Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Invited Review

Optimization with constraint learning: A framework and survey

Adejuyigbe O. Fajemisin, Donato Maragno, Dick den Hertog[*]

Amsterdam Business School, University of Amsterdam, Amsterdam 1018TV, the Netherlands

ARTICLE INFO

Article history:
Received 17 December 2021
Accepted 27 April 2023

Keywords:
Analytics
Optimization
Constraint learning
Machine learning

ABSTRACT

Many real-life optimization problems frequently contain one or more constraints or objectives for which there are no explicit formulae. If however data on feasible and/or infeasible states are available, these data can be used to learn the constraints. The benefits of this approach are clearly seen, however, there is a need for this process to be carried out in a structured manner. This paper, therefore, provides a framework for Optimization with Constraint Learning (OCL) which we believe will help to formalize and direct the process of learning constraints from data. This framework includes the following steps: (i) setup of the conceptual optimization model, (ii) data gathering and preprocessing, (iii) selection and training of predictive models, (iv) resolution of the optimization model, and (v) verification and improvement of the optimization model. We then review the recent OCL literature in light of this framework and highlight current trends, as well as areas for future research.

1. Introduction

1.1. The synergy between optimization and machine learning

The synergy between mathematical optimization and Machine Learning (ML) has come to the forefront in recent years and has been discussed in depth in the recent literature. Sun et al. (2019) provide a recent survey of optimization methods from an ML perspective. In another survey, Gambella et al. (2021) present fundamental ML tasks such as classification, regression, clustering, adversarial learning, and others as optimization problems. In the optimization field, more and more attention is being given to using ML to aid the formulation and solution of optimization problems. Practical optimization problems can be large and difficult to solve, and more and more authors are exploiting the flexibility and usefulness of ML to aid in the solution of difficult optimization problems (e.g. Abbasi et al., 2020; Fischetti & Fraccaro, 2019). In addition to speeding up the solution of problems, ML and deep learning have also been used to speed up tree search in Hottung et al. (2020), accelerate the Branch-and-Price Algorithm in Vaclavik et al. (2018), and guide branching in Mixed Integer Programs (MIP) in Khalil et al. (2016). ML has also been used to improve meta-heuristic solution approaches (Karimi-Mamaghan et al., 2022). A recently published survey by Bengio et al. (2021) provides an overview of recent attempts to leverage machine learning to solve combinatorial optimization problems.

1.2. Motivation for this study

The examples in the previous section use ML to learn or obtain solutions to optimization problems in a more efficient manner. In our view, however, there is even more value in using ML for optimization: ML can be used to learn constraints. Many real-life optimization problems contain one or more constraints or objectives for which there are no explicit formulae. However when there is data available, one can use the data to learn the constraints. Two such examples are given in Section 2. These examples show the difficulty of formulating some constraints using explicit formulae and highlight the usefulness of being able to learn such constraints from available data. In addition to using traditional ML approaches to learn constraints for optimization problems (e.g. regression trees and linear regression in Verwer et al., 2017 and neural networks in Villarrubia et al., 2018), other techniques such as genetic programming (e.g. in Pawlak & Krawiec, 2018), and mixed-integer linear programming (MILP) (e.g. in Pawlak & Krawiec, 2017a) have also been used. In our literature search, we have noticed several works of this kind from several different application areas and even several scientific fields. However, it seems beneficial to have these works collated and discussed in one place. Additionally, we propose to take a more holistic view of the entire constraint learning process, starting from defining the base problem, to the resolution and validation of the final optimization model.

[*] Corresponding author.
E-mail addresses: a.o.fajemisin2@uva.nl (A.O. Fajemisin), d.maragno@uva.nl (D. Maragno), d.denhertog@uva.nl (D. den Hertog).

https://doi.org/10.1016/j.ejor.2023.04.041

This paper is therefore intended to provide both a survey of literature on constraint learning for optimization, as well as offer a framework for effectively facilitating Optimization with Constraint Learning (OCL). The five steps of the proposed framework are: (i) setup of the conceptual optimization model, (ii) data gathering and processing, (iii) selection and training of the predictive model(s), (iv) resolution of the optimization model, and (v) validation/improvement of the optimization model. These steps will be discussed in more detail throughout this paper. We use the term “Constraint Learning” to make clear that we are interested in cases in which most of the optimization model is known, but some of the constraints are difficult to model explicitly. This is to differentiate from other terms such as “model learning” (Donti et al., 2017) and “model seeking” (Beldiceanu & Simonis, 2016) seen in the literature. We should also note that there are several approaches in the literature that are similar to *Constraint Learning* but do not fit in this survey for several reasons. In Constraint Programming (CP) for example, the process of learning CP models from data (positive and/or negative examples) has been widely studied (Beldiceanu & Simonis, 2016; Bessiere et al., 2004; Galassi et al., 2018; Kolb, 2016; O’Sullivan, 2010). These are not included as we are interested in cases where predictive models are embedded as constraints in mathematical optimization.

We also exclude papers on the subject of Satisfiability Modulo Theories (SMT) (Nieuwenhuis & Oliveras, 2006) from our survey. Additionally, papers such as Bertsimas & Kallus (2020); Demirovic et al. (2019); Elmachtoub & Grigas (2022) and Villarrubia et al. (2018) which focus on predicting problem *parameters*, or use predictive models to approximate *known* objective functions and constraints are excluded. We also exclude papers where the structure of the problem is known, but predictive models are used to replace the problem in order to speed up computation (Nagata & Chu, 2003; Nascimento et al., 2000; Venzke & Chatzivasileiadis, 2020; Venzke et al., 2020). Similarly, papers like Shang et al. (2017) and Han et al. (2021) which use predictive models to learn the uncertainty set for robust optimization problems are not considered. Finally, papers which show an interplay between ML and optimization in some way but do not focus on constraint learning (e.g. Bagloee et al., 2018; Gilan & Dilkina, 2015; Say et al., 2020; Say & Sanner, 2018) are also out of the scope of this survey.

While Lombardi & Milano (2018) and Kotary et al. (2021) provide short surveys on OCL, we present a more extensive framework for constraint learning and consider a wider range of the literature. Papers identified in our literature search will be discussed in light of the proposed framework, and gaps in knowledge and scope for future lines of research will be identified. Running examples will be used to illustrate the steps of the framework. It is envisaged that in addition to providing a review as well as the framework, our work will bring together ideas from diverse fields.

1.3. Outline

This framework and survey is structured as follows. In Section 2, we first present the running examples that will be used to illustrate our framework. We then describe this framework in Section 3 and review the recent literature on light of it in Section 4. Areas for further research are presented in Section 5, with general conclusions given in Section 6.

2. Running examples

In order to illustrate the concepts presented in later sections, we first present two practical optimization problems in which constraint learning plays an important role.

2.1. Palatability considerations in WFP food baskets

This first example deals with the issue of providing palatable rations in the food baskets delivered by the World Food Programme (WFP) to population groups in need. These food baskets are designed to meet the nutritional requirements of a population and take into account existing levels of malnutrition and disease, climatic conditions, activity levels, and so on (UNWFP, 2021). In addition to considering the nutritional content, we would also like to be able to consider the palatability of the rations. This problem is particularly relevant in that the palatability of a food basket can change drastically between different groups of beneficiaries. Since the palatability constraint is difficult to model manually, and a data set with different food baskets and their palatability scores is available, a predictive model can be used to derive the palatability constraint from the data, which can then be embedded into the original food basket optimization problem.

Let $\mathcal{K}$ be the set of commodities (such as rice, flour, chickpeas, sugar, etc.) that may be included in a food basket, and $\mathcal{L}$ be the set of nutrients (such as vitamins, minerals, proteins, etc.) which are important for maintaining good health. If the nutritional requirement for nutrient $l \in \mathcal{L}$ (in grams/person/day) is denoted by $Nutreq_l$, and the nutritional value for nutrient l (per grams of commodity $k \in \mathcal{K}$) is denoted by $Nutval_{kl}$, then a simple model for food basket optimization problem can be given as

$$\min_{\boldsymbol{x}} \boldsymbol{c}^T \boldsymbol{x} \tag{1a}$$

$$\text{s.t.} \sum_{k\in\mathcal{K}} Nutval_{kl}\, x_k \geq Nutreq_l \qquad \forall l \in \mathcal{L} \tag{1b}$$

$$Palatability\_score(\boldsymbol{x}) \geq P_{\min} \tag{1c}$$

$$x_k \geq 0 \qquad \forall k \in \mathcal{K}, \tag{1d}$$

where x_k is the amount of each commodity $k \in \mathcal{K}$ (measured in grams) included in the food ration, $\boldsymbol{c} \in \mathbb{R}^{|\mathcal{K}|}$ is the cost vector associated with the commodities (in \$/gram), and the objective function (1a) minimizes the ration cost. Constraint (1b) is used to ensure a minimum level of nutritional requirements in the rations. Constraint (1c) has been learned using a predictive model and is used to ensure that the palatability of the ration is at least equal to $P_{\min}$. The diet problem in (1a) to (1d) is only a part of a much bigger model (see Peters et al., 2021) that also includes the sourcing, facility location and transportation problems. The constraint to be learned here is a function of relatively few variables ($|\mathcal{K}|$ in this case), compared with the thousands of variables in the full model.

The predictive model is trained on a sparse data set $\boldsymbol{D}$ in $\mathbb{R}^{n\times(|\mathcal{K}|+1)}$ composed of n samples, with each sample composed of $|\mathcal{K}|$ features and one label. Each sample is a different food basket where the $|\mathcal{K}|$ features are the amounts of each commodity k that make up the food basket, and the label corresponds to the palatability score associated with each specific food basket. This label can either be numeric or categorical, resulting in $Palatability\_score(\boldsymbol{x})$ being either a regression or classification model respectively. The data and code for this example are available at https://github.com/hwiberg/OptiCL.

2.2. Radiotherapy optimization

The second example problem we will use to illustrate the framework is radiotherapy (RT) optimization. In RT, ionizing radiation beams are used to control tumor growth by damaging cancerous tissue. Healthy, non-cancerous tissue (also known as organs-at-risk (OAR)) are however present in close proximity to the tumors, and consequently incur unavoidable damage. The tumor damage can be quantified using the Tumor Control Probability (TCP), while the unavoidable damage to the OARs can be quantified using the Normal Tissue Complications Probability (NTCP). The final optimization problem is to maximize the TCP subject to a set of constraints ensuring that the NTCP does not exceed a specified level. If $\mathcal{Q}$ is the set of OARs (e.g. stomach, liver, spinal cord) that need to be protected, then the optimization problem can be written as

$$\max_{\boldsymbol{x},\boldsymbol{d}} \ \mathrm{TCP}(\boldsymbol{d},\boldsymbol{w}) \tag{2a}$$

$$\text{s.t. } \mathrm{NTCP}_q(\boldsymbol{d},\boldsymbol{w}) \le \delta_q \qquad \forall q \in \mathcal{Q} \tag{2b}$$

$$f(\boldsymbol{d}) \le 0 \tag{2c}$$

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{d} \tag{2d}$$

$$\boldsymbol{x} \ge 0, \tag{2e}$$

where $\boldsymbol{w}$ is a vector containing the biological characteristics of the tumor and the OARs, as well as other patient-specific information. The maximum allowable damage to OAR q is given by δ_q, while constraint (2c) is a placeholder for other constraints on the dose distribution $\boldsymbol{d}$ (e.g. minimum dose, maximum dose, etc.) and is usually convex or can be converted to a convex form. The decision variables $\boldsymbol{x}$ and $\boldsymbol{d}$ are the set of the radiation intensities and the doses to be delivered over the course of the treatment respectively, while $\mathbf{A}$ is the dose deposition matrix. A more detailed treatment of intensity-modulated radiation therapy optimization can be found in Hoffmann et al. (2008).

In the current literature, explicit expressions are used for the TCP and NTCP functions (Gay & Niemierko, 2007). These however are surrogate functions based on population-wide estimates, and as such, the response of different patients to the same treatment can be quite varied (Scott et al., 2021). Thus, while some patients experience complete remission with minimal damage to OARs, others suffer from radiation poisoning or tumor recurrence. It has been shown in Tucker et al. (2013) that the inclusion of patient-specific characteristics (such as genetic predispositions) can improve the prediction of patients' response to RT treatment. We therefore intend to use data to learn (2a) and (2b), leading to the creation of more effective treatment plans for patients. This problem can be reformulated following the approach given in Section 3.1 to learn these objectives and constraints.

The above two examples have illustrated the potential benefits of OCL. In the next section, we will describe the general framework for OCL, as well as show how to apply the steps of the proposed framework to optimization problems.

3. General description of framework

The proposed framework is the result of a careful analysis of many approaches adopted in the literature covered by this survey. It is the formalization of a procedure that (partially) recurs in most approaches to learning constraints. We believe that a complete view of all the steps that characterize the framework will help future research see the big picture of constraint learning, and the impact that each step has on the final performance.

3.1. Step 1: setup of the conceptual optimization model

This initial step is usually done in collaboration with domain experts. This step begins with (i) defining the decision variables and parameters involved, and (ii) defining those constraints that are easy to model manually. After this, the constraint(s) to be learned can be included. Finally, any additional auxiliary constraints required to facilitate the constraint learning process may be added. A broad formulation of all the fundamental aspects that characterize optimization with constraint learning is given as:

$$\min_{\boldsymbol{x},\boldsymbol{y}} \ f(\boldsymbol{x}) \tag{3a}$$

$$\text{s.t. } \boldsymbol{g}(\boldsymbol{x}) \le 0 \tag{3b}$$

$$\boldsymbol{y} = \hat{\boldsymbol{h}}(\boldsymbol{x},\boldsymbol{w},\boldsymbol{z}) \tag{3c}$$

$$z_i = s_i(\boldsymbol{x}) \qquad i = 1,\dots,m \tag{3d}$$

$$\boldsymbol{\theta}(\boldsymbol{y}) \le 0 \tag{3e}$$

$$\boldsymbol{x} \in X, \tag{3f}$$

where the vector $\boldsymbol{x}$ is the vector of decision variables, (3a) is the known objective function and (3b) are the constraints that are easy to model. In (3c), $\boldsymbol{y}$ is the difficult part of the model to design manually and is the output of a predictive model $\hat{\boldsymbol{h}}(\cdot)$ which takes as input the decision variables $\boldsymbol{x}$, and $\boldsymbol{z}$, which is the vector of variables designed as a combination of the $\boldsymbol{x}$ variables. The predictive model's performance can be often improved with the addition of other features $\boldsymbol{w}$ that are not part of the decision-making process, but convey additional or contextual information about the problem (Bertsimas et al., 2022; Chen et al., 2020a). For example in the radiotherapy optimization example in Section 2.2, although the decision variables are radiation intensities $\boldsymbol{x}$, additional information is conveyed by the vector $\boldsymbol{w}$ which contains patient-specific information (such as gene expressions). Although $\boldsymbol{w}$ is known in this example, it may be unknown in other problems and as such, we may have to optimize for the average or worst case. It may also be possible to treat any uncertainties in $\boldsymbol{w}$ using a robust optimization (Gorissen et al., 2015) or stochastic programming (Birge & Louveaux, 2011) approach.

The vector $\boldsymbol{x}$ can also be exploited to build more complex features using a process known as *feature engineering* (Kuhn & Johnson, 2019). The idea behind this process is to use effective combinations of decision variables as new features to improve the performance of the predictive model, reduce the number of features used in the learning process, or embed prior knowledge into the predictive model. These new features $\boldsymbol{z} = (z_1,\dots,z_m)$ can be expressed using constraint (3d), where s_i is the specific function used to extract the feature z_i. The palatability constraint in Section 2.1 is an example of such a case. It is not simply a function of the decision variables, but more accurately a function of aggregates of $\boldsymbol{x}$. These z_i variables represent the amount of food per macro category (Cereals & Grains, Pulses & Vegetables, Oils & Fats, Mixed & Blended foods, Meat & Fish & Dairy), so that (1c) becomes $Palatability\_score(\boldsymbol{z}) \ge P_{\min}$. It should be noted that some part of the known constraint (3b) might also be a function of $\boldsymbol{y}$, i.e. $\boldsymbol{g}(\boldsymbol{x})$ could be $\boldsymbol{g}(\boldsymbol{x},\boldsymbol{y})$, however we will continue with the definition where the known and unknown constraints are written separately. Similarly, some part of the objective function might also depend on the quantity to be learned (i.e. $f(\boldsymbol{x},\boldsymbol{y})$), but we will also continue with the definition where $f(\boldsymbol{x})$ is known. Finally (3e) contains any required constraints on $\boldsymbol{y}$.

In order to avoid getting a solution far from what is seen in the data, we can restrict the optimization to a region that is densely populated by samples that have been used to train and test the predictive model. This region is known as a Trust Region (TR), and is briefly mentioned by Biggs et al. (2018) and expounded upon in Maragno et al. (2021). Here, the authors constrain the optimal solution to be within the convex hull of all samples (training and test samples regardless of their label). Although the computation of the convex hull is burdensome in a high-dimensional space, the authors circumnavigate the problem by constraining the optimal solution to be a convex combination of the samples. Thus, the computation of the convex hull facets is unnecessary and the problem distills to constraining the optimal solution with $O(n)$ constraints, where n is the dimensionality of the decision variable. We refer to Maragno et al. (2021) for full details.

Ideally, a trust region should account for outliers and prevent low-density regions to be considered trustworthy. Maragno et al. (2021) propose a two-step approach that first identifies high-density clusters and then defines the trust region as a combination of multiple convex hulls, one for each cluster. Smaller and denser trust regions lead to worse solutions in terms of objective function value, but also to greater confidence in the predictive model evaluation. A similar approach in terms of clustering high-density regions before constraint learning is seen in Sroka & Pawlak (2018), and Karmelita & Pawlak (2020).

3.2. Step 2: data gathering and preprocessing

The next step is to focus on collecting and preprocessing the data which will be used to learn the constraints (3c). The characteristics of the data inform the tasks to be performed and have a direct impact not only on the predictive model, but also on the final optimization model. For example, the amount of data available might influence the choice of the ML method used to learn the constraint, which may then have an influence on the approach used to solve the final optimization model. If the right data are available, then they can be used directly (with or without preprocessing) to learn the constraints (3c). If the data are not yet available, decisions have to be made on how to collect the data. Data collection or generation strategies should follow established principles such as Design of Experiments (DoE) techniques (which are applicable for physical experiments) or Design of Computer Experiments (DoCE) techniques (applicable for computational experiments). In the WFP application of Section 2.1 for example, collecting data on the palatability of rations during an epidemic or in a war zone might not be possible. In this case, the generation or simulation of data following DoCE principles can be done. A recent review of DoCE is given in Garud et al. (2017).

Data collection or generation can take place either in a single run or in multiple runs. Performing data collection multiple times can make the learning and optimization process more accurate. For example, if the simulation of a process from which data is obtained takes several hours, then the choice of input parameters for the simulator might need to be carefully considered. Adaptive DoCE techniques such as those in Crombecq et al. (2011) and Xu et al. (2014) may be used in such cases. In Derivative-Free Optimization (DFO) only a few or even one new data point is collected in each iteration to minimize the number of data points needed to obtain an optimal solution. We refer to Conn et al. (2009) for a detailed treatment on DFO. In addition to the availability or sourcing of the data, the size of the dataset also informs the choice of the learning approach. It is known that deep neural networks are able to exploit large datasets better than traditional algorithms such as logistic regression or support vector machines (Ng, 2020). In contrast, methods such as kriging (Matheron, 1963) can be used with smaller datasets. Regardless of the size of the datasets, most data (especially those not generated with the use of simulators) need to be pre-processed before they can be used in the next step (learning). Examples of various pre-processing methods are given in McKinney (2017). Whatever data collection, generation, and preprocessing approaches are used, the end result of this second step is a dataset that is ready to be used for training, testing, and validating the predictive algorithms. A review of the various data generation and preprocessing approaches used in the context of constraint learning is given in Section 4.2.

3.3. Step 3: selection and training of predictive model(s)

Once the data gathering and preprocessing activities have been completed, it is necessary to select, train, validate, and test the predictive model that will subsequently be used to represent one or more constraints in the final optimization model. Selection and training are two activities that go hand in hand. It is not possible to select the best predictive model without first training and testing it on the available dataset. Correspondingly, the poor performance of a model after training might require a different predictive model to be selected. The selection of the predictive model is based on six main factors:

i Classification or regression: The choice of the predictive model will depend on whether the constraint to be learned results in a discrete set of values (classification), or a continuous spectrum of values (regression). The use of a classification model results in a feasible set being learned, while the use of a regression model results in a function being learned.

ii Computational complexity: The choice of the predictive model directly affects the computational complexity of the final optimization model. In contrast to a linear regression model, deep learning models require several mathematical or logical statements to represent them (see Section 4.4) and undoubtedly increase the complexity of the final optimization problem. A decision might also be made to select a predictive model so that the complexity of the final optimization problem does not increase. In the WFP example, the underlying problem is linear so a linear predictive model may be chosen to learn (1c).

iii Model performance: In addition to computational complexity, the predictive model has to perform adequately with respect to some performance measure (e.g. accuracy or mean squared error). A simple but inaccurate model can have worse consequences than a complex but accurate model. The performance should be evaluated on both a test set and in the final optimization model. This means that the performance evaluation has to be performed both in step 3 and step 5.

iv Confidence intervals: If the predictive models used provide confidence intervals, it might be possible to use this information in approaches such as stochastic optimization or robust optimization.

v Data quality: Despite the attention focused on the dataset in Step 2, it may be necessary to choose a predictive algorithm that is robust to noise or missing data. This is especially the case if the data to be used has been provided by a third party.

vi Interpretability: The selection of a predictive model might also depend on whether or not the practitioner requires the model to be interpretable. In the radiotherapy optimization problem of Section 2.2, clinicians may prefer highly explainable predictive models to be used to learn the $\mathrm{TCP}(\boldsymbol{d},\boldsymbol{w})$ and $\mathrm{NTCP}_q(\boldsymbol{d},\boldsymbol{w})$ functions due to the seriousness of the application. There are however pros and cons to consider. Decision trees, for example, are known to be highly explainable but may require the linearization of several disjunctions to encode them in the optimization problem. Refer to Section 4.4 for an example.

Table 1 shows the pros and cons of the predictive models generally used in constraint learning. It can be seen that while linear models are very good in terms of explainability and computa-

Table 1
Pros and cons of predictive models commonly used in constraint learning.

	Explainability	Complexity	Data required	Performance
Linear models*	++	++	++	−−
Decision Trees	+	+	+	+
Tree ensembles	−	−	−	+
Neural Networks	−−	−−	−−	++

++: very good; +: good; −: bad; −−: very bad.
* Linear regression, linear support vector machine.

tional complexity, their performance is lacking. Conversely, while neural networks perform quite well, they require large amounts of labeled data to train (Ng, 2020) and are not considered explainable (Molnar, 2020). A review of the various predictive models used for constraint learning in optimization is given in Section 4.3.

The implementation process is a cycle of training the predictive model, evaluating the model's performance, and improving the model in terms of the data and the model parameters. If a feed-forward neural network is used, for example, parameters such as the number of layers, the number of nodes per layer, the activation functions, whether or not to use dropout regularization, etc., need to be determined. Out-of-sample testing should be done to ensure the adequate performance of the model, with changes made to improve performance if necessary. In the context of constraint learning, under the same computational complexity, the best predictive model is the one with higher performance in the final optimization model. This is related to the concept of predictive versus prescriptive models and is explained further in Bertsimas & Kallus (2020). The goal should be to generate prediction models that aim to minimize decision error, and not just prediction error (Elmachtoub & Grigas, 2022).

3.4. Step 4: resolution of the optimization model

Once the predictive model has been used to learn the difficult-to-model constraint, it will need to be embedded into the rest of the optimization problem. Recall that the predictive model $\hat{\boldsymbol{h}}(\cdot)$ in (3c) may not be easy to add as a constraint and may need to be encoded in such a way that it can be handled by commercial or conventional solvers. This however depends on the type of model used, as a linear or quadratic function can be embedded more easily than a neural network for example. The integration of the learned constraints is often not trivial and may result in a significant reformulation of the model. If possible, the predictive model may be incorporated either by providing gradients directly to the solver, or by reformulating it such that it becomes linear, convex, conic, etc., and can more readily be incorporated into the optimization problem. If desired, the optimization models may be modified to take into account any uncertainty measures provided by the predictive models. A successful embedding can also include providing additional information (such as bounds) to the solver of choice in order to facilitate a more efficient solution process. A successful embedding should make it possible to take full advantage of the optimization solver's capabilities. A deeper look at the various methods used in the literature to encode predictive models is given in Section 4.4.

3.5. Step 5: verification and improvement of the optimization model

In this final step, the performance of the constraint learning process is evaluated with respect to three things: the final optimization model, the predictive model, and the data. For the optimization model, the focus is on both the goodness of the solution (i.e. the objective function value), as well as the time taken to find this solution. For the performance of the predictive model, the focus is on ensuring that the learned constraints accurately represent the difficult-to-model constraints. The data used to learn constraints must also be examined. If, for example, the data points used are far from the boundary, this might have a negative effect on the constraint learned and the optimal solution (Thams et al., 2017). If it is possible to generate data, then using different data to improve the predictive model can in turn help to improve the optimal solution. One way of doing this is via a process called active learning (Settles, 2009). Here, the dataset is continually modified by adding new solutions (output of the optimization model) to it, which is then used to re-train the predictive model. A review of the approaches used to verify and improve models with learned constraints is provided in Section 4.5.

Table 2 shows the emphasis placed on the different steps in the literature. While all steps are generally present in the literature, some steps are often emphasized more than others. It can be seen that the most emphasis is placed on Step 4, i.e., learning and reformulating constraints so that the final optimization problem can be easily solved. While the above framework steps have been given in sequential order, it should be noted that some of these steps may be repeated. For example, if it is seen in step 5 that the learned constraint is a poor representation of reality, we may need to go back to step 1. Similarly, the incorporation of new data will mean that the predictive model needs to be retrained and re-embedded, and so on. Internal loops are also possible — it may be necessary to iterate between steps 2 and 3 until an acceptable predictive model is developed, or between steps 4 and 5 until satisfactory results are achieved.

4. Review

4.1. Setup of the conceptual model

The papers by Lombardi et al. (2017) and Maragno et al. (2021) provide very detailed approaches to formalizing the process of OCL. There is also a clear structure provided in De Angelis et al. (2003) for learning unknown constraints. The methodology in Lombardi et al. (2017), called Empirical Model Learning (EML), is described as a way of obtaining components of a prescriptive model from machine learning. Similar to the steps given in Section 3.1, they define a conceptual model including known constraints, constraints to be learned, as well as the definition of constraints used for feature extraction. Maragno et al. (2021) also provide a general framework for OCL that can be applied every time data is available. A key focus of their approach is to learn constraints and embed them in such a way that the computational complexity of the final model is not increased. Padmanabhan & Tuzhilin (2003) and Deng et al. (2018) also conceptually discuss the use of data mining to define objectives and constraints.

While other papers which use a predictive element to learn constraints do not have as detailed a setup, they incorporate some of the elements of Step 1 in their approach. Some authors (e.g. Verwer et al., 2017) explicitly define constraints to perform feature extraction, while most of the papers reviewed simply identify which constraints are difficult to model manually and re-

Table 2
Emphasis placed on the different steps of the framework.

	Step 1	Step 2	Step 3	Step 4	Step 5
Bergman et al. (2022)	●	●	●	●	●
Biggs et al. (2018)	●	●	●	●	●
Chatzivasileiadis (2020)	●	●	●	●	●
Chen et al. (2020b)	◐	◐	●	●	◐
Chi et al. (2007)	○	●	●	●	●
Cozad et al. (2014)	◐	●	●	●	●
Cremer et al. (2018)	◐	◐	●	●	●
De Angelis et al. (2003)	●	●	●	◐	●
Fahmi & Cremaschi (2012)	●	●	●	●	◐
Garg et al. (2018)	●	◐	●	●	●
Grimstad & Andersson (2019)	●	●	●	●	●
Gutierrez-Martinez et al. (2010)	●	◐	●	●	●
Halilbašić et al. (2018)	◐	●	●	●	●
Jalali et al. (2019)	●	●	●	●	●
Karmelita & Pawlak (2020)	●	●	n/a	●	●
Kudła & Pawlak (2018)	●	●	●	●	●
Kumar et al. (2019a)	●	●	n/a	●	●
Kumar et al. (2019b)	●	●	n/a	●	●
Kumar et al. (2021)	●	●	n/a	●	●
Lombardi et al. (2017)	●	●	●	●	●
Maragno et al. (2021)	●	●	●	●	●
Mišić (2020)	●	◐	●	●	●
Murzakhanov et al. (2020)	●	●	●	●	●
Paulus et al. (2021)	●	●	●	●	●
Pawlak & Krawiec (2017a)	●	◐	n/a	●	●
Pawlak & Krawiec (2017b)	●	●	n/a	●	●
Pawlak & Krawiec (2018)	●	●	n/a	●	●
Pawlak (2019)	●	●	n/a	●	●
Pawlak & Litwiniuk (2021)	●	●	●	●	●
Pawlak & O'Neill (2021)	●	●	n/a	●	●
Prat & Chatzivasileiadis (2020)	◐	●	●	●	◐
Say et al. (2017)	◐	●	●	●	◐
Schede et al. (2019)	●	○	●	●	●
Schweidtmann & Mitsos (2019)	◐	◐	●	●	◐
Sroka & Pawlak (2018)	●	●	n/a	●	●
Thams et al. (2017)	◐	●	●	●	●
Verwer et al. (2017)	●	◐	◐	●	●
Xavier et al. (2021)	◐	●	●	●	●
Yang & Bequette (2021)	●	●	●	●	●

●: most emphasis; ◐: some emphasis; ○: less emphasis; n/a: not applicable.

place them with predictive models. This will be discussed later in Sections 4.3 and 4.4.

The approach taken to formalize the constraint learning process in Pawlak & Krawiec (2017a) is different in that their framework is a MILP. Similarly, other authors also leverage MILP (Schede et al., 2019) or Integer Programming (IP) (Cozad et al., 2014) frameworks for constraint learning. Sroka & Pawlak (2018) use local search, and Kumar et al. (2021) use an approach based on stochastic local search. A number of authors also use a genetic programming framework to obtain constraints from data (Pawlak & Krawiec, 2017b; 2018). Pawlak (2019) use evolutionary strategies, and Pawlak & O'Neill (2021) use grammatical evolution (O'Neil & Ryan, 2003). In these different setups, the authors give the relative simplicity of their approaches as their key justification — no predictive models have to be learned, no constraints for feature extraction have to be included, and no complex embedding or reformulating procedures need to be followed. The disadvantage of these approaches however is that they do not make use of the learning power of predictive models which have been shown to be able to learn very complex decision boundaries well.

4.2. Data gathering and preprocessing

Of the papers considered in this survey, Thams et al. (2017) has a detailed approach with regards to the data generation and preprocessing phase detailed in Section 3.2. Their procedure, also used by Halilbašić et al. (2018) and Murzakhanov et al. (2020), is designed to either make use of available data from sensors or generate them using a simulator. Possible operating points of the system under consideration are fed into a simulator, and the output of the simulator is analyzed and classified as either *stable* or *unstable*. These operating points along with their classifications are then stored in a database. The data is only generated once, and they attempt to keep the generated database as small as possible for computational reasons. Although the generated data is small, they ensure that enough data is collected so that their predictive models can be trained. They do this by focusing on generating points close to the boundary of their two-class problem while neglecting operating points far away from the boundary. In this way, they balance exploration and exploitation when generating data. In addition to this, class imbalance in the dataset is addressed using several strategies, and feature selection is done so that only easily accessible variables are considered.

With regards to following DoCE principles, the authors of Lombardi et al. (2017) use factorial designs to generate multiple training sets from a simulator. Data generation here, although time-consuming, only needs to be done once. The data is also normalized so that the input features fall within a certain range. The approach developed in Cozad et al. (2014) also takes DoCE principles into account by using either Latin Hypercube Sampling (LHS) or two-level factorial design (Mukerjee & Wu, 2006). They also use adaptive sampling to iteratively refine the models generated and provide discussions on whether to use exploration-based or exploitation-based sampling. De Angelis et al. (2003) use small data collected using an on-field survey (an expensive process) to find a probability distribution that is later used in a simulator that generates enough data to train the predictive model. In Fahmi & Cremaschi (2012), the data is obtained from multiple simulators. Some transformation is done to the data to ensure that the distribution of the dependent variable is as close as possible to a uniform distribution. The authors here note that a large amount of data was needed to train their neural networks for higher accuracy. Data may also be collated from multiple available sources as in the case of Bertsimas et al. (2016).

In terms of the sizes of the datasets surveyed, it can be seen that not many of the problems considered have large datasets. Data containing 500 or fewer examples are common. In Pawlak & Krawiec (2017a), the total amount of data generated using uniform sampling varied from 10 to 500 examples. Similarly, Schweidtmann & Mitsos (2019) use data with 500 or fewer examples. Data of a similar size is also seen in Kudła & Pawlak (2018); Yang & Bequette (2021) and several others. For one of the test problems in Schweidtmann & Mitsos (2019), only 46 examples are used. On the larger side, Chen et al. (2020b) use data containing 10,000 instances, Murzakhanov et al. (2020) use 36,144 data points generated using LHS, while the data used in Say et al. (2017) contains 100,000 data points. For certain constraint learning approaches, the availability of a large amount of data is a must. The local search approach used in Sroka & Pawlak (2018) is such an example, where the test set contains 500,000 examples generated via uniform sampling.

Some approaches for constraint learning necessitate the significant preprocessing of the data. In order to learn ellipsoidal constraints in Pawlak & Litwiniuk (2021), the uniformly generated data first needs to be clustered and standardized before Principal Component Analysis (PCA) is applied to the data. This clustering and PCA procedure shapes the data so that it can be represented using ellipsoidal constraints. In other cases, although preprocessing is done, it is not crucial to the constraint learning approach (e.g. in Biggs et al., 2018; Pawlak, 2019).

In some cases, not much emphasis is placed on the data generation and preprocessing phase. This is sometimes because the data is fully available, is neither big nor dirty, and has no need for preprocessing. For example, the situation of having high-dimensional data, where the number of examples is much less than the number of features, is only explicitly mentioned in Mišić (2020). In a few cases, however, the data step is an afterthought. Even when data is generated, this is cursorily mentioned without giving details or following exhaustive DoCE principles. We believe that our framework will help to provoke more thought on this phase of the constraint learning process.

4.3. Selection and training of ML models

We first note that papers that use regression functions as the main constraint learning methods have been excluded from Table 3, as they are too numerous to be included in this paper. Regression with first- and second-order polynomials is used a lot in engineering, especially in combination with DoE and DoCE. We refer to Kleijnen (2015) for a detailed treatment. Kriging models are also used a lot in engineering, especially when the objective function values can be obtained by (expensive) computer simulations. The big advantage of kriging is that it has very high predictive power even for a relatively small number of data points. The disadvantages of kriging are (i) it costs much computing time to obtain the kriging model (ii) the data points have to be space-filling (iii) embedding kriging functions in the optimization model makes it non-convex and hard to solve. See Forrester et al. (2008) for a detailed treatment on kriging.

It can be seen from Table 3 that Artificial Neural Networks (ANN) and Decision Trees (DT) are the most popularly used predictive models in constraint learning (with the exception of regression models). This could be because these methods have been shown to have high representative power, can be used for both regression and classification, and generally perform well. Reviews on ANNs and DTs are given in Cao et al. (2018) and Carrizosa et al. (2021) respectively. Training, testing, and evaluation are generally done as prescribed in the ML literature, and the methods are generally evaluated using metrics such as the Mean Square Error (MSE), accuracy, and so on. For example, to learn objective functions in Biggs et al. (2018), the data is split in a 70:30 ratio between training and test sets, and prediction accuracy used as the evaluation metric. See Chapter 2 of James et al. (2013) for more information on training and testing ML models.

In terms of the size of the ANNs, with the exception of Yang & Bequette (2021) who use a network with six hidden layers, most networks used in constraint learning are generally small. One reason for this could be to reduce the burden of embedding large networks as constraints — it will be seen from (5a) to (5e) in Section 4.4 that the number of binary variables required is proportional to the size of the network. Also, as the input of a current layer is the output of a previous one, using multiple layers results in nested activation functions in the final model. This could lead to issues regarding convexity and/or differentiability, therefore smaller networks are used. Another reason for small neural network sizes is that using larger networks provides no benefit in some cases. Lombardi et al. (2017) for example use a network with one hidden layer (with two neurons) as increasing the network size did not improve the solution quality of the optimization problem. To view the effect of network size on computation times, Schweidtmann & Mitsos (2019) vary both the number of neurons and the number of layers. Results showed that the solution time increases approximately exponentially with the number of layers used to learn the constraints. Deep neural networks required more processing time than shallow neural networks for the same total number of neurons.

An alternative to using small networks could be to use sparsification. Murzakhanov et al. (2020) use a network with three hidden layers and fifty neurons in each layer to learn constraints, however they reduce the size of the network by enforcing 70% of the weights to be zero. This sparsification procedure ensures that the network was not too large and also avoided over-fitting during training. Say et al. (2017) use a similar technique to remove connections to and from neurons with very small weights in order to strengthen the MILP formulation. Another alternative could be to ensure that the constraint learning procedure results in a convex optimization problem. Chen et al. (2020b) and Yang & Bequette (2021) use Input Convex Neural Networks (ICNN) (Amos et al., 2017) as their predictive models. An ICNN requires that all weights in the network are non-negative and that all activation functions are convex and non-decreasing. Although these networks require more training iterations and have lower representation power than conventional networks, their use guarantees that the final problem is a convex optimization problem. This allows the authors to balance the trade-off between model accuracy and computational tractability.

The activation functions used in the hidden layers in the constraint learning literature are usually either the Rectified Linear Unit (ReLU) (e.g. in Say et al., 2017), the sigmoid function (e.g. in Chen et al., 2020b), (some form of) the hyperbolic tangent function (e.g. in Fahmi & Cremaschi, 2012), or the softplus activation function (e.g. in Chen et al., 2020b). The output layer activation function is usually an identity linear function. An overview of the most common types of activation functions is given in Sharma (2017).

Table 3
Methods used for constraint learning.

	Neural networks	Decision trees	Random forest	Other ensemble	Support vector machines	Clustering	(M)ILP	Other
Bergman et al. (2022)	x							x
Biggs et al. (2018)		x	x					
Chatzivasileiadis (2020)	x	x						
Chen et al. (2020b)	x							
Chi et al. (2007)					x			
Cozad et al. (2014)							x	x
Cremer et al. (2018)		x		x				
De Angelis et al. (2003)	x							
Fahmi & Cremaschi (2012)	x							x
Garg et al. (2018)					x			
Grimstad & Andersson (2019)	x							
Gutierrez-Martinez et al. (2010)	x							
Halilbašić et al. (2018)		x						
Jalali et al. (2019)					x			
Karmelita & Pawlak (2020)						x		x
Kudła & Pawlak (2018)		x						
Kumar et al. (2019a)								x
Kumar et al. (2019b)							x	x
Kumar et al. (2021)								x
Lombardi et al. (2017)	x	x						
Maragno et al. (2021)	x	x	x	x	x	x		x
Mišić (2020)		x	x					
Murzakhanov et al. (2020)	x							
Paulus et al. (2021)	x							
Pawlak & Krawiec (2017a)							x	
Pawlak & Krawiec (2017b)								x
Pawlak & Krawiec (2018)								x
Pawlak (2019)								x
Pawlak & Litwiniuk (2021)						x		x
Pawlak & O'Neill (2021)							x	x
Prat & Chatzivasileiadis (2020)		x						
Say et al. (2017)	x							
Schede et al. (2019)		x					x	
Schweidtmann & Mitsos (2019)	x							
Sroka & Pawlak (2018)						x		x
Thams et al. (2017)		x						
Verwer et al. (2017)		x						x
Xavier et al. (2021)					x			
Yang & Bequette (2021)	x							

In addition to ANNs, DTs are also popularly used in constraint learning. DTs are able to capture non-convex and discontinuous feasibility spaces (Chatzivasileiadis, 2020), and provide these rules in an easily understandable format (Prat & Chatzivasileiadis, 2020). For this reason, Thams et al. (2017) and Halilbašić et al. (2018) specifically choose decision trees as their method of choice to learn constraints in an optimal power flow problem. Their rationale is that as electricity is a crucial utility, using easily interpretable rules in their analysis can lower the barrier to power plant operators adopting their methods. This interpretability requirement could also favor the use of decision trees in the radiotherapy example of Section 2.2.

As random forests are a natural extension of using decision trees, they have also been used in constraint learning. Biggs et al. (2018) learn objective functions using random forest while Mišić (2020) optimize tree ensembles. AdaBoost (Freund & Schapire, 1997) (with DTs as base learners) is also used in Cremer et al. (2018) to learn a probabilistic description of a constraint.

Support Vector Machines (SVM) can be expressed very clearly as optimization problems, and as a result, there is a significant amount of optimization literature on SVMs. Most of the literature however deals with subjects such as optimal feature selection (e.g. Jiménez-Cordero et al., 2021), or mathematical formulation and solution of the SVM problem (e.g. Baldomero-Naranjo et al., 2020), or improving predictions (e.g. Yao et al., 2017). There are however a few cases where SVM is used in the constraint learning process. Jalali et al. (2019) use SVM to learn power inverter control rules. Chi et al. (2007) use SVM for regression with a second order polynomial kernel function. A feature selection process is further used to remove insignificant terms and improve model accuracy. Xavier et al. (2021) also use SVM with linear kernels to learn hyper-planes within which the solution of the optimization problem is very likely to exist.

Clustering is also used in constraint learning, although it is often used in addition to other methods. Pawlak & Litwiniuk (2021) combine clustering with PCA and ellipsoidal constraints to get an MIQCP, while Sroka & Pawlak (2018) combine clustering with local search to search the clusters for LP constraints that represent each cluster.

Combinations of predictive models are often used. Verwer et al. (2017) combine regression trees and regression models for learning revenue constraints for an auction optimization problem. In addition to using ANNs to learn most of the constraints required, Fahmi & Cremaschi (2012) also use non-linear regression models for specific constraints. Paulus et al. (2021) use both integer programming and neural networks to learn both the cost terms and the constraints for integer programs. Although Schede et al. (2019) use an LP formulation to learn polytopes that cover feasible examples, they also include a decision tree heuristic to model the importance of examples to be considered. Additionally, Maragno et al. (2021) are able to use different predictive models to learn different constraints of the same optimization problem in order to achieve better performance.

There are also approaches that do not use predictive models to learn constraints. Pawlak & Krawiec (2017b, 2018) and Pawlak (2019) use genetic programming as their approach of choice, Pawlak & Krawiec (2017a) use a mixed-integer formulation and Cozad et al. (2014) combine simple basis functions using a MILP formulation to learn constraints.

Although RF is an ensemble method, the column "Other Ensemble" in Table 3 is separated from the "Random Forest" column to highlight the fact that there is an opportunity to use other ensemble methods where different individual base learners can be used and their predictors combined. There is also the opportunity to try a wider variety of predictive models such as symbolic regression or Bayesian methods, especially if they are more easily represented using mathematical functions. Studies could be done to see if certain families of models perform better for certain types of problems. Finally, methods that provide some measure of uncertainty (such as confidence intervals or conditional probabilities) might also be chosen in order to be able to incorporate this uncertainty into the final mathematical model.

4.4. Resolution of the optimization model

With regression functions, the process of embedding them as constraints is relatively straightforward as the functions are simply a weighted sum of the decision variables (Bertsimas et al., 2016; Verwer et al., 2017). With other predictive models, however, the embedding process is more involved. With ANNs for example, their embedding is also dependent on the activation functions used. The output $\hat{y}$ of a fully connected neural network with h hidden layers can be written as:

$$\mathbf{H}_1 = \sigma_1(\mathbf{W}_1\mathbf{X}) \tag{4a}$$

$$\mathbf{H}_i = \sigma_i(\mathbf{W}_i\mathbf{H}_{i-1}) \tag{4b}$$

$$\hat{y} = \sigma_{h+1}(\mathbf{W}_{h+1}\mathbf{H}_h), \tag{4c}$$

where $\mathbf{X}$ is the training data, $\mathbf{W}_i$ are the weights associates with layer i, and σ_i is the activation function used in layer i. In order to incorporate a trained neural network with ReLU activation functions (i.e. $\sigma(x) = \max(0, x)$) as a constraint, the *max*() function can be replaced by introducing binary variables b_j for each neuron in the network:

$$y_j = \max\left(0, \hat{y}_j\right) \Rightarrow \begin{cases} y_j \le \hat{y}_j - \hat{y}_j^{\min}\left(1 - b_j\right) & \text{(5a)} \\ y_j \ge \hat{y}_j & \text{(5b)} \\ y_j \le \hat{y}_j^{\max} b_j & \text{(5c)} \\ y_j \ge 0 & \text{(5d)} \\ b_j \in \{0, 1\}^{N_j}. & \text{(5e)} \end{cases}$$

If the input to a neuron $\hat{y}_j > 0$, then $b_j = 1$ and (5a) and (5b) force the neuron output y_j to the input $\hat{y}_j$. On the other hand, if the input to the neuron $\hat{y}_j \le 0$ then $b_j = 0$, and (5) and (5d) force the output y_j to 0. The resulting MILP can then be solved with off-the-shelf solvers such as Gurobi (Gurobi Optimization, LLC, 2022) or CPLEX (IBM, 2022). This is done by authors such as Murzakhanov et al. (2020) and Bergman et al. (2022).

The disadvantages of this approach are twofold. Firstly, large networks require a correspondingly large number of binary variables. Secondly, the bounds $\hat{y}_j^{\min}$ and $\hat{y}_j^{\max}$ have to be chosen to be as tight as possible for the MILP solver. Anderson et al. (2020) present a MIP formulation that produces strong relaxations for ReLUs. They start with the big-M formulation and then add strengthening inequalities as needed. Say et al. (2017) also encode ReLU into MILP using big-M constraints, but include reformulations to strengthen the bounds. They also use sparsification to remove neurons with very small weights, as constraints with very small coefficients can be difficult to solve for commercial solvers (D'andreagiovanni & Gleixner, 2016).

The embedding of trained ANNs as constraints is not always as complicated as mentioned above. Equations (4a)–(4c) can be used directly as constraints, as long as the solver can handle the activation function σ. This approach is used in Fahmi & Cremaschi (2012) with the hyperbolic tangent activation function, as well as in Gutierrez-Martinez et al. (2010) where the activation function is differentiable so that the constraint can be easily handled by conventional solvers. It is nevertheless noted in Lombardi et al. (2017) that several commercial solvers rely on convexity for providing globally optimal results and that this approach may result in locally optimal solutions depending on the σ used. In Chen et al. (2020b), the use of ICNN guarantees that the optimization problem is convex and will converge to a globally optimal solution. They do however speed up this process by using a smoothened version of the ReLU, called the Softplus activation function. A large value of the parameter t in the Softplus function allows it to be smooth enough, while still being convex. The final problem is solved using dual decomposition (Chiang et al., 2007; Yu & Lui, 2006). Yang & Bequette (2021) also use ICNNs to get convex optimization problems which are then solved using sequential quadratic programming.

Noticing the use of ANNs in various optimization applications, Schweidtmann & Mitsos (2019) provide an efficient method for deterministic global optimization problems which have ANNs embedded in them. As the only non-linearities of ANNs are their activation functions (the tanh function in their case), the tightness of their relaxations is influenced by the relaxations of the tanh functions. The key idea of their paper is to therefore hide the equations that describe the ANN and its variables from the Branch-and-Bound (B&B) solver using McCormick relaxations of the activation functions. These relaxations are once continuously differentiable, and the lower bounds are calculated by automatic propagation of McCormick relaxations through the network equations. This "Reduced-Space (RS)" formulation results in significantly fewer variables and equalities than the original formulation (by two orders of magnitude in some cases), with significantly shorter computation times in most cases. They also see that shallow ANNs show much tighter relaxations than deep ANNs. Grimstad & Andersson (2019) also provide bound tightening procedures for ReLU networks to reduce solution times.

The embedding of kriging functions in the final model is done in a direct way: substituting the kriging function leads to an explicitly formulated, but highly non-convex constraint. First and second-order derivatives can be calculated, and therefore standard nonlinear solvers can be used. However, due to the non-convexity, solvers might end up in local optima. See also den Hertog & Stehouwer (2002), and Stinstra & den Hertog (2008) for examples.

With DTs, the paths from the root to leaf nodes can be represented using disjunctions of conjunctions. These rules can either be linearized or embedded as constraints using a framework like Generalized Disjunctive Programming (GDP) (GAMS Development Corporation, 2021; Grossmann, 2009). Although Lombardi et al. (2017) do not embed DTs into MINLPs due to the potential for poor bounds being obtained as a result of the extensive linearization of disjunctions required, other authors have. In Kudła & Pawlak (2018), branches of the tree from root to leaf nodes are encoded as linear constraints using a big-M formulation. Paths that end up in feasible decisions are denoted with the (+) sign (see Fig. 1). These paths are then converted into linear constraints using auxiliary binary variables to show which path in the tree is followed. For example, at the top of the tree, if a binary variable $b_1 = 1$, then c_1 is active and (6) holds. Otherwise if $b_1 = 0$, then c_2 is active and (7) holds. Mathematically,

$$c_1 : x_1 \le a_1 \rightarrow x_1 \le a_1 + M(1 - b_1) \tag{6}$$

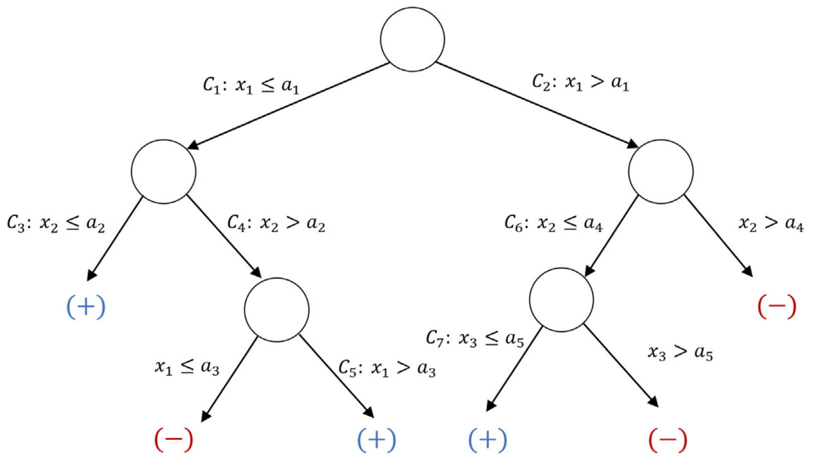

Fig. 1. Extracting constraints from a decision tree (adapted from Kudła & Pawlak, 2018).

$$c_2 : x_1 > a_1 \rightarrow x_1 > a_1 - Mb_1. \quad (7)$$

This process is applied to the whole tree until a set of linear constraints capturing the tree is obtained. A similar approach is used in Verwer et al. (2017) to embed regression trees as constraints. In Thams et al. (2017), each path from the root to a leaf node is represented by two constraints, with one binary variable per path to determine if a path is chosen. The resulting MILP is solved with Gurobi. Halilbašić et al. (2018) also follow a similar procedure except that the resulting problem is further reformulated as a Mixed-Integer Second-Order Cone Program (MISOCP) because of the presence of additional non-linear constraints. The resulting MISOCP and MINLP are then solved using commercial solvers.

To embed random forests as constraints, multiple DTs can be used as above, with an additional constraint that selects a consensus based on the outcome chosen by most of the individual decision trees (Bonfietti et al., 2015). Biggs et al. (2018) maximize a random forest as the objective. For each tree, they use logical constraints to determine which leaf a solution lies in. For the forest, they use a constraint to ensure that only one leaf for each tree is active. To ensure that the feasible set is bounded and that the solution is not too dissimilar from the observed data, the solution is constrained to be within the convex hull. This is similar to the approach in Mišić (2020) which also has an objective function expressed as the prediction of a tree ensemble, and also uses some form of Benders reformulation with lazy constraint generation to solve the problem. Biggs et al. (2018) also compare solving the MIP for the whole random forest, versus splitting the forest into a number of subsets, and solving each subset using the MIP, and a cross-validation heuristic procedure to achieve performance improvements. In Cremer et al. (2018) where the AdaBoost ensemble learning method is used (with DTs as the weak learners), there is one set of constraints for each DT, and exactly one disjunction must be selected for each learner. Probability estimates for all learners are combined in a weighted sum, and the resulting MILP is solved using a commercial solver. For SVMs, linear or quadratic polynomials can be used directly in the optimization problem, providing the solvers used can handle quadratic constraints.

The advantage of using LP or MILP approaches to learn constraints is seen in this step of the framework, as there are no complex embedding procedures to be followed. In Pawlak & Krawiec (2017a), the resolution of the final optimization problem here is relatively easy as the entire constraint learning process was formulated as a MILP from the very beginning. This is also seen in papers like Sroka & Pawlak (2018) and Schede et al. (2019). In Pawlak & Litwiniuk (2021), the data is first clustered, and then PCA is used to give each cluster an ellipsoidal shape. These ellipsoids can then be represented using quadratic constraints. Their approach produces Mixed Integer Quadratically Constrained Programs (MIQCP), which can then be solved by any solver that supports quadratic programming.

In summary, it can be seen that this step of the constraint learning process is the most complex one. Incorporating predictive models sometimes results in a significant reformulation of the original model. The embedding of the predictive models should however be done in a way that takes full advantage of the optimization solver's capabilities. This can be done by providing gradients or other useful information directly to the solver or reformulating the problem to become linear, convex, conic quadratic, etc. The approach by Garg et al. (2018) results in a quadratic program with linear constraints that is solved using off-the-shelf solvers. Constraint learning approaches that use MILP or LP frameworks such as in Pawlak & Krawiec (2017a) avoid these complex embedding procedures, however, the disadvantages of this approach could include a lack of flexibility and limited learning ability. There is therefore a trade-off between simplicity and performance to be considered. Wherever possible, it is desirable to keep the final optimization problem in the same complexity class as the original problem.

4.5. Verification and improvement of the optimization model

The validation of the final optimization model is straightforward when dealing with benchmark problems, as the ground truth is already known. This approach to verifying models is seen in a lot of the literature. In optimal power flow, for example, the availability of benchmark problems allows Prat & Chatzivasileiadis (2020) to find solutions to problems that were previously intractable. Garg et al. (2018) are able to show that their methods obtain near-optimal solutions. The use of benchmark problems also allows for the comparison of the number of synthesized constraints with the actual number of constraints from the benchmark problems (Karmelita & Pawlak, 2020; Pawlak, 2019; Pawlak & Krawiec, 2017a; 2018). Moreover, they are able to assess the similarity of the syntaxes by computing the mean angle between weight vectors of the corresponding constraints in the synthesized and actual models.

The feasible regions of the synthesized models can also be compared to those of the actual feasible regions using the Jaccard Index (Jaccard, 1912), as it is used to measure the similarity between sets. This is used in Kudła & Pawlak (2018); Pawlak & Krawiec (2017a) and others. Visual comparisons (limited to 3-dimensions) have also been used to illustrate how well a constraint learning approach captures the real feasible region (e.g. in Pawlak & Litwiniuk, 2021). In Schede et al. (2019), the authors compute the probabilities that feasible (and infeasible) points of the benchmark problem lie in the feasible (and infeasible) region of the learned problem. They also compare the differences between the optimum values of the original and learned problems. Cremer et al. (2018); Gutierrez-Martinez et al. (2010) and Schweidtmann & Mitsos (2019) similarly also test their approaches on benchmark problems, and compare their results to the known solutions of the problems. A similar approach can be taken to evaluate the effect of constraint learning on the generated treatment plans for the radiotherapy example of Section 2.2. The new treatment plans can be compared to conventional ones using a dose-volume histogram (DVH) (Drzymala et al., 1991). In addition to using benchmark problems, Xavier et al. (2021) also evaluate the performance of the method by using out-of-distribution data to measure the robustness against moderate dataset shift.

Using synthetic or well-known benchmark problems allows one to accurately determine how far learned constraints are from the real ones (Kumar et al., 2019b), however, this is not always possible. Consequently, other model verification procedures are used in the literature. Simulators can also be used to evaluate constraint learning approaches by comparing learned solutions to solutions from the simulator (e.g. in Verwer et al., 2017).

Authors also compare the constraint learning approaches with the typical approaches used to solve those types of problems. This is done in Say et al. (2017), for example, where they compare their constraint learning approach with the rule-based approach commonly used to solve their type of problem. Cozad et al. (2014) implement model improvement by iteratively generating new data and updating the model with these data. The process continues until the error is less than a specified tolerance value. An overview of the approaches used for verification and improvement of the learned models is given in Table 4.

5. Challenges and opportunities in constraint learning

As the process of constraint learning is relatively new, it stands to reason that there are several challenges or limitations that need to be taken into consideration. The primary challenge is that predictive models are designed for predictive purposes, and not to be embedded into an optimization model. Other challenges that new practitioners should keep in mind include:

1. Constraint violation: The optimal solution can be infeasible in reality if the learned constraints do not approximate correctly the true constraints.
2. Explainability: Although optimization models are usually considered explainable, and therefore understandable to humans, the use of learned constraints makes the optimization model difficult to understand and the optimal solution potentially difficult to explain.
3. Data availability: A dataset of historical solutions is required to fit a predictive model. Some predictive models require more data than others to have acceptable performances. Obtaining the data for training purposes might be difficult in certain cases. Additionally in certain applications, one-class data (i.e., data on only feasible (or infeasible states)) may be more available than two-class data (i.e., data on both feasible and infeasible states). Potential practitioners of OCL should be aware of what approaches work best in either case.
4. Computational complexity: Some predictive models require auxiliary variables and many additional constraints in order to be embedded into the optimization model. The number and class of additional constraints and auxiliary variables have a direct effect on the computational complexity of the optimization model.
5. Confounders: The use of predictive models to define part of the optimization model should account for potential confounders. It has been noted by de Mast et al. (2022) that machine learning models are correlational models and not causal models. The fact that certain values of the features co-occur with certain values of the outcome does not imply that changing the value of the features would change the outcome. These features could be causal, however, there is no guarantee of this. Therefore, whenever a causal model on the features space is available, it should be considered in the embedding of the predictive model.

In light of these challenges, some of the opportunities for further research are presented below. The opportunities are discussed in light of the steps of our proposed framework.

Data gathering and preprocessing There is an opportunity to understand the effects of the data generation and preprocessing phase on the solution of the optimization problem. Is there a particular strategy (e.g. DoCE, Kriging, etc.) that performs best for constraint learning? It is also necessary to understand the size and type of data that is needed in order to get a good predictive model for constraint learning, and not just for prediction.

Selection and training Firstly, the choice of which model to use to learn constraints should also be further investigated. Most of the approaches used to learn constraints are the most commonly known methods (regression, ANNs, DTs, etc.), however, there is an opportunity to see how less well-known approaches perform in terms of solution quality, as well as ease of embedding as constraints. Secondly, predictive models often provide a measure of the uncertainty of their predictions. Neural network classifiers, for example, can have Softmax scores (Grave et al., 2017) while decision trees report the number of misclassified examples at their leaf nodes. Although Cremer et al. (2018) have used probabilistic information from their classifier to good results, further research on how to incorporate such measures of uncertainty into optimization problems should be carried out. Approaches such as robust optimization or stochastic programming might be useful here. Thirdly, as noted above, predictive models are designed for predictive purposes, and not to be embedded into an optimization model. There could be an opportunity to develop models designed to be embedded, as well as models that perform well with small amounts of data. Finally, only a few approaches have used ensemble methods to learn constraints. While not all problems will require the additional predictive power that an ensemble approach brings, it is to be expected that as the best predictive models in ML applications are often ensembles, it makes sense to incorporate their superior abilities into constraint learning.

Resolution of the optimization model Several approaches for embedding ML models as constraints have been seen, however, can these models be embedded in more efficient ways? The difficulty of using larger neural networks or larger decision trees to learn constraints might be overcome if more efficient embedding procedures are developed. In Schweidtmann & Mitsos (2019), difficult parts of the neural network (tanh activation functions) were hidden from the B&B solver. Further research on hiding unnecessary information from solvers is needed. Further research is also needed on providing additional useful information (such as pre-computed derivatives) to commercial solvers so as to improve solution time and quality. This will be useful when the incorporation of predictive models makes these computations impractical or expensive.

Verification and improvement In the reviewed literature, methods for model verification and improvement, usually consist of either comparing to benchmark problems, or to other competing

Table 4
Approaches used for model verification and improvement.

	Benchmark problems	Compare constraints and/or feasible regions	Simulator results	Compare to other approaches	Adaptive sampling	Other
Bergman et al. (2022)	x					
Chatzivasileiadis (2020)	x			x		
Chen et al. (2020b)	x		x	x		
Chi et al. (2007)			x			
Cozad et al. (2014)			x		x	
Cremer et al. (2018)	x		x	x		
De Angelis et al. (2003)			x		x	
Fahmi & Cremaschi (2012)			x			
Garg et al. (2018)	x			x		
Grimstad & Andersson (2019)	x					
Gutierrez-Martinez et al. (2010)	x			x		
Halilbašić et al. (2018)	x			x		
Jalali et al. (2019)	x			x		
Karmelita & Pawlak (2020)	x	x		x		
Kudła & Pawlak (2018)	x	x		x		
Kumar et al. (2019a)	x					
Kumar et al. (2019b)	x			x		
Kumar et al. (2021)	x			x		
Lombardi et al. (2017)			x	x		
Maragno et al. (2021)	x			x		
Mišić (2020)				x		
Murzakhanov et al. (2020)	x			x		
Paulus et al. (2021)	x					
Pawlak & Krawiec (2017a)	x	x				
Pawlak & Krawiec (2017b)	x	x				
Pawlak & Krawiec (2018)	x	x				
Pawlak (2019)	x	x		x		
Pawlak & Litwiniuk (2021)	x	x		x		
Pawlak & O'Neill (2021)	x	x		x		
Prat & Chatzivasileiadis (2020)	x			x		
Say et al. (2017)	x			x		
Schede et al. (2019)	x	x		x		
Schweidtmann & Mitsos (2019)	x			x		
Sroka & Pawlak (2018)	x	x		x		
Thams et al. (2017)	x			x		
Verwer et al. (2017)			x			
Xavier et al. (2021)	x					x
Yang & Bequette (2021)	x			x		

approaches. It could be promising to look into developing formal approaches for verifying learned constraint models. A framework for verifying neural network behavior was developed in Venzke & Chatzivasileiadis (2020), and research could be done to see if similar frameworks can be applied during the constraint learning process. In terms of model improvement, more focus should be put on improving models via processes such as active learning or adaptive sampling. In such cases, it would be interesting to see the effects (if any) of exploration versus exploitation when generating data on the overall solution of the optimization problem. Questions also arise with respect to the robustness of the solution – What is the sensitivity of the optimal solution with respect to the uncertainty in the learned constraint? How is the robustness of the solution related to the noise in the dataset? A structured and formal approach for verifying optimization problems with learned constraints can help to answer questions such as these.

6. Conclusions

OCL helps to capture constraints that would otherwise have been difficult to capture (Kumar et al., 2019a), providing there is data available. The benefits are clearly stated by Halilbašić et al. (2018) where their data-driven approach increases the feasible space, identifies the global optimum, and takes less time than conventional methods. There is however a need to go about the process of OCL in an organized manner, in order to avoid potential pitfalls. We have therefore provided a framework for OCL, which we believe will help to formalize and direct the process of OCL. Besides providing the framework, we reviewed the literature in light of the different steps of the framework, highlighting common trends in constraint learning, as well as possible areas for future research. It is our belief that this paper will help to guide future efforts in OCL, and be of benefit to the wider OR community.

Acknowledgements

This work was supported by the Dutch Scientific Council (NWO) grant OCENW.GROOT.2019.015, Optimization for and with Machine Learning (OPTIMAL). The authors would also like to thank Ilker Birbil for his comments during the preparation of the manuscript, and the anonymous referees for useful comments that improved the paper.

References

Abbasi, B., Babaei, T., Hosseinifard, Z., Smith-Miles, K., & Dehghani, M. (2020). Predicting solutions of large-scale optimization problems via machine learning: A case study in blood supply chain management. *Computers & Operations Research, 119*, 104941.

Amos, B., Xu, L., & Kolter, J. Z. (2017). Input convex neural networks. In *International conference on machine learning*. In *Proceedings of machine learning research* (pp. 146–155). PMLR.

De Angelis, V., Felici, G., & Impelluso, P. (2003). Integrating simulation and optimisation in health care centre management. *European Journal of Operational Research, 150*(1), 101–114.

Anderson, R., Huchette, J., Ma, W., Tjandraatmadja, C., & Vielma, J. P. (2020). Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming, 183*, 3–39.

Bagloee, S. A., Asadi, M., Sarvi, M., & Patriksson, M. (2018). A hybrid machine-learning and optimization method to solve bi-level problems. *Expert Systems with Applications, 95*, 142–152.

Baldomero-Naranjo, M., Martínez-Merino, L. I., & Rodríguez-Chía, A. M. (2020). Tightening big Ms in integer programming formulations for support vector machines with ramp loss. *European Journal of Operational Research, 286*(1), 84–100.

Beldiceanu, N., & Simonis, H. (2016). Modelseeker: Extracting global constraint models from positive examples. In *Data mining and constraint programming* (pp. 77–95). Springer.

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

Bergman, D., Huang, T., Brooks, P., Lodi, A., & Raghunathan, A. U. (2022). JANOS: An integrated predictive and prescriptive modeling framework. *INFORMS Journal on Computing, 34*(2), 807–816.

Bertsimas, D., & Kallus, N. (2020). From predictive to prescriptive analytics. *Management Science, 66*(3), 1025–1044.

Bertsimas, D., McCord, C., & Sturt, B. (2022). Dynamic optimization with side information. *European Journal of Operational Research*. https://doi.org/10.1016/j.ejor.2022.03.030.

Bertsimas, D., O'Hair, A., Relyea, S., & Silberholz, J. (2016). An analytics approach to designing combination chemotherapy regimens for cancer. *Management Science, 62*(5), 1511–1531.

Bessiere, C., Coletta, R., Freuder, E. C., & O'Sullivan, B. (2004). Leveraging the learning power of examples in automated constraint acquisition. In *International conference on principles and practice of constraint programming* (pp. 123–137). Springer.

Biggs, M., Hariss, R., & Perakis, G. (2018). Optimizing objective functions determined from random forests. Available at SSRN, 10.2139/ssrn.2986630.

Birge, J. R., & Louveaux, F. (2011). *Introduction to stochastic programming*. Springer Science & Business Media.

Bonfietti, A., Lombardi, M., & Milano, M. (2015). Embedding decision trees and random forests in constraint programming. In *International conference on ai and or techniques in constraint programming for combinatorial optimization problems* (pp. 74–90). Springer.

Cao, W., Wang, X., Ming, Z., & Gao, J. (2018). A review on neural networks with random weights. *Neurocomputing, 275*, 278–287.

Carrizosa, E., Molero-Río, C., & Romero Morales, D. (2021). Mathematical optimization in classification and regression trees. *Top, 29*(1), 5–33.

Chatzivasileiadis, S. (2020). From decision trees and neural networks to MILP: Power system optimization considering dynamic stability constraints. In *2020 European control conference (ECC)* (p. 594). IEEE.

Chen, X., Wang, Y., & Zhou, Y. (2020a). Dynamic assortment optimization with changing contextual information. *Journal of Machine Learning Research, 21*(216), 1–44.

Chen, Y., Shi, Y., & Zhang, B. (2020b). Input convex neural networks for optimal voltage regulation.

Chi, H.-M., Ersoy, O. K., Moskowitz, H., & Ward, J. (2007). Modeling and optimizing a vendor managed replenishment system using machine learning and genetic algorithms. *European Journal of Operational Research, 180*(1), 174–193.

Chiang, M., Low, S. H., Calderbank, A. R., & Doyle, J. C. (2007). Layering as optimization decomposition: A mathematical theory of network architectures. *Proceedings of the IEEE, 95*(1), 255–312.

Conn, A., Scheinberg, K., & Vicente, L. (2009). Introduction to derivative-free optimization. In *MPS-SIAM series on optimization*.

Cozad, A., Sahinidis, N. V., & Miller, D. C. (2014). Learning surrogate models for simulation-based optimization. *AIChE Journal, 60*(6), 2211–2227.

Cremer, J. L., Konstantelos, I., Tindemans, S. H., & Strbac, G. (2018). Data-driven power system operation: Exploring the balance between cost and risk. *IEEE Transactions on Power Systems, 34*(1), 791–801.

Crombecq, K., Laermans, E., & Dhaene, T. (2011). Efficient space-filling and non–collapsing sequential design strategies for simulation-based modeling. *European Journal of Operational Research, 214*(3), 683–696.

Demirovic, E., Stuckey, P. J., Bailey, J., Chan, J., Leckie, C., Ramamohanarao, K., & Guns, T. (2019). Predict+ optimise with ranking objectives: Exhaustively learning linear functions. In *International joint conference on artificial intelligence (IJCAI)* (pp. 1078–1085).

Deng, Y., Liu, J., & Sen, S. (2018). Coalescing data and decision sciences for analytics. In *Recent advances in optimization and modeling of contemporary problems* (pp. 20–49). INFORMS.

Donti, P., Amos, B., & Kolter, J. Z. (2017). Task-based end-to-end model learning in stochastic optimization. In *Advances in neural information processing systems* (pp. 5484–5494).

Drzymala, R., Mohan, R., Brewster, L., Chu, J., Goitein, M., Harms, W., & Urie, M. (1991). Dose-volume histograms. *International Journal of Radiation Oncology*Biology*Physics, 21*(1), 71–78.

D'andreagiovanni, F., & Gleixner, A. M. (2016). Towards an accurate solution of wireless network design problems. In *International symposium on combinatorial optimization* (pp. 135–147). Springer.

Elmachtoub, A. N., & Grigas, P. (2022). Smart "predict, then optimize". *Management Science, 68*(1), 9–26.

Fahmi, I., & Cremaschi, S. (2012). Process synthesis of biodiesel production plant using artificial neural networks as the surrogate models. *Computers and Chemical Engineering, 46*, 105–123.

Fischetti, M., & Fraccaro, M. (2019). Machine learning meets mathematical optimization to predict the optimal production of offshore wind parks. *Computers & Operations Research, 106*, 289–297.

Forrester, A., Sobester, A., & Keane, A. (2008). *Engineering design via surrogate modelling: Apractical guide*. Wiley.

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences, 55*(1), 119–139.

Galassi, A., Lombardi, M., Mello, P., & Milano, M. (2018). Model agnostic solution of CSPs via deep learning: A preliminary study. In *International conference on the integration of constraint programming, artificial intelligence, and operations research* (pp. 254–262). Springer.

Gambella, C., Ghaddar, B., & Naoum-Sawaya, J. (2021). Optimization problems for machine learning: A survey. *European Journal of Operational Research, 290*(3), 807–828.

GAMS Development Corporation (2021). Disjunctive programming. https://www.gams.com/latest/docs/UG_EMP_DisjunctiveProgramming.html.

Garg, A., Jalali, M., Kekatos, V., & Gatsis, N. (2018). Kernel-based learning for smart inverter control. In *2018 IEEE global conference on signal and information processing (GLOBALSIP)* (pp. 875–879). IEEE.

Garud, S. S., Karimi, I. A., & Kraft, M. (2017). Design of computer experiments: A review. *Computers and Chemical Engineering, 106*, 71–95.

Gay, H. A., & Niemierko, A. (2007). A free program for calculating EUD-based NTCP and TCP in external beam radiotherapy. *Physica Medica, 23*(3), 115–125.

Gilan, S. S., & Dilkina, B. (2015). Sustainable building design: A challenge at the intersection of machine learning and design optimization. In *AAAI workshop: Computational sustainability* (pp. 101–106).

Gorissen, B. L., Yanıkoğlu, İ., & den Hertog, D. (2015). A practical guide to robust optimization. *Omega, 53*, 124–137.

Grave, É., Joulin, A., Cissé, M., Grangier, D., & Jégou, H. (2017). Efficient softmax approximation for GPUs. In D. Precup, & Y. W. Teh (Eds.), *Proceedings of the 34th international conference on machine learning*. In *Proceedings of machine learning research: vol. 70* (pp. 1302–1310). PMLR.

Grimstad, B., & Andersson, H. (2019). Relu networks as surrogate models in mixed-integer linear programs. *Computers and Chemical Engineering, 131*, 106580.

Grossmann, I. E. (2009). *Generalized disjunctive programming*. In C. A. Floudas, & P. M. Pardalos (Eds.) (pp. 1180–1183)). Boston, MA: Springer US.

Gurobi Optimization, LLC (2022). Gurobi Optimizer Reference Manual. https://www.gurobi.com.

Gutierrez-Martinez, V. J., Cañizares, C. A., Fuerte-Esquivel, C. R., Pizano-Martinez, A., & Gu, X. (2010). Neural-network security-boundary constrained optimal power flow. *IEEE Transactions on Power Systems, 26*(1), 63–72.

Halilbašić, L., Thams, F., Venzke, A., Chatzivasileiadis, S., & Pinson, P. (2018). Data–driven security-constrained AC-OPF for operations and markets. In *2018 power systems computation conference (PSCC)* (pp. 1–7). IEEE.

Han, B., Shang, C., & Huang, D. (2021). Multiple kernel learning-aided robust optimization: Learning algorithm, computational tractability, and usage in multi-stage decision-making. *European Journal of Operational Research, 292*(3), 1004–1018.

den Hertog, D., & Stehouwer, H. (2002). Optimizing color picture tubes by high–cost non-linear programming. *European Journal of Operational Research, 140*(2), 197–211.

Hoffmann, A. L., den Hertog, D., Siem, A. Y., Kaanders, J. H., & Huizenga, H. (2008). Convex reformulation of biologically-based multi-criteria intensity-modulated radiation therapy optimization including fractionation effects. *Physics in Medicine and Biology, 53*(22), 6345.

Hottung, A., Tanaka, S., & Tierney, K. (2020). Deep learning assisted heuristic tree search for the container pre-marshalling problem. *Computers & Operations Research, 113*, 104781.

IBM (2022). User's Manual for CPLEX. https://www.ibm.com/docs/en/icos/12.8.0.0?topic=cplex-users-manual.

Jaccard, P. (1912). The distribution of the flora in the alpine zone. 1. *New Phytologist, 11*(2), 37–50.

Jalali, M., Kekatos, V., Gatsis, N., & Deka, D. (2019). Designing reactive power control rules for smart inverters using support vector machines. *IEEE Transactions on Smart Grid, 11*(2), 1759–1770.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning*. Springer.

Jiménez-Cordero, A., Morales, J. M., & Pineda, S. (2021). A novel embedded min-max approach for feature selection in nonlinear support vector machine classification. *European Journal of Operational Research, 293*(1), 24–35.

Karimi-Mamaghan, M., Mohammadi, M., Meyer, P., Karimi-Mamaghan, A. M., & Talbi, E.-G. (2022). Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. *European Journal of Operational Research, 296*(2), 393–422.

Karmelita, M., & Pawlak, T. P. (2020). CMA-ES for one-class constraint synthesis. In *Proceedings of the 2020 genetic and evolutionary computation conference* (pp. 859–867).

Khalil, E., Le Bodic, P., Song, L., Nemhauser, G., & Dilkina, B. (2016). Learning to branch in mixed integer programming. *Proceedings of the AAAI Conference on Artificial Intelligence, 30*, 1.

Kleijnen, J. (2015). *Design and analysis of simulation experiments*. Springer.

Kolb, S. M. (2016). Learning constraints and optimization criteria. In *Workshops at the thirtieth aaai conference on artificial intelligence* (pp. 403–409).

Kotary, J., Fioretto, F., Hentenryck, P. V., & Wilder, B. (2021). End-to-end constrained optimization learning: A survey.

Kudła, P., & Pawlak, T. P. (2018). One-class synthesis of constraints for mixed-integer linear programming with c4.5 decision trees. *Applied soft computing, 68*, 1–12.

Kuhn, M., & Johnson, K. (2019). *Feature engineering and selection: A practical approach for predictive models*. CRC Press.

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

Kumar, M., Teso, S., De Causmaecker, P., & De Raedt, L. (2019a). Automating personnel rostering by learning constraints using tensors. In *2019 ieee 31st international conference on tools with artificial intelligence (ICTAI)* (pp. 697–704). IEEE.

Kumar, M., et al., (2019b). Acquiring integer programs from data. In *Proceedings of the twenty-eighth international joint conference on artificial intelligence* (pp. 1130–1136). IJCAI.

Lombardi, M., & Milano, M. (2018). Boosting combinatorial problem modeling with machine learning. In *Proceedings of the twenty-seventh international joint conference on artificial intelligence, IJCAI-18* (pp. 5472–5478). International Joint Conferences on Artificial Intelligence Organization.

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

de Mast, J., Steiner, S. H., Nuijten, W. P. M., & Kapitan, D. (2022). Analytical problem solving based on causal, correlational and deductive models. *The American Statistician*, 1–11.

Maragno, D., Wiberg, H., Bertsimas, D., Birbil, S. I., den Hertog, D., & Fajemisin, A. (2021). Mixed-integer optimization with constraint learning.

Matheron, G. (1963). Principles of geostatistics. *Economic Geology, 58*(8), 1246–1266.

McKinney, W. (2017). *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. O'Reilly Media, Inc..

Mišić, V. V. (2020). Optimization of tree ensembles. *Operations Research, 68*(5), 1605–1624.

Molnar, C. (2020). *Interpretable machine learning*. Lulu.com.

Mukerjee, R., & Wu, C.-F. (2006). *A modern theory of factorial design*. Springer.

Murzakhanov, I., Venzke, A., Misyris, G. S., & Chatzivasileiadis, S. (2020). Neural networks for encoding dynamic security-constrained optimal power flow to mixed-integer linear programs.

Nagata, Y., & Chu, K. H. (2003). Optimization of a fermentation medium using neural networks and genetic algorithms. *Biotechnology Letters, 25*(21), 1837–1842.

Nascimento, C. A. O., Giudici, R., & Guardani, R. (2000). Neural network based approach for optimization of industrial chemical processes. *Computers and Chemical Engineering, 24*(9–10), 2303–2314.

Ng, A. (2020). Neural networks and deep learning. https://www.coursera.org/specializations/deep-learning.

Nieuwenhuis, R., & Oliveras, A. (2006). On SAT modulo theories and optimization problems. In *International conference on theory and applications of satisfiability testing* (pp. 156–169). Springer.

O'Sullivan, B. (2010). Automated modelling and solving in constraint programming. In *Proceedings of the twenty-fourth AAAI conference on artificial intelligence AAAI-10: vol. 24* (pp. 1493–1497).

O'Neil, M., & Ryan, C. (2003). Grammatical evolution: Evolutionary automatic programming in an arbitrary language. *Kluwer Academic Publishers, 10*, 978-1.

Padmanabhan, B., & Tuzhilin, A. (2003). On the use of optimization for data mining: Theoretical interactions and ecrm opportunities. *Management Science, 49*(10), 1327–1343.

Paulus, A., Rolínek, M., Musil, V., Amos, B., & Martius, G. (2021). Comboptnet: Fit the right np-hard problem by learning integer programming constraints. In *International conference on machine learning* (pp. 8443–8453). PMLR.

Pawlak, T. P. (2019). Synthesis of mathematical programming models with one-class evolutionary strategies. *Swarm and Evolutionary Computation, 44*, 335–348.

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

Pawlak, T. P., & Krawiec, K. (2017b). Synthesis of mathematical programming constraints with genetic programming. In *European conference on genetic programming* (pp. 178–193). Springer.

Pawlak, T. P., & Krawiec, K. (2018). Synthesis of constraints for mathematical programming with one-class genetic programming. *IEEE Transactions on Evolutionary Computation, 23*(1), 117–129.

Pawlak, T. P., & Litwiniuk, B. (2021). Ellipsoidal one-class constraint acquisition for quadratically constrained programming. *European Journal of Operational Research, 293*(1), 36–49.

Pawlak, T. P., & O'Neill, M. (2021). Grammatical evolution for constraint synthesis for mixed-integer linear programming. *Swarm and Evolutionary Computation, 64*, 100896.

Peters, K., Silva, S., Gonçalves, R., Kavelj, M., Fleuren, H., den Hertog, D., … Freeman, M. (2021). The nutritious supply chain: Optimizing humanitarian food assistance. *Informs Journal on Optimization, 3*(2), 200–226.

Prat, E., & Chatzivasileiadis, S. (2020). Learning active constraints to efficiently solve bilevel problems. arXiv preprint arXiv:2010.06344.

Say, B., Devriendt, J., Nordström, J., & Stuckey, P. J. (2020). Theoretical and experimental results for planning with learned binarized neural network transition models. In *International conference on principles and practice of constraint programming* (pp. 917–934). Springer.

Say, B., & Sanner, S. (2018). Planning in factored state and action spaces with learned binarized neural network transition models. In *Proceedings of the twenty-seventh international joint conference on artificial intelligence IJCAI-18* (pp. 4815–4821).

Say, B., Wu, G., Zhou, Y. Q., & Sanner, S. (2017). Nonlinear hybrid planning with deep net learned transition models and mixed-integer linear programming. In *Proceedings of the twenty-sixth international joint conference on artificial intelligence IJCAI-17* (pp. 750–756).

Schede, E. A., Kolb, S., & Teso, S. (2019). Learning linear programs from data. In *2019 ieee 31st international conference on tools with artificial intelligence (ICTAI)* (pp. 1019–1026). IEEE.

Schweidtmann, A. M., & Mitsos, A. (2019). Deterministic global optimization with artificial neural networks embedded. *Journal of Optimization Theory and Applications, 180*(3), 925–948.

Scott, J. G., Sedor, G., Scarborough, J. A., Kattan, M. W., Peacock, J., Grass, G. D., … Waller, A., et al., (2021). Personalizing radiotherapy prescription dose using genomic markers of radiosensitivity and normal tissue toxicity in nsclc. *Journal of Thoracic Oncology, 16*(3), 428–438.

Settles, B. (2009). Active learning literature survey. *Computer Sciences Technical Report*. University of Wisconsin–Madison.

Shang, C., Huang, X., & You, F. (2017). Data-driven robust optimization based on kernel learning. *Computers and Chemical Engineering, 106*, 464–479.

Sharma, S. (2017). Activation functions in neural networks. shorturl.at/ACOP26.

Sroka, D., & Pawlak, T. P. (2018). One-class constraint acquisition with local search. In *Proceedings of the genetic and evolutionary computation conference* (pp. 363–370).

Stinstra, E., & den Hertog, D. (2008). Robust optimization using computer experiments. *European Journal of Operational Research, 191*(3), 816–837.

Sun, S., Cao, Z., Zhu, H., & Zhao, J. (2019). A survey of optimization methods from a machine learning perspective. *IEEE transactions on cybernetics, 50*(8), 3668–3681.

Thams, F., Halilbasic, L., Pinson, P., Chatzivasileiadis, S., & Eriksson, R. (2017). Data–driven security-constrained OPF. In *Proceedings of the 10th bulk power systems dynamics and control symposium* (pp. 1–10).

Tucker, S. L., Li, M., Xu, T., Gomez, D., Yuan, X., Yu, J., … Liao, Z. (2013). Incorporating single-nucleotide polymorphisms into the Lyman model to improve prediction of radiation pneumonitis. *International Journal of Radiation Oncology*Biology*Physics, 85*(1), 251–257.

UNWFP, U. (2021). The WFP food basket. https://www.wfp.org/wfp-food-basket.

Vaclavik, R., Novak, A., Sucha, P., & Hanzlek, Z. (2018). Accelerating the branch-and-price algorithm using machine learning. *European Journal of Operational Research, 271*(3), 1055–1069.

Venzke, A., & Chatzivasileiadis, S. (2020). Verification of neural network behaviour: Formal guarantees for power system applications. *IEEE Transactions on Smart Grid, 12*(1), 383–397.

Venzke, A., Qu, G., Low, S., & Chatzivasileiadis, S. (2020). Learning optimal power flow: Worst-case guarantees for neural networks. In *2020 ieee international conference on communications, control, and computing technologies for smart grids (smartgridcomm)* (pp. 1–7). IEEE.

Verwer, S., Zhang, Y., & Ye, Q. C. (2017). Auction optimization using regression trees and linear models as integer programs. *Artificial Intelligence, 244*, 368–395.

Villarrubia, G., De Paz, J. F., Chamoso, P., & De la Prieta, F. (2018). Artificial neural networks used in optimization problems. *Neurocomputing, 272*, 10–16.

Xavier, Á. S., Qiu, F., & Ahmed, S. (2021). Learning to solve large-scale security–constrained unit commitment problems. *INFORMS Journal on Computing, 33*(2), 739–756.

Xu, S., Liu, H., Wang, X., & Jiang, X. (2014). A robust error-pursuing sequential sampling approach for global metamodeling based on Voronoi diagram and cross validation. *Journal of Mechanical Design, 136*(7), 071009.

Yang, S., & Bequette, B. W. (2021). Optimization-based control using input convex neural networks. *Computers and Chemical Engineering, 144*, 107143.

Yao, X., Crook, J., & Andreeva, G. (2017). Enhancing two-stage modelling methodology for loss given default with support vector machines. *European Journal of Operational Research, 263*(2), 679–689.

Yu, W., & Lui, R. (2006). Dual methods for nonconvex spectrum optimization of multicarrier systems. *IEEE Transactions on Communications, 54*(7), 1310–1322.