proposed approach is related to the literature on constraint acquisition, mainly conducted in constraint programming (CP). In our study, the core optimization model is mainly known, and the only missing part is a subset of constraints. Thus, for the sake of conciseness, we will mainly review the literature in the area of constraint learning and acquisition, whereas the studies that focus on model learning or model seeking (e.g., Beldiceanu and Simonis (2016), Kumar et al. (2021)). Studies in constraint acquisition focus on learning a set of constraints based on several feasible and infeasible examples. Most of these approaches attempt to match the examples with a combination of rules in a given library and suggest constraints compatible with the solutions in hand (Bessiere et al. 2017, Tsouros and Stergiou 2020). Although most constraint acquisition methods synthesize constraints for CP models, a few studies, such as Pawlak and Krawiec (2017) and Schede et al. (2019), learn constraints for MILP or nonlinear programming (NLP) models. A comprehensive survey of related literature is represented in Fajemisin et al. (2021). What differentiates their work from the current study is that these approaches do not use statistical learning and predictive models to learn complex feasibility regions.

Since our approaches embed predictive models into optimization ones, we provide a short overview of papers considering predictive models as constraints of an optimization problem. Liu et al. (2021) introduces a framework to integrate travel-time prediction and order-assignment optimization to capture the drivers' routing decision-making and include it in the optimization problem. They employ linear regression and random forest as two compatible prediction models with optimization. They further reformulate the problem to be efficiently solved by the branch-and-price algorithm. Moreover, they discuss the robust integrated model of the problem and provide two heuristics to solve the multi-period setting of the problem. The methodology introduced in Lombardi et al. (2017), called empirical model learning (EML), obtains components of a complex prescriptive method using machine learning by learning relations between decidable and observable. In this conceptual model, they generate data by simulation and propose embedding a trained neural network and decision tree (DT) into a combinatorial problem such as mixed integer non-linear programming (MINLP), CP, and local search. The papers provide an embedding for linear regression in MILP but do not present any result for this case. Moreover, for the case of DT, it only provides embedding for local search, CP, and SAT. More recently,

two general packages have emerged to allow for the automatic linearization of prediction models and their integration into MILP. JANOS (Bergman et al. 2022) currently supports neural networks, logistic regression, and linear regression. OMLT (Ceccon et al. 2022) is another open-source software package known as an optimization and machine learning toolkit allowing integration of neural networks and gradient-boosted trees in large-scale optimization problems. It is compatible with the training models from packages like Keras, PyTorch, and TensorFlow and has various activation functions.

3 Description of the solution framework

We first describe the framework introduced in Hewitt and Frejinger (2020), which is built based on the following three main steps:

1. Defining the general structure of the problem, referred to as the general optimization problem (GOP).
2. Training a machine learning model on historical modifications applied to each decision variable of the previous optimal solutions of the GOP.
3. Embedding the trained models in the GOP as soft constraints and providing an augmented model, which we refer to as the adapted optimization problem (AOP).

We assume the decision support software company knows the GOP of the customer, which is a traditional MILP for each decision point t:

$$\max \quad \boldsymbol{r}^t \boldsymbol{x} \tag{1}$$
$$\text{s.t.} \quad \boldsymbol{A}^t \boldsymbol{x} \leq \boldsymbol{b}^t \tag{2}$$
$$\boldsymbol{x} \in \mathcal{X}, \tag{3}$$

where $\boldsymbol{r}^t$, $\boldsymbol{A}^t$ and $\boldsymbol{b}^t$ are the parameters of the problem. We also assume historical data for a set of time periods ($t : 1...T$) is available. We then compare the optimal solution ($\bar{\boldsymbol{x}}^t$) and the actionable plan ($\bar{\boldsymbol{y}}^t$) at decision points t. The assumption is that the customer has applied business rules to the optimal solutions that altered them to actionable plans. These business rules act as side constraints $\boldsymbol{G}^t \boldsymbol{x} + \boldsymbol{H}^t \boldsymbol{v} \leq \boldsymbol{u}^t$ that are missing in the GOP. These constraints consist of the GOP regular variables ($\boldsymbol{x}$) and possible extra decision variables ($\boldsymbol{v} \in \mathcal{V}$) such as ones related to if-then rules or precedence in scheduling. It should be remarked that the matrices and vectors are shown in bold text throughout this work.

The second task revolves around training a predictive model on historical data. The objective is to keep the AOP as a MILP to maintain the existing solution pipeline. We thereby opt for machine learning prediction models that are linearizable and tractable. Hewitt and Frejinger (2020) employ linear regression in their proposed framework. For each element of the variables of the optimization problem, the regression equation is learned from the data set ($\bar{\boldsymbol{x}}, \bar{y}_i$) as follows:

$$y_i - \left(\hat{\beta}^i + \sum_{j \in N} \left(\beta^i_j x_j \right) \right) = 0 \qquad \forall i \in N : 1, \ldots, n, \tag{4}$$

where $\hat{\beta}^i$ and $\boldsymbol{\beta}^i$ are the interception and coefficients of the ith equation, respectively, the least squares method is used throughout their work.

The third step of the framework proposed in Hewitt and Frejinger (2020) is to embed the learning model in the GOP to create the AOP. It is possible to embed a machine learning model into an optimization problem only if a suitable encoding has been defined. They define δ_i as the difference between each pair of predictions of the executable plan (y_i) and optimal solution (x_i). The AOP is therefore represented as following for each decision point t:

$$\max \quad r^t \boldsymbol{x} - c\Delta \tag{5}$$

$$
\begin{aligned}
\text{s.t.} \quad & A^t \boldsymbol{x} \leq b^t && && (6)\\
& y_i - (\hat{\beta}^i + \sum_{j \in N} (\beta^i_j x_j)) = 0 && \forall i \in N : 1, \ldots, n && (7)\\
& \delta^i \geq y_i - x_i && \forall i \in N : 1, \ldots, n && (8)\\
& \delta^i \geq x_i - y_i && \forall i \in N : 1, \ldots, n && (9)\\
& \Delta = \sum_{i \in N} \delta^i && \forall i \in N : 1, \ldots, n && (10)\\
& \boldsymbol{x} \in \mathcal{X} \subseteq \mathcal{R}^n, \boldsymbol{y} \subseteq \mathcal{R}^n && && (11)\\
& \delta^i \geq 0 && \forall i \in N : 1, \ldots, n && (12)\\
& \Delta \geq 0, && && (13)
\end{aligned}
$$

$\hat{\beta}^i$ and $\boldsymbol{\beta}^i$ are fixed learned parameters, and Δ is a variable that captures the sum of all differences between predictions and solutions. Moreover, the objective function is designed to stimulate the AOP to obtain solutions close to predictions of executable plans. c is a given penalty for the AOP. This penalty term balances the importance of the original GOP objective and the difference between predictions and solutions and needs to be tuned for each problem. By solving the AOP, we obtain an optimal solution ($\boldsymbol{x}$). The promise is that the new solution satisfies the business rules for more instances than the original solutions.

This framework assumes that the underlying problem can be modeled as an optimization problem and that the parameters are known in advance. A dataset of historical records comprising optimal solutions (intended decisions) and executed plans (actual decisions) exists. To evaluate the approach's effectiveness, the following additional steps need to be considered in this study: generating data and validating the framework. As shown in the framework depicted in Figure 1, the first step is data

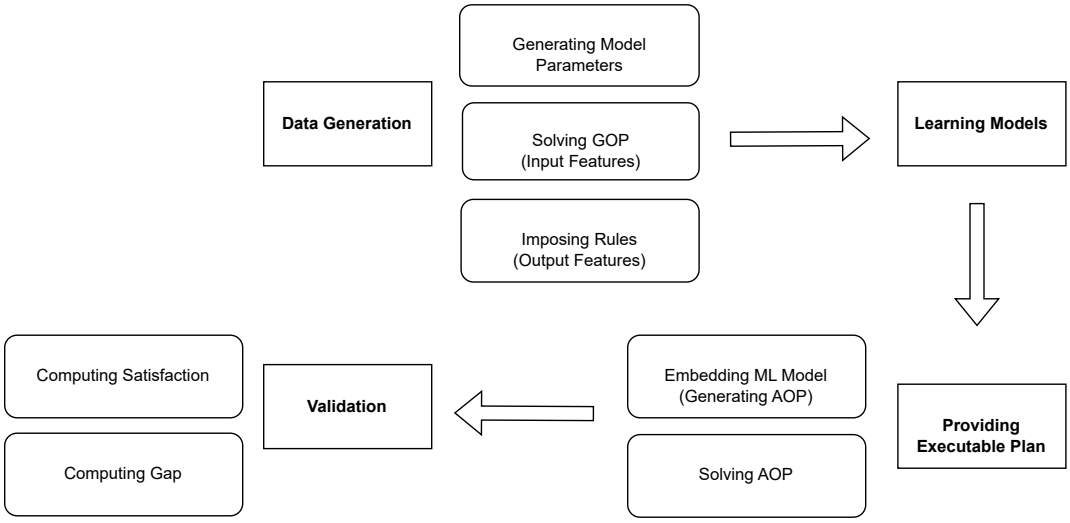


Figure 1: Framework overview

generation. The experiment is a controlled setting since, in this study, the true data generation is known. The collected data is partitioned into training and test sets. The former is utilized to acquire knowledge on the linear representation of a mapping. With the latter, we evaluate the effectiveness of these learned mathematical entities.

The parameters are generated randomly for each problem instance, and the constructed problem is solved optimally. Once the optimal solutions are obtained, the predefined rules and preferences are applied, creating the new altered solutions. A machine learning model is then trained and embedded

into the optimization model in the next step. As such, plans that are likely more executable are generated by solving the new optimization problem (AOP). To validate the method, we evaluate two criteria. Firstly, we measure the percentage of feasible instances regarding the unmodeled rules after solving the AOP. Secondly, assuming the rules are known, we compute the difference between the cost of adding the learned constraints to the model and the cost of adding the true rules directly to the model. This performance measure is associated with the degradation of the objective function because the predictive model is inherently imperfect.

The following explains this study's contributions to the framework's learning phase. Section 3.1 briefly describes the regularizations used in the linear regression. Section 3.2 details how we can model and incorporate decision trees instead of linear regression in the framework.

3.1 Regularized linear regression

Hewitt and Frejinger (2020) use a sparse matrix of coefficients to generate the AOP by removing the coefficients with p-value higher than 0.05. We also believe that using a sparser model has some advantages. Regularization leads to obtaining a trained model based on the more important features rather than all of them. So, each variable y is influenced by a limited number of variables x. Regularization reduces the number of variables in each constraint. This model results in a more easily interpretable and computationally tractable AOP (Bottmer et al. 2022). We explore a regularized version of the described linear regression to pursue this idea. As such, we apply lasso regression, ridge regression, and elastic Net (eNet) (Zou and Hastie 2005). While the regularized linear regression equation is the same as the linear regression, lasso regression extends it by adding an $L1$-norm penalty to the model's coefficients ($\lambda \,||\, \boldsymbol{\beta} \,||_1$). The shrinkage parameter, λ, controls the sparsity of the coefficients matrix by setting less important features to zero. On the other hand, the $L2$-norm penalty term in ridge regression limits the magnitude of weights and shrinks them toward zero but does not set them exactly to zero. Elastic Net employs a combination of $L1$ and $L2$ regularization, which can help overcome some of the limitations of each separate regularization.

3.2 Decision trees

This study suggests using decision trees as a machine learning technique to predict latent (possibly nonlinear) business rules compatible with MILP. In decision trees, the feature space is partitioned into subspaces, and each subspace has its specific fitted prediction model. Therefore, a piece-wise linear model can be constructed to approximate nonlinear rules. Regression trees can be used when dealing with fractional variables, whereas classification trees would be beneficial with discrete variables (Dunn 2018). Inspired by the studies of Bertsimas and Dunn (2017) and Dunn (2018), one can embed decision trees in an optimization model by defining auxiliary binary variables $\boldsymbol{z}$, and big-M constraints:

$$\max \quad \boldsymbol{r}^t\boldsymbol{x} - c\Delta \tag{14}$$

$$\text{s.t.} \quad \boldsymbol{A}^t\boldsymbol{x} \leq \boldsymbol{b}^t \tag{15}$$

$$y_i - (\hat{\beta}^{il} + \sum_{j\in N}(\beta_j^{il}x_j)) \leq M(1 - z^{il}) \qquad \forall i \in N, \forall l \in L \tag{16}$$

$$y_i - (\hat{\beta}^{il} + \sum_{j\in N}(\beta_j^{il}x_j)) \geq -M(1 - z^{il}) \qquad \forall i \in N, \forall l \in L \tag{17}$$

$$\sum_{j\in N}(h_j^{im}x_j) \leq g^{im} + (1 - z^{il}) \qquad \forall i \in N, \forall l \in L, \forall m \in Left(l) \tag{18}$$

$$\sum_{j\in N}(h_j^{im}x_j - h_j^{im}\epsilon_j) \geq g^{im} - (1 - \epsilon_{max})(1 - z^{il}) \qquad \forall i \in N, \forall l \in L, \forall m \in Right(l) \tag{19}$$

$$\sum_{l\in L} z^{il} = 1 \qquad \forall i \in N \tag{20}$$

$$\delta^i \geq y_i - x_i \qquad \forall i \in N \tag{21}$$

$$\delta^i \geq x_i - y_i \qquad \forall i \in N \tag{22}$$

$$\Delta = \sum_{i \in N} \delta^i \qquad \forall i \in N \tag{23}$$

$$\boldsymbol{x} \in \mathcal{X} \subseteq \mathcal{R}^n, \boldsymbol{y} \subseteq \mathcal{R}^n, \boldsymbol{z} \in \{0,1\}, \quad \delta^i \geq 0 \qquad \forall i \in N, \Delta \geq 0 \tag{24}$$

Since the problem is a multi-output classification, we train n separate decision trees ($i : 0, \ldots, n-1$). It is important to note that, in practice, training the model for all of the n variables is not necessarily required. Instead, one can train decision trees (or any other machine learning models) for only the variables that vary between the executed plans and the optimal solutions. Assuming each tree consists of $|L|$ leaf nodes, we define z^{il} to represent the selection of leaf node $l \in L$ for the output i. At each leaf node l in each tree, we make the prediction using Equations (16) and (17), where $\hat{\beta}^{il}$ and $\boldsymbol{\beta}^{il}$ are learned parameters and M is a sufficiently large constant. Here, (16) and (17) are represented in the most general format when we have a linear regression prediction for each leaf node. Nonetheless, they can be merely constant or 0-1 predictions for integer programming through a classification tree. Constraints (18) and (19) enforce branching by traversing each leaf node's left ancestors and right ancestors, respectively. m indicates intermediate nodes. If the learning method applies hyperplane splittings, several coefficients h can have nonzero values simultaneously. In this study, we only investigate the default machine learning method of Scikit-learn 0.21.2, CART (Breiman et al. 2017). In this library, univariate splitting is used for regression and classification trees. Hence, each node has only one x_j with a nonzero coefficient. The constant values g are thresholds of branching in the trees, and ϵ and ϵ_{max} are defined to generate compatible inequality constraints for MILP solvers. Moreover, selecting only one leaf node for each output of every problem instance is guaranteed according to constraints (20).

Although applying a random forest (Breiman 2001) instead of a single tree may result in a more accurate predictor, embedding a random forest in an optimization problem requires adding many extra variables and constraints. This dramatically expands the size of the problem.

4 Test problems and computational experiments

This section presents an experimental study to investigate the performance of the framework. To this end, we consider the fractional and binary knapsack problems and the nurse rostering problem. We evaluate the framework on the test instances of these problems in terms of both the feasibility of the solutions and the degradation in the objective function.

In each part, we briefly define the test problem, and the data generation procedure is later explained. The computational results for each test problem are detailed in the last section.

GUROBI (Gurobi Optimization, Inc. 2012) version 9.0.1 is our benchmark MIP solver for solving the GOP and the AOP models. All of the algorithms and models were implemented in Python. The machine learning models are trained using Scikit-learn (Pedregosa et al. 2011) 0.21. We divide the data into a training set (80%) and a test set (20%). Tenfold cross-validation and GridSearch are performed for hyper-parameter tuning for all models. In lasso and ridge regressions, we tune the shrinkage parameter. In Elastic Net, we assign a value and tune the $l1\_ratio$ parameter that combines lasso and ridge penalties. The GridSearch hyper-parameter tuning process for shrinkage parameter is performed over a discrete set in the range of 0.0001 to 10. In addition, the $l1\_ratio$ tuning is achieved using a step-size approach over the range of $0 \leq l1\_ratio \leq 1$. The Scikit-learn library is utilized for decision trees as well. For decision trees, we optimize the maximum depth and the criterion as measures of the splits' quality. In our hyper-parameters search procedure, we tune the model where the maximum depth changes over the range of 1 to 10, and the criterion is selected between gini and entropy.

4.1 Linear and binary Knapsack problems

We consider the problem of loading different products in a container with capacity b. Each product i has a size of a_i and yields a profit of r_i to maximize the overall profit. We assume this problem has been solved multiple times with different parameters for each day or each decision point t. The framework is implemented on both fractional and binary knapsack problems. In the former, a portion of a product can be loaded in the container. On the contrary, only the whole product can be loaded in the latter. Therefore, the GOP is precisely the MILP introduced in (1) - (3) for the linear knapsack problem and by setting $\boldsymbol{x^t} \in \{0, 1\}$, the model for the binary knapsack problem is attained. For the linear knapsack problem, we compare the results of the linear regression model embedded in the framework with decision tree modeling. In contrast, we investigate the discrete knapsack problem using linear regression and classification tree models.

4.1.1 Data generation

We adopted the data generation process presented in Hewitt and Frejinger (2020) while creating more complex sets of concealed operational rules. First, we construct an optimization model for each decision point t. The first step is to generate the parameters of the knapsack problems. We consider a small data set with $T = 500$ instances and $n = 15$ products and a larger one with $T = 500$ and $n = 50$. The parameters b and r_i are randomly generated from $[500, 1000]$ and $[50, 300]$, respectively, following a uniform distribution. We carry out our experiments considering different combinations of these parameters. a_i is also randomly sampled from a triangular distribution of $(8, b/n, b)$. In Addition, the big-M value is set to $\max(y_i) - \min(y_i)$. In the case of the knapsack problem, since $0 \leq y_i \leq 1$, the big-M equals 1.

According to Figure 1, the next step toward data generation is to solve the generated linear and binary knapsack problems. The obtained optimal solutions create the input features. ($\bar{\boldsymbol{x}}$). To provide the output data, $\bar{\boldsymbol{y}}$, we generate an executed plan for each decision point by altering the optimal solution. To this end, several business rules are defined and applied to the optimal solution. Table 1 shows the set of rules used in this work. Some settings have only one rule, while others combine several operational rules. The settings that are considered in Hewitt and Frejinger (2020) are either similar to `Set1` with one rule or equivalent to `Set4` and `Set5` with several regulations in the form of $(x_i \geq x_j)$. They also investigate a setting similar to `Set7`, where the corresponding business rule may vary depending on the day. Nonetheless, they choose between two rules randomly. Moreover, the authors do not consider testing on a setting similar to `Set2` and `Set3`, which are in place for all problem variables. As presented in the table, we attempt to evaluate the framework's performance on different types of conditional and non-conditional business rules.

Table 1: Rule settings for knapsack problem

Setting	Mathematical Representation of Rules	Setting	Mathematical Representation of Rules	Setting	Mathematical Representation of Rules
`Set1`	$x_1 \geq x_2$	`Set2`	if $x_i + x_{i+1} = 2 \rightarrow x_i = 0 \quad \forall i$	`Set3`	if $x_i + x_{i+1} + x_{i+2} \geq 2 \rightarrow x_i = 0 \quad \forall i$
`Set4`	$x_1 \geq x_2$ $x_{10} \geq x_{20}$ $x_{11} \geq x_{25}$ $x_{22} \geq x_{40}$ $x_{45} \geq x_{14}$ $x_{36} \geq x_{23}$	`Set5`	$x_1 \geq x_2, \quad x_{10} \geq x_{20}$ $x_{11} \geq x_{25}, \quad x_{22} \geq x_{40}$ $x_{45} \geq x_{14}, \quad x_{36} \geq x_{23}$ $x_{49} \geq x_{50}, \quad x_7 \geq x_{32}$ $x_{41} \geq x_{29}, \quad x_9 \geq x_{31}$	`Set6`	if $x_1 = 1$ & $x_3 = 1$ & $x_7 = 0 \rightarrow x_6 = 1$ if $x_1 = 0$ & $x_2 = 1$ & $x_5 = 1 \rightarrow x_3 = 0$ if $x_1 = 0$ & $x_3 = 0 \rightarrow x_6 = 1$ if $x_4 = 1$ & $x_7 = 0 \rightarrow x_9 = 0$ if $x_4 = 1$ & $x_7 = 1 \rightarrow x_9 = 1$ if $x_1 = 1$ & $x_2 = 1 \rightarrow x_3 = 0$
`Set7`	if $x_2 \leq 0.5 \rightarrow x_1 \geq x_2$ if $x_2 > 0.5 \rightarrow x_1 = 0$	`Set8`	if $x_2 > 0$ & $x_1 = 0 \rightarrow x_3 = 0$ if $x_2 = 0$ & $x_1 = 0 \rightarrow x_3 \geq x_4$	`Set9`	if $x_4 \leq 0.5 \rightarrow x_1 \geq x_2$ if $x_4 > 0.5 \rightarrow a_1x_1 + a_2x_2 + a_3x_3 \leq 0.4b$

In our experiments, we provide several rules close to business rules in industries. For instance, if the rules in `Set2` are applied in the case of the binary knapsack model, they restrict the loading of consecutive products together. As another example, consider the second rule of `Set9`. If the condition occurs, the total loaded portion of the first three products could not exceed 0.4 of the total container

capacity. Therefore, the output $\bar{\boldsymbol{y}}$ is generated by applying the rules on the optimal solutions. For example, imagine `Set1` is imposed in a company. If the solution of the knapsack problem suggests, $\bar{x_1} = 0$ and $\bar{x_2} = 0.5$, then the associated executed plan is considered as $\bar{y_1} = \bar{y_2} = 0.5$, even if the executed plan is not feasible for the GOP. In Table 2, we represent the functional mapping of some examples of the rules applied to the knapsack problem to elaborate on how we generate output data according to the obtained optimal solutions.

Table 2: Functional mapping of rules for knapsack problem

Mathematical Representationof Rule	Functional Mapping
$x_1 \geq x_2$	$\begin{cases} \bar{y_i} = max[\bar{x_1}, \bar{x_2}] & i = 1 \\ \bar{y_i} = \bar{x_i} & i = 2, \ldots, n \end{cases}$
$a_1x_1 + a_2x_2 + a_3x_3 \leq 0.4b$	$\begin{cases} \text{if } a_1\bar{x_1} + a_2\bar{x_2} + a_3\bar{x_3} \leq 0.4b \rightarrow \bar{y_i} = \bar{x_i} \quad i = 1, \ldots, n \\ \text{if } a_1\bar{x_1} + a_2\bar{x_2} + a_3\bar{x_3} > 0.4b \rightarrow \bar{y_i} = \begin{cases} min[\bar{x_3}, 0.4b/a_3] & i = 3 \\ min[\bar{x_2}, (0.4b - a_3\bar{y_3})/a_2] & i = 2 \\ min[\bar{x_1}, (0.4b - a_3\bar{y_3} - a_2\bar{y_2})/a_1] & i = 1 \\ \bar{x_i} & i = 4, \ldots, n \end{cases} \end{cases}$

4.1.2 Results

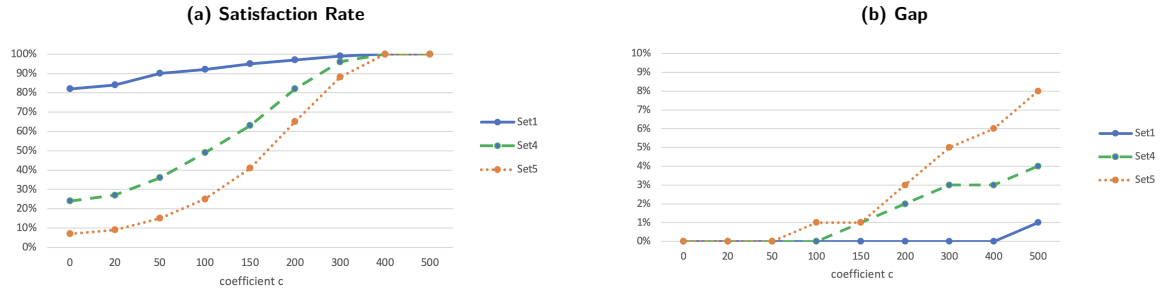
This section presents the framework's efficiency for linear and discrete knapsack problems under several rules defined in Table 1. All of the settings are investigated on the small data set with 15 variables, while `Set1`, `Set4`, and `Set5` are imposed on the more extensive data set with 50 variables. As mentioned in Section 3, there are two criteria to assess the quality of the obtained solutions by AOP. With the first measure, we determine the frequency of the instances in which the AOP provides a feasible solution regarding the hidden unmodeled constraints ($\boldsymbol{G}^t\boldsymbol{x} + \boldsymbol{H}^t\boldsymbol{v} \leq \boldsymbol{u}^t$) and we show it as a percentage of the whole data set. The following measure denotes the level of degradation in the objective function. We first need to solve the GOP for each instance to compute this measure, which we call the gap in the results in Table 3 and Table 4.

Next, we add the mathematical representation of the concealed rules as soft constraints. Then, we calculate the cost of the solutions obtained by the AOP and provide the percentage of the gap between these two objective functions by computing $(Z^t_{GOP-BR} - Z^t_{AOP})/Z^t_{GOP-BR}$. In this ratio, Z^t_{GOP-BR} represents the objective value of the GOP where it contains $\boldsymbol{G}^t\boldsymbol{x} + \boldsymbol{H}^t\boldsymbol{v} \leq \boldsymbol{u}^t$, and Z^t_{AOP} is calculated as $\sum_{j=1}^{n} r_j x_j^{AOP}$ for each instance t. This measure is only considered for the instances which pass the feasibility test, therefore, we report an average gap of all the satisfied instances in the results.

When solving the AOP, we change the penalty coefficient c value in the objective function and evaluate the performance for each c. In most experiments, increasing c boosts the satisfaction measure, but it stabilizes beyond a certain value of c. However, the degradation in the objective value usually escalates by growing c. To elaborate more, consider Figure 2 in which we fit elastic Net regression on data of solving linear knapsack problem instances, and then we employ the framework on our test set. We evaluate the framework for different values of c ranging from 0 to 500 for each of the three investigated sets of rules in the plots. They are all in the form of $x_i \leq x_j$, yet by different numbers of hidden operational rules. All the graphs in Figure 2 acknowledge that the satisfaction rate augments by increasing the value of c. In addition, we see that for all numbers of hidden business rules, the solutions of the AOP are far more likely to satisfy the regulations than the GOP ($c = 0$). Although this satisfaction converges to 100% in all three sets of rules, the gap typically surges according to Figure 2. Therefore, the best coefficient for `Set1` is 300 with no degradation in the objective value, while the best results are obtained for `Set4` and `Set5` in $c = 400$ with a gap equal to 3% and 6%, respectively. Comparing the graphs demonstrates that increasing the number of rules decreases quality measures.

We demonstrate the efficiency metrics for six settings tested on the linear knapsack problem as well as six settings on the binary knapsack problem in Tables 3 and 4, respectively, considering that the reported results are associated to the test instances. Remark that we only report the best results

Figure 2: Effect of coefficient c on the quality of the solutions of eNet linear regression on the linear knapsack problem



in terms of feasibility obtained by altering c and the associated gap. In the tables, we highlight the best-obtained satisfaction rate among all the methods as the more important measure; hence, if a tie occurs, we also highlight the best gap.

The first column of the results denoted in both tables of this section is related to the percentage of instances already feasible in the GOP regarding the concealed constraints ($c = 0$). In the following columns of the tables, we compare the performance of linear regression, lasso regression, ridge regression, and elastic Net regression on both linear and binary knapsack problems. The last column of each learning method reports the sparsity of the learned coefficient matrix. Furthermore, we investigate applying regression trees on the linear knapsack problem and classification trees on the 0-1 knapsack problem.

Based on the results in Table 3, we can see that, in most of the experiments, applying the framework enhances the feasibility level compared to the solutions of GOP ($c = 0$). Next, we compare the quality of the solutions regarding the objective function. Solving the AOP obtained by different methods yields a solution with objective function values close to the GOP values (the maximum average difference is 10%). The table suggests that the performance of lasso and eNet are similar, and, for the first three settings, they satisfy the constraints for all the instances. We observe that even simple linear regression without regularization boosts the feasibility rate. The only setting not significantly improved through the framework is `Set7`. One hypothesis is that since the regression score is not as good as the other settings, the regression equations cannot represent the constraints well. However, investigating all the settings with different machine learning methods, we could not concretely conclude that there is always a significant, meaningful relation between the learning score and the framework's performance. Another possible effective attribute is the sparsity of the coefficient matrix. Looking into both Tables 3 and 4 depicts that in most cases, regression methods with a sparser coefficient matrix outperform those employing a denser coefficient matrix. As an example, consider `Set4`, in which, although the learning scores are very close to each other, $r^2 = 0.85$ for linear regression and $r^2 = 0.86$ for eNet, the sparsity of the coefficient matrix in eNet results in a better quality of solutions in terms of satisfaction rate.

Table 3: Linear Knapsack problem results

Setting	Satisfaction c = 0	Linear Regression			Lasso Linear Regression			Ridge Linear Regression			eNet Linear Regression			Regression Tree	
		Satisfaction	Gap	Sparsity	Satisfaction	Gap	Sparsity	Satisfaction	Gap	Sparsity	Satisfaction	Gap	Sparsity	Satisfaction	Gap
Set1	81%	92%	0%	0%	**100%**	**0%**	74%	93%	0%	0%	**100%**	**0%**	80%	82%	0%
Set4	24%	54%	3%	0%	**100%**	**3%**	88%	62%	4%	0%	**100%**	**3%**	93%	24%	0%
Set5	7%	27%	10%	0%	**100%**	**6%**	88%	40%	7%	0%	**100%**	**6%**	91%	7%	0%
Set7	85%	85%	0%	0%	85%	0%	40%	85%	0%	0%	85%	0%	87%	**86%**	0%
Set8	75%	**86%**	**4%**	0%	82%	3%	67%	**86%**	**4%**	0%	82%	3%	67%	80%	0%
Set9	75%	**97%**	3%	0%	**97%**	**2%**	76%	**97%**	**2%**	0%	**97%**	**2%**	73%	80%	0%

The next set of experiments belongs to the binary knapsack problem depicted in Table 4. One notable observation is that the table confirms that the framework is even more efficient for the binary knapsack problem. Regardless of the machine learning method type, most solutions are satisfied after

Table 4: Binary Knapsack problem results

Setting	Satisfaction c = 0	Linear Regression			Lasso Linear Regression			Ridge Linear regression			eNet Linear Regression			Classification Tree	
		Satisfaction	Gap	Sparsity	Satisfaction	Gap	Sparsity	Satisfaction	Gap	Sparsity	Satisfaction	Gap	Sparsity	Satisfaction	Gap
Set2	23%	**100%**	15%	0%	**100%**	12%	62%	**100%**	20%	0%	**100%**	8%	69%	**100%**	**0%**
Set3	6%	**100%**	30%	0%	**100%**	29%	58%	**100%**	30%	0%	**100%**	29%	70%	**100%**	**19%**
Set6	52%	96%	29%	0%	98%	24%	42%	98%	29%	0%	98%	27%	38%	**100%**	12%
Set7	90%	**100%**	1%	0%	**100%**	1%	67%	**100%**	1%	0%	**100%**	1%	67%	**100%**	**0%**
Set8	80%	**100%**	5%	0%	**100%**	**3%**	33%	**100%**	5%	0%	**100%**	**3%**	53%	**100%**	**3%**
Set9	77%	97%	3%	0%	97%	3%	71%	97%	3%	0%	97%	3%	44%	**99%**	0%

employing the framework. The AOP could pass the feasibility test for all the instances for four of the six tested settings. This is the case even for Set2 and Set3, with much less frequency of feasibility for the GOP solutions ($c = 0$). Next, we will turn to more detailed comparisons between the methods. Although lasso and eNet regressions still outperform ridge and linear regressions, the classification tree model achieves the best results in satisfaction and gap, specifically, for the first three sets of operational rules. The average gap obtained by the classification tree is very competitive. At the same time, the solutions satisfy the hidden constraints for all of the test instances.

Looking at both tables simultaneously and comparing quality measures for all the methods confirms the superiority of eNet and lasso linear regression over the other methods. The coefficient matrices are always very sparse in these models. In addition, the classification tree model is the most efficient method for 0-1 knapsack problem instances.

4.2 Nurse rostering problem

We investigate the framework's performance on a practical and complex optimization problem. Nurse rostering (De Causmaecker and Vanden Berghe 2011) is an essential aspect of healthcare management for scheduling nurses with different skills and has received considerable attention in recent years. Nurse rostering involves assigning shifts to nurses for a time horizon while restricting them with hard and soft constraints. The main objective is to optimize the utilization of limited resources, thereby enhancing the hospital's operational efficiency. However, it is essential to prioritize the job satisfaction of nurses in the process. The nurse rostering problem (NRP) is a complex combinatorial optimization problem known as NP-hard (De Causmaecker and Vanden Berghe 2011). Generally, hospitals dedicate considerable effort to creating high-quality rosters for their nurses, including using decision-support tools and expert judgments.

In modeling the NRP, the constraints taken into account and the objectives could vary from hospital to hospital or during different planning periods within a hospital. This diversity leads to a wide range of nurse rostering optimization models. Moreover, the solution methodologies proposed in the literature for this problem originate in different categories of MILP, CP, heuristics, meta-heuristics, stochastic programming, etc. We refer the reader to Ngoo et al. (2022) and the references therein for more details.

This section introduces a basic NRP formulated and solved using MILP. We then examine utilizing the proposed data-driven methodology to tackle this problem. Note that this study represents an initial attempt in this field, and our objective is to gauge the method's efficacy when confronted with a limited amount of training data. Hence, we restrict our analysis to scenarios where all nurses are identical regarding their skills and preferences. This decision allows us to provide additional training data from the available dataset, and we outline the details of how this is achieved later in this section.

Inspired by the models presented in Ceschia et al. (2019), Strandmark et al. (2020), Goh et al. (2022), a MILP formulation of the problem is given below:

The objective function is to minimize the total under and overstaffing:

$$\min \quad \sum_{j=1}^{s} \sum_{d=1}^{m} (\alpha w_{jd} + \alpha' w'_{jd}) \tag{25}$$

Parameters	
N	Set of nurses: $\{1, \ldots, n\}$.
P	Set of days in the planning horizon: $\{1, \ldots, p\}$.
S	Set of shift types: $\{1, \ldots, s\}$ = (morning, evening, over-night).
D_{jd}	Nurse demand for shift j in day d.
α	Penalty for understaffing.
α'	Penalty for overstaffing.
K_1	The minimum number of shifts each nurse must be assigned.
K_2	The maximum number of shifts each nurse can be assigned.
K_3	The maximum number of consecutive days each nurse can work.
K_4	The minimum number of free days each nurse must have after 2 consecutive overnight shift.
K_5	The maximum number of overnight shifts each nurse can be assigned.
Decision Variables	
$x_{ijd} \in \{0, 1\}$	1 if nurse i assigned to shift j on day d, 0 otherwise.
$w_{jd} \geq 0$	Total understaffing for shift j on day d.
$w'_{jd} \geq 0$	Total overstaffing for shift j on day d.

Hence, the covering constraints to compute the understaffing and overstaffing are given:

$$w_{jd} \geq D_{jd} - \sum_i x_{ijd} \qquad \forall d, \forall j \tag{26}$$

$$w'_{jd} \geq \sum_i x_{ijd} - D_{jd} \qquad \forall d, \forall j \tag{27}$$

The hard constraints of the problem which have to be satisfied are as follows:

- **C1:** Any nurse can work no more than one shift per day.

$$\sum_{j=1}^{s} x_{ijd} \leq 1 \qquad \forall i, \forall d \tag{28}$$

(29)

- **C2:** The total number of shifts assigned to each nurse must be between minimum and maximum.

$$K_1 \leq \sum_{j=1}^{s} \sum_{d=1}^{p} x_{ijd} \leq K_2 \qquad \forall i \tag{30}$$

- **C3:** The number of consecutive working days assigned to a nurse must not exceed the maximum allowed.

$$\sum_{j=1}^{s} \sum_{d=k}^{k+K_3} x_{ijd} \leq K_3 \qquad \forall i, \forall k : 1, \ldots, (p - K_3) \tag{31}$$

- **C4:** A nurse has to have a minimum number of free days after 2 consecutive night shifts.

$$\sum_{d=k}^{k+1} x_{i3d} + \sum_{j=1}^{s} x_{ij(k+1+r)} \leq 2 \qquad \forall i, \forall k : 1, \ldots, (p - K_4 - 1), \forall r : 1, \ldots, K4 \tag{32}$$

- **C5:** A nurse cannot exceed a maximum number of night shifts during the planning horizon.

$$\sum_{d=1}^{p} x_{i3d} \leq K_5 \qquad \forall i \tag{33}$$

This is the basic model we specify as our GOP of nurse rostering. In the next subsection, we clarify the business rules we consider for testing and how we generate input and output for the machine learning models.

4.2.1 Data generation

To generate our historical data, similar to the procedure carried out for the knapsack problem, we need to solve the NRP for each planning period t. Without loss of generality, we assume the regulations and demand can vary from time to time in a hospital. Therefore, for each planning period, we randomly generate our model's parameters $D_{j,d}, K_1, K_2, K_3, K_4$ and K_5 from uniform distributions of $[0,3], [4,8], [8,9], [5,6], [1,2]$ and $[2,5]$, respectively. Note that we implement and report two separate sets of tests, wherein one of them, we consider penalty coefficients α and α' equal to 1, meaning that the hospital attempts to avoid understaffing and overstaffing with the same weight. In the other set of tests, replacing α by 3 and α' by 1, we assess the scenario in which understaffing is more unfavorable for the hospital than overstaffing.

As discussed in Section 4.2, in the current study, we assume the nurses are identical so that they can replace one another, and therefore, the solutions of the NRP are symmetric. This assumption generates more training data from an initial set of solved optimization instances. All generated data sets consist of 200 initial planning periods divided into 80% for training and 20% for testing. However, after solving this set of instances, we use the optimal solutions to provide more training data by considering every schedule for each nurse as a training instance. Thus, the new set of cases is multiplied by the number of nurses.

By solving the generated NRP instances, we acquire the input features of our learning models. Hence, following the same steps as the knapsack problem, we alter the optimal solution based on some assumed operational rules to generate output features. In Table 5, we display three different business rules not included in the GOP, and executable plans should satisfy them. To indicate the usability of the proposed framework in a real industrial problem, we devise some business rules that are practical and interpretable in the NRP (Haspeslagh et al. 2010, Goh et al. 2022). In addition to the logical representation of the rules in this table, the constraints added to the MILP model are described in Table 6. In this table, for each set of rules, we represent the counterpart linearized constraint where z_i is a binary variable. We add the linear constraints to the NRP to solve the problem with known constraints and evaluate the performance of learned constraints by computing gap.

Table 5: Rule setting for nurse rostering problem

Setting	Description of Rule	Logical/Mathematical Representation of Rule
`Set1`	After a night shift on, the following morning shift is off	if $x_{i3d} = 1 \rightarrow x_{i1(d+1)} = 0$ $\forall i, \forall d : 1, \ldots, p-1$
`Set2`	After a night shift on the following morning and afternoon shifts are off	if $x_{i3d} = 1 \rightarrow x_{i1(d+1)} + x_{i2(d+1)} = 0$ $\forall i, \forall d : 1, \ldots, p-1$
`Set3`	After a shift in a weekend on, the two first week-days are off	if $\sum_j (x_{i,j,7q-1} + x_{i,j,7q}) \geq 1 \rightarrow \sum_j (x_{i,j,7q+1} + x_{i,j,7q+2}) = 0$ $\forall i, \forall q : 1, \ldots, \lfloor p/7 \rfloor$

Table 6: MIP constraint representation of rules for nurse rostering problem

Setting	Added MIP Constraints	
`Set1`	$x_{i,3,d} + x_{i,1,d+1} \leq 1$	$\forall i \in N, d : 1, \ldots, p-1$
`Set2`	$2(1 - x_{i,3,d}) \geq x_{i,1,d+1} + x_{i,2,d+1}$	$\forall i \in N, d : 1, \ldots, p-1$
`Set3`	$\sum_j (x_{i,j,7q-1} + x_{i,j,7q}) \leq 5z_i + 0.001z_i + 0.999$	$\forall i \in N, \forall q : 1, \ldots, \lfloor p/7 \rfloor$
	$\sum_j (x_{i,j,7q} + x_{i,j,7q+1}) \leq 6(1 - z_i)$	$\forall i \in N, \forall q : 1, \ldots, \lfloor p/7 \rfloor$
	$\sum_j (x_{i,j,7q} + x_{i,j,7q+1}) \geq 1 - z_i$	$\forall i \in N, \forall q : 1, \ldots, \lfloor p/7 \rfloor$

We note that where we test the rules associated with `Set1` and `Set2`, we solve the NRP model of (25–33); nonetheless, to test `Set3` and to deal with a more complicated problem, we consider `Set2` as a part of constraints in the GOP model of NRP in addition to constraints (C1–C5).

Since the NRP problem's principal decision variable, x_{ijd}, has three indices, the regression and branching equations in the corresponding AOP slightly differ from the knapsack problem. As an instance, the Equation 16 is replaced by the following:

$$y_{ijd} - (\hat{\beta}^{ijdl} + \sum_{efg}(\beta_{efg}^{ijdl} x_{efg})) \leq M(1 - z^{ijdt}) \qquad \forall i, j, d \; \forall l \in L \tag{34}$$

And in the case of branching instead of (18) in the AOP, we embed the following equation:

$$\sum_{efg}(h_{efg}^{ijdm} x_{efg}) \leq g^{ijdm} + (1 - z^{ijdl}) \qquad \forall i, j, d \; \forall l \in leaf, \forall m \in Left(l) \tag{35}$$

Since the nurses are interchangeable, we also need to learn the models based on two indices, j and d. Nevertheless, the equations in the AOP must be compatible with the true decision variable of the problem x_{ijd}. Therefore, the Equations (34) and (35) are reformulated as below, respectively:

$$y_{ijd} - (\hat{\beta}^{jdl} + \sum_{fg}(\beta_{fg}^{jdl} x_{ifg})) \leq M(1 - z^{jdl}) \qquad \forall i, j, d \; \forall l \in L \tag{36}$$

$$\sum_{fg}(h_{fg}^{jdm} x_{ifg}) \leq g^{jdm} + (1 - z^{jdl}) \qquad \forall i, j, d \; \forall l \in leaf, \forall m \in Left(l) \tag{37}$$

4.2.2 Results

To showcase the framework's performance for the nurse rostering problem, we provide information on four types of gaps and the percentage of rule satisfaction for the test instances. The gaps are defined in Table 7. The first computed gap is similar to the previously reported for the knapsack problems. It illustrates the degradation in the objective function of the problem and is expressed as a percentage of the aim of AOP value. The second type of gap measures the difference between the AOP and the GOP solutions with known constraints regarding the total deviation from the demand. The understaffing and overstaffing gaps are included to provide information on the variations between the AOP and the GOP solutions with additional known constraints regarding nurse shortage and over-coverage during the planning period. We use the absolute value for overstaffing and understaffing gaps, and all four reported gaps are the average of the gaps of all test instances.

Table 7: Definition of computed gaps for nurse rostering problem

Different Computed Gaps	Definition	Computation
Gap1	**Objective Function Degradation**	$(Z^t_{AOP-BR} - Z^t_{GOP})/Z^t_{AOP-BR}$
Gap2	**Total Demand Deviation Gap**	$\sum_{jd}(w + w')^{\text{AOP}} - \sum_{jd}(w + w')^{\text{GOP}}$
Gap3	**Understaffing Gap**	$\sum_{jd}(w)^{\text{AOP}} - (w)^{\text{GOP}}$
Gap4	**Overstaffing Gap**	$\sum_{jd}(w')^{\text{AOP}} - (w')^{\text{GOP}}$

To compare the quality of solutions obtained by the AOP modeled by linear regression and the AOP modeled by classification tree, we provide two tables for the NRP. Table 8 is associated with the nurse rostering problem, where avoiding understaffing is more critical for the decision-makers in the hospital. In contrast, Table 9 displays the results where understaffing and overstaffing have equal weight. For both experiments, we test the framework on various rule settings and problem sizes, where n is the number of nurses and p corresponds with the planning horizon. Like the knapsack problem experiments, we tune the coefficient c in the process. In this problem, we vary c from 0.1 to 15, yet only report the best results.

To ensure meaningful and comparable results, we include a column in the tables that indicates the average nurse demand for the planning period in the test set. This enables us to compare the gaps to the demands for each test and draw conclusions based on the findings.

Table 8: Results of applying the framework on different settings for nurse rostering problem – $\alpha = 3, \alpha' = 1$

Setting	Problem Size	Satisfaction c = 0	Demand	Linear Regression						eNet Linear Regression						Classification Tree				
				Satisfaction	Gap1	Gap2	Gap3	Gap4	Sparsity	Satisfaction	Gap1	Gap2	Gap3	Gap4	Sparsity	Satisfaction	Gap1	Gap2	Gap3	Gap4
Set1	$n = 6$ p=10	62%	44.2	**100%**	30%	4.30	3.18	1.12	0%	**100%**	2%	0.12	0.10	0.02	43%	**100%**	**0%**	**0**	**0**	**0**
Set2	$n = 6$ p=10	27%	44.2	**100%**	60%	10.25	6.30	3.95	0%	**100%**	56%	8.62	6.05	2.57	49%	**100%**	**0%**	**0**	**0**	**0**
Set2	$n = 7$ p=12	15%	54.1	**100%**	76%	21.57	13.57	8.00	0%	**100%**	74%	18.70	12.82	5.87	46%	**100%**	**0%**	**0**	**0**	**0**
Set3	$n = 7$ p=12	0%	54.1	**100%**	42%	8.12	5.00	3.12	0%	**100%**	41%	7.67	4.80	2.87	57%	**100%**	**1%**	**0.10**	**0.08**	**0.02**

Table 9: Results of applying the framework on different settings for nurse rostering problem – $\alpha = \alpha' = 1$

Setting	Problem Size	Satisfaction c = 0	Demand	Linear Regression						eNet Linear Regression						Classification Tree				
				Satisfaction	Gap1	Gap2	Gap3	Gap4	Sparsity	Satisfaction	Gap1	Gap2	Gap3	Gap4	Sparsity	Satisfaction	Gap1	Gap2	Gap3	Gap4
Set1	$n = 6$ p=10	50%	44.2	**100%**	46%	5.25	2.97	2.27	0%	**100%**	44%	4.62	4.17	0.45	43%	**100%**	**0%**	**0**	**0**	**0**
Set2	$n = 6$ p=10	30%	44.2	**100%**	72%	19.05	11.70	7.35	0%	**100%**	62%	10.90	8.10	2.80	50%	**100%**	**0%**	**0**	**0**	**0**
Set2	$n = 7$ p=12	7%	54.1	**100%**	79%	24.82	16.37	8.45	0%	**100%**	78%	23.72	16.02	7.70	48%	**100%**	**0%**	**0**	**0**	**0**
Set3	$n = 7$ p=12	0%	54.1	**100%**	40%	7.10	4.65	2.45	0%	**100%**	38%	6.75	4.40	2.35	56%	**100%**	**1%**	**0.10**	**0.08**	**0.02**

One remark that should be considered is that solving the nurse rostering problem using the framework with an embedded classification tree is very fast, taking less than 20 seconds for each instance. However, we have noticed that solving some instances using linear regression equations to optimality is very time-consuming. As a result, in our experiments, we limit the time for solving each AOP model to 600 seconds.

Looking into the satisfaction rates presented in Tables 8 and 9, the AOP with linear regression models and classification trees is feasible for all instances' operational rules. This satisfaction is obtained when we increase the coefficient c to some extent. Next, we focus on the various reported gaps in the tables. We observe that the classification tree consistently outperforms the embedded linear regression models regarding all gap types for all test settings. A closer look into the result tables reveals that even for the settings where the objective function degradation for linear regression models exceeds 70%, the classification tree achieves 0% gap. This is because we need to increase the coefficient c significantly for embedded linear regression models to align the optimal solution with the prediction of the executable plan and satisfy the hidden rules. In contrast, embedding the obtained classification tree in the framework allows us to achieve feasible solutions regarding the operational rules with a small value of c (e.g., $c = 0.1$). Although the optimal solution obtained by the GOP with known rules and the AOP based on classification trees may vary, the demand covering, and consequently, the objective value remains almost the same for all instances. Here, we only evaluate the performance of linear regression without regularization and eNet, as they represent the least and most effective approaches, respectively. We confirm the superiority of eNet over linear regression without regularization for the NRP and the knapsack problem.

Considering both tables, it can be deduced that although adding more complicated business rules and increasing the size of the problem results in less rule satisfaction in the GOPs, embedding the machine learning models remains efficient in satisfying the hidden rules. More significantly, using the framework with the classification tree is highly effective even in the last setting with a larger instance size regarding satisfaction and all types of gaps.

5 Conclusion

In this study, we extend a data-driven framework that was recently proposed to learn and incorporate customization constraints of the classical linear programming models using linear regression and statistical learning. We include more complex machine learning models (lasso, ridge, eNet regularization, and decision tree) into the framework, evaluate their performance on more complex discrete models, and test conditional rules.

Furthermore, we empirically validate the method on discrete problems for the first time. We investigate the framework's performance on the binary knapsack problem and a more realistic and complex problem in nurse rostering, where operational rules are more practical and interpretable.

The study's notable outcome is that using different types of regularized linear regression enhances the results in most instances of all problems compared to linear regression without regularization. Moreover, we conclude that embedding classification trees in binary problems outperform embedding regression models to a great extent.

Finally, this study is still at an exploratory level. As an example, while learning linear models and embedding them in the framework is less computationally expensive, investigating the application of other machine learning methods like neural networks is of interest (Fischetti and Jo 2018). Moreover, a possible future research direction is learning and implementing separating hyperplanes (Bertsimas and Dunn 2017) in the framework for more complex rules.

References

Augustin A, Jouvet P, Lahrichi N, Lodi A, Rousseau LM (2022) A data-driven approach to include availability of icu beds in the planning of the operating room. Omega 109:102608.

Beldiceanu N, Simonis H (2016) Modelseeker: Extracting global constraint models from positive examples. Data Mining and Constraint Programming, 77–95 (Springer).

Bengio Y, Lodi A, Prouvost A (2021) Machine learning for combinatorial optimization: a methodological tour d'horizon. European Journal of Operational Research 290(2):405–421.

Bergman D, Huang T, Brooks P, Lodi A, Raghunathan AU (2022) Janos: an integrated predictive and prescriptive modeling framework. INFORMS Journal on Computing 34(2):807–816.

Bertsimas D, Dunn J (2017) Optimal classification trees. Machine Learning 106:1039–1082.

Bessiere C, Koriche F, Lazaar N, O'Sullivan B (2017) Constraint acquisition. Artificial Intelligence 244:315–342.

Bottmer L, Croux C, Wilms I (2022) Sparse regression for large data sets with outliers. European Journal of Operational Research 297(2):782–794.

Breiman L (2001) Random forests. Machine learning 45:5–32.

Breiman L, Friedman JH, Olshen RA, Stone CJ (2017) Classification and regression trees (Routledge).

Ceccon F, Jalving J, Haddad J, Thebelt A, Tsay C, Laird CD, Misener R (2022) Omlt: Optimization & machine learning toolkit. The Journal of Machine Learning Research 23(1):15829–15836.

Ceschia S, Dang N, De Causmaecker P, Haspeslagh S, Schaerf A (2019) The second international nurse rostering competition. Annals of Operations Research 274(1–2):171–186.

De Causmaecker P, Vanden Berghe G (2011) A categorisation of nurse rostering problems. Journal of Scheduling 14:3–16.

Dunn JW (2018) Optimal trees for prediction and prescription. Ph.D. thesis, Massachusetts Institute of Technology.

Elmachtoub AN, Grigas P (2022) Smart "predict, then optimize". Management Science 68(1):9–26.

Fajemisin A, Maragno D, Hertog Dd (2021) Optimization with constraint learning: A framework and survey. arXiv preprint arXiv:2110.02121.

Ferreira KJ, Lee BHA, Simchi-Levi D (2016) Analytics for an online retailer: Demand forecasting and price optimization. Manufacturing & service operations management 18(1):69–88.

Fischetti M, Jo J (2018) Deep neural networks and mixed integer linear optimization. Constraints 23(3):296–309.

Goh SL, Sabar NR, Abdullah S, Kendall G, et al. (2022) A 2-stage approach for the nurse rostering problem. IEEE Access 10:69591–69604.

Gurobi Optimization, Inc (2012) Gurobi optimizer reference manual. URL http://www.gurobi.com.

Haspeslagh S, De Causmaecker P, Stølevik M, Schaerf A (2010) First international nurse rostering competition 2010 (august 10-13, 2010, belfast, uk). PATAT 2010-Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling, Belfast, Northern-Ireland, UK.

Hewitt M, Frejinger E (2020) Data-driven optimization model customization. European Journal of Operational Research 287(2):438–451.

Junior Mele U, Maria Gambardella L, Montemanni R (2021) Machine learning approaches for the traveling salesman problem: A survey. 2021 The 8th International Conference on Industrial Engineering and Applications (Europe), 182–186.

Kafaei P, Cappart Q, Renaud MA, Chapados N, Rousseau LM (2021) Graph neural networks and deep reinforcement learning for simultaneous beam orientation and trajectory optimization of cyberknife. Physics in Medicine & Biology 66(21):215002.

Khaniyev T (2018) Data-driven structure detection in optimization: Decomposition, hub location, and brain connectivity.

Kumar M, Kolb S, De Raedt L, Teso S (2021) Learning mixed-integer linear programs from contextual examples. arXiv preprint arXiv:2107.07136 .

Liu S, He L, Max Shen ZJ (2021) On-time last-mile delivery: Order assignment with travel-time predictors. Management Science 67(7):4095–4119.

Lombardi M, Milano M, Bartolini A (2017) Empirical decision model learning. Artificial Intelligence 244:343–367.

Ngoo CM, Goh SL, Sabar NR, Abdullah S, Kendall G, et al. (2022) A survey of the nurse rostering solution methodologies: The state-of-the-art and emerging trends. IEEE Access .

Nguyen DT, Adulyasak Y, Cordeau JF, Ponce SI (2022) Data-driven operations and supply chain management: established research clusters from 2000 to early 2020. International journal of production research 60(17):5407–5431.

Nicosia G, Pardalos P, Giuffrida G, Umeton R (2017) Machine Learning, Optimization, and Big Data: Third International Conference, MOD 2017, Volterra, Italy, September 14–17, 2017, Revised Selected Papers, volume 10710 (Springer).

Ning C, You F (2018) Data-driven stochastic robust optimization: General computational framework and algorithm leveraging machine learning for optimization under uncertainty in the big data era. Computers & Chemical Engineering 111:115–133.

Pawlak TP, Krawiec K (2017) Automatic synthesis of constraints from examples using mixed integer linear programming. European Journal of Operational Research 261(3):1141–1157.

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12:2825–2830.

Pham TS, Legrain A, De Causmaecker P, Rousseau LM (2023) A prediction-based approach for online dynamic appointment scheduling: A case study in radiotherapy treatment. INFORMS Journal on Computing.

Schede EA, Kolb S, Teso S (2019) Learning linear programs from data. 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI), 1019–1026 (IEEE).

Strandmark P, Qu Y, Curtois T (2020) First-order linear programming in a column generation-based heuristic approach to the nurse rostering problem. Computers & Operations Research 120:104945.

Tsouros DC, Stergiou K (2020) Efficient multiple constraint acquisition. Constraints 25(3):180–225.

Vanderschueren T, Verdonck T, Baesens B, Verbeke W (2022) Predict-then-optimize or predict-and-optimize? an empirical evaluation of cost-sensitive learning strategies. Information Sciences 594:400–415.

Zou H, Hastie T (2005) Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society Series B: Statistical Methodology 67(2):301–320.