Mixed-Integer Optimization with Constraint Learning

Donato Maragno*

Amsterdam Business School, University of Amsterdam, 1018 TV Amsterdam, Netherlands d.maragno@uva.nl

Holly Wiberg*

Operations Research Center, Massachusetts Institute of Technology, Cambridge MA 02139 hwiberg@mit.edu

Dimitris Bertsimas

Sloan School of Management, Massachusetts Institute of Technology, Cambridge MA 02139 dbertsim@mit.edu

Ş. İlker Birbil, Dick den Hertog, Adejuyigbe Fajemisin

Amsterdam Business School, University of Amsterdam, 1018 TV Amsterdam, Netherlands
s.i.birbil@uva.nl d.denhertog@uva.nl a.o.fajemisin2@uva.nl

We establish a broad methodological foundation for mixed-integer optimization with learned constraints. We propose an end-to-end pipeline for data-driven decision making in which constraints and objectives are directly learned from data using machine learning, and the trained models are embedded in an optimization formulation. We exploit the mixed-integer optimization-representability of many machine learning methods, including linear models, decision trees, ensembles, and multi-layer perceptrons. The consideration of multiple methods allows us to capture various underlying relationships between decisions, contextual variables, and outcomes. We also characterize a decision trust region using the convex hull of the observations, to ensure credible recommendations and avoid extrapolation. We efficiently incorporate this representation using column generation and clustering. In combination with domain-driven constraints and objective terms, the embedded models and trust region define a mixed-integer optimization problem for prescription generation. We implement this framework as a Python package (`OptiCL`) for practitioners. We demonstrate the method in both chemotherapy optimization and World Food Programme planning. The case studies illustrate the benefit of the framework in generating high-quality prescriptions, the value added by the trust region, the incorporation of multiple machine learning methods, and the inclusion of multiple learned constraints.

Key words: mixed-integer optimization, machine learning, constraint learning, prescriptive analytics

---

* These authors contributed equally.

1. Introduction

Mixed-integer optimization (MIO) is a powerful tool that allows us to optimize a given objective subject to various constraints. This general problem statement of optimizing under constraints is nearly universal in decision-making settings. Some problems have readily quantifiable and explicit objectives and constraints, in which case MIO can be directly applied. The situation becomes more complicated, however, when the constraints and/or objectives are not explicitly known.

For example, suppose we deal with cancerous tumors and want to prescribe a treatment regimen with a limit on toxicity; we may have observational data on treatments and their toxicity outcomes, but we have no natural function that relates the treatment decision to its resultant toxicity. We may also encounter constraints that are not directly quantifiable. Consider a setting where we want to recommend a diet, defined by a combination of foods and quantities, that is sufficiently "palatable." Palatability cannot be written as a function of the food choices, but we may have qualitative data on how well people "like" various potential dietary prescriptions. In both of these examples, we cannot directly represent the outcomes of interest as functions of our decisions, but we have *data* that relates the outcomes and decisions. This raises a question: how can we consider data to learn these functions?

In this work, we tackle the challenge of data-driven decision making through a combined machine learning (ML) and MIO approach. ML allows us to learn functions that relate decisions to outcomes of interest directly through data. Importantly, many popular ML methods result in functions that are MIO-representable, meaning that they can be embedded into MIO formulations. This MIO-representable class includes both linear and nonlinear models, allowing us to capture a broad set of underlying relationships in the data. While the idea of learning functions directly from data is core to the field of ML, data is often underutilized in MIO settings due to the need for functional relationships between decision variables and outcomes. We seek to bridge this gap through *constraint learning*; we propose a general framework that allows us to learn constraints and objectives directly from data, using ML, and to optimize decisions accordingly, using MIO.

Once the learned constraints have been incorporated into the larger MIO, we can solve the problem directly using off-the-shelf solvers.

The term *constraint learning*, used several times throughout this work, captures both constraints and objective functions. We are fundamentally learning functions to relate our decision variables to the outcome(s) of interest. The predicted values can then either be incorporated as constraints or objective terms; the model learning and embedding procedures remain largely the same. For this reason, we refer to them both under the same umbrella of *constraint learning*. We describe this further in Section 2.2.

1.1. Literature review

Previous work has demonstrated the use of various ML methods in MIO problems and their utility in different application domains. The simplest of these methods is the regression function, as the approach is easy to understand and easy to implement. Given a regression function learned from data, the process of incorporating it into an MIO model is straightforward, and the final model does not require complex reformulations. As an example, Bertsimas et al. (2016) use regression models and MIO to develop new chemotherapy regimens based on existing data from previous clinical trials. Kleijnen (2015) provides further information on this subject.

More complex ML models have also been shown to be MIO-representable, although more effort is required to represent them than simple regression models. Neural networks which use the ReLU activation function can be represented using binary variables and big-M formulations (Amos et al. 2016, Grimstad and Andersson 2019, Anderson et al. 2020, Chen et al. 2020, Spyros 2020, Venzke et al. 2020). Where other activation functions are used (Gutierrez-Martinez et al. 2011, Lombardi et al. 2017, Schweidtmann and Mitsos 2019), the MIO representation of neural networks is still possible, provided the solvers are capable of handling these functions.

With decision trees, each path in the tree from root to leaf node can be represented using one or more constraints (Bonfietti et al. 2015, Verwer et al. 2017, Halilbasic et al. 2018). The number of constraints required to represent decision trees is a function of the tree size, with larger

trees requiring more linearizations and binary variables. The advantage here, however, is that decision trees are known to be highly interpretable, which is often a requirement of ML in critical application settings (Thams et al. 2017). Random forests (Biggs et al. 2021, Mišić 2020) and other tree ensembles (Cremer et al. 2019) have also been used in MIO in the same way as decision trees, with one set of constraints for each tree in the forest/ensemble along with one or more additional aggregate constraints.

Data for constraint learning can either contain information on continuous data, feasible and infeasible states (two-class data), or only one state (one-class data). The problem of learning functions from one-class data and embedding them into optimization models has been recently investigated with the use of decision trees (Kudła and Pawlak 2018), genetic programming (Pawlak and Krawiec 2019), local search (Sroka and Pawlak 2018), evolutionary strategies (Pawlak 2019), and a combination of clustering, principal component analysis and wrapping ellipsoids (Pawlak and Litwiniuk 2021).

The above selected applications generally involve a single function to be learned and a fixed ML method for the model choice. Verwer et al. (2017) use two model classes (decision trees and linear models) in a specific auction design application, but in this case the models were determined a priori. Some authors have presented a more general framework of embedding learned ML models in optimization problems (Lombardi et al. 2017, Bergman et al. 2019), but in practice these works are restricted to limited problem structures and learned model classes. Recently, Bergman et al. (2019) introduced a software to embed neural networks and logistic and linear regression models as objective terms in an MIO formulation. These works can be viewed as special cases of our framework and cannot be directly applied in our case studies. We take a broader perspective, proposing a comprehensive end-to-end pipeline that encompasses the full ML and optimization components of a data-driven decision making problem.

Our work falls under the umbrella of prescriptive analytics. Bertsimas and Kallus (2020) and Elmachtoub and Grigas (2021) leverage ML model predictions as inputs into an optimization problem. Our approach is distinct from existing work in that we directly embed ML models rather than

extracting predictions, allowing us to optimize our decisions over the model. In the broadest sense, our framework relates to work that jointly harnesses ML and MIO, an area that has garnered significant interest in recent years in both the optimization and machine learning communities (Bengio et al. 2021).

1.2. Contributions

Our work unifies several research areas in a comprehensive manner. Our key contributions are as follows:

1. We develop an end-to-end framework that takes data and directly implements model training, model selection, integration into a larger MIO, and ultimately optimization. We make this available as an open-source software, `OptiCL` (Optimization with Constraint Learning) to provide a practitioner-friendly tool for making better data-driven decisions. The code is available at `https://github.com/hwiberg/OptiCL`.

2. We implement a model selection procedure that allows us to capture quite general functional relationships between contextual variables, treatments, and outcomes. We use cross-validation to select from a broad set of ML methods, assuming no single model's dominance, and further allow for the combined use of different algorithms for different outcomes. Our framework supports models for both regression and classification functions, and handles constraint learning in cases with both one-class and two-class data. Additionally, we give mathematical representations of the ML functions to enable their use in MIO applications.

3. Due to the uncertainty associated with learning from data, we introduce a concept which we call a trust region. This allows us to restrict the solution of the optimization problem to be consistent with the domain of the predictive models. Defining this trust region in cases where there is a huge amount of data to learn from can be computationally intensive, so we also provide a column selection algorithm that significantly improves the computation time. We furthermore propose a clustering heuristic for a general MIO formulation that shows significant computational gains while obtaining near-optimality. These approaches allow us to reduce the computational burden of our approach while keeping the benefits of the trust region.

4. We demonstrate the power of our method in two real-world case studies, using data from the World Food Programme and chemotherapy clinical trials. We pose relevant questions in the respective areas and formalize them as constraint learning problems. We implement our framework and subsequently evaluate the quantitative performance and scalability of our methods in these settings.

2. Methodology

Suppose we have data $\mathcal{D} = \{(\bar{\boldsymbol{x}}_i, \bar{\boldsymbol{w}}_i, \bar{\boldsymbol{y}}_i)\}_{i=1}^N$, with observed treatment decisions $\bar{\boldsymbol{x}}_i$, contextual information $\bar{\boldsymbol{w}}_i$, and outcomes of interest $\bar{\boldsymbol{y}}_i$ for sample i. Following the guidelines proposed in Fajemisin et al. (2021), we present a framework that, given data $\mathcal{D}$, learns functions for the outcomes of interest ($\boldsymbol{y}$) that are to be constrained or optimized. These learned representations can then be used to generate predictions for a new observation with context $\boldsymbol{w}$. Figure 1 outlines the complete pipeline, which is detailed in the sections below.

2.1. Conceptual model

Given the decision variable $\boldsymbol{x} \in \mathbb{R}^n$ and the fixed feature vector $\boldsymbol{w} \in \mathbb{R}^p$, we propose model M($\boldsymbol{w}$)

$$
\begin{aligned}
\min_{\boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{y} \in \mathbb{R}^k} \quad & f(\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{y}) \\
\text{s.t.} \quad & \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{y}) \leq \boldsymbol{0}, \\
& \boldsymbol{y} = \hat{\boldsymbol{h}}_{\mathcal{D}}(\boldsymbol{x}, \boldsymbol{w}), \\
& \boldsymbol{x} \in \mathcal{X}(\boldsymbol{w}),
\end{aligned}
\tag{1}
$$

where $f(., \boldsymbol{w}, .) : \mathbb{R}^{n+k} \mapsto \mathbb{R}$, $\boldsymbol{g}(., \boldsymbol{w}, .) : \mathbb{R}^{n+k} \mapsto \mathbb{R}^m$, and $\hat{\boldsymbol{h}}_{\mathcal{D}}(., \boldsymbol{w}) : \mathbb{R}^n \mapsto \mathbb{R}^k$. Explicit forms of f and $\boldsymbol{g}$ are known but they may still depend on the predicted outcome $\boldsymbol{y}$. Here, $\hat{\boldsymbol{h}}_{\mathcal{D}}(\boldsymbol{x}, \boldsymbol{w})$ represents the predictive models, one per outcome of interest, which are ML models trained on $\mathcal{D}$. Although our subsequent discussion mainly revolves around linear functions, we acknowledge the significant progress in nonlinear (convex) integer solvers. Our discussion can be easily extended to nonlinear models that can be tackled by those ever-improving solvers.

Figure 1   Constraint learning and optimization pipeline.
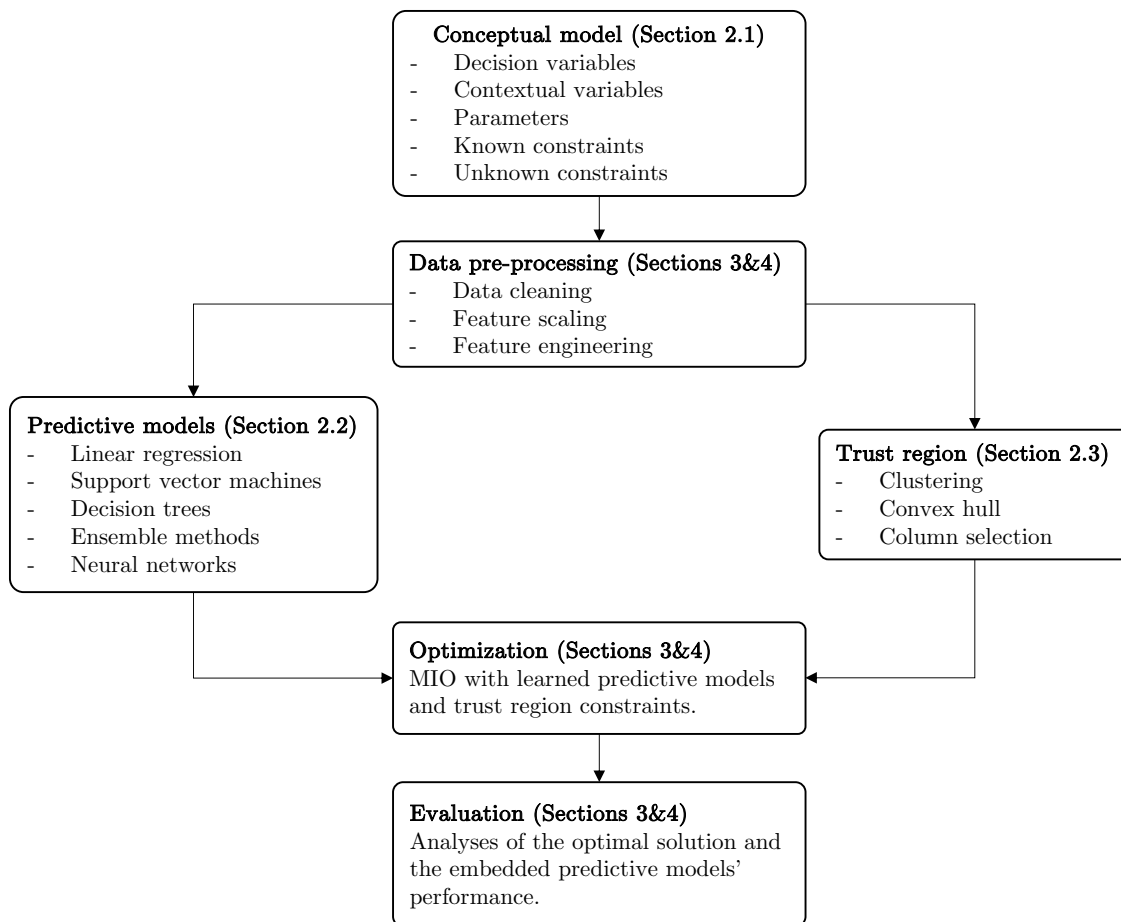
We note that the embedding of a single learned outcome may require multiple constraints and auxiliary variables; the embedding formulations are described in Section 2.2. For simplicity, we omit $\mathcal{D}$ in further notation of $\hat{\boldsymbol{h}}$ but note that all references to $\hat{\boldsymbol{h}}$ implicitly depend on the data used to train the model. Finally, the set $\mathcal{X}(\boldsymbol{w})$ defines the trust region, *i.e.*, the set of solutions for which we trust the embedded predictive models. In Section 2.3, we provide a detailed description of how the trust region $\mathcal{X}(\boldsymbol{w})$ is obtained from the observed data. We refer to the final MIO formulation with the embedded constraints and variables as $\mathrm{EM}(\boldsymbol{w})$.

Model $\mathrm{M}(\boldsymbol{w})$ is quite general and encompasses several important *constraint learning* classes:

1. **Regression.** When the trained model results from a regression problem, it can be constrained by a specified upper bound τ, *i.e.*, $g(y) = y - \tau \leq 0$, or lower bound τ, *i.e.*, $g(y) = -y + \tau \leq 0$.

If $\boldsymbol{y}$ is a vector (*i.e.*, multi-output regression), we can likewise provide a threshold vector $\boldsymbol{\tau}$ for the constraints.

2. **Classification.** If the trained model is obtained with a binary classification algorithm, in which the data is labeled as "feasible" (1) or "infeasible" (0), then the prediction is generally a probability $y \in [0, 1]$. We can enforce a lower bound on the feasibility probability, *i.e.*, $y \geq \tau$. A natural choice of τ is 0.5, which can be interpreted as enforcing that the result is more likely feasible than not. This can also extend to the multi-class setting, say k classes, in which the output $\boldsymbol{y}$ is a k-dimensional unit vector, and we apply the constraint $y_i \geq \tau$ for whichever class i is desired. When multiple classes are considered to be feasible, we can add binary variables to ensure that a solution is feasible, only if it falls in one of these classes with sufficiently high probability.

3. **Objective function.** If the objective function has a term that is also learned by training an ML model, then we can introduce an auxiliary variable $t \in \mathbb{R}$, and add it to the objective function along with an epigraph constraint. Suppose for simplicity that the model involves a single learned objective function, $\hat{h}$, and no learned constraints. Then the general model becomes

$$\min_{\boldsymbol{x} \in \mathbb{R}^n, y \in \mathbb{R}, t \in \mathbb{R}} t$$

$$\text{s.t. } \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{w}) \leq 0,$$

$$y = \hat{h}(\boldsymbol{x}, \boldsymbol{w}),$$

$$y - t \leq 0,$$

$$\boldsymbol{x} \in \mathcal{X}(\boldsymbol{w}).$$

Although we have rewritten the problem to show the generality of our model, it is quite common in practice to use y in the objective and omit the auxiliary variable t.

We observe that constraints on learned outcomes can be applied in two ways depending on the model training approach. Suppose that we have a continuous scalar outcome y to learn and we

want to impose an upper bound of $\tau \in \mathbb{R}$ (it may also be a lower bound without loss of generality). The first approach is called *function learning* and concerns all cases where we learn a regression function $\hat{h}(\boldsymbol{x}, \boldsymbol{w})$ without considering the feasibility threshold (τ). The resultant model returns a predicted value $y \in \mathbb{R}$. The threshold is then applied as a constraint in the optimization model as $y \leq \tau$. Alternatively, we could use the feasibility threshold τ to binarize the outcome of each sample in $\mathcal{D}$ into feasible and infeasible, that is $\bar{y}_i := \mathbb{I}(\bar{y}_i \leq \tau), \ i = 1, \ldots, N$, where $\mathbb{I}$ stands for the indicator function. After this relabeling, we train a binary classification model $\hat{h}(\boldsymbol{x}, \boldsymbol{w})$ that returns a probability $y \in [0, 1]$. This approach, called *indicator function learning*, does not require any further use of the feasibility threshold τ in the optimization model, since the predictive models directly encode feasibility.

The function learning approach is particularly useful when we are interested in varying the threshold τ as a model parameter. Additionally, if the fitting process is expensive and therefore difficult to perform multiple times, learning an indicator function for each potential τ might be infeasible. In contrast, the indicator function learning approach is necessary when the raw data contains binary labels rather than continuous outcomes, and thus we have no ability to select or vary τ.

2.2. MIO-representable predictive models

Our framework is enabled by the ability to embed learned predictive models into an MIO formulation with linear constraints. This is possible for many classes of ML models, ranging from linear models to ensembles, and from support vector machines to neural networks. In this section, we detail the embedding procedure. In all cases, the model has been *pre-trained*; we embed the trained model $\hat{h}(\boldsymbol{x}, \boldsymbol{w})$ into our larger MIO formulation to allow us to constrain or optimize the resultant predicted value. Consequently, the optimization model is not dependent on the complexity of the model training procedure, but solely the size of the final trained model. Without loss of generality, we assume that y is one-dimensional; *i.e.*, we are learning a single model, and this model returns a scalar, not a multi-output vector.

All of the methods below can be used to learn constraints that apply upper or lower bounds to y, or to learn y that we incorporate as part of the objective. We present the model embedding procedure for both cases when $\hat{h}(\boldsymbol{x}, \boldsymbol{w})$ is a continuous or a binary predictive model, where relevant. We assume that either regression or classification models can be used to learn feasibility constraints, as described in Section 2.1.

Linear Regression. Linear regression (LR) is a natural choice of predictive function given its inherent linearity and ease of embedding. A regression model can be trained to predict the outcome of interest, y, as a function of $\boldsymbol{x}$ and $\boldsymbol{w}$. The algorithm can optionally use regularization; the embedding only requires the final coefficient vectors $\boldsymbol{\beta_x} \in \mathbb{R}^n$ and $\boldsymbol{\beta_w} \in \mathbb{R}^p$ (and intercept term β_0) to describe the model. The model can then be embedded as

$$y = \beta_0 + \boldsymbol{\beta_x^\top} \boldsymbol{x} + \boldsymbol{\beta_w^\top} \boldsymbol{w}.$$

Support Vector Machines. A support vector machine (SVM) uses a hyper-plane split to generate predictions, both for classification (Cortes and Vapnik 1995) and regression (Drucker et al. 1997). We consider the case of *linear* SVMs, since this allows us to obtain the prediction as a linear function of the decision variables $\boldsymbol{x}$. In linear support vector regression (SVR), which we use for function learning, we fit a linear function to the data. The setting is similar to linear regression, but the loss function only penalizes residuals greater than an ϵ threshold (Drucker et al. 1997). As with linear regression, the trained model returns a linear function with coefficients $\boldsymbol{\beta_x}$, $\boldsymbol{\beta_w}$, and β_0. The final prediction is

$$y = \beta_0 + \boldsymbol{\beta_x^\top} \boldsymbol{x} + \boldsymbol{\beta_w^\top} \boldsymbol{w}.$$

For the classification setting, linear support vector classification (SVC) identifies a hyper-plane that best separates positive and negative samples (Cortes and Vapnik 1995). A trained SVC model similarly returns coefficients $\boldsymbol{\beta_x}$, $\boldsymbol{\beta_w}$, and β_0, where a sample's prediction is given by

$$y = \begin{cases} 1, & \text{if } \beta_0 + \boldsymbol{\beta_x^\top} \boldsymbol{x} + \boldsymbol{\beta_w^\top} \boldsymbol{w} \geq 0; \\ 0, & \text{otherwise.} \end{cases}$$

In SVC, the output variable y is binary rather than a probability. In this case, the constraint can be embedded as $y \geq 1$.

Decision Trees. Decision trees partition observations into distinct *leaves* through a series of *feature splits*. These algorithms are popular in predictive tasks due to their natural interpretability and ability to capture nonlinear interactions among variables. Breiman et al. (1984) first introduced Classification and Regression Trees (CART), which constructs trees through parallel splits in the feature space. Decision tree algorithms have subsequently been adapted and extended. Bertsimas and Dunn (2017) propose an alternative decision tree algorithm, Optimal Classification Trees (and Optimal Regression Trees), that improves on the basic decision tree formulation through an optimization framework that approximates globally optimal trees. Optimal trees also support multi-feature splits, referred to as *hyper-plane splits*, that allow for splits on a linear combination of features (Bertsimas, D. and Dunn, J. 2018).

A generic decision tree of depth 2 is shown in Figure 2. A split at node i is described by an inequality $A_i^\top \boldsymbol{x} \leq b_i$. We assume that A can have multiple non-zero elements, in which we have the hyper-plane split setting; if there is only one non-zero element, this creates a parallel (single feature) split. Each terminal node j (*i.e.*, leaf) yields a prediction (p_j) for its observations. In the case of regression, the prediction is the average value of the training observations in the leaf, and in binary classification, the prediction is the proportion of leaf members with the feasible class. Each leaf can be described as a polyhedron, namely a set of linear constraints that must be satisfied by all leaf members. For example, for node 3, we define $\mathcal{P}_3 = \left\{ x : A_1^\top x \leq b_1, A_2^\top x \leq b_2 \right\}$.

Suppose that we wish to constrain the predicted value of this tree to be at most τ, a fixed constant. After obtaining the tree in Figure 2, we can identify which paths satisfy the desired bound ($p_i \leq \tau$). Suppose that p_3 and p_6 do satisfy the bound, but p_4 and p_7 do not. In this case, we can enforce that our solution belongs to $\mathcal{P}_3$ or $\mathcal{P}_6$. This same approach applies if we only have access to two-class data (feasible vs. infeasible); we can directly train a binary classification algorithm and

Figure 2   A decision tree of depth 2 with four terminal nodes (leaves).
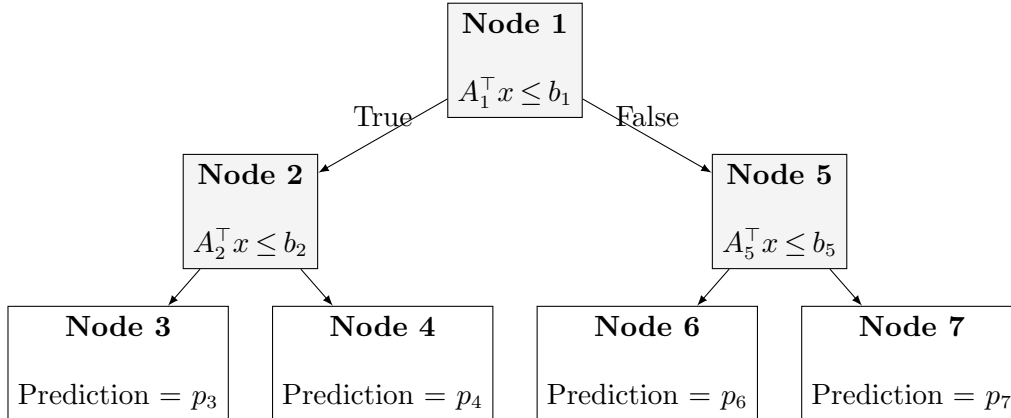
enforce that the solution lies within one of the "feasible" prediction leaves (determined by a set probability threshold).

If the decision tree provides our only learned constraint, we can decompose the problem into multiple separate MIOs, one per feasible leaf. The conceptual model for the subproblem of leaf i then becomes

$$\min_{\boldsymbol{x}} \; f(\boldsymbol{x}, \boldsymbol{w})$$

$$\text{s.t. } \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{w}) \leq 0,$$

$$(\boldsymbol{x}, \boldsymbol{w}) \in \mathcal{P}_i,$$

where the learned constraints for leaf i's subproblem are implicitly represented by the polyhedron $\mathcal{P}_i$. These subproblems can be solved in parallel, and the minimum across all subproblems is obtained as the optimal solution. Furthermore, if all decision variables $\boldsymbol{x}$ are continuous, these subproblems are linear optimization problems (LOs), which can provide substantial computational gains. This is explored further in Appendix A.2.

In the more general setting where the decision tree forms one of many constraints, or we are interested in varying the τ limit within the model, we can directly embed the model into a larger MIO. We add binary variables representing each leaf, and set y to the predicted value of the assigned leaf. An observation can only be assigned to a leaf, if it obeys all of its constraints; the structure of the tree guarantees that exactly one path will be fully satisfied, and thus, the leaf

assignment is uniquely determined. A solution belonging to $\mathcal{P}_3$ will inherit $y = p_3$. Then, y can be used in a constraint or objective. The full formulation for the embedded decision tree is included in Appendix A.1.

Ensemble Methods. Ensemble methods, such as random forests (RF) and gradient-boosting machines (GBM) consist of many decision trees that are aggregated to obtain a single prediction for a given observation. These models can thus be implemented by embedding many "sub-models" (Breiman 2001). Suppose we have a forest with P trees. Each tree can be embedded as a single decision tree (see previous paragraph) with the constraints from Appendix A.1, which yields a predicted value y_i.

RF models typically generate predictions by taking the average of the predictions from the individual trees:

$$y = \frac{1}{P} \sum_{i=1}^{P} y_i.$$

This can then be used as a term in the objective, or constrained by an upper bound as $y \leq \tau$; this can be done equivalently for a lower bound. In the classification setting, the prediction averages the probabilities returned by each model ($y_i \in [0,1]$), which can likewise be constrained or optimized.

Alternatively, we can further leverage the fact that unlike the other model classes, which return a single prediction, the RF model generates P predictions, one per tree. When constraining the prediction, we can optionally impose a violation limit, enforcing that the constraint must hold for most of the trees within the forest, but can be violated by a proportion $\alpha \in [0,1]$. This allows for a degree of robustness to individual model predictions by discarding a small number of potential outlier predictions:

$$\frac{1}{P} \sum_{i=1}^{P} \mathbb{I}(y_i \leq \tau) \geq 1 - \alpha.$$

Note that $\alpha = 0$ enforces the bound for all trees within the forest, yielding the most conservative estimate, whereas $\alpha = 1$ removes the constraint entirely.

In the case of GBM, we have an ensemble of base-learners which are not necessarily decision trees. The model output is then computed as

$$y = \sum_{i=1}^{P} \beta_i y_i,$$

where y_i is the predicted value of the i-th regression model $\hat{h}_i(\boldsymbol{x}, \boldsymbol{w})$, β_i is the weight associated with the prediction. Although trees are typically used as base-learners, in theory we might use any of the MIO-representable predictive models discussed in this section.

Neural Networks. We implement multi-layer perceptrons (MLP) with a rectified linear unit (ReLU) activation function, which form an MIO-representable class of neural networks (Grimstad and Andersson 2019, Anderson et al. 2020). These networks consist of an input layer, $L-2$ hidden layer(s), and an output layer. In a given hidden layer l of the network, with nodes N^l, the value of a node $i \in N^l$, denoted as v_i^l, is calculated using the weighted sum of the previous layer's node values, followed by the ReLU activation function, $\mathrm{ReLU}(x) = \max\{0, x\}$. The value is given as

$$v_i^l = \max\left\{0, \beta_{i0}^l + \sum_{j \in N^{l-1}} \beta_{ij}^l v_j^{l-1}\right\},$$

where $\boldsymbol{\beta}_i^l$ is the coefficient vector for node i in layer l. This nonlinear transformation of the input space over multiple nodes (and layers) allows MLPs to capture complex functions that other algorithms cannot adequately encode, making them a powerful class of models.

Critically, the ReLU operator, $v = \max\{0, x\}$, can be encoded using linear constraints, as detailed in Appendix A.3. The constraints for an MLP network can be generated recursively starting from the input layer, with a set of ReLU constraints for each node in each internal layer, $l \in \{2, \ldots, L-1\}$. This allows us to embed a trained MLP with an arbitrary number of hidden layers and nodes into an MIO. In a regression setting, the output layer L consists of a single node that is a linear combination of the node values in layer $L-1$, so it can be encoded directly as

$$y = v^L = \beta_0^L + \sum_{j \in N^{L-1}} \beta_j^L v_j^{L-1}.$$

In the binary classification setting, the output layer requires one neuron with a sigmoid activation function, $S(x) = \frac{1}{1+e^{-x}}$. The value is given as

$$v^L = \frac{1}{1 + e^{-(\beta_0^L + \boldsymbol{\beta}^{L\top} \boldsymbol{v}^{L-1})}}$$

with $v^L \in (0,1)$. This function is nonlinear, and thus, cannot be directly embedded into our formulation. However, if τ is our desired probability lower bound, it will be satisfied when $\beta_0^L + \boldsymbol{\beta}^{L\top} \boldsymbol{v}^{L-1} \geq \ln\left(\frac{\tau}{1-\tau}\right)$. Therefore, the neural network's output, binarized with a threshold of τ, is given by

$$y = \begin{cases} 1, & \text{if } \beta_0^L + \boldsymbol{\beta}^{L\top} \boldsymbol{v}^{L-1} \geq \ln\left(\dfrac{\tau}{1-\tau}\right); \\ \\ 0, & \text{otherwise.} \end{cases}$$

For example, at a threshold of $\tau = 0.5$, the predicted value is 1 when $\beta_0^L + \boldsymbol{\beta}^{L\top} \boldsymbol{v}^{L-1} \geq 0$. Here, τ can be chosen according to the minimum necessary probability to predict 1. As for the SVC case, y is binary and the constraint can be embedded as $y \geq 1$. We refer to Appendix A.4 for the case of neural networks trained for multi-class classification.

2.3. Convex hull as trust region

As the optimal solutions of optimization problems are often at the extremes of the feasible region, this can be problematic for the validity of the trained ML model. Generally speaking the accuracy of a predictive model deteriorates for points that are farther away from the data points in $\mathcal{D}$ (Goodfellow et al. 2015). To mitigate this problem, we use a so-called trust region that prevents the predictive model from extrapolating. According to Ebert et al. (2014), when data is enclosed by a boundary of convex shape, the region inside this boundary is known as an interpolation region. This interpolation region is also referred to as the convex hull, and by excluding solutions outside the convex hull, we prevent extrapolation. If $\mathcal{Z} = \{\bar{\boldsymbol{z}}_i\}_{i=1}^N$ is the set of observed input data with $\bar{\boldsymbol{z}}_i = (\bar{\boldsymbol{x}}_i, \bar{\boldsymbol{w}}_i)$, we define the trust region as the convex hull of this set and denote it by $\mathrm{CH}(\mathcal{Z})$. Recall that $\mathrm{CH}(\mathcal{Z})$ is the smallest convex polytope that contains the set of points $\mathcal{Z}$. It is well-known that computing the convex hull is exponential in time and space with respect to the number

Figure 3   Use of the two-step approach to remove low-density regions.

(a) CH($\mathcal{Z}$) with single region.   (b) CH($\mathcal{Z}$) with clustered regions.
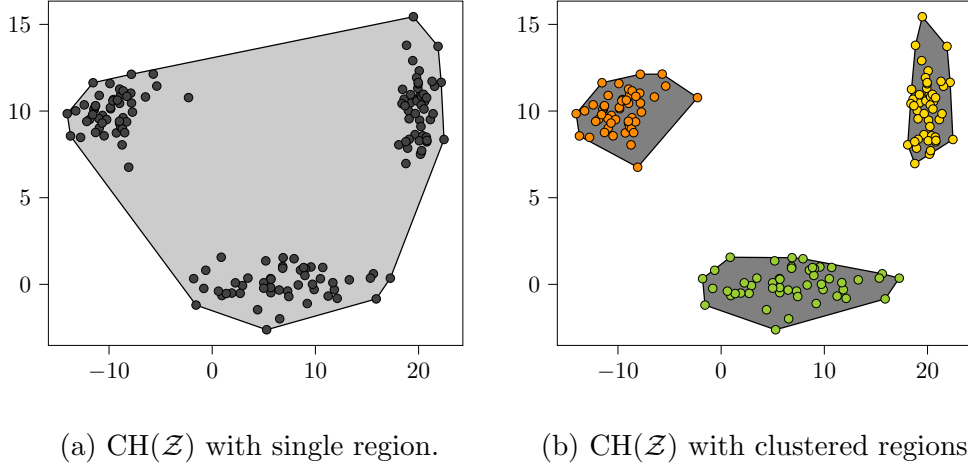
of samples and their dimensionality (Skiena 2008). However, since the convex hull is a polytope, explicit expressions for its facets are not necessary. More precisely, CH($\mathcal{Z}$) is represented as

$$\text{CH}(\mathcal{Z}) = \left\{ z \,\middle|\, \sum_{i \in \mathcal{I}} \lambda_i \bar{z}_i = z, \ \sum_{i \in \mathcal{I}} \lambda_i = 1, \ \boldsymbol{\lambda} \geq 0 \right\}, \tag{2}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^N$, and $\mathcal{I} = \{1, \dots, N\}$ is the index set of samples in $\mathcal{Z}$.

In situations such as the one shown in Figure 3a, CH($\mathcal{Z}$) includes regions with few or no data points (low-density regions). Blindly using CH($\mathcal{Z}$) in this case can be problematic if the solutions are found in the low-density regions. We therefore advocate the use of a two-step approach. First, clustering is used to identify distinct high-density regions, and then the trust region is represented as the union of the convex hulls of the individual clusters (Figure 3b).

We can either solve EM($\boldsymbol{w}$) for each cluster, or embed the union of the $|\mathcal{K}|$ convex hulls into the MIO given by

$$\bigcup_{k \in \mathcal{K}} \text{CH}(\mathcal{Z}_k) = \left\{ z \,\middle|\, \sum_{i \in \mathcal{I}_k} \lambda_i \bar{z}_i = z, \ \sum_{i \in \mathcal{I}_k} \lambda_i = u_k \ \forall k \in \mathcal{K}, \sum_{k \in \mathcal{K}} u_k = 1, \ \boldsymbol{\lambda} \geq 0, \ \boldsymbol{u} \in \{0,1\}^{|\mathcal{K}|} \right\}, \tag{3}$$

where $\mathcal{Z}_k \subseteq \mathcal{Z}$ refers to subset of samples in cluster $k \in \mathcal{K}$ with the index set $\mathcal{I}_k \subseteq \mathcal{I}$. The union of convex hulls requires the binary variables u_k to constrain a feasible solution to be exactly in one of the convex hulls. More precisely, $u_k = 1$ corresponds to the convex hull of the k-th cluster. As we show in Section 3, solving EM($\boldsymbol{w}$) for each cluster may be done in parallel, which has a

positive impact on computation time. We note that both formulations (2) and (3) assume that $\bar{z}$ is continuous. These formulations can be extended to datasets with binary, categorical and ordinal features. In the case of categorical features, extra constraints on the domain and one-hot encoding are required.

In addition to embedding the trust region for predictive models, this approach offers independent value in one-class constraint learning, which is an often studied problem in the literature (Pawlak 2019, Pawlak and Litwiniuk 2021). Here, data is composed of only feasible samples, so the predictive models discussed in Section 2.2 (which require both feasible and infeasible samples) are no longer suitable. A typical example occurs in real-world business processes like machine scheduling. Most of the schedules created by the machine shop supervisor are feasible, even if they may not be optimal. Thus, infeasible schedules are so infrequent that they may not be part of the dataset. We handle this one-class constraint learning task by employing the two-step approach, where we first cluster to identify separate high-density regions, and then use the union of convex hulls to represent the trust region.

Although the convex hull can be represented by linear constraints, the number of variables in $\text{EM}(\boldsymbol{w})$ increases with the increase in the dataset size, which may make the optimization process prohibitive when the number of samples becomes too large. We therefore provide a column selection algorithm that selects a small subset of the samples. This algorithm can be used for $\text{EM}(\boldsymbol{w})$ when all variables are continuous. Figure 4 visually demonstrates the procedure; we begin with an arbitrary sample of the full data, and use column selection to iteratively add samples $\bar{\boldsymbol{z}}_i$ until no improvement can be found. In Appendix B.2, we provide a full description of the approach, as well as a formal lemma which states that in each iteration of column selection, the selected sample from $\mathcal{Z}$ is also a vertex of $\text{CH}(\mathcal{Z})$. In synthetic experiments, we observe that the algorithm scales well with the dataset size. The computation time required by solving the optimization problem with the algorithm is near-constant and minimally affected by the number of samples in the dataset. The experiments in Appendix B.2 show optimization with column selection to be significantly faster than a traditional approach, which makes it an ideal choice when dealing with massive datasets.

Figure 4   Visualization of the column selection algorithm. Known and learned constraints define the infeasible

region. The column selection algorithm starts using only a subset of data points (red filled circles),

$\mathcal{Z}' \subseteq \mathcal{Z}$ to define the trust region. In each iteration a vertex of $\mathrm{CH}(\mathcal{Z})$ is selected (red hollow circle)

and included in $\mathcal{Z}'$ until the optimal solution (star) is within the feasible region, namely the convex

hull of $\mathcal{Z}'$. Note that with column selection we do not need the complete dataset to obtain the optimal

solution, but rather only a subset.
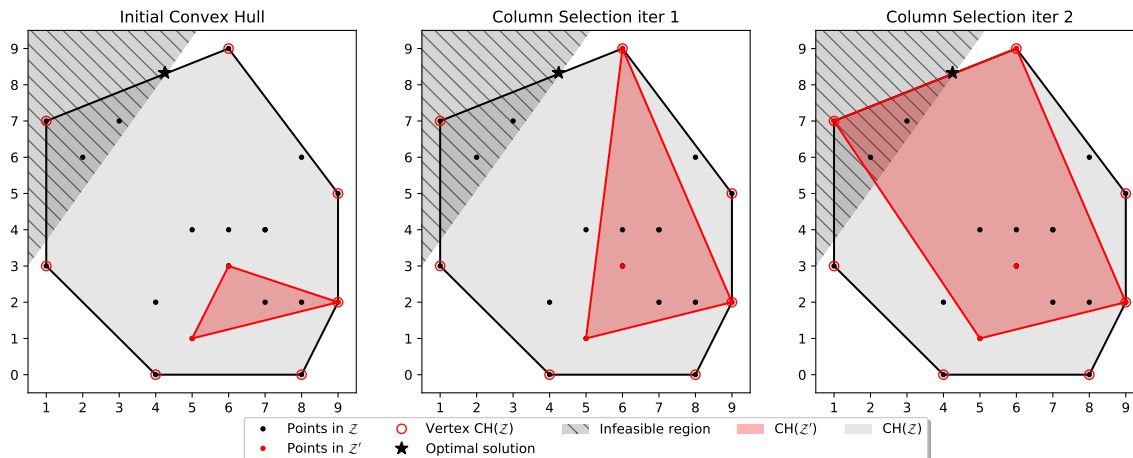
3. Case study: a palatable food basket for the World Food Programme

In this case study, we use a simplified version of the model proposed by Peters et al. (2021),

which seeks to optimize humanitarian food aid. Its extended version aims to provide the World

Food Programme (WFP) with a decision-making tool for long-term recovery operations, which

simultaneously optimizes the food basket to be delivered, the sourcing plan, the delivery plan,

and the transfer modality of a month-long food supply. The model proposed by Peters et al.

(2021) enforces that the food baskets address the nutrient gap and are palatable. To guarantee a

certain level of palatability, the authors use a number of "unwritten rules" that have been defined

in collaboration with nutrition experts. In this case study, we take a step further by inferring

palatability constraints directly from data that reflects local people's opinions. We use the specific

case of Syria for this example. The conceptual model presents an LO structure with only the food

palatability constraint to be learned. Data on palatability is generated through a simulator, but

the procedure would remain unchanged if data were collected in the field, for example through

surveys. The structure of this problem, which is an LO and involves only one learned constraint,

allows the following analyses: (1) the effect of the trust-region on the optimal solution, and (2) the effect of clustering on the computation time and the optimal objective value. Additionally, the use of simulated data provides us with a ground truth to use in evaluating the quality of the prescriptions.

3.1. Conceptual model

The optimization model is a combination of a capacitated, multi-commodity network flow model, and a diet model with constraints for nutrition levels and food basket palatability.

The sets used to define the constraints and the objective function are displayed in Table 1. We have three different sets of nodes, and the set of commodities contains all the foods available for procurement during the food aid operation.

Table 1 Definition of the sets used in the WFP model.

Sets	
$\mathcal{N}_{\mathcal{S}}$	Set of source nodes
$\mathcal{N}_{\mathcal{T}}$	Set of transshipment nodes
$\mathcal{N}_{\mathcal{D}}$	Set of delivery nodes
$\mathcal{K}$	Set of commodities ($k \in \mathcal{K}$)
$\mathcal{L}$	Set of nutrients ($l \in \mathcal{L}$)

The parameters used in the model are displayed in Table 2. The costs used in the objective function concern transportation (p^T) and procurement (p^P). The amount of food to deliver depends on the demand (d) and the number of feeding days ($days$). The nutritional requirements ($nutreq$) and nutritional values ($nutrval$) are detailed in Appendix C. Here, the parameter γ is needed to convert the metric tons used in the supply chain constraints to the grams used in the nutritional constraints. The parameter t is used as a lower bound on the food basket palatability.

The decision variables are shown in Table 3. The flow variables F_{ijk} are defined as the metric tons of a commodity k transported from node i to j. The variable x_k represents the average daily ration per beneficiary for commodity k. The variable y refers to the palatability of the food basket.

Parameters	
γ	Conversion rate from metric tons (mt) to grams (g)
d_i	Number of beneficiaries at delivery point $i \in \mathcal{N}_\mathcal{D}$
$days$	Number of feeding days
$nutreq_l$	Nutritional requirement for nutrient $l \in \mathcal{L}$ (grams/person/day)
$nutval_{kl}$	Nutritional value for nutrient $l \in \mathcal{L}$ per gram of commodity $k \in \mathcal{K}$
p_{ik}^P	Procurement cost (in \$ / mt) of commodity k from source $i \in \mathcal{N}_\mathcal{S}$
p_{ijk}^T	Transportation cost (in \$ / mt) of commodity k from node $i \in \mathcal{N}_\mathcal{S} \cup \mathcal{N}_\mathcal{T}$ to node $j \in \mathcal{N}_\mathcal{T} \cup \mathcal{N}_\mathcal{D}$
t	Palatability lower bound

Table 3   Definition of the variables used in the WFP model.

Variables	
F_{ijk}	Metric tons of commodity $k \in \mathcal{K}$ transported between node i and node j
x_k	Grams of commodity $k \in \mathcal{K}$ in the food basket
y	Food basket palatability

The full model formulation is as follows:

$$\min_{\boldsymbol{x}, y, \boldsymbol{F}} \sum_{i \in \mathcal{N}_\mathcal{S}} \sum_{j \in \mathcal{N}_\mathcal{T} \cup \mathcal{N}_\mathcal{D}} \sum_{k \in \mathcal{K}} p_{ik}^P F_{ijk} + \sum_{i \in \mathcal{N}_\mathcal{S} \cup \mathcal{N}_\mathcal{T}} \sum_{j \in \mathcal{N}_\mathcal{T} \cup \mathcal{N}_\mathcal{D}} \sum_{k \in \mathcal{K}} p_{ijk}^T F_{ijk} \tag{4a}$$

$$\text{s.t.} \sum_{j \in \mathcal{N}_\mathcal{T}} F_{ijk} = \sum_{j \in \mathcal{N}_\mathcal{T}} F_{jik}, \quad i \in \mathcal{N}_\mathcal{T}, \ k \in \mathcal{K}, \tag{4b}$$

$$\sum_{j \in \mathcal{N}_\mathcal{S} \cup \mathcal{N}_\mathcal{T}} \gamma F_{jik} = d_i x_k days, \quad i \in \mathcal{N}_\mathcal{D}, \ k \in \mathcal{K}, \tag{4c}$$

$$\sum_{k \in \mathcal{K}} Nutval_{kl} x_k \geq Nutreq_l, \quad l \in \mathcal{L}, \tag{4d}$$

$$x_{salt} = 5, \tag{4e}$$

$$x_{sugar} = 20, \tag{4f}$$

$$y \geq t, \tag{4g}$$

$$y = \hat{h}(\boldsymbol{x}), \tag{4h}$$

$$F_{ijk}, x_k \geq 0, \quad i, j \in \mathcal{N}, \ k \in \mathcal{K}. \tag{4i}$$

The objective function consists of two components, procurement costs and transportation costs. Constraints (4b) are used to balance the network flow, namely to ensure that the inflow and the outflow of a commodity are equal for each transhipment node. Constraints (4c) state that flow into a delivery node has to be equal to its demand, which is defined by the number of beneficiaries

times the daily ration for commodity k times the feeding days. Constraints (4d) guarantee an optimal solution that meets the nutrition requirements. Constraints (4e) and (4f) force the amount of salt and sugar to be 5 grams and 20 grams respectively. Constraint (4g) requires the food basket palatability (y), defined by means of a predictive model (4h), to be greater than a threshold (t). Lastly, non-negativity constraints (4i) are added for all commodity flows and commodity rations.

3.2. Dataset and predictive models

To evaluate the ability of our framework to learn and implement the palatability constraints, we use a simulator to generate diets with varying palatabilities. Each sample is defined by 25 features representing the amount (in grams) of all commodities that make up the food basket. We then use a ground truth function to assign each food basket a palatability between 0 and 1, where 1 corresponds to a perfectly palatable basket, and 0 to an inedible basket. This function is based on suggestions provided by WFP experts. The data is then balanced to ensure that a wide variety of palatability scores are represented in the dataset. The final data used to learn the palatability constraint consists of 121,589 samples. Two examples of daily food baskets and their respective palatability scores are shown in Table 4. In this case study, we use a palatability lower bound (t) of 0.5 for our learned constraint.

The next step of the framework involves training and choosing the predictive model that best approximates the unknown constraint. The predictive models used to learn the palatability constraints are those discussed in Section 2, namely LR, SVM, CART, RF, GBM with decision trees as base-learners, and MLP with ReLU activation function.

3.3. Optimization results

The experiments are executed using `OptiCL` jointly with Gurobi v9.1 (Gurobi Optimization, LLC 2021) as the optimization solver. Table 5 reports the performances of the predictive models evaluated both for the validation set and for the prescriptions after being embedded into the optimization model. The table also compares the performance of the optimization with and without the trust

Table 4   Two examples of daily food baskets.

Commodity	Basket 1 Amount (g)	Basket 2 Amount (g)
DSM	31.9	33.9
Chickpeas	–	75.7
Lentils	41	–
Maize meal	48.9	–
Meat	–	17.2
Oil	22	28.6
Salt	5	5
Sugar	20	20
Wheat	384.2	131.2
Wheat flour	–	261.3
WSB	67.3	59.8
Palatability Score	0.436	0.741

DSM=dried skim milk, WSB=wheat soya blend.

region. The column "Validation MSE" gives the Mean Squared Error (MSE) of each model obtained in cross-validation during model selection. While all scores in this column are desirably low, the MLP model significantly achieves the lowest error during this validation phase. The column "MSE" gives the MSE of the predictive models once embedded into the optimization problem to evaluate how well the predictions for the optimal solutions match their true palatabilities (computed using the simulator). It is found using 100 optimal solutions of the optimization model generated with different cost vectors. The MLP model exhibits the best performance (0.055) in this context, showing its ability to model the palatability constraint better than all other methods.

Table 5   Predictive models performances for the validation set ("Validation MSE"), and for the prescriptions after being embedded into the optimization model with ("MSE-TR") and without the trust region ("MSE"). The last two columns show the average computation time in seconds and its standard deviation (SD) required to solve the optimization model with ("Time-TR") and without the trust region ("Time").

Model	Validation MSE	MSE	MSE-TR	Time (SD)	Time-TR (SD)
LR	0.046	0.256	0.042	0.003 (0.0008)	1.813 (0.204)
SVM	0.019	0.226	0.027	0.003 (0.0006)	1.786 (0.208)
CART	0.014	0.273	0.059	0.012 (0.0030)	7.495 (5.869)
RF	0.018	0.252	0.025	0.248 (0.1050)	30.128 (13.917)
GBM	0.006	0.250	0.017	0.513 (0.4562)	60.032 (41.685)
MLP	0.001	0.055	0.001	14.905 (41.764)	28.405 (23.339)

Runtimes reported using an Intel i7-8665U 1.9 GHz CPU, 16 GB RAM (Windows 10 environment).

Benefit of trust region. Table 5 shows that when the trust region is used ("MSE-TR"), the MSEs obtained by all models are now much closer to the results from the validation phase. This shows

the benefit of using the trust region as discussed in Section 2.3 to prevent extrapolation. With the trust region included, the MLP model also exhibits the lowest MSE (0.001). The improved performance seen with the inclusion of the trust region does come at the expense of computation speed. The column "Time-TR" shows the average computation time in seconds and its standard deviation (SD) with trust region constraints included. In all cases, the computation time has clearly increased when compared against the computation time required without the trust region (column "Time"). This is however acceptable, as significantly more accurate results are obtained with the trust region.

Benefit of clustering. The large dataset used in this case study makes the use of the trust region expensive in terms of time required to solve the final optimization model. While the column selection algorithm described in Section 2.3 is ideal for significantly reducing the computation time, optimization models that require binary variables, either for embedding an ML model or to represent decision variables, cannot use the column selection algorithm. However, in this more general MIO case, it is possible to divide the dataset into clusters and solve in parallel an MIO for each cluster. By using parallelization, the total solution time can be expected to be equal to the longest time required to solve any single cluster's MIO. Contrary to column selection, the use of clusters can result in sub-optimal solutions; the trust region gets smaller with more clusters and prevents the model from finding solutions that are convex combinations of members of different clusters. However, as described in Section 2.3, optimal solutions that lie between clusters may in fact reside in low-density areas of the feature space that should not be included in the trust region. In this sense, the loss in optimality might actually coincide with more trustable solutions.

Figure 5 shows the effect of clusters in solving the model (4a-4i) with GBM as the predictive model used to learn the palatability constraint. K-means is used to partition the dataset into K clusters, and the reported values are averaged over 100 iterations. In the left graph, we report the maximum runtime distribution across clusters needed to solve the different MIOs in parallel. In the right graph, we have the distributions of optimality gap, *i.e.*, the relative difference between

the optimal solution obtained with clusters compared to the solution obtained with no clustering. In this case study, the use of clusters significantly decreases the runtime (89.2% speed up with $K = 50$) while still obtaining near-optimal solutions (less then 0.25% average gap with $K = 50$). We observe that the trends are not necessarily monotonic in K. It is possible that a certain choice of K may lead to a suboptimal solution, whereas a larger value of K may preserve the optimal solution as the convex combination of points within a single cluster.

Figure 5   Effect of the number of clusters (K) on the computation time and the optimality gap across clusters, with bootstrapped 95% confidence intervals.
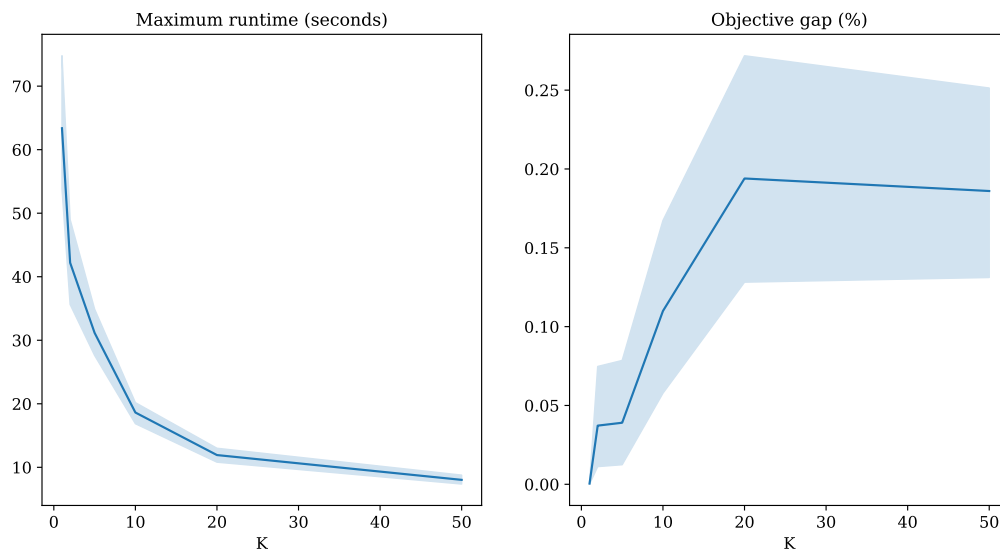
4. Case study: chemotherapy regimen design

In this case study, we extend the work of Bertsimas et al. (2016) in the design of chemotherapy regimens for advanced gastric cancer. Late stage gastric cancer has a poor prognosis with limited treatment options (Yang et al. 2011). This has motivated significant research interest and clinical trials (National Cancer Institute 2021). In Bertsimas et al. (2016), the authors pose the question of algorithmically identifying promising chemotherapy regimens for new clinical trials based on existing trial results. They construct a database of clinical trial treatment arms which includes cohort and study characteristics, the prescribed chemotherapy regimen, and various outcomes.

Given a new study cohort and study characteristics, they optimize a chemotherapy regimen to maximize the cohort's survival subject to a constraint on overall toxicity. The original work uses linear regression models to predict survival and toxicity, and it constrains a single toxicity measure. In this work we leverage a richer class of ML methods and more granular outcome measures. This offers benefits through higher performing predictive models and more clinically-relevant constraints.

Chemotherapy regimens are particularly challenging to optimize, since they involve multiple drugs given at potentially varying dosages, and they present risks for multiple adverse events that must be managed. This example highlights the generalizability of our framework to complex domains with multiple decisions and learned functions. The treatment variables in this problem consist of both binary and continuous elements, which are easily incorporated through our use of MIO. We have several learned constraints which must be simultaneously satisfied, and we also learn the objective function directly as a predictive model.

4.1. Conceptual model

The use of clinical trial data forces us to consider each cohort as an observation, rather than an individual, since only aggregate measures are available. Thus, our model optimizes a cohort's treatment. The contextual variables ($\boldsymbol{w}$) consist of various cohort and study summary variables. The inclusion of fixed, *i.e.*, non-optimization, features allows us to account for differences in baseline health status and risk across study cohorts. These features are included in the predictive models but then are fixed in the optimization model to reflect the group for whom we are generating a prescription. We assume that there are no unobserved confounding variables in this prescriptive setting.

The treatment variables ($\boldsymbol{x}$) encode a chemotherapy regimen. A regimen is defined by a set of drugs, each with an administration schedule of potentially varied dosages throughout a chemotherapy cycle. We characterize a regimen by drug indicators and each drug's average daily dose and maximum instantaneous dose in the cycle:

$$\boldsymbol{x}_b^d = \mathbb{I}(\text{drug } d \text{ is administered}),$$

$$\boldsymbol{x}_a^d = \text{average daily dose of drug } d,$$

$$\boldsymbol{x}_i^d = \text{maximum instantaneous dose of drug } d.$$

This allows us to differentiate between low-intensity, high-frequency and high-intensity, low-frequency dosing strategies. The outcomes of interest ($\boldsymbol{y}$) consist of overall survival, to be included as the objective (y_{OS}), and various toxicities, to be included as constraints ($y_i, \ i \in \mathcal{Y}_C$).

To determine the optimal chemotherapy regimen $\boldsymbol{x}$ for a new study cohort with characteristics $\boldsymbol{w}$, we formulate the following MIO:

$$
\begin{aligned}
\min_{\boldsymbol{x},\boldsymbol{y}} \ & y_{OS} \\
\text{s.t.} \ & y_i \leq \tau_i, & i \in \mathcal{Y}_C, \\
& y_i = \hat{h}_i(\boldsymbol{x},\boldsymbol{w}), & i \in \mathcal{Y}_C, \\
& y_{OS} = \hat{h}_{OS}(\boldsymbol{x},\boldsymbol{w}), \\
& \sum_d \boldsymbol{x}_b^d \leq 3, \\
& \boldsymbol{x} \in \mathcal{X}(\boldsymbol{w}).
\end{aligned}
$$

In this case study, we learn the full objective. However, this model could easily incorporate deterministic components to optimize as additional weighted terms in the objective. We include one domain-driven constraint, enforcing a maximum regimen combination of three drugs.

The trust region, $\mathcal{X}(\boldsymbol{w})$, plays two crucial roles in the formulation. First, it ensures that the predictive models are applied within their valid bounds and not inappropriately extrapolated. It also naturally enforces a notion of "clinically reasonable" treatments. It prevents drugs from being prescribed at doses outside of previously observed bounds, and it requires that the drug combination must have been previously seen (although potentially in different doses). It is nontrivial to explicitly characterize what constitutes a realistic treatment, and the convex hull provides a data-driven solution that integrates directly into the model framework.

4.2. Dataset

Our data consists of 495 clinical trial arms from 1979-2012 (Bertsimas et al. 2016). We consider nine contextual variables, including the average patient age and breakdown of primary cancer site. We include several "dose-limiting toxicities" (DLTs) for our constraint set: Grade 3/4 constitutional toxicity, gastrointestinal toxicity, and infection, as well as Grade 4 blood toxicity. As the name suggests, these are chemotherapy side effects that are severe enough to affect the course of treatment. There are 28 unique drugs that appear in multiple arms of the training set, yielding 84 decision variables.

We apply a temporal split, training the predictive models on trial arms through 2008 and generating prescriptions for the trial arms in 2009-2012. The final training set consists of 320 observations, and the final testing set consists of 96 observations. The full feature set, inclusion criteria, and data processing details are included in Appendix D.1.

To define the trust region, we take the convex hull of the treatment variables ($\boldsymbol{x}$) on the training set. This aligns with the temporal split setting, in which we are generating prescriptions going forward based on an existing set of past treatment decisions. In general it is preferable to define the convex hull with respect to both $\boldsymbol{x}$ and $\boldsymbol{w}$ as discussed in Appendix B.1, but this does not apply well with a temporal split. Our data includes the study year as a feature to incorporate temporal effects, and so our test set observations will definitionally fall outside of the convex hull defined by the observed $(\boldsymbol{x}, \boldsymbol{w})$ in our training set.

4.3. Predictive models

Several ML models are trained for each outcome of interest using cross-validation for parameter tuning, and the best model is selected based on the validation criterion. We employ function learning for all toxicities, directly predicting the toxicity incidence and applying an upper bound threshold within the optimization model.

Based on the model selection procedure, overall DLT, gastrointestinal toxicity, and overall survival are predicted using GBM models. Blood toxicity and infection are predicted using linear

models, and constitutional toxicity is predicted with a RF model. This demonstrates the advantage of learning with multiple model classes; no single method dominates in predictive performance. A complete comparison of the considered models is included in Appendix D.2.

4.4. Evaluation framework

We generate prescriptions using the optimization model outlined in Section 4.1, with the embedded model choices specified in Section 4.3. In order to evaluate the quality of our prescriptions, we must estimate the outcomes under various treatment alternatives. This evaluation task is notoriously challenging due to the lack of counterfactuals. In particular, we only know the true outcomes for observed cohort-treatment pairs and do not have information on potential unobserved combinations. We propose an evaluation scheme that leverages a "ground truth" ensemble (GT ensemble). We train several ML models using all data from the study. These models are not embedded in an MIO model, so we are able to consider a broader set of methods in the ensemble. We then predict each outcome by averaging across all models in the ensemble. This approach allows us to capture the maximal knowledge scenario. Furthermore, such a "consensus" approach of combining ML models has been shown to improve predictive performance and is more robust to individual model error (Bertsimas et al. 2021). The full details of the ensemble models and their predictive performances are included in Appendix D.3.

4.5. Optimization results

We evaluate our model in multiple ways. We first consider the performance of our prescriptions against observed (given) treatments. We then explore the impact of learning multiple sub-constraints rather than a single aggregate toxicity constraint. All optimization models have the following shared parameters: toxicity upper bound of 0.6 quantile (as observed in training data) and maximum violation of 25% for RF models. We report results for all test set observations with a feasible solution.

Table 6 reports the predicted outcomes under two constraint approaches: (1) constraining each toxicity separately ("All Constraints"), and (2) constraining a single aggregate toxicity measure

("DLT Only"). For each cohort in the test set, we generate predictions for all outcomes of interest under both prescription schemes and compute the relative change of our prescribed outcome from the given outcome predictions.

Benefit of prescriptive scheme. We begin by evaluating our proposed prescriptive scheme ("All Constraints") against the observed actual treatments. For example, under the GT ensemble scheme, 84.7% of cohorts satisfied the overall DLT constraint under the given treatment, compared to 94.1% under the proposed treatment. This yields an improvement of 11.10%. We obtain a significant improvement in survival (11.40%) while also improving toxicity limit satisfaction across all individual toxicities. Using the GT ensemble, we see toxicity satisfaction improvements between 1.3%-25.0%.

Table 6 Comparison of outcomes under given treatment regimen, regimen prescribed when only constraining the aggregate toxicity, and regimen prescribed under our full model.

		All Constraints		DLT Only	
	Given (SD)	Prescribed (SD)	% Change	Prescribed (SD)	% Change
Any DLT	0.847 (0.362)	0.941 (0.237)	11.10%	0.906 (0.294)	6.90%
Blood	0.812 (0.393)	0.824 (0.383)	1.40%	0.706 (0.458)	-13.00%
Constitutional	0.953 (0.213)	1.000 (0.000)	4.90%	1.000 (0.000)	4.90%
Infection	0.882 (0.324)	0.894 (0.310)	1.30%	0.800 (0.402)	-9.30%
Gastrointestinal	0.800 (0.402)	1.000 (0.000)	25.00%	1.000 (0.000)	25.00%
Overall Survival	10.855 (1.939)	12.092 (1.470)	11.40%	12.468 (1.430)	14.90%

We report the mean and standard deviation (SD) of constraint satisfaction (binary indicator) and overall survival (months) across the test set. The relative change is reported against the given treatment.

Benefit of multiple constraints. Table 6 also illustrates the value of enforcing constraints on each individual toxicity rather than as a single measure. When only constraining the aggregate toxicity measure ("DLT Only"), the resultant prescriptions actually have lower constraint satisfaction for blood toxicity and infection than the baseline given regimens. By constraining multiple measures, we are able to improve across all individual toxicities. The fully constrained model actually improves the overall DLT measure satisfaction, suggesting that the inclusion of these "sub-constraints" also makes the aggregate constraint more robust. This improvement does come at the expense of slightly lower survival between the "All" and "DLT Only" models (-0.38 months) but we note that

incurring the individual toxicities that are violated in the "DLT Only" model would likely make the treatment unviable.

5. Discussion

Our experimental results illustrate the benefits of our constraint learning framework in data-driven decision making in two problem settings: food basket recommendations for the WFP and chemotherapy regimens for advanced gastric cancer. The quantitative results show the improvement in predictive performance when incorporating the trust region and learning from multiple candidate model classes. We also see a benefit in incorporating multiple learned constraints over a single aggregate measure. Our framework scales to large problem sizes, enabled by efficient formulations and tailored approaches to specific problem structures. Our approach for efficiently learning the trust region also has broad applicability in one-class constraint learning.

We recognize several opportunities to further extend this framework. Our work naturally relates to the causal inference literature and individual treatment effect estimation (Athey and Imbens 2016, Shalit et al. 2017). These methods do not directly translate to our problem setting; existing work generally assumes highly structured treatment alternatives (*e.g.*, binary treatment vs. control) or a single continuous treatment (*e.g.*, dosing), whereas we allow more general decision structures. In future work, we are interested in incorporating ideas from causal inference to relax the assumption of unobserved confounders.

Additionally, our framework is dependent on the quality of the underlying predictive models. We constrain and optimize point predictions from our embedded models. This can be problematic in the case of model misspecification, a known shortcoming of "predict-then-optimize" methods (Elmachtoub and Grigas 2021). We mitigate this concern in two ways. First, our model selection procedure allows us to obtain higher quality predictive models by capturing several possible functional relationships. Second, our inclusion of the violation limit in constrained ensemble models allows us to directly parametrize how conservative our predictions are and our robustness to the predictions of individual learners. This concept could be extended to a more general ensemble, in which we

embed multiple separate models for an outcome of interest and enforce the constraint over some subset of these models. In future work, there is an opportunity to incorporate ideas from robust optimization to directly account for prediction uncertainty in the constraints.

In this work, we present a unified framework for optimization with learned constraints that leverages both ML and MIO for data-driven decision making. Our work flexibly learns problem constraints and objectives with supervised learning, and incorporates them into a larger optimization problem of interest. We also learn the trust region, providing more credible recommendations and improving predictive performance, and accomplish this efficiently using column generation and unsupervised learning. The generality of our method allows us to tackle quite complex decision settings, such as chemotherapy optimization, but also includes tailored approaches for more efficiently solving specific problem types. Finally, we implement this as a Python software package (`OptiCL`) to enable practitioner use. We envision that `OptiCL`'s methodology will be added to state-of-the-art optimization modeling software packages.

Acknowledgments

References

Amos B, Xu L, Kolter JZ (2016) Input convex neural networks URL `http://ariv.org/abs/1609.07152`.

Anderson R, Huchette J, Ma W, Tjandraatmadja C, Vielma JP (2020) Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming* 183(1-2):3–39, ISSN 14364646, URL `http://dx.doi.org/10.1007/s10107-020-01474-5`.

Athey S, Imbens G (2016) Recursive partitioning for heterogeneous causal effects. *Proceedings of the National Academy of Sciences* 113(27):7353–7360, ISSN 0027-8424, URL `http://dx.doi.org/10.1073/pnas.1510489113`.

Bengio Y, Lodi A, Prouvost A (2021) Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research* 290(2):405–421, ISSN 0377-2217, URL `http://dx.doi.org/https://doi.org/10.1016/j.ejor.2020.07.063`.

Bergman D, Huang T, Brooks P, Lodi A, Raghunathan AU (2019) JANOS: An integrated predictive and prescriptive modeling framework. URL `http://arxiv.org/abs/1911.09461`.

Bertsimas D, Borenstein A, Mingardi L, Nohadani O, Orfanoudaki A, Stellato B, Wiberg H, Sarin P, Varelmann DJ, Estrada V, Macaya C, Gil IJ (2021) Personalized prescription of ACEI/ARBs for hypertensive COVID-19 patients. *Health Care Management Science* 24(2):339–355, ISSN 15729389, URL `http://dx.doi.org/10.1007/s10729-021-09545-5`.

Bertsimas D, Dunn J (2017) Optimal classification trees. *Machine Learning* 106(7):1039–1082, ISSN 15730565, URL `http://dx.doi.org/10.1007/s10994-017-5633-9`.

Bertsimas D, Kallus N (2020) From Predictive to Prescriptive Analytics. *Management Science* 66(3):1025–1044, ISSN 0025-1909, URL `http://dx.doi.org/10.1287/mnsc.2018.3253`.

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, ISSN 15265501, URL `http://dx.doi.org/10.1287/mnsc.2015.2363`.

Bertsimas, D and Dunn, J (2018) *Machine Learning under a Modern Optimization Lens* (Belmont: Dynamic Ideas).

Biggs M, Hariss R, Perakis G (2021) Optimizing objective functions determined from random forests. *SSRN Electronic Journal* 1–46, ISSN 1556-5068, URL `http://dx.doi.org/10.2139/ssrn.2986630`.

Bonfietti A, Lombardi M, Milano M (2015) Embedding decision trees and random forests in constraint programming. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9075:74–90, ISSN 16113349, URL `http://dx.doi.org/10.1007/978-3-319-18008-3_6`.

Breiman L (2001) Random forests. *Machine Learning* 45(1):5–32, ISSN 08856125, URL `http://dx.doi.org/10.1023/A:1010933404324`.

Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) *Classification and regression trees* (Routledge), ISBN 978-0412048418, URL http://dx.doi.org/10.1201/9781315139470.

Cancer Therapy Evaluation Program (2006) Common terminology criteria for adverse events v3.0. URL https://ctep.cancer.gov/protocoldevelopment/electronic_applications/docs/ctcaev3.pdf.

Chen Y, Shi Y, Zhang B (2020) Input convex neural networks for optimal voltage regulation. URL http://arxiv.org/abs/2002.08684.

Cortes C, Vapnik V (1995) Support-vector networks. *Machine Learning* 20(3):273–297.

Cremer JL, Konstantelos I, Tindemans SH, Strbac G (2019) Data-driven power system operation: Exploring the balance between cost and risk. *IEEE Transactions on Power Systems* 34(1):791–801, ISSN 08858950, URL http://dx.doi.org/10.1109/TPWRS.2018.2867209.

Drucker H, Surges CJ, Kaufman L, Smola A, Vapnik V (1997) Support vector regression machines. *Advances in Neural Information Processing Systems* 1:155–161, ISSN 10495258.

Ebert T, Belz J, Nelles O (2014) Interpolation and extrapolation: Comparison of definitions and survey of algorithms for convex and concave hulls. *2014 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, 310–314, URL http://dx.doi.org/10.1109/CIDM.2014.7008683.

Elmachtoub AN, Grigas P (2021) Smart "Predict, then Optimize". *Management Science* 1–46, ISSN 0025-1909, URL http://dx.doi.org/10.1287/mnsc.2020.3922.

Fajemisin A, Maragno D, den Hertog D (2021) Optimization with constraint learning: A framework and survey. URL https://arxiv.org/abs/2110.02121.

Goodfellow IJ, Shlens J, Szegedy C (2015) Explaining and harnessing adversarial examples. *CoRR* abs/1412.6572.

Grimstad B, Andersson H (2019) ReLU networks as surrogate models in mixed-integer linear programs. *Computers and Chemical Engineering* 131:106580, ISSN 00981354, URL http://dx.doi.org/10.1016/j.compchemeng.2019.106580.

Gurobi Optimization, LLC (2021) Gurobi Optimizer Reference Manual. URL https://www.gurobi.com.

Gutierrez-Martinez VJ, Cañizares CA, Fuerte-Esquivel CR, Pizano-Martinez A, Gu X (2011) Neural-network security-boundary constrained optimal power flow. *IEEE Transactions on Power Systems* 26(1):63–72, ISSN 08858950, URL http://dx.doi.org/10.1109/TPWRS.2010.2050344.

Halilbasic L, Thams F, Venzke A, Chatzivasileiadis S, Pinson P (2018) Data-driven security-constrained AC-OPF for operations and markets. *20th Power Systems Computation Conference, PSCC 2018* URL http://dx.doi.org/10.23919/PSCC.2018.8442786.

Kleijnen JP (2015) Design and analysis of simulation experiments. *International Workshop on Simulation*, 3–22 (Springer).

Kudła P, Pawlak TP (2018) One-class synthesis of constraints for Mixed-Integer Linear Programming with C4.5 decision trees. *Applied Soft Computing Journal* 68:1–12, ISSN 15684946, URL http://dx.doi.org/10.1016/j.asoc.2018.03.025.

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

Mišić VV (2020) Optimization of tree ensembles. *Operations Research* 68(5):1605–1624, ISSN 15265463, URL http://dx.doi.org/10.1287/opre.2019.1928.

MOSEK (2019) *MOSEK Optimizer API for Python 9.3.7*. URL https://docs.mosek.com/latest/pythonapi/index.html.

National Cancer Institute (2021) Treatment clinical trials for gastric (stomach) cancer. URL https://www.cancer.gov/about-cancer/treatment/clinical-trials/disease/stomach-cancer/treatment.

Pawlak TP (2019) Synthesis of mathematical programming models with one-class evolutionary strategies. *Swarm and Evolutionary Computation* 44:335–348, ISSN 2210-6502, URL http://dx.doi.org/https://doi.org/10.1016/j.swevo.2018.04.007.

Pawlak TP, Krawiec K (2019) Synthesis of constraints for mathematical programming with one-class genetic programming. *IEEE Transactions on Evolutionary Computation* 23(1):117–129, URL http://dx.doi.org/10.1109/TEVC.2018.2835565.

Pawlak TP, Litwiniuk B (2021) Ellipsoidal one-class constraint acquisition for quadratically constrained programming. *European Journal of Operational Research* 293(1):36–49, ISSN 03772217, URL `http://dx.doi.org/10.1016/j.ejor.2020.12.018`.

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

Schweidtmann AM, Mitsos A (2019) Deterministic global optimization with artificial neural networks embedded. *Journal of Optimization Theory and Applications* 180(3):925–948, ISSN 15732878, URL `http://dx.doi.org/10.1007/s10957-018-1396-0`.

Shalit U, Johansson FD, Sontag D (2017) Estimating individual treatment effect: generalization bounds and algorithms. *International Conference on Machine Learning*, 3076–3085 (PMLR).

Skiena SS (2008) *The Algorithm Design Manual* (Springer Publishing Company, Incorporated), 2nd edition.

Spyros C (2020) From Decision Trees and Neural Networks to MILP: power system optimization considering dynamic stability constraints. *2020 European Control Conference (ECC)*, 594–594 (IEEE), ISBN 978-3-90714-402-2, URL `http://dx.doi.org/10.23919/ECC51009.2020.9143834`.

Sroka D, Pawlak TP (2018) One-class constraint acquisition with local search. *GECCO 2018 - Proceedings of the 2018 Genetic and Evolutionary Computation Conference* 363–370, URL `http://dx.doi.org/10.1145/3205455.3205480`.

Thams F, Halilbaši L, Pinson P, Chatzivasileiadis S, Eriksson R (2017) Data-driven security-constrained OPF. *Proc. 10th Bulk Power Syst. Dyn. Control Symp.*, 1–10, URL `http://irep2017.inesctec.pt/conference-papers/conference-papers/paper65r7z1aplj.pdf`.

UNHCR, UNICEF, WFP, WHO (2002) Food and nutrition needs in emergencies. URL `https://www.who.int/nutrition/publications/emergencies/a83743/en/`.

Venzke A, Viola DT, Mermet-Guyennet J, Misyris GS, Chatzivasileiadis S (2020) Neural networks for encoding dynamic security-constrained optimal power flow to mixed-integer linear programs URL `http://arxiv.org/abs/2003.07939`.

Verwer S, Zhang Y, Ye QC (2017) Auction optimization using regression trees and linear models as integer

programs. *Artificial Intelligence* 244:368–395, ISSN 00043702, URL `http://dx.doi.org/10.1016/j.`

`artint.2015.05.004`.

Wolfe P (1961) A duality theorem for non-linear programming. *Quarterly of Applied Mathematics*

19(3):239–244.

Yang D, Hendifar A, Lenz C, Togawa K, Lenz F, Lurje G, Pohl A, Winder T, Ning Y, Groshen S, Lenz

HJ (2011) Survival of metastatic gastric cancer: Significance of age, sex and race/ethnicity. *Journal of*

Gastrointestinal Oncology 2(2):77–84, ISSN 2219-679X, URL `http://dx.doi.org/10.3978/j.issn.`

`2078-6891.2010.025`.

Appendix A: Methodology

A.1. Embedding a decision tree

Consider the leaves in Figure 2. An observation will be assigned to the leftmost leaf (node 3) if $A_1^\top \boldsymbol{x} \leq b_1$ and $A_2^\top \boldsymbol{x} \leq b_2$. An observation would be assigned to node 4 if $A_1^\top \boldsymbol{x} \leq b_1$ and $A_2^\top \boldsymbol{x} > b_2$, or equivalently, $-A_2^\top \boldsymbol{x} < -b_2$. Furthermore, we can remove the strict inequalities using a sufficiently small ϵ parameter, so that $-A_2^\top \boldsymbol{x} \leq -b_2 - \epsilon$. We can then encode the leaf assignment of observation $\boldsymbol{x}$ through the following constraints:

$$A_1^\top \boldsymbol{x} - M(1 - l_3) \leq b_1, \tag{5a}$$

$$A_2^\top \boldsymbol{x} - M(1 - l_3) \leq b_2, \tag{5b}$$

$$A_1^\top \boldsymbol{x} - M(1 - l_4) \leq b_1, \tag{5c}$$

$$-A_2^\top \boldsymbol{x} - M(1 - l_4) \leq -b_2 - \epsilon, \tag{5d}$$

$$-A_1^\top \boldsymbol{x} - M(1 - l_6) \leq -b_1 - \epsilon, \tag{5e}$$

$$A_5^\top \boldsymbol{x} - M(1 - l_6) \leq b_5, \tag{5f}$$

$$-A_1^\top \boldsymbol{x} - M(1 - l_7) \leq -b_1 - \epsilon, \tag{5g}$$

$$-A_5^\top \boldsymbol{x} - M(1 - l_7) \leq -b_5 - \epsilon, \tag{5h}$$

$$l_3 + l_4 + l_6 + l_7 = 1, \tag{5i}$$

$$y - (p_3 l_3 + p_4 l_4 + p_6 l_6 + p_7 l_7) = 0, \tag{5j}$$

where l_3, l_4, l_6, l_7 are binary variables associated with the corresponding leaves. For a given $\boldsymbol{x}$, if $A_1^\top \boldsymbol{x} \leq b_1$, Constraints (5e) and (5h) will force l_6 and l_7 to zero, respectively. If $A_2^\top \boldsymbol{x} \leq b_2$, constraint (5d) will force l_4 to 0. The assignment constraint (5i) will then force $l_3 = 1$, assigning the observation to leaf 3 as desired. Finally, constraint (5j) sets y to the prediction of the assigned leaf (p_3). We can then constrain the value of y using our desired upper bound of τ (or lower bound, without loss of generality).

More generally, consider a decision tree $\hat{h}(\boldsymbol{x}, \boldsymbol{w})$ with a set of leaf nodes $\mathcal{L}$ each described by a binary variable l_i and a prediction score p_i. Splits take the form $(A_x)^\top \boldsymbol{x} + (A_w)^\top \boldsymbol{w} \leq b$, where A_x gives the coefficients for the optimization variables $\boldsymbol{x}$ and A_w gives the coefficients for the non-optimization (fixed) variables $\boldsymbol{w}$. Let $\mathcal{S}^l$ be the set of nodes that define the splits that observations in leaf i must obey. Without loss of generality, we can write these all as $(\bar{A}_x)_j^\top \boldsymbol{x} + (\bar{A}_w)_j^\top \boldsymbol{w} - M(1 - l_i) \leq \bar{b}_j$, where $\bar{A}$ is A if leaf i follows the left split of j and $-A$ otherwise. Similarly, $\bar{b}$ equals b if the leaf falls to the left split, and $-b - \epsilon$ otherwise, as established above. This decision tree can then be embedded through the following constraints:

$$(\bar{A}_x)_j^\top \boldsymbol{x} + (\bar{A}_w)_j^\top \boldsymbol{w} - M(1 - l_i) \leq \bar{b}_j, \ i \in \mathcal{L}, j \in \mathcal{S}^l, \tag{6a}$$

$$\sum_{i \in \mathcal{L}} l_i = 1, \tag{6b}$$

$$y - \sum_{i \in \mathcal{L}} p_i l_i = 0. \tag{6c}$$

Here, M can be selected for each split by considering the maximum difference between $(\bar{A}_x)_j^\top \boldsymbol{x} + (\bar{A}_w)_j^\top \boldsymbol{w}$ and b_j. A prescription solution $\boldsymbol{x}$ for a patient with features $\boldsymbol{w}$ must obey the constraints determined by its split path, *i.e.* only the splits that lead to its assigned leaf i. If $l_i = 0$ for some leaf i, the corresponding split constraints need not be considered. If $l_i = 1$, constraint (6a) will enforce that the solution obeys all split constraints leading to leaf i. If $l_i = 0$, no constraints related to leaf i should be applied. When $l_i = 0$, constraint (6a) will be nonbinding at node j if $M \geq (\bar{A}_x)_j^\top \boldsymbol{x} + (\bar{A}_w)_j^\top \boldsymbol{w} - \bar{b}_j$. Thus we can find the minimum necessary value of M by maximizing these expressions over all possible values of $\boldsymbol{x}$ (for the patient's fixed $\boldsymbol{w}$). For a given patient with features $\boldsymbol{w}$ for whom we wish to optimize treatment, $\text{EM}(\boldsymbol{w})$ is the solution of

$$\max_{\boldsymbol{x}} (\bar{A}_x)_j^\top \boldsymbol{x} + (\bar{A}_w)_j^\top \boldsymbol{w} - \bar{b}_j \tag{7a}$$

$$\text{s.t. } \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{w}) \leq 0, \tag{7b}$$

$$\boldsymbol{x} \in \mathcal{X}(\boldsymbol{w}). \tag{7c}$$

Note that the non-learned constraints on $\boldsymbol{x}$, namely constraint (7b), and the trust region constraint (7c) allow us to reduce the search space when determining M.

A.2. MIO vs. LO formulation for decision trees

In Section 2, we proposed two ways of embedding a decision tree as a constraint. The first uses an LO to represent each feasible leaf node in the tree, while the second directly uses the entire MIO representation of the tree as a constraint. To compare the performance of these two approaches, we learn the palatability constraint using a decision tree (CART) grown to various depths (from a maximum depth of 3 to 20) and solve the optimization model with both approaches. Figure 6 shows that as the maximum allowable tree depth is increased, the number of LOs to be solved also increases. This is because there are more feasible leaves which need to be represented using LOs. Once the tree has reached the optimal depth (selected via cross-validation), increasing the maximum allowable depth of the tree does not cause the tree to grow any further. At this point, the number of LOs to be solved remains constant. When comparing the solution times (averaged over 10 runs), the right graph in Figure 6 shows that the MIO approach is relatively consistent in terms of solution time regardless of the tree depth. With the LO approach however, as the depth of the tree grows, the number of LOs to be solved also grows. While the solution time of a single LO

Figure 6   Comparison of MIO and multiple LO approach to tree representation, as a function of allowable tree depth.
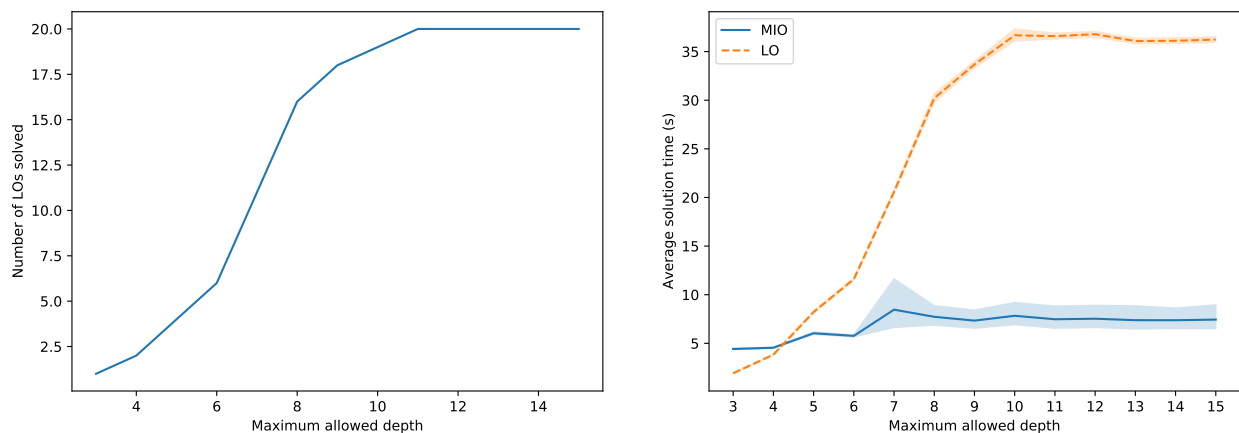
is very low, solving multiple LOs sequentially might be heavily time consuming. A way to speed up the process is to solve the LOs in parallel. A tree of depth 3 requires only one LO to be solved, which takes 1.8 seconds in this problem setting. By parallelizing the solution of the LOs, the total solution time can be expected to take only as long as it takes for the slowest LO to be solved.

A.3. MIO representation of the ReLU activation function

We can represent the ReLU operator, $v = \max\{0, x\}$ the following way:

$$v \geq x, \tag{8a}$$

$$v \leq x - M_L(1 - z), \tag{8b}$$

$$v \leq M_U z, \tag{8c}$$

$$v \geq 0, \tag{8d}$$

$$z \in \{0, 1\}, \tag{8e}$$

where $M_L < 0$ is a lower bound on all possible values of x, and $M_U > 0$ is an upper bound. While this embedding relies on a big-M formulation, it can be improved in multiple ways. The model can be tightened by careful selection of M_L and M_U. Furthermore, Anderson et al. (2020) recently proposed an additional iterative cut generation procedure to improve the strength of the basic big-M formulation.

A.4. Embedding a multi-layer perceptron for multi-class classification

In multi-class classification, the outputs are traditionally obtained by applying a *softmax* activation function, $S(\boldsymbol{x})_i = e^{x_i} / \left(\sum_{k=1}^{K} e^{x_k} \right)$, to the final layer. This function ensures that the outputs sum to one and can thus be interpreted as probabilities. In particular, suppose we have a K-class

classification problem. Each node in the final layer has an associated weight vector $\boldsymbol{\beta}_i$, which maps the nodes of layer $L-1$ to the output layer by $\boldsymbol{\beta}_i^\top \boldsymbol{v}^{L-1}$. The softmax function rescales these values, so that class i will be assigned probability

$$v_i^L = \frac{e^{\boldsymbol{\beta}_i^\top \boldsymbol{v}^{L-1}}}{\sum_{k=1}^K e^{\boldsymbol{\beta}_k^\top \boldsymbol{v}^{L-1}}}.$$

We cannot apply the softmax function directly in an MIO framework with linear constraints. Instead, we use an *argmax* function to directly return an indicator of the highest probability class, similar to the approach with SVC and binary classification MLP. In other words, the output $\boldsymbol{y}$ is the identity vector with $y_i = 1$ for the most likely class. Class i has the highest probability if and only if

$$\beta_{i0}^L + \boldsymbol{\beta}_i^{L\top} \boldsymbol{v}^{L-1} \geq \beta_{k0}^L + \boldsymbol{\beta}_k^{L\top} \boldsymbol{v}^{L-1}, \quad k = 1, \ldots, K.$$

We can constrain this with a big-M constraint as follows:

$$\beta_{i0}^L + \boldsymbol{\beta}_i^{L\top} \boldsymbol{v}^{L-1} \geq \beta_{k0}^L + \boldsymbol{\beta}_k^{L\top} \boldsymbol{v}^{L-1} - M(1 - y_i), \quad k = 1, \ldots, K, \tag{9a}$$

$$\sum_{k=1}^K y_k = 1. \tag{9b}$$

Constraint (9a) forces $y_i = 0$, if the constraint is not satisfied for some $k \in \{1, \ldots, K\}$. Constraint (9b) ensures that $y_i = 1$ for the highest likelihood class. We can then constrain the prediction to fall in our desired class i by enforcing $y_i = 1$.

Appendix B: Trust region

As we explain in Section 2.3, the trust region prevents the predictive models from extrapolating. It is defined as the convex hull of the set $\mathcal{Z} = \{(\bar{\boldsymbol{x}}_i, \bar{\boldsymbol{w}}_i)\}_{i=1}^N$, with $\bar{\boldsymbol{x}}_i \in \mathbb{R}^n$ observed treatment decisions, and $\bar{\boldsymbol{w}}_i \in \mathbb{R}^p$ contextual information. In Section B.1, we explain the importance of using both $\bar{\boldsymbol{x}}$ and $\bar{\boldsymbol{w}}$ in the formulation of the convex hull. When the number of samples (N) is too large, the optimization model trust region constraints may become computationally expensive. In this case, we propose a column selection algorithm which is detailed in Section B.2.

B.1. Defining the convex hull

We characterize the feasible decision space using the convex hull of our observed data. In general, we recommend defining the feasible region with respect to both $\bar{\boldsymbol{x}}$ and $\bar{\boldsymbol{w}}$. This ensures that our prescriptions are reasonable with respect to the contextual variables as well. Note that for different values of $\boldsymbol{w}$, the convex hull in the $\boldsymbol{x}$ space may be different. In Figure 7, the shaded region represents the convex hull of $\mathcal{Z}$ formed by the dataset (blue dots), and the red line represents the set of trusted solutions when $\boldsymbol{w}$ is fixed to a certain value. In Figure 7a, we see that the set of

Figure 7   Effect of $\bar{\boldsymbol{w}}$ on the trust region.
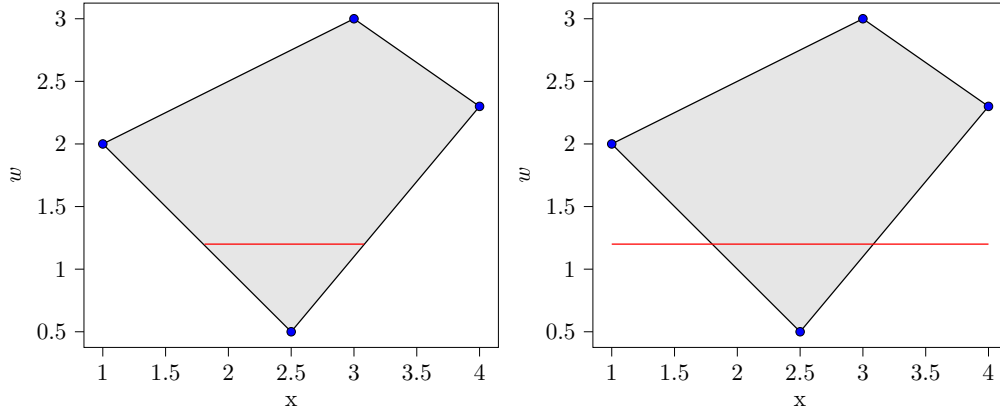
(a) Solutions lie within trust region.

(b) Solutions may lie outside the trust region.

trusted solutions (red line) lies within $\mathrm{CH}(\mathcal{Z})$ when we include $\bar{\boldsymbol{w}}$. If we leave out $\bar{\boldsymbol{w}}$ in the definition of the trust region, then we end up with the undesired situation shown in Figure 7b, where the solution may lie outside of $\mathrm{CH}(\mathcal{Z})$. We observe that in some cases we must define the convex hull with a subset of variables. This is true in cases where the convex hull constraint leads to excessive data thinning, in which case it may be necessary to define the convex hull on treatment variables only.

B.2. Column selection

Let $\mathrm{P}_{\mathcal{I}}$ be a convex and continuously differentiable model consisting of an objective function and constraints that may be known a priori as well as learned from data. Like in Section 2.3, we denote the index set of samples by $\mathcal{I}$. As part of the constraints, the trust region is defined on the entire set $\mathcal{Z}$. We start with the matrix $\boldsymbol{Z} \in \mathbb{R}^{N \times (n+p)}$, where each row corresponds to a given data point in $\mathcal{Z}$. Then, model $\mathrm{P}_{\mathcal{I}}$ is given as

$$\min_{\boldsymbol{\lambda}} \; f(\boldsymbol{Z}^{\top}\boldsymbol{\lambda}) \tag{10a}$$

$$\text{s.t.} \; g_j(\boldsymbol{Z}^{\top}\boldsymbol{\lambda}) \leq 0, \quad j = 1, \ldots, m, \qquad\qquad \perp \boldsymbol{\mu}, \tag{10b}$$

$$\sum_{i \in \mathcal{I}} \lambda_i = 1, \qquad\qquad \perp \rho, \tag{10c}$$

$$\lambda_i \geq 0, \quad i \in \mathcal{I}, \qquad\qquad \perp \boldsymbol{v}, \tag{10d}$$

where the decision variable $\boldsymbol{x}$ is replaced by $\boldsymbol{Z}^{\top}\boldsymbol{\lambda}$. Constraints (10b) include both *known* and *learned* constraints, while constraints (10c) and (10d) are used for the trust region. The dual variables

associated with with constraints (10b), (10c), and (10d) are $\boldsymbol{\mu} \in \mathbb{R}^m, \rho \in \mathbb{R},$ and $\boldsymbol{v} \in \mathbb{R}^N,$ respectively. Note that for readability, we omit the contextual variables ($\boldsymbol{w}$) without loss of generality.

When we deal with huge datasets, solving $P_{\mathcal{I}}$ may be computationally expensive. Therefore, we propose an iterative column selection algorithm (Algorithm 1) that can be used to speed up the optimization while still obtaining a global optima.

Algorithm 1 Column Selection

Input: $\mathcal{I}$ ▷ Index set of columns of $\boldsymbol{Z}^\top$

Output: $\boldsymbol{\lambda}^*$ ▷ Optimal solution

1: $\mathcal{I}' \leftarrow \mathcal{I}^0$ ▷ Initial column pool

2: **while** TRUE **do**

3:      $\boldsymbol{\lambda}^*, (\boldsymbol{\mu}^*, \rho^*, \boldsymbol{v}^*) \leftarrow P_{\mathcal{I}'}$

4:      $\bar{\mathcal{I}} \leftarrow \text{WolfeDual}(\boldsymbol{\lambda}^*, (\boldsymbol{\mu}^*, \rho^*, \boldsymbol{v}^*), \mathcal{I}', \mathcal{I})$ ▷ Column(s) selection

5:      **if** $\bar{\mathcal{I}} \neq \emptyset$ **then**

6:          $\mathcal{I}' \leftarrow \mathcal{I}' \cup \bar{\mathcal{I}}$

7:      **else**

8:          Break

9:      **end if**

10: **end while**

The algorithm starts by initializing $\mathcal{I}' \subseteq \mathcal{I}$ with an arbitrarily small subset of samples $\mathcal{I}^0$ and iteratively solves the restricted master problem $P_{\mathcal{I}'}$ and the WolfeDual function. By solving $P_{\mathcal{I}'}$, we get the primal and dual optimal solutions $\boldsymbol{\lambda}^*$ and $(\boldsymbol{\mu}^*, \rho^*, \boldsymbol{v}^*),$ respectively. The primal and dual optimal solutions, together with $\mathcal{I}$ and $\mathcal{I}',$ are given as input to WolfeDual which returns a set of samples $\bar{\mathcal{I}} \subseteq \mathcal{I} \setminus \mathcal{I}'$ with negative reduced cost. If $\bar{\mathcal{I}}$ is not empty it is added to $\mathcal{I}'$ and a new iteration starts, otherwise the algorithm stops, and $\boldsymbol{\lambda}^*$ (with the corresponding $\boldsymbol{x}^*$) is returned as the global optima of $P_{\mathcal{I}}$. A visual interpretation of Algorithm 1 is shown in Figure 4.

In function WolfeDual, samples $\bar{\mathcal{I}}$ are selected using the Karush–Kuhn–Tucker (KKT) stationary condition which corresponds to the equality constraint in the Wolfe dual formulation of $P_{\mathcal{I}}$ (Wolfe 1961). The KKT stationary condition of $P_{\mathcal{I}'}$ is

$$\nabla_{\boldsymbol{\lambda}} f(\tilde{\boldsymbol{Z}}^\top \boldsymbol{\lambda}^*) + \sum_{i=1}^m \mu_i^* \nabla_{\boldsymbol{\lambda}} g_i(\tilde{\boldsymbol{Z}}^\top \boldsymbol{\lambda}^*) - \boldsymbol{e}\rho^* - \boldsymbol{v}^* = \boldsymbol{0}, \tag{11}$$

where $\tilde{\boldsymbol{Z}}$ is the matrix constructed with samples in $\mathcal{I}'$, and $\boldsymbol{e}$ is an N'-dimensional vector of ones with $N' = |\mathcal{I}'|$. Equation (11) can be rewritten as

$$\tilde{\boldsymbol{Z}}\nabla_{\boldsymbol{x}}f(\tilde{\boldsymbol{Z}}^\top\boldsymbol{\lambda}^*) + \sum_{i=1}^{m}\mu_i^*\tilde{\boldsymbol{Z}}\nabla_{\boldsymbol{x}}g_i(\tilde{\boldsymbol{Z}}^\top\boldsymbol{\lambda}^*) - \boldsymbol{e}\rho^* - \boldsymbol{v}^* = \boldsymbol{0}. \tag{12}$$

Equation (12) is used to evaluate the reduced cost related to each sample $\bar{\boldsymbol{z}} \in \mathcal{Z}$ which is not in matrix $\tilde{\boldsymbol{Z}}$. Consider a new sample $\bar{\boldsymbol{z}}$ in (12), with its associated $\lambda_{N'+1}$ set equal to zero. $(\lambda_1^*, \ldots, \lambda_{N'}^*, \lambda_{N'+1})$ is still a feasible solution of the restricted master problem $\mathrm{P}_{\mathcal{I}'}$, since it does not affect the value of $\boldsymbol{x}$. As a consequence, $\boldsymbol{\mu}$ and ρ will not change their value, nor will f and $\boldsymbol{g}$. The only unknown variable is $v_{N'+1}$, namely the reduced cost of $\bar{\boldsymbol{z}}$. However, we can write it as

$$\begin{pmatrix} \boldsymbol{v}^* \\ v_{N'+1} \end{pmatrix} = \begin{pmatrix} \tilde{\boldsymbol{Z}} \\ \bar{\boldsymbol{z}}^\top \end{pmatrix}\nabla_{\boldsymbol{x}}f(\tilde{\boldsymbol{Z}}^\top\boldsymbol{\lambda}^*) + \sum_{i=1}^{t}\mu_i^*\begin{pmatrix} \tilde{\boldsymbol{Z}} \\ \bar{\boldsymbol{z}}^\top \end{pmatrix}\nabla_{\boldsymbol{x}}g_i(\tilde{\boldsymbol{Z}}^\top\boldsymbol{\lambda}^*) - \boldsymbol{e}\rho^*. \tag{13}$$

If $v_{N'+1}$ is negative it means that we may improve the incumbent solution of $\mathrm{P}_{\mathcal{I}'}$ by including the sample $\bar{\boldsymbol{z}}$ in $\tilde{\boldsymbol{Z}}$.

LEMMA 1. *After solving the convex and continuously differentiable problem $P_{\mathcal{I}'}$, the sample in $\mathcal{I} \setminus \mathcal{I}'$ with the most negative reduced cost is a vertex of the convex hull CH($\mathcal{Z}$).*

Proof From equation (13) we have

$$v_{N'+1} = \bar{\boldsymbol{z}}^\top\nabla_{\boldsymbol{x}}f(\tilde{\boldsymbol{Z}}^\top\boldsymbol{\lambda}^*) + \bar{\boldsymbol{z}}^\top\nabla_{\boldsymbol{x}}\boldsymbol{g}(\tilde{\boldsymbol{Z}}^\top\boldsymbol{\lambda}^*)\boldsymbol{\mu}^* - \rho^*. \tag{14}$$

The problem of finding $\bar{\boldsymbol{z}}$, such that its reduced cost is the most negative one, can be written as a linear program where equation (14) is being minimized, and a solution must lie within CH($\mathcal{Z}$). That is,

$$\begin{aligned} \min_{\boldsymbol{z}, \boldsymbol{\lambda}} \quad & \boldsymbol{z}^\top\nabla_{\boldsymbol{x}}f(\tilde{\boldsymbol{Z}}^\top\boldsymbol{\lambda}^*) + \boldsymbol{z}^\top\nabla_{\boldsymbol{x}}\boldsymbol{g}(\tilde{\boldsymbol{Z}})\boldsymbol{\mu}^* - \rho^* \\ \text{s.t.} \quad & \boldsymbol{Z}^\top\boldsymbol{\lambda} = \boldsymbol{z}, \\ & \sum_{j \in \mathcal{I}}\lambda_j = 1, \\ & \lambda_j \geq 0, \quad j \in \mathcal{I}, \end{aligned} \tag{15}$$

where $\boldsymbol{z}$ and $\boldsymbol{\lambda}$ are the decision variables, and $\boldsymbol{\mu}^*$, $\boldsymbol{\lambda}^*$, ρ^* are fixed parameters. Since the objective function is linear with respect to $\boldsymbol{z}$, the optimal solution of (15) will necessarily be a vertex of CH($\mathcal{Z}$). □

To illustrate the benefits of column selection, consider the following convex optimization problem that we shall refer to as P_{exp}:

$$\min_{\boldsymbol{x}} \boldsymbol{c}^\top\boldsymbol{x} \tag{16a}$$

Figure 8   Effect of column selection on computation time. Solution times are reported for three different sizes of

problem P_{exp}. Small-scale: $n = 5$, $k = 10$. Medium-scale: $n = 10$, $k = 50$. Large-scale: $n = 20$, $k = 100$.

The number of samples goes from 500 to 5×10^5. In each iteration, the sample with most negative

reduced cost is selected. The same problem is solved using MOSEK (2019) with conic reformulation for

10 different instances where c, A, and b are randomly generated.
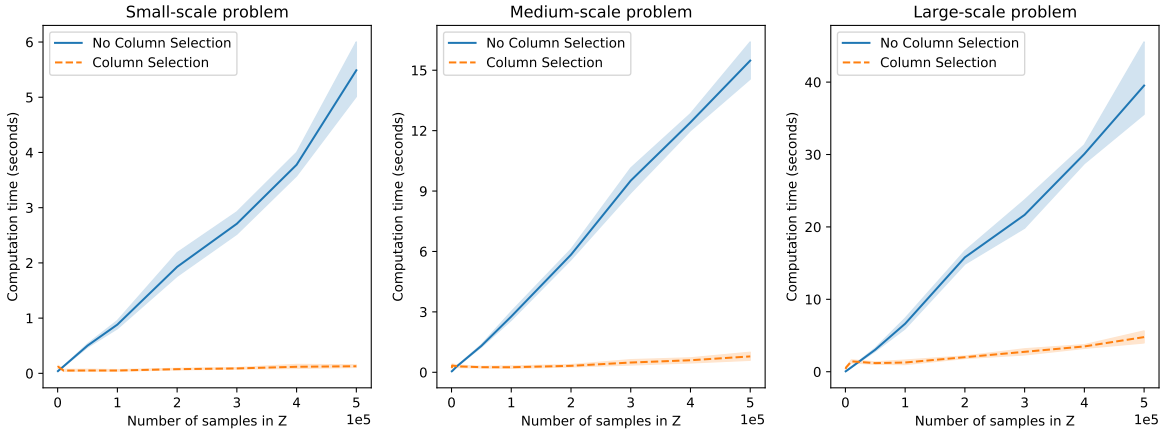
$$\text{s.t. } log(\sum_{i=1}^{n} e^{x_i}) \leq t, \tag{16b}$$

$$\boldsymbol{Ax} \leq b, \tag{16c}$$

$$\sum_{i=1}^{N} \lambda_i \bar{\boldsymbol{z}}_i = \boldsymbol{x}, \tag{16d}$$

$$\sum_{j=1}^{N} \lambda_j = 1, \tag{16e}$$

$$\lambda_j \geq 0, \quad j = 1 \dots N. \tag{16f}$$

Without a loss of generality, we assume that the constraint (16b) is known a priori, and constraints (16c) are the linear embeddings of learned constraints with $\boldsymbol{A} \in \mathbb{R}^{k \times n}$ and $\boldsymbol{b} \in \mathbb{R}^k$. Constraints (16d-16f) define the trust region based on N datapoints. Figure 8 shows the computation time required to solve P_{exp} with different values of n, k, and N. The "No Column Selection" approach consists of solving P_{exp} using the entire dataset. The "Column Selection" approach makes use of Algorithm 1 to solve the problem, starting with $|\mathcal{I}^0| = 100$, and selecting only one sample at each iteration, *i.e.*, the one with the most negative reduced cost. It can be seen that in all cases, the use of column selection results in significantly improved computation times. This allows us to more quickly define the trust region for problems with large amounts of data.

Table 7 Nutritional contents per gram for different foods.

Food	Eng(kcal)	Prot(g)	Fat(g)	Cal(mg)	Iron(mg)	VitA(ug)	ThB1(mg)	RibB2(mg)	NicB3(mg)	Fol(ug)	VitC(mg)	Iod(ug)
Beans	335	20	1.2	143	8.2	0	0.5	0.22	2.1	180	0	0
Bulgur	350	11	1.5	23	7.8	0	0.3	0.1	5.5	38	0	0
Cheese	355	22.5	28	630	0.2	120	0.03	0.45	0.2	0	0	0
Fish	305	22	24	330	2.7	0	0.4	0.3	6.5	16	0	0
Meat	220	21	15	14	4.1	0	0.2	0.23	3.2	2	0	0
Corn-soya blend	380	18	6	513	18.5	500	0.65	0.5	6.8	0	40	0
Dates	245	2	0.5	32	1.2	0	0.09	0.1	2.2	13	0	0
Dried skim milk	360	36	1	1257	1	1,500	0.42	1.55	1	50	0	0
Milk	360	36	1	912	0.5	280	0.28	1.21	0.6	37	0	0
Salt	0	0	0	0	0	0	0	0	0	0	0	1000000
Lentils	340	20	0.6	51	9	0	0.5	0.25	2.6	0	0	0
Maize	350	10	4	13	4.9	0	0.32	0.12	1.7	0	0	0
Maize meal	360	9	3.5	10	2.5	0	0.3	0.1	1.8	0	0	0
Chickpeas	335	22	1.4	130	5.2	0	0.6	0.19	3	100	0	0
Rice	360	7	0.5	7	1.2	0	0.2	0.08	2.6	11	0	0
Sorghum/millet	335	11	3	26	4.5	0	0.34	0.15	3.3	0	0	0
Soya-fortified bulgur wheat	350	17	1.5	54	4.7	0	0.25	0.13	4.2	74	0	0
Soya-fortified maize meal	390	13	1.5	178	4.8	228	0.7	0.3	3.1	0	0	0
Soya-fortified sorghum grits	360	360	1	40	2	0	0.2	0.1	1.7	50	0	0
Soya-fortified wheat flour	360	16	1.3	211	4.8	265	0.66	0.36	4.6	0	0	0
Sugar	400	0	0	0	0	0	0	0	0	0	0	0
Oil	885	0	100	0	0	0	0	0	0	0	0	0
Wheat	330	12.3	1.5	36	4	0	0.3	0.07	5	51	0	0
Wheat flour	350	11.5	1.5	29	3.7	0	0.28	0.14	4.5	0	0	0
Wheat-soya blend	370	20	6	750	20.8	498	1.5	0.6	9.1	0	40	0

Eng = Energy, Prot = Protein, Cal = Calcium, VitA = Vitamin A, ThB1 = ThiamineB1, RibB2 = RiboflavinB2, NicB3 = NicacinB3, Fol = Folate, VitC = Vitamin C, Iod = Iodine

Table 8 Nutrient requirements used in optimization model.

Type	Eng(kcal)	Prot(g)	Fat(g)	Cal(mg)	Iron(mg)	VitA(ug)	ThB1(mg)	RibB2(mg)	NicB3(mg)	Fol(ug)	VitC(mg)	Iod(ug)
Avg person day	2100	52.5	89.25	1100	22	500	0.9	1.4	12	160	0	150

Eng = Energy, Prot = Protein, Cal = Calcium, VitA = Vitamin A, ThB1 = ThiamineB1, RibB2 = RiboflavinB2, NicB3 = NicacinB3, Fol = Folate, VitC = Vitamin C, Iod = Iodine

Appendix C: WFP case study

Table 7 and Table 8 show the nutritional value of each food and our assumed nutrient requirements, respectively. The values adopted are based on the World Health Organization (WHO) guidelines (UNHCR et al. 2002).

### C.1. Effect of ensemble violation limit

Figure 9 reports the effect of the ensemble violation limit (α) on the objective (Total Cost) and constraint (Palatability) in the WFP case study. As expected, we see a tradeoff between the total cost of the recommended WFP food baskets and the achieved palatability. Higher violation limits (larger α) obtain lower costs at the expense of lower palatability. Lower α values result in more conservative solutions with higher cost but better palatability. This parametrization of individual model violation tolerance allows us to directly quantify this tradeoff and can provide a useful tool in assessing solution alternatives.

Appendix D: Chemotherapy regimen design

D.1. Data Processing

The data for this case study includes three components, study cohort characteristics ($\boldsymbol{w}$), treatment variables ($\boldsymbol{x}$), and outcomes ($\boldsymbol{y}$). The raw data was obtained from Bertsimas et al. (2016), in which the authors manually curated data from 495 clinical trial arms for advanced gastric cancer. Our feature space was processed as follows:

Figure 9   Effect of violation limit on objective (total cost) and constraint (palatability). The average is reported

over 100 problem instances.
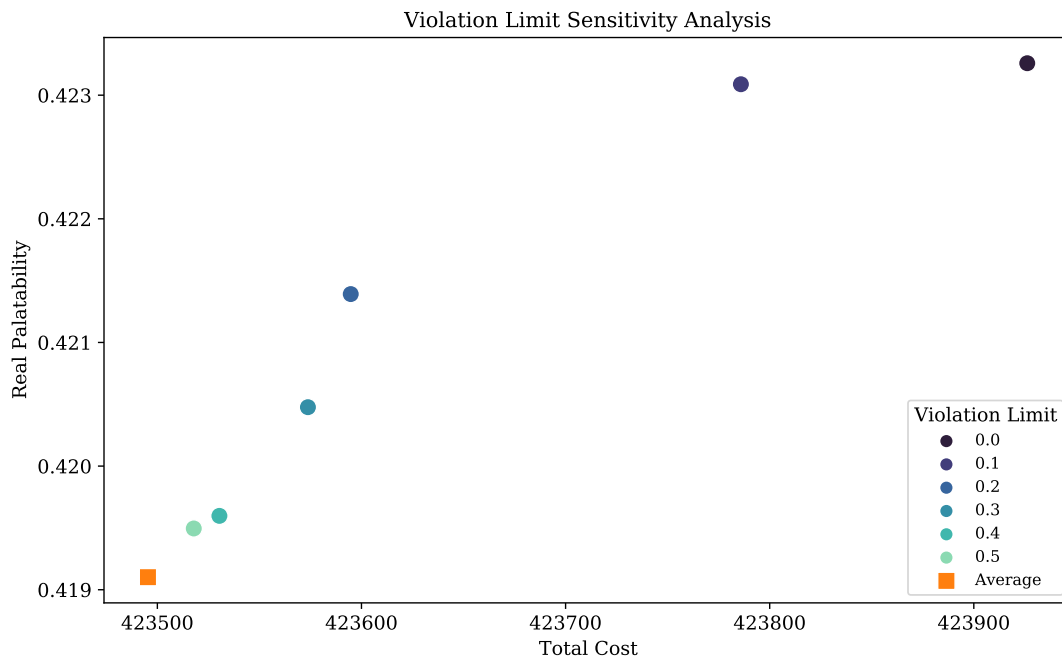
Cohort Characteristics. We included several cohort characteristics to adjust for the study context: fraction of male patients, median age, primary site breakdown (Stomach vs. GEJ), fraction of patients receiving prior palliative chemotherapy, and mean ECOG score. We also included variables for the study context: the study year, country, and number of patients. Missing data was imputed using multiple imputation based on the other contextual variables; 20% of observations had one missing feature and 6% had multiple missing features.

Treatment Variables. Chemotherapy regimens involve multiple drugs being delivered at potentially varied frequencies over the course of a chemotherapy cycle. As a result, multiple dimensions of the dosage must be encoded to reflect the treatment strategy. As in Bertsimas et al. (2016), we include three variables to represent each drug: an indicator (1 if the drug is used in the regimen), instantaneous dose, and average dose.

Outcomes. We use Overall Survival (OS) as our survival metric, as reported in the clinical trials. Any observations with unreported OS are excluded. We consider several "dose-limiting toxicities" (DLTs): Grade 3/4 constitutional, gastrointestinal, infection, and neurological toxicities, as well as Grade 4 blood toxicities. The toxicities reported in the original clinical trials are aggregated according to the CTCAE toxicity classes (Cancer Therapy Evaluation Program 2006). We also include a variable for the occurrence of any of the four individual toxicities (t_i for each toxicity

Table 9 Comparison of out-of-sample R^2 all considered models for learned outcomes in chemotherapy regimen selection problem.

Outcome	Linear	SVM	CART	ORT-H	RF	GBM
Any DLT	0.268	-0.094	-0.016	0.000	0.152	0.202
Blood	0.196	-1.102	0.012	0.026	0.153	0.105
Constitutional	0.106	0.144	0.157	-0.179	0.194	0.136
Infection	0.082	-0.511	-0.222	0.000	0.070	0.035
Gastrointestinal	0.141	-0.196	-0.023	-0.067	0.066	0.083
Overall Survival	0.448	0.385	0.474	0.505	0.496	0.450

$i \in T$, called DLT proportion; we treat these toxicity groups as independent and thus define the DLT proportion as

$$DLT = 1 - \prod_{i \in T}(1 - t_i).$$

We define Grade 4 blood toxicity as the maximum of five individual blood toxicities (related to neutrophils, leukocytes, lymphocytes, thrombocytes, anemia). Observations missing all of these toxicities were excluded; entries with partial missingness were imputed using multiple imputation based on other blood toxicity columns. Similarly, observations with no reported Grade 3/4 toxicities were excluded; those with partial missingness were imputed using multiple imputation based on the other toxicity columns. This exclusion criteria resulted in a final set of 461 (of 495) treatment arms.

We split the data into training/testing sets temporally. The training set consists of all clinical trials through 2008, and the testing set consists of all 2009-2012 trials. We exclude trials from the testing set if they use new drugs not seen in the training data (since we cannot evaluate these given treatments). We also identify sparse treatments (defined as being only seen once in the training set) and remove all observations that include these treatments. The final training set consists of 320 observations, and the final testing set consists of 96 observations.

D.2. Predictive Models

Table 9 shows the out-of-sample performance of all considered methods in the model selection pipeline. We note that model choice is based on the 5-fold validation performance, so it does not necessarily correspond to the highest test set performance.

D.3. Prescription Evaluation

Table 10 shows the performance of the models that comprise the ground truth ensemble used in the evaluation framework. These models trained on the full data. We see that the ensemble models, particularly RF and GBM, have the highest performance. These models are trained on more data

Table 10   Performance (R^2) of individual models in ground truth
ensemble for model evaluation.

outcome	Linear	SVM	CART	RF	GBM	XGB
Any DLT	0.301	0.330	0.250	0.573	0.670	0.323
Blood	0.287	0.351	0.211	0.701	0.813	0.446
Constitutional	0.139	0.224	0.246	0.602	0.682	0.285
Infection	0.217	0.303	0.139	0.514	0.588	0.247
Gastrointestinal	0.201	0.328	0.238	0.563	0.733	0.475
Overall Survival	0.528	0.469	0.421	0.815	0.827	0.756

and include more complex parameter options (*e.g.*, deeper trees, larger forests) since they are not required to be embedded in the MIO and are rather used directly to generate predictions. For this reason, the GT ensemble could also be generalized to consider even broader method classes that are not directly MIO-representable, such as neural networks with alternative activation functions, providing an additional degree of robustness.