Noname manuscript No.
(will be inserted by the editor)

The Voice of Optimization

Dimitris Bertsimas and Bartolomeo Stellato

Received: date / Accepted: date

Abstract We introduce the idea that using optimal classification trees (OCTs) and optimal classification trees with-hyperplanes (OCT-Hs), interpretable machine learning algorithms developed by **??**, we are able to obtain insight on the strategy behind the optimal solution in continuous and mixed-integer convex optimization problem as a function of key parameters that affect the problem. In this way, optimization is not a black box anymore. Instead, we redefine optimization as a multiclass classification problem where the predictor gives insights on the logic behind the optimal solution. In other words, OCTs and OCT-Hs give optimization a voice. We show on several realistic examples that the accuracy behind our method is in the 90%-100% range, while even when the predictions are not correct, the degree of suboptimality or infeasibility is very low. We compare optimal strategy predictions of OCTs and OCT-Hs and feedforward neural networks (NNs) and conclude that the performance of OCT-Hs and NNs is comparable. OCTs are somewhat weaker but often competitive. Therefore, our approach provides a novel insightful understanding of optimal strategies to solve a broad class of continuous and mixed-integer optimization problems.

Keywords Parametric optimization · Interpretability · Sampling · Multiclass classification

D. Bertsimas
Operations Research Center and Sloan School of Management,
Massachusetts Institute of Technology,
Cambridge, MA 02139,
E-mail: dbertsim@mit.edu

B. Stellato
Operations Research Center and Sloan School of Management,
Massachusetts Institute of Technology,
Cambridge, MA 02139,
E-mail: stellato@mit.edu

1 Introduction

Optimization has a long and distinguished history that has had and continues to have genuine impact in the world. In a typical optimization problem in the real world, practitioners see optimization as a black-box tool where they formulate the problem and they pass it to a solver to find an optimal solution. Especially in high dimensional problems typically encountered in real world applications, it is currently not possible to interpret or intuitively understand the optimal solution. However, independent from the optimization algorithm used, practitioners would like to understand how problem parameters affect the optimal decisions in order to get intuition and interpretability behind the optimal solution. Moreover, in almost all real world applications of optimization the main objective is not to solve just one problem but to solve multiple similar instances that vary slightly from each other. In fact, in most real-world applications we solve similar optimization problems multiple times with varying data depending on problem-specific parameters.

Our goal in this paper is to propose a framework to predict the optimal solution as parameters of the problem vary and do so in an interpretable way. A naive approach could be to learn directly the optimal solution from the problem parameters. However, this method is computationally *intractable* and *imprecise*: intractable, because it would require a potentially high dimensional predictor with hundreds of thousands of components depending on the decision variable size; imprecise, because it would involve a regression task that would naturally carry a generalization error leading to suboptimal or even infeasible solutions. Instead, our approach encodes the optimal solution with a small amount of information that we denote as *strategy*. Depending on the problem class, a strategy can have different meanings but it always corresponds to the complete information needed to efficiently recover the optimal solution.

In recent times, machine learning has also had significant impact to the world. Optimization methods has been a major driver of its success (**??**). In this paper, we apply machine learning to optimization with the objective to give a voice to optimization, that is to provide interpretability and intuition behind optimal solutions.

First, we solve an optimization problem for many parameter combinations and obtain the optimal strategy: the set of active constraints as well as the value of the discrete variables. Second, we encode the strategies with unique integer scalars. In this way, we have a mapping from parameters to the optimal solution strategies, which gives rise to a multiclass classification problem (**?**). We solve the multiclass classification problem using optimal classification trees (OCTs) and optimal classification trees with-hyperplanes (OCT-Hs) developed by **??**, interpretable state of the art classification algorithms as well as neural networks (NNs), which, while not interpretable, serve as a benchmark to compare the accuracy of OCTs and OCT-Hs.

1.1 Related Work

Machine learning for optimization. There has been as significant recent interest from both the computer science and the optimization communities to systematically analyze and solve optimization problems using machine learning (**?**).

We have so far seen two main approaches in applying machine learning to optimization. The first one involves learning heuristics to improve the performance of optimization algorithms. Most optimization methods consist of iterative routines with repeated decisions based on expert knowledge and manual tuning. For example, branch-and-bound (B&B) involves several decision rules about the branching behavior that are hand-tuned into the solvers. However, this tuning can be very complex because it concerns several aspects of the problem that are not known a priori. To overcome these limitations **?** propose to learn the branching rules showing performance improvements over commercial hand-tuned algorithms. Similarly, **?** approximate strong branching rules with learning methods. Machine learning has been useful also to select reformulations and decompositions for mixed-integer optimization (MIO). **?** learn in which cases it is more efficient to solve mixed-integer quadratic optimization problem (MIQO) by linearizing or not the cost function. They model it as a classification problem showing advantages compared to how this choice is made heuristically inside state-of-the-art solvers. **?** propose a similar method applied to decomposition selection for MIO. The second approach applying machine learning to decision making sees optimization problems as control tasks to tackle using reinforcement learning (**?**). Problems suitable to this framework include knapsack-like or network problems with multistage decisions. **?** develop a method to learn heuristics over graph problems. In this way the node selection criterion becomes the output of a specialized neural network that does not depend on the graph size (**?**). Every time a new node is visited, **?** feed a graph representation of the problem to the NN obtaining a criterion suggesting the next node to select in the optimal solution.

Learning parametric programs. Even though recent approaches for integrating machine learning and optimization show promising results, they do not consider the parametric nature of the problems appearing in real-world applications. It is often the case that practitioners solve the same problem with slightly varying parameters multiple times generating a large amount of data describing how the parameters affect the optimal solution. There are only a few recent papers exploiting this information to build better solution algorithms. The idea of learning the set of active constraints for parametric online optimization has been proposed by **?**. The authors frame the learning procedure as a sampling scheme where they collect all the active sets appearing from the parameters. However, we found several limitations of their approach. First, in the online phase, they evaluate all the relevant active sets in parallel and pick the best one (**?**, *ensemble policy*). Therefore, they do not solve the optimization problem as a multiclass classification problem and they are not able to gain insights on how the parameters affect the optimal strategy. In addition, they do not tackle mixed-integer optimization problems but only continuous convex ones. Finally, the sampling strategy by **?** has to be tuned for each specific problem since it depends on at least four different parameters. This is because the authors compute the probabilistic guarantees based on how many new active sets appear over the samples in a window of a specific size (**?**, Section 3). In this work, instead, we provide a concise Good-Turing estimator (**?**) for the probability of finding new unseen strategies which can be directly applied to many different problem instances. In the field of model predictive control (MPC), **?** warm-start an online active set method for solving quadratic optimization problems (QOs). However, in that work there is no rigorous sampling scheme with probability bounds

to obtain the different active sets. Instead, the authors either simulate the dynamical controlled system or provide an alternative gridding method to search for the relevant active sets. Furthermore, the method by **?** is tailored to a specific linear control problem in the form of QO and cannot tackle general convex or mixed-integer convex problem. Learning for parametric programs can also speedup the online solution algorithms. **?** apply the framework in this paper to online mixed-integer optimization. By focusing on speed instead of interpretability, they obtain significant speedups compared to state-of-the-art solvers.

Sensitivity analysis. The study of how changes in the problem parameters affect the optimal solution has for long been studied in sensitivity analysis, (**?**, Chapter 5) and (**?**, Section 5.6) for introductions on the topic. While sensitivity analysis is related to our work as it analyzes the effects of changes in problem parameters, it is fundamentally different both in philosophy and applicability. In sensitivity analysis the problem parameters are uncertain and the goal is to understand how their perturbations affect the optimal solution. This aspect is important when, for example, the problem paramters are not known with high accuracy and we would like to understand how the solution would change in case of perturbations. In this work instead, we consider problems without uncertainty and use previous data to learn how the problem parameters affect the optimal solution. Therefore, our problems are deterministic and we are not considering perturbations around any nominal value. As a matter of fact, the data we use for training are not restricted to lie close to any nominal point. In addion, sensitivity analysis usually studies continuous optimization problems since it relies on the dual variables at the optimal solution to determine the effect of parameter perturbations. This is why there has been only limited work on sensitivity analysis for MIO. In contrast, we show that our method can be directly applied to problems with integer variables.

1.2 Contributions

In this paper, we propose a learning framework to give a voice to continuous and mixed-integer convex optimization problems. With our approach we can reliably sample the occurring strategies using the Good-Turing estimator, learn an interpretable classifier using OCTs and OCT-Hs, interpret the dependency between the optimal strategies and the key parameters from the resulting tree and solve the optimization problem using the learned predictor.

Our specific contributions include:

1. We introduce a new framework for gaining insights on the solution of optimization problems as a function of their key parameters. The optimizer becomes an interpretable machine learning classifier using OCTs and OCT-Hs which highlights the relationship between the optimal solution strategy and the problem parameters. In this way optimization is no longer a black box and our approach gives it a voice that provides intuition and interpretability on the optimal solution.
2. We show that our method can be applied to a broad collection of optimization problems including convex continuous and convex mixed-integer optimization problem. We do not pose any assumption on the dependency of the cost and constraints on the problem parameters.

3. We introduce a new exploration scheme based on the Good-Turing estimator (**?**) to discover the strategies arising from the problem parameters. This scheme allows us to reliably bound the probability of encountering unseen strategies in order to be sure our classifier accurately represents the optimization problem.
4. In several realistic examples we show that the sample accuracy of our method is in the 90%-100% range, while even in the cases where the prediction is not correct the degree of suboptimality or infeasibility is very low. We also compare the performance of OCTs and OCT-Hs to NNs obtaining comparable out-of-sample accuracy.

In other words, our approach provides a novel, reliable and insightful framework for understanding the strategies to compute the optimal solution of a broad class of continuous and mixed-integer optimization problems.

1.3 Paper Structure

The structure of the paper is as follows: In Section **??**, we define an optimal strategy to solve continuous and mixed-integer optimization problems and present several concrete examples that demonstrate what we call the voice of optimization. In Section **??**, we outline the core part of our approach of using multiclass classification to learn the mapping from parameters to optimal strategies. We further present an approach to estimate how likely it is to encounter a parameter that leads to an optimal strategy that has not yet been seen. In Section **??**, we outline our Python implementation MLOPT (Machine Learning Optimizer). In Section **??**, we test our approach on multiple examples from continuous and mixed-integer optimization and present the accuracy of predicting the optimal strategy of OCTs and OCT-Hs in comparison with a NNs implementation. Section **??** summarizes our conclusions. Appendices **??** and **??** briefly present optimal classification trees (OCTs and OCT-Hs) and NNs, respectively to make the paper self-contained.

2 The Voice of Optimization

In this section, we introduce the notion of an optimal strategy to solve continuous and mixed-integer optimization problems. Given a parameter $\theta \in \mathbf{R}^p$ we define *strategy* $s(\theta)$ as the complete information needed to efficiently compute the optimal solution of an optimization problem given the parameter θ. We assume that the problem is always feasible for every encountered value of θ.

Note that in the illustrative examples in this section we omit the details of the learning algorithm which we explain in Section **??**. Instead, we focus on the interpretation of the resulting strategies which correspond to a decision tree. For these examples the accuracy of our approach to find the optimal solution was always 100%, confirming that the resulting models accurately recover the optimal solution. For simplicity of exposition, we sample the problem parameters in the examples of this section in simple regions such as intervals or balls around specific points but this is not a strict requirement for our approach as it will be more clear in the next sections. In the examples in this section we used OCTs since their

accuracy was 100% and they are more in terpretable than OCT-Hs. In the last example, we used both an OCT and an OCT-H for comparison.

2.1 Optimal Strategies in Continuous Optimization

Consider the continuous optimization problem

$$\begin{array}{ll} \text{minimize} & f(\theta, x) \\ \text{subject to} & g(\theta, x) \leq 0, \end{array} \tag{1}$$

where $x \in \mathbf{R}^n$ is the vector of decision variables and $\theta \in \mathbf{R}^p$ the vector of parameters affecting the problem data. Functions $f : \mathbf{R}^p \times \mathbf{R}^n \to \mathbf{R}$ and $g : \mathbf{R}^p \times \mathbf{R}^n \to \mathbf{R}^m$ are assumed to be convex in x. Given a parameter θ we denote the optimal primal solution as $x^\star(\theta)$ and the optimal cost function value as $f(\theta, x^\star(\theta))$.

Tight constraints. Let us define the *tight constraints* $\mathcal{T}(\theta)$ as the set of constraints that are satisfied as equalities at optimality,

$$\mathcal{T}(\theta) = \{i \in \{1, \dots, m\} \mid g_i(\theta, x^\star) = 0\}. \tag{2}$$

Given the tight constraints, all the other constraints are no longer needed to solve the original problem.

For non-degenerate problems, the tight constraints correspond to the *support constraints*, *i.e.*, the set of constraints that, if removed, would allow a decrease in $f(\theta, x^\star(\theta))$ (**?**, Definition 2.1). In the case of linear optimization problems (LOs) the support constraints are the linearly independent constraints defining a basic feasible solution (**?**, Definition 2.9). An important property of support constraints is that they cannot be more than the dimension n of the decision variable (**?**, Proposition 1), (**?**, Lemma 2.2). This fact plays a key role in our method to reduce the complexity to predict the solution of parametric optimization problems. The benefits are more evident when the number of constraints is much larger than the number of variables, *i.e.*, $n \ll m$.

Multiple optimal solutions and degeneracy. In practice we can encounter problems with multiple optimal solutions or degeneracy. When the cost function is not strongly convex in x, we can have multiple optimal solutions. In these cases we consider only one of the optimal solutions to be $x^\star$ since it is enough for our training purposes. Note that most solvers such as (**?**) return anyway only one solution and not the complete set of optimal solutions. With degenerate problems, we can have more tight constraints than support constraints for an optimal solution $x^\star$. This is because the support constraints set is no longer unique in case of degeneracy. However, we use the set of tight constraints which remains unique since it includes by definition all the constraints that are satisfied as equalities independently from being support constraints or not. In addition, our method is still efficient because the number of tight constraints is in practice much lower than the total number of constraints, even in case of degeneracy. Therefore, we can directly apply our framework to problems with multiple optimal solutions and affected by degeneracy.

Solution strategy. We can now define our strategy as the index of tight constraints at the optimal solution, *i.e.*, $s(\theta) = \mathcal{T}(\theta)$. Given the optimal strategy, solving (**??**) corresponds to solving

$$\begin{array}{ll} \text{minimize} & f(\theta, x) \\ \text{subject to} & g_i(\theta, x) \leq 0, \quad \forall i \in \mathcal{T}(\theta). \end{array} \tag{3}$$

This problem is easier than (**??**), especially when $n \ll m$. Note that we can enforce the components g_i that are linear in x as equalities. This further simplifies (**??**) while preserving its convexity. In case of LO and QO when the cost f is linear or quadratic and the constraints g are all linear, the solution of (**??**) corresponds to solving a linear system of equations defined by the KKT conditions (**?**, Section 10.2).

Inventory management

Consider an inventory management problem with horizon $t = 0, \ldots, T-1$ with $T = 30$. The decision variables are u_t describing how much we order at time t and x_t describing the inventory level at time t. $c = 2$ is the cost if ordering, $h = 1$ the holding cost and $p = 3$ the shortage cost. We define the maximum quantity we can order each time as $M = 3$. The parameters are the product demand d_t at time t and the initial value of the inventory x_{init}. The optimization problem can be written as follows

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{t=0}^{T-1} \max(hx_t, -px_t) + cu_t \\ \text{subject to} & x_{t+1} = x_t + u_t - d_t, \quad t = 0, \ldots, T-1 \\ & x_0 = x_{\text{init}} \\ & 0 \leq u_t \leq M, \quad t = 0, \ldots, T-1. \end{array} \tag{4}$$

Depending on d_t with $t = 0, \ldots, T-1$ and x_{init}, we need to adapt our ordering policy to minimize the cost. We assume that $d_t \in [1, 3]$ and $x_{\text{init}} \in [7, 13]$.

The strategy selection is summarized in Figure **??** and can be easily interpreted: $u_t = 0$ for $t \leq t_0$ and then $u_t = d_t$ for $t > t_0$. A inventory level trajectory example can be seen in Figure **??**. Let us outline the strategies depicted in Figure **??**. For example, if the initial inventory level x_{init} is below 7.91, we should apply Strategy 4, where we wait only $t_0 = 3$ time steps before ordering. Otherwise, if $7.91 \leq x_{\text{init}} < 9.97$ we should wait for $t_0 = 5$ time steps before ordering because the initial inventory level is higher. The other branches can be similarity interpreted. Note that for this problem instance the decision is largely independent from d_t. The only exception comes with d_5 that determines the choice of strategies in the right-hand side of the tree.

The strategies we derived are related to the (s, S) policies for inventory management (**?**): if the inventory level falls below value s, we order so that the level becomes S. The optimal decisions from our approach are as simple as the (s, S) policies because we can describe them with *if-then-else* rules in Figure **??**. However, they give better performance because they take into account future predictions and they can deal with more complex problem formulations with harder constraints.

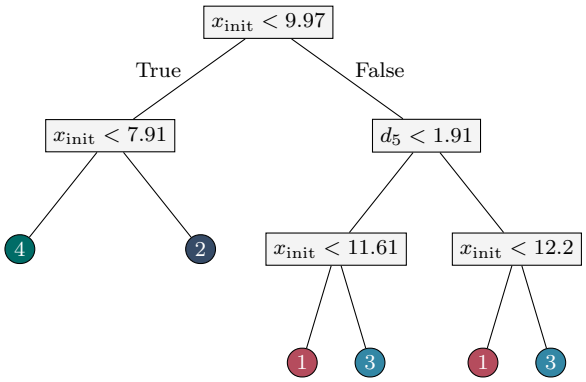

Fig. 1 Decision strategy for the inventory example. *Strategy 1*: do not order for the first 4 time steps, then order matching the demand. *Strategy 2*: do not order for the first 5 time steps, then order matching the demand. *Strategy 3*: do not order for the first 6 time steps, then order matching the demand. *Strategy 4*: do not order for the first 3 time steps, then order matching the demand.

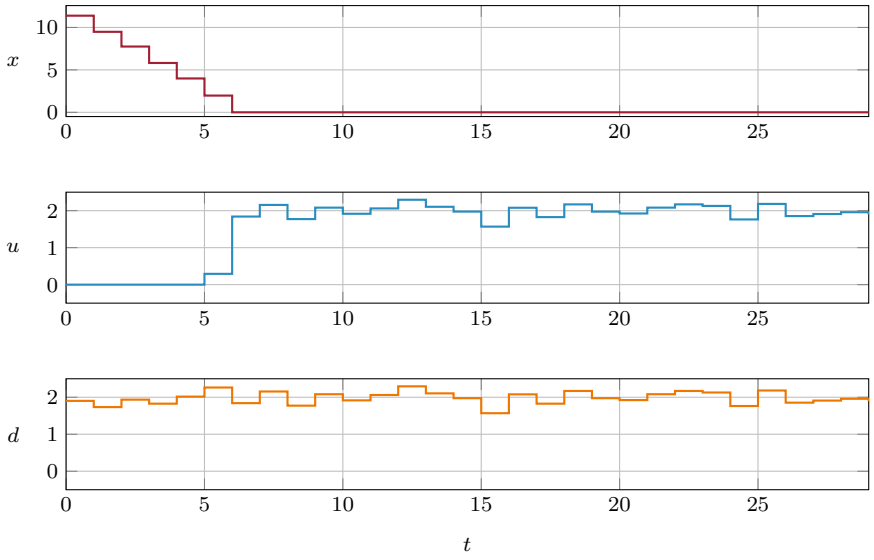

Fig. 2 Inventory behavior with Strategy 2. The lower bound on u_t is active for the first 5 steps. Then $u_t = d_t$.

2.2 Optimal Strategies in Mixed-Integer Optimization

When dealing with integer variables we address the following problem

$$\begin{aligned} &\text{minimize} && f(\theta, x) \\ &\text{subject to} && g(\theta, x) \leq 0 \\ &&& x_{\mathcal{I}} \in \mathbf{Z}^d. \end{aligned} \tag{5}$$

where $\mathcal{I}$ is the set of indices for the variables constrained to be integer and $|\mathcal{I}| = d$.

Tight constraints. In this case the set of tight constraints does not uniquely define the optimal solution because of the integrality of some components of x. However, when we fix the integer variables to their optimal values $x^\star_{\mathcal{I}}(\theta)$, (**??**) becomes a continuous convex optimization problem of the form

$$\begin{array}{ll} \text{minimize} & f(\theta, x) \\ \text{subject to} & g(\theta, x) \leq 0 \\ & x_{\mathcal{I}} = x^\star_{\mathcal{I}}(\theta). \end{array} \tag{6}$$

Note that the optimal cost of (**??**) and (**??**) are the same, however, optimal solutions may be different as there may be alternative optima. After fixing the integer variables, the tight constraints of problem (**??**) uniquely define an optimal solution.

Solution strategy. For this class of problems the strategy corresponds to a tuple containing the index of tight constraints of the continuous reformulation (**??**) and the optimal value of the integer variables, *i.e.*, $s(\theta) = (\mathcal{T}(\theta), x^\star_{\mathcal{I}}(\theta))$. Compared to continuous problems, we must also include the value of the integer variables to recover the optimal solution $x^\star(\theta)$.

Given the optimal strategy, problem (**??**) corresponds to solving the continuous problem

$$\begin{array}{lll} \text{minimize} & f(\theta, x) & \\ \text{subject to} & g_i(\theta, x) \leq 0, & \forall i \in \mathcal{T}(\theta) \\ & x_{\mathcal{I}} = x^\star_{\mathcal{I}}(\theta). & \end{array} \tag{7}$$

Solving this problem is much less computationally demanding than (**??**) because it is continuous, convex and has smaller number of constraints, especially when $n \ll m$. As for the continuous case (**??**), the components g_i that are linear in x can be enforced as equalities further simplifying (**??**) while preserving its convexity.

Similarly to Section **??**, in case of mixed-integer linear optimization problem (MILO) and MIQO when the cost f is linear or quadratic and the constraints g are all linear, the solution of (**??**) corresponds to solving a linear system of equations defined by the KKT conditions (**?**, Section 10.2). This means that we can solve these problems online without needing any optimization solver (**?**).

The knapsack problem

Consider the knapsack problem

$$\begin{array}{ll} \text{maximize} & c^T x \\ \text{subject to} & a^T x \leq b \\ & 0 \leq x \leq u \\ & x \in \mathbf{Z}^n, \end{array} \tag{8}$$

with $n = 10$. The decision variables are $x = (x_1, \ldots, x_{10})$ indicating the quantity to pick for each item $i = 1, \ldots, 10$. We chose the cost vector $c = (0.42, 0.72, 0, 0.3, 0.15, 0.09, 0.19, 0.35, 0.4, 0.54)$. The knapsack capacity is $b = 5$. The weights $a = (a_1, \ldots, a_{10})$ and the maximum order quantity $u = (u_1, \ldots, u_{10})$ are our parameters. We assume that a is in a ball $B(a_0, r_0)$ and $u \in B(u_0, r_0)$ where $u_0 = a_0 = (2, 2, \ldots, 2)$ and $r_0 = 1$.

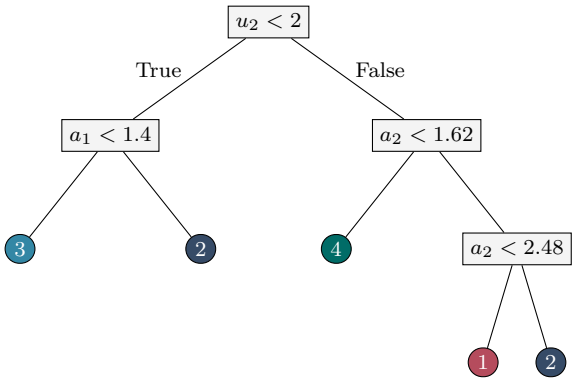

Fig. 3 Example knapsack decision strategies. *Strategy 1*: $x_i = 0$ for $i \neq 2$ and $x_2 = 2$. *Strategy 2*: $x_i = 0$ for $i \neq 2, 10$ and $x_2 = x_{10} = 1$. *Strategy 3*: $x_i = 0$ for $i \neq 1, 2$ and $x_1 = 2$ and $x_2 = 1$. *Strategy 4*: $x_i = 0$ for $i \neq 2, 10$ and $x_2 = 2$ and $x_{10} = 1$.

With this setup we obtain the solution strategy outlined in Figure **??**. For this problem each strategy uniquely identifies the integer variables and is straightforward to analyze. For instance, the left part of the tree outlines what happens if the upper bound u_2 is strictly less than 2. Moreover, if the weight $a_1 < 1.4$, then it is easier to include x_1 in the knapsack solution and for this reason Strategy 3 has $x_1 = 2$ and $x_2 = 1$. On the contrary, if $a_1 \geq 1.4$, then Strategy 2 applies $x_1 = 0$ and $x_2 = x_{10} = 1$. Even though all these rules are simple, they capture the complexity of a hard combinatorial problem for the parameters that affect it. Note that only a few parameters affect the strategy selection, *i.e.*, u_2, a_1 and a_2, while the others are not relevant for this class of problem instances that have $a \in B(a_0, r_0)$ and $u \in B(u_0, r_0)$.

Supplier selection

Consider the problem of selecting $n = 5$ suppliers in order to satisfy a known demand d. For each supplier i we have a per-unit cost $c = (0.42, 0.72, 0, 0.3, 0.15)$ and a maximum quantity to order $m = (1.09, 1.19, 1.35, 1.4, 1.54)$, while $\gamma = 0.1$. We are interested in understanding the optimal strategy as a function of the demand d and the supplier delivery times τ_i, which are the problem parameters. The optimization problem is as follows:

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{i=1}^{n} c_i u_i + \gamma \max_i(\{\tau_i x_i\}) \\ \text{subject to} & \displaystyle\sum_{i=1}^{n} u_i \geq d \\ & 0 \leq u_i \leq x_i m_i \\ & x \in \{0,1\}^n, u_i \in \mathbf{R}. \end{array} \tag{9}$$

Specifically, we are interested in $d \in [1, 3]$ and $\tau = (\tau_1, \ldots, \tau_5) \in B(\tau_0, r_0)$ with the center of the ball being $\tau_0 = (2, 3, 2.5, 5, 1)$ and radius $r_0 = 0.5$. With these parameters we obtain the solution described in Figure **??**. As before, the decision rules are simple and the solution strategy is intuitive and interpretable.

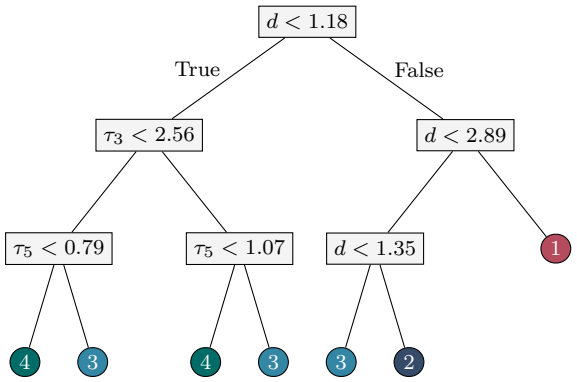

Fig. 4 Example vendor decision strategies using OCT. *Strategy 1*: $x = (1, 0, 1, 0, 1)$, $u_2 = u_4 = 0$. $u_3 = m_3, u_5 = m_5$ and u_1 is free. *Strategy 2*: $x = (0, 0, 1, 0, 1)$. $u_1 = u_2 = u_4 = 0$ and $u_3 = m_3$. u_5 is free. *Strategy 3*: $x = (0, 0, 1, 0, 0)$. $u_1 = u_2 = u_4 = u_5 = 0$ and u_3 is free. *Strategy 4*: $x = (0, 0, 0, 0, 1)$. $u_1 = u_2 = u_3 = u_4 = 0$. u_5 is free.

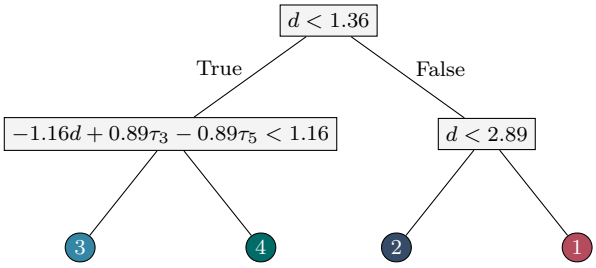

Fig. 5 Example vendor decision strategies using OCT-H. The strategies are the same as in Figure **??** but accuracy is the same or higher and the tree depth is reduced. Smaller tree depth can sometimes help in interpreting the classification.

In Figure **??**, we show the tree that OCT-H gave for this example. Even though the accuracy of the OCT in Figure **??** is 100%, the OCT-H depth is smaller which can sometimes make some classification tasks more interpretable despite the multiple coefficients on the hyperplanes.

3 Machine Learning

In this section, we introduce the core part of our approach: learning the mapping from parameters to optimal strategies. After the training, the mapping will replace the core part of standard optimization algorithms – the optimal strategy search – with a multiclass classification problem where each strategy corresponds to a class label.

3.1 Multiclass Classifier

We would like to solve a multiclass classification problem with data (θ_i, s_i), $i = 1, \ldots, N$ where $\theta_i \in \mathbf{R}^p$ are the parameters and $s_i \in \mathcal{S}$ the corresponding labels

identifying the optimal strategies. $\mathcal{S}$ represents the set of strategies of cardinality $|\mathcal{S}| = M$.

In this work we apply two supervised learning techniques for multiclass classification to compare their predictive performance: optimal classification trees (OCTs, OCT-Hs) (**??**) and neural networks (NNs) (**??**). As we mentioned earlier OCTs, OCT-Hs are interpretable and can be described using simple rules as in the examples in the previous section, while NNs are not interpretable since they represent a composition of multiple nonlinear functions. A more detailed description of OCTs and OCT-Hs can be found in Appendix **??** and of NNs can be found in Appendix **??**.

3.2 Strategies Exploration

In this section, we estimate how likely it is to find a new parameter θ whose optimal strategy does not lie among the ones we have already seen. If it is unlikely to find new strategies, then we can be sure that our classification problem includes all the possible strategies arising in practice. Otherwise, we must collect more data to have a more representative classification problem.

Estimating the probability of finding unseen strategies. Given N independent samples $\Theta_N = \{\theta_1, \ldots, \theta_N\}$ drawn from an unknown discrete distribution $\mathcal{P}$ with the corresponding strategies $(s(\theta_1), \ldots, s(\theta_N))$, we find M unique strategies $\mathcal{S}(\Theta_N) = \{s_1, \ldots, s_M\}$. We are interested in bounding the probability of finding unseen strategies

$$\mathbf{P}(\theta_{N+1} \in \mathbf{R}^p \mid s(\theta_{N+1}) \notin \mathcal{S}(\Theta_N)), \tag{10}$$

with confidence at least $1 - \beta$ where $\beta > 0$.

This problem started from the seminal work by Turing and Good (**?**) in the context of decrypting the Enigma codes during World War II. The Enigma machine was a German navy encryption device used for secret military communications. Part of Enigma's encryption key was a three letter sequence (a word) selected from a book containing all the possible ones in random order. The number of possible words was so large that it was impossible to test all the combinations with the computing power available at that time. In order to decrypt the Enigma machine without testing all the possible words, Turing wanted to check only a subset of them while estimating that the likelihood of finding a new unseen word was low. This is how the Good-Turing estimator was developed. It was a fundamental step towards the Enigma machine decryption which is believed to have shortened World War II of at least two years (**?**). In addition, this class of estimators have become standard in a wide range of natural language processing applications.

Good-Turing estimator. Let N_r be the number of strategies that appeared exactly r times in $(s(\theta_1), \ldots, s(\theta_N))$. The Good-Turing estimator for the probability of having an unseen strategy is given by **?**

$$G = N_1/N, \tag{11}$$

which corresponds to the ratio between the number of distinct strategies that have appeared exactly once, over the total number of samples. Despite the elegant

result, Good and Turing did not provide a convergence analysis of this estimator for finite samples. Only few decades later the first theoretical work on that topic appeared in (**?**). Using **?**'s inequality the authors derived a high probability confidence interval. That result can be directly applied to our problem with the following theorem.

Theorem 1 (Missing strategies bound) *The probability of encountering a parameter θ_{N+1} corresponding to an unseen strategy $s(\theta_{N+1})$ satisfies with confidence at least $1-\beta$*

$$\mathbf{P}(\theta_{N+1} \in \mathbf{R}^p \mid s(\theta_{N+1}) \notin \mathcal{S}(\Theta_N)) \leq G + c\sqrt{(1/N)\ln(3/\beta)},$$

where G corresponds to the Good-Turing estimator (**??**) *and $c = (2\sqrt{2}+\sqrt{3})$.*

Proof Proof of Theorem **??** The result follows directly from (**?**, Theorem 9).

Exploration algorithm. Given the bound in Theorem **??** we construct Algorithm **??** to compute the different strategies appearing in our problem.

Algorithm 1 Strategies exploration

given $\epsilon, \beta, \Theta = \emptyset, \mathcal{S} = \emptyset, u = \infty$
for $k = 1, \ldots,$ **do**
  Sample θ_k and compute $s(\theta_k)$ ▷ Sample parameter and strategy.
  $\Theta \leftarrow \Theta \cup \{\theta_k\}$ ▷ Update set of samples.
  if $s(\theta_k) \notin \mathcal{S}$ **then**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{s(\theta_k)\}$ ▷ Update strategy set if new strategy found
  if $G + c\sqrt{(1/k)\ln(3/\beta)} \leq \epsilon$ **then** ▷ Break if bound less than ϵ
    break
return $k, \Theta, \mathcal{S}$

The algorithm keeps on sampling until we encounter a large enough set of distinct strategies. It terminates when the probability of encountering a parameter with an unseen optimal strategy is less than ϵ with confidence at least $1-\beta$. Note that in practice, instead of sampling one point per iteration k, it is more convenient to sample more points to avoid having to recompute the class frequencies which becomes computationally intensive with several thousands of samples and hundreds of classes. Algorithm **??** imposes no assumptions on the optimization problem nor the data distribution.

4 Machine Learning Optimizer

We implemented our method in the Python software tool MLOPT (Machine Learning Optimizer). It is integrated with CVXPY (**?**) to formulate and solve the problems. CVXPY allows the user to easily define mixed-integer convex optimization problems performing all the required reformulations needed by the optimizers while keeping track of the original constraints and variables. This makes it ideal for identifying which constraints are tight or not at the optimal solution.

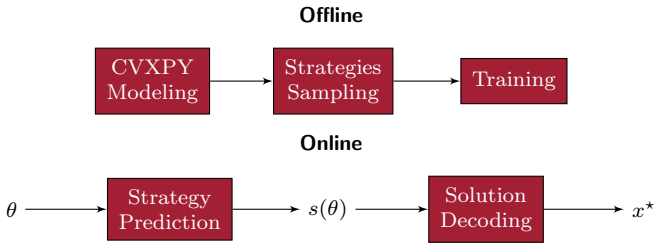

Fig. 6 Algorithm implementation

We used the Gurobi Optimizer (**?**) to find the tight constraints because it provides a good tradeoff between solution accuracy and computation time. Note that from Section **??**, in case of LO, MILO, QO and MIQO when the cost f is linear or quadratic and the constraints g are all linear, the online solution corresponds to solving a linear system of equations defined by the KKT conditions (**?**, Section 10.2) on the reduced subproblem. This means that we can solve those parametric optimization problems without the need to apply any optimization solver.

MLOPT performs the iterative sampling procedure outlined in Section **??** to obtain the strategies required for the classification task. We sample 5000 new points at each iteration and compute their strategies until the Good Turing estimate is below $\epsilon_{\text{GT}} = 0.005$. The strategy computation is fully parallelized across samples.

We interfaced MLOPT to the machine learning libraries OptimalTrees.jl (**?**) on multi-core CPUs and PyTorch (**?**) for both CPUs and GPUs. In the training phase we automatically tune the predictor parameters by splitting the data points in 90%/10% training/validation. We tune the OCTs and OCT-Hs with maximal depth for values $5, 10, 15$ and minimum bucket size for values $1, 5, 10$. For the NNs we validate the stochastic gradient descent learning rate for values $0.001, 0.01, 0.1$, batch size for values $32, 128$ and number of epochs for values $50, 100$. We described the complete algorithm in Figure **??**.

5 Computational Benchmarks

In this section, we test our approach on multiple examples from continuous and mixed-integer optimization. We benchmarked the predictive performance of OCTs, OCT-Hs (see Appendix **??**) and NNs (see Appendix **??**) depending on the problem type and size. We executed the numerical tests on a Dell R730 cluster with 28 Intel E5-2680 CPUs with a total of 256GB RAM and a Nvidia Tesla K80 GPU. To facilitate the data collection, we generated the training samples from distributions on intervals or on hyperballs around specified points. In this way the distributions are unimodal and therefore closer to the ones encountered in practice but we could have chosen other distributions such as multimodal ones. We evaluated the performance metrics on 100 unseen samples drawn from the same distribution of θ.

Infeasibility and suboptimality. After the learning phase, we compared the predicted solution $\hat{x}_i^\star$ to the optimal one $x_i^\star$ obtained by solving the instances from scratch. Given a parameter θ_i, we say that the predicted solution is infeasible if the constraints are violated more than a predefined tolerance $\epsilon_{\text{inf}} = 10^{-3}$ according to the infeasibility metric

$$p(\hat{x}_i^\star) = \|(g(\theta_i, \hat{x}_i^\star))_+\|_2 / r(\theta_i, \hat{x}_i^\star),$$

where $r(\theta, x)$ normalizes the violation depending on the size of the summands of g. For example, if $g(\theta, x) = A(\theta)x - b$, then $r(\theta, x) = \max(\|A(\theta)x\|_2, \|b\|_2)$. If the predicted solution $\bar{x}_i$ is feasible, we define its suboptimality as

$$d(\hat{x}_i^\star) = (f(\hat{x}_i^\star) - f(x_i^\star))/|f(x_i^\star)|.$$

Note that $d(\hat{x}_i^\star) \geq 0$ by construction. We consider a predicted solution to be accurate if it is feasible and if the suboptimality is less than the tolerance $\epsilon_{\text{sub}} = 10^{-3}$. For each instance we report the maximum infeasibility $\max_i p(\hat{x}_i^\star)$ and the maximum suboptimality $\max_i d(\hat{x}_i^\star)$. Note that for notation ease and to simplify the max operation, if a point is infeasible we consider its suboptimality 0 and ignore it.

Predicting the optimal strategy. Once the learning phase is completed, the predictor outputs the three (3) most likely optimal strategies and picks the best one according to infeasibility and suboptimality after solving the associated reduced problems.

5.1 Transportation Optimization

Consider a standard transportation problem with n warehouses and m retail stores. Let x_{ij} be the quantity we ship from warehouse i to store j. s_i denotes the supply for warehouse i and d_j the demand from store j. We define the cost of transporting a unit from warehouse i to store j as c_{ij}. The optimization problem can be formulated as follows:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} \\
\text{subject to} \quad & \sum_{j=1}^{m} x_{ij} \leq s_i, \quad \forall i \\
& \sum_{i=1}^{n} x_{ij} \geq d_j \quad \forall j \\
& x_{ij} \geq 0 \qquad \forall i, j.
\end{aligned}$$

The first constraint ensures we respect the supply for each warehouse i. The second constraint enforces the sum of all the shipments to match at least the demand for store j. Finally, the quantity of shipped product has to be nonnegative. Our learning parameters are the demands d_j.

Table 1 Transportation benchmarks.

n	m	n_{var}	n_{con}	learner	N	GT	$\lvert\mathcal{S}\rvert$	acc [%]	$\max_i p(\bar{x}_i)$	$\max_i d(\bar{x}_i)$
20	20	400	440	NN	10000	1.00×10^{-4}	10	100.00	8.88×10^{-5}	0.00
20	20	400	440	OCT	10000	1.00×10^{-4}	10	100.00	8.88×10^{-5}	0.00
20	20	400	440	OCT-H	10000	1.00×10^{-4}	10	100.00	8.88×10^{-5}	0.00
20	10	200	230	NN	10000	4.00×10^{-4}	9	100.00	0.00	0.00
20	10	200	230	OCT	10000	4.00×10^{-4}	9	99.00	1.00	0.00
20	10	200	230	OCT-H	10000	4.00×10^{-4}	9	100.00	0.00	0.00
40	40	1600	1680	NN	10000	3.00×10^{-4}	9	100.00	1.29×10^{-16}	0.00
40	40	1600	1680	OCT	10000	3.00×10^{-4}	9	100.00	1.29×10^{-16}	0.00
40	40	1600	1680	OCT-H	10000	3.00×10^{-4}	9	100.00	1.29×10^{-16}	0.00
40	20	800	860	NN	10000	1.00×10^{-4}	7	100.00	0.00	0.00
40	20	800	860	OCT	10000	1.00×10^{-4}	7	100.00	0.00	0.00
40	20	800	860	OCT-H	10000	1.00×10^{-4}	7	100.00	0.00	0.00
60	60	3600	3720	NN	10000	6.00×10^{-4}	17	100.00	1.86×10^{-5}	0.00
60	60	3600	3720	OCT	10000	6.00×10^{-4}	17	100.00	1.59×10^{-16}	0.00
60	60	3600	3720	OCT-H	10000	6.00×10^{-4}	17	100.00	1.59×10^{-16}	0.00
60	30	1800	1890	NN	10000	1.00×10^{-4}	13	100.00	0.00	0.00
60	30	1800	1890	OCT	10000	1.00×10^{-4}	13	100.00	0.00	0.00
60	30	1800	1890	OCT-H	10000	1.00×10^{-4}	13	100.00	0.00	0.00
80	80	6400	6560	NN	15000	1.60×10^{-3}	93	92.00	7.34×10^{-3}	0.00
80	80	6400	6560	OCT	15000	1.60×10^{-3}	93	73.00	1.00	0.00
80	80	6400	6560	OCT-H	15000	1.60×10^{-3}	93	96.00	1.00	0.00
80	40	3200	3320	NN	10000	0.00	7	100.00	0.00	0.00
80	40	3200	3320	OCT	10000	0.00	7	100.00	0.00	0.00
80	40	3200	3320	OCT-H	10000	0.00	7	100.00	0.00	0.00

Problem instances We generate the transportation problem instances by varying the number of warehouses n and stores m. The cost vectors c_i are distributed as $\mathcal{U}(0,5)$ and the supplies s_i as $\mathcal{U}(3,13)$. The parameter vector $d = (d_1, \ldots, d_m)$ was sampled from a uniform distribution within the ball $B(\overline{d}, 0.75)$ with center $\overline{d} \sim \mathcal{N}(3,1)$.

Results. The results appear in Table **??**. Independently from the problem instances the prediction accuracy is very high for both optimal trees and neural networks. Note that even though we solve problems with thousands of variables and constraints, the number of strategies is always within few tens or less. Moreover, both NNs and OCT-Hs provide optimal and feasible solutions with almost perfect accuracy. OCT performed comparably, except in one instance.

5.2 Portfolio Optimization

Consider the problem of allocating assets to minimize the risk adjusted return considered in **?** who formulated it as a QO

$$\begin{array}{ll} \text{maximize} & \mu^T x - \gamma(x^T \Sigma x) \\ \text{subject to} & \mathbf{1}^T x = 1 \\ & x \geq 0, \end{array}$$

where the variable $x \in \mathbf{R}^n$ represents the investments to make.

Table 2 Continuous portfolio benchmarks.

n	p	n_{con}	learner	N	GT	$\lvert\mathcal{S}\rvert$	acc [%]	$\max_i p(\bar{x}_i)$	$\max_i d(\bar{x}_i)$
100	10	101	NN	10000	4.00×10^{-4}	12	100.00	1.85×10^{-14}	1.82×10^{-6}
100	10	101	OCT	10000	4.00×10^{-4}	12	100.00	5.89×10^{-5}	1.53×10^{-12}
100	10	101	OCT-H	10000	4.00×10^{-4}	12	100.00	5.89×10^{-5}	1.53×10^{-12}
200	20	201	NN	10000	0.00	2	100.00	7.53×10^{-5}	4.21×10^{-11}
200	20	201	OCT	10000	0.00	2	100.00	7.53×10^{-5}	4.21×10^{-11}
200	20	201	OCT-H	10000	0.00	2	100.00	7.53×10^{-5}	4.21×10^{-11}
300	30	301	NN	10000	2.00×10^{-4}	14	100.00	2.69×10^{-14}	1.02×10^{-5}
300	30	301	OCT	10000	2.00×10^{-4}	14	100.00	2.69×10^{-14}	1.02×10^{-5}
300	30	301	OCT-H	10000	2.00×10^{-4}	14	100.00	9.13×10^{-5}	7.10×10^{-6}
400	40	401	NN	10000	0.00	2	100.00	3.02×10^{-14}	2.24×10^{-12}
400	40	401	OCT	10000	0.00	2	100.00	3.02×10^{-14}	2.24×10^{-12}
400	40	401	OCT-H	10000	0.00	2	100.00	3.02×10^{-14}	2.24×10^{-12}
500	50	501	NN	10000	0.00	4	100.00	6.85×10^{-5}	4.42×10^{-11}
500	50	501	OCT	10000	0.00	4	100.00	6.85×10^{-5}	4.42×10^{-11}
500	50	501	OCT-H	10000	0.00	4	100.00	6.85×10^{-5}	4.42×10^{-11}

Our learning parameter is the vector of expected returns $\mu \in \mathbf{R}^n$. In addition we denote the risk aversion coefficient as $\gamma > 0$, and the covariance matrix for the risk model as $\Sigma \in \mathbf{S}^n_+$. Σ is assumed to be

$$\Sigma = FF^T + D,$$

where $F \in \mathbf{R}^{n\times p}$ is the factor loading matrix and $D \in \mathbf{R}^{n\times n}$ is a diagonal matrix describing the asset-specific risk.

Problem instances. We generated portfolio instances for different number of factors p and assets n. We chose the elements of F with 50 % nonzeros and $F_{ij} \sim \mathcal{N}(0,1)$. The diagonal matrix D has elements $D_{ii} \sim \mathcal{U}(0,\sqrt{p})$. The return vector parameters were randomly sampled from a uniform distribution within the ball $B(\overline{\mu}, 0.15)$ with $\overline{\mu} \sim \mathcal{N}(0,1)$. The risk-aversion coefficient is $\gamma = 1$.

Results. Results are shown in Table **??**. The prediction accuracy for OCTs, OCT-Hs and NNs is 100%, the number of strategies is less than 15, and the infeasibility and suboptimality are very low.

5.3 Facility location

Let $i \in I$ the set of possible locations for facilities such as factories or warehouses. We denote as $j \in J$ the set of delivery locations. The cost of transporting a unit of goods from facility i to location j is c_{ij}. The construction cost for building facility i is f_i. For each location j, we define the demand d_j. The capacity of facility i is s_i. The main goal of this problem is to find the best tradeoff between transportation

costs versus construction costs. We can write the model as a MILO

$$\begin{array}{ll} \text{minmimize} & \displaystyle\sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij} + \sum_{i\in I} f_i y_i \\ \text{subject to} & \displaystyle\sum_{i\in I} x_{ij} \geq d_j, \quad \forall j \in J \\ & \displaystyle\sum_{j\in J} x_{ij} \leq s_i y_i, \quad \forall i \in I \\ & x_{ij} \geq 0 \\ & y_i \in \{0,1\}. \end{array} \tag{12}$$

The decision variables x_{ij} describe the amount of goods sent from facility i to location j. The binary variables y_i determine if we build facility i or not.

Problem instances. We generated the facility location instances for different values of facilities n and warehouses m. The costs c_{ij} are sampled from a uniform distribution $\mathcal{U}(0,1)$ and f_i from $\mathcal{U}(0,10)$. We chose capacities s_i as $\mathcal{U}(8,18)$ to ensure the problem is always feasible. The demands parameters $d = (d_1,\ldots,d_m)$ were sampled from a uniform distributions within the ball $B(\overline{d}, 0.25)$ with $\overline{d} \sim \mathcal{N}(3,1)$.

Results. Table **??** shows the benchmark examples. We notice a predictive performance degradation when the number of strategies is higher. We suspect this is related to more samples needed to properly train the models. However, the infeasibility and suboptimality measures are low even when the prediction is not correct. As before OCTs, OCT-Hs and NNs are comparable except in one instance where OCT-H significantly outperformed.

5.4 Hybrid Vehicle Control

Consider the hybrid-vehicle control problem in (**?**, Section 3.2). The model consists of a battery, an electric motor/generator and an engine. We assume to know the demanded power P_t^{des} over the horizon $t = 0,\ldots,T-1$. The goal is to plan the battery and the engine power outputs P_t^{batt} and P_t^{eng} so that they match at least the demand,

$$P_t^{\text{batt}} + P_t^{\text{eng}} \geq P_t^{\text{des}}.$$

The status of the battery is modeled with the internal energy E_t evolving as

$$E_{t+1} = E_t - \tau P_t^{\text{batt}},$$

where $\tau > 0$ is the time interval discretization. The battery capacity is limited by $E^{\max}$ and its initial value is E_{init}.

We now model the cost function. We penalize the terminal energy state of the battery with the function

$$g(E) = \eta(E^{\max} - E)^2.$$

with $\eta \geq 0$. At each time t we model the on-off state of the engine with the binary variable z_t. When the engine is off ($z_t = 0$) we do not consume any energy, thus we have $0 \leq P_t^{\text{eng}} \leq P^{\max} z_t$. When the engine is on ($z_t = 1$) it consumes

Table 3 Facility location benchmarks.

n	m	n_{var}	n_{con}	learner	N	GT	$\lvert\mathcal{S}\rvert$	acc [%]	$\max_i p(\bar{x}_i)$	$\max_i d(\bar{x}_i)$
20	20	420	480	NN	10000	0.00	3	100.00	1.14×10^{-16}	1.19×10^{-7}
20	20	420	480	OCT	10000	0.00	3	100.00	1.14×10^{-16}	8.56×10^{-8}
20	20	420	480	OCT-H	10000	0.00	3	100.00	1.14×10^{-16}	8.57×10^{-8}
20	10	220	270	NN	10000	0.00	1	100.00	0.00	1.22×10^{-16}
20	10	220	270	OCT	10000	0.00	1	100.00	0.00	0.00
20	10	220	270	OCT-H	10000	0.00	1	100.00	0.00	1.22×10^{-16}
40	40	1640	1760	NN	10000	0.00	12	95.00	1.53×10^{-2}	2.02×10^{-16}
40	40	1640	1760	OCT	10000	0.00	12	93.00	2.01×10^{-2}	2.02×10^{-16}
40	40	1640	1760	OCT-H	10000	0.00	12	99.00	2.78×10^{-3}	2.02×10^{-16}
40	20	840	940	NN	10000	0.00	2	100.00	2.54×10^{-16}	4.86×10^{-8}
40	20	840	940	OCT	10000	0.00	2	100.00	2.54×10^{-16}	4.58×10^{-8}
40	20	840	940	OCT-H	10000	0.00	2	100.00	2.54×10^{-16}	1.09×10^{-8}
60	60	3660	3840	NN	10000	8.00×10^{-4}	28	100.00	8.28×10^{-5}	2.25×10^{-16}
60	60	3660	3840	OCT	10000	8.00×10^{-4}	28	100.00	2.74×10^{-16}	2.25×10^{-16}
60	60	3660	3840	OCT-H	10000	8.00×10^{-4}	28	100.00	2.74×10^{-16}	2.25×10^{-16}
60	30	1860	2010	NN	10000	0.00	2	100.00	1.90×10^{-16}	1.40×10^{-16}
60	30	1860	2010	OCT	10000	0.00	2	100.00	1.90×10^{-16}	1.40×10^{-16}
60	30	1860	2010	OCT-H	10000	0.00	2	100.00	1.90×10^{-16}	1.40×10^{-16}
80	80	6480	6720	NN	10000	5.00×10^{-4}	25	64.00	2.72×10^{-2}	5.63×10^{-3}
80	80	6480	6720	OCT	10000	5.00×10^{-4}	25	66.00	2.72×10^{-2}	5.45×10^{-3}
80	80	6480	6720	OCT-H	10000	5.00×10^{-4}	25	88.00	5.46×10^{-3}	5.39×10^{-3}
80	40	3280	3480	NN	10000	1.00×10^{-4}	10	98.00	4.27×10^{-3}	6.26×10^{-4}
80	40	3280	3480	OCT	10000	1.00×10^{-4}	10	93.00	4.41×10^{-3}	2.65×10^{-3}
80	40	3280	3480	OCT-H	10000	1.00×10^{-4}	10	99.00	3.47×10^{-16}	6.26×10^{-4}

$\alpha P_t^{\text{eng}} + \beta P_t^{\text{eng}} + \gamma$ units of fuel, with $\alpha, \beta, \gamma > 0$. We define the stage power cost as

$$f(P, z) = \alpha P^2 + \beta P + \gamma z.$$

We also introduce a cost of turning the engine on at time t as $\delta(z_t - z_{t-1})_+$.

The hybrid vehicle control problem can be formulated as a mixed integer quadratic optimization problem:

$$\begin{array}{ll}
\text{minimize} & \eta(E_T - E^{\max})^2 + \displaystyle\sum_{t=0}^{T-1} f(P_t^{\text{eng}}, z_t) + \delta(z_t - z_{t-1})_+ \\
\text{subject to} & E_{t+1} = E_t - \tau P_t^{\text{batt}}, \qquad t = 0, \ldots, T-1 \\
& 0 \le E_t \le E^{\max}, \qquad t = 0, \ldots, T \\
& E_0 = E_{\text{init}} \\
& 0 \le P_t^{\text{eng}} \le P^{\max}, \qquad t = 0, \ldots, T-1 \\
& P_t^{\text{batt}} + P_t^{\text{eng}} \ge P_t^{\text{des}}, \qquad t = 0, \ldots, T-1 \\
& z_t \in \{0, 1\}, \qquad t = 0, \ldots, T-1.
\end{array}$$

The initial state E_0 and demanded power P_t^{des} are our parameters.

Problem instances. We generated the control instances with varying horizon length T. The discretization time interval is $\tau = 4$. We chose the cost function parameters parameters $\alpha = \beta = \gamma = 1$, and $\delta = 0.1$. The maximum charge is $E^{\max} = 50$ and the maximum power $P^{\max} = 1$. The parameter E_0 was sampled from a

Table 4 Hybrid control benchmarks.

T	n_{var}	n_{con}	learner	N	GT	$\lvert\mathcal{S}\rvert$	acc [%]	$\max_i p(\bar{x}_i)$	$\max_i d(\bar{x}_i)$
5	21	43	NN	10000	2.00×10^{-4}	10	99.00	9.43×10^{-15}	2.84×10^{-3}
5	21	43	OCT	10000	2.00×10^{-4}	10	97.00	9.43×10^{-15}	9.83×10^{-2}
5	21	43	OCT-H	10000	2.00×10^{-4}	10	100.00	9.43×10^{-15}	1.20×10^{-15}
10	41	83	NN	10000	3.00×10^{-4}	14	100.00	1.17×10^{-14}	3.81×10^{-15}
10	41	83	OCT	10000	3.00×10^{-4}	14	100.00	1.17×10^{-14}	3.53×10^{-15}
10	41	83	OCT-H	10000	3.00×10^{-4}	14	100.00	1.17×10^{-14}	3.53×10^{-15}
20	81	163	NN	10000	6.00×10^{-4}	14	99.00	2.92×10^{-4}	6.26×10^{-15}
20	81	163	OCT	10000	6.00×10^{-4}	14	99.00	1.51×10^{-4}	6.33×10^{-15}
20	81	163	OCT-H	10000	6.00×10^{-4}	14	100.00	6.18×10^{-5}	6.33×10^{-15}
30	121	243	NN	10000	2.00×10^{-4}	22	94.00	2.34×10^{-3}	9.86×10^{-4}
30	121	243	OCT	10000	2.00×10^{-4}	22	83.00	6.42×10^{-3}	5.31×10^{-4}
30	121	243	OCT-H	10000	2.00×10^{-4}	22	94.00	1.55×10^{-13}	5.23×10^{-4}
40	161	323	NN	10000	1.00×10^{-4}	34	91.00	3.78×10^{-3}	4.24×10^{-4}
40	161	323	OCT	10000	1.00×10^{-4}	34	65.00	2.59×10^{-2}	1.97×10^{-3}
40	161	323	OCT-H	10000	1.00×10^{-4}	34	80.00	1.07×10^{-2}	6.19×10^{-4}

uniform distribution within the ball $B(40, 0.5)$. The demand P^{des} also comes from a uniform distribution within the ball $B(\overline{P}^{\text{des}}, 0.5)$ where

$$\begin{aligned}\overline{P}_t^{\text{des}} = (&0.05, 0.30, 0.55, 0.80, 1.05, 1.30, 1.55, 1.80, 1.95, 1.70, 1.45, 1.20, 1.02,\\ &1.12, 1.22, 1.32, 1.42, 1.52, 1.62, 1.72, 1.73, 1.38, 1.03, 0.68, 0.33, -0.02,\\ &-0.37, -0.72, -0.94, -0.64, -0.34, -0.04, 0.18, 0.08, -0.02, -0.12,\\ &0.22, -0.32, -0.42, -0.52)\end{aligned}$$

is the desired power in **?**. Depending on the horizon length T we choose only the first T elements of $\overline{P}^{\text{des}}$ to sample P^{des}.

Results. We present the results in Table **??**. Also in this case the number of strategies $|\mathcal{S}|$ is much less than the number of variables or the worst case number of control input combinations. The maximum infeasibility and suboptimality are low even when the predictions are not exact and the performance of OCT-Hs and NNs is comparable, except on one instance where NNs were stronger. The performance of OCT was weaker especially as the dimension of the problem increased.

6 Conclusions

We introduced the idea that using OCTs and OCT-Hs we obtain insight on the strategy of the optimal solution in many optimization problems as a function of key parameters. In this way, optimization is not a black box anymore, but rather it has a voice, *i.e.*, we are able to provide insights on the logic behind the optimal solution. The class of optimization problems that these ideas apply to is rather broad since it includes any continuous and mixed-integer convex optimization problems without any assumption on the parameters dependency. The accuracy of our approach is in the 90%-100%, while even when it does not provide the optimal solution the degree of suboptimality or infeasibility is extremely low. Comparisons on several examples show that the out-of-sample strategy prediction accuracy of OCT-Hs is

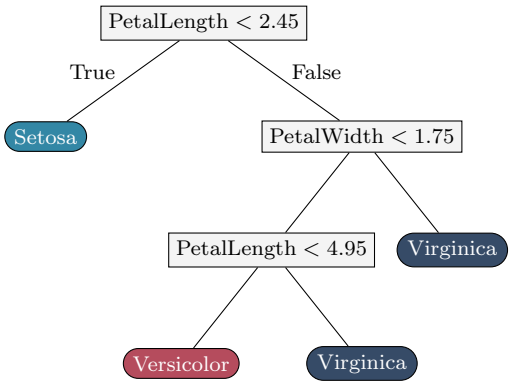

Fig. 7 Example OCT for the Iris dataset **?**.

comparable to the NNs. Therefore our method provides a reliable and insightful understanding of optimal solutions in a broad class of optimization problems.

A Optimal Classification Trees

OCTs and OCT-Hs developed by **??** are a recently proposed generalization of classification and regression trees (CARTs) developed by **?** that construct decision trees that are near optimal with significantly higher prediction accuracy while retaining their interpretability. **?** have shown that given a NN (feedforward, convolutional and recurrent), we can construct an OCT-H that has the same in sample accuracy, that is OCT-Hs are at least as powerful as NNs. The constructions can sometimes generate OCT-Hs with high depth. However, they also report computational results that show that OCT-Hs and NNs have comparable performance in practice even for depths of OCT-Hs below 10.

Architecture. OCTs recursively partition the feature space $\mathbf{R}^p$ to construct hierarchical disjoint regions. A tree can be defined as a set of nodes $t \in \mathcal{T}$ of two types $\mathcal{T} = \mathcal{T}_B \cup \mathcal{T}_L$:

Branch nodes Nodes $t \in \mathcal{T}_B$ at the tree branches describe a split of the form $a_t^T \theta < b_t$ where $a_t \in \mathbf{R}^p$ and $b_t \in \mathbf{R}$. They partition the space in two subsets: the points on the left branch satisfying the inequality and the remaining ones points on the right branch. If splits involve a single variable we denote them as *parallel* and we refer to the tree as optimal classification tree (OCT). This is achieved by enforcing all components of a_t to be all 0 except from one. Otherwise, if the components of a_t can be freely nonzero, we denote the splits as *hyperplanes* and we refer to the tree as optimal classification tree with-hyperplanes (OCT-H).

Leaf nodes Nodes $t \in \mathcal{T}_L$ at the tree leaves make a class prediction for each point falling into that node.

An example OCT for the Iris dataset appears in Figure **??** (**?**). For each new data point it is straightforward to follow which hyperplanes are satisfied and to make a prediction. This characteristic makes OCTs and OCT-Hs highly interpretable. Note that the level of interpretability of the resulting trees can be also tuned by changing minimum sparsity of a_t. The two extremes are maximum sparsity OCTs and minimum sparsity OCT-Hs but we can specify anything in between.

Learning. With the latest computational advances in MIO, **?** were able to exactly formulate the tree training as a MIO problem and solve it in a reasonable amount of time for problem sizes arising in real-world applications.

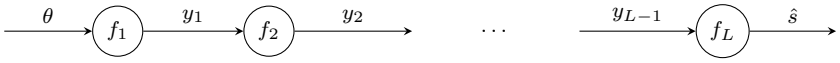

Fig. 8 Example Feedforward Neural Network with functions $f_i,\ i = 1, \ldots, L$ defined in (**??**).

The OCT cost function is a tradeoff between the misclassification error at each leaf and the tree complexity

$$\mathcal{L}_{OCT} = \sum_{t \in \mathcal{T}_L} L_t + \alpha \sum_{t \in \mathcal{T}_B} \|a_t\|_1,$$

where the L_t is the misclassification error at node t and the second term represents the complexity of the tree measured as the sum the norm of the hyperplane coefficients in all the splits. The parameter $\alpha > 0$ regulates the tradeoff. For more details about the cost function and the constraints of the problem, we refer the reader to (**?**, Section 2.2, Section 3.1).

Bertsimas and Dunn apply a local search method (**?**, Section 2.3) that manages to solve OCT problems for realistic sizes in fraction of the time an off-the-shelf optimization solver would take. The algorithm proposed iteratively improves and refines the current tree until a local minimum is reached. By repeating this search from different random initialization points the authors compute several local minima and then take the best one as the resulting tree. This heuristic showed remarkable performance both in terms of computation time and quality of the resulting tree becoming the algorithm included in the OptimalTrees.jl Julia package (**?**).

B Neural Networks

NNs have recently become one of the most prominent machine learning techniques revolutionizing fields such as speech recognition (**?**) and computer vision (**?**). The wide range of applications of these techniques recently extended to autonomous driving (**?**) and reinforcement learning (**?**). The popularity of neural networks is also due to the widely used open-source libraries learning on CPUs and GPUs coming from both academia and industry such as TensorFlow (**?**), Caffe (**?**) and PyTorch (**?**). We use feedforward neural networks which offer a good tradeoff between simplicity and accuracy without resorting to more complex architectures such as convolutional or recurrent neural networks (**?**).

Architecture. Given L layers, a neural network is a composition of functions of the form

$$\hat{s} = f_L(f_{L-1}(\ldots f_1(\theta))),$$

where each function consists of

$$y_l = f(y_{l-1}) = g(W_l y_{l-1} + b_l). \tag{13}$$

The number of nodes in each layer is n_l and corresponds to the dimension of the vector $y_l \in \mathbf{R}^{n_l}$. Layer $l = 1$ is defined as the *input* layer and $l = L$ as the *output* layer. Consequently $y_1 = \theta$ and $y_L = \hat{s}$. The linear transformation in each layer is composed of an affine transformation with parameters $W_l \in \mathbf{R}^{n_l \times n_{l-1}}$ and $b_l \in \mathbf{R}^{n_l}$. The activation function $g : \mathbf{R}^{n_l} \to \mathbf{R}^{n_l}$ models nonlinearities. We chose as activation function the rectified linear unit (ReLU) defined as

$$g(x) = \max(x, 0),$$

for all the layers $l = 1, \ldots, L-1$. Note that the max operator is intended element-wise. We chose a ReLU because on the one hand it provides sparsity to the model since it is 0 for the negative components of x and because on the other hand it does not suffer from the vanishing gradient issues of the standard sigmoid functions (**?**). An example neural network can be found in Figure **??**.

For the output layer we would like the network to output not only the predicted class, but also a normalized ranking between them according to how likely they are to be the right one. This can be achieved with a softmax activation function in the output layer defined as

$$g(x)_j = \frac{e^{x_j}}{\sum_{j=1}^{M} e^{x_j}},$$

where $j = 1, \ldots, M$ are the elements of $g(x)$. Hence, $0 \leq g(x) \leq 1$ and the predicted class is $\text{argmax}(\hat{s})$.

Learning. Before training the network, we rewrite the labels for the neural network learning using a one-hot encoding, *i.e.*, $s_i \in \mathbf{R}^M$ where M is the total number of classes and all the elements of s_i are 0 except the one corresponding to the class which is 1.

We define a smooth cost function amenable to algorithms such as gradient descent, *i.e.*, the cross-entropy loss

$$\mathcal{L}_{\text{NN}} = \sum_{i=1}^{N} -s_i^T \log(\hat{s}_i),$$

where log is intended element-wise. The cross-entropy loss $\mathcal{L}$ can also be interpreted as the distance between the predicted probability density of the labels compared to the true one.

The actual training phase consists of applying stochastic gradient descent using the derivatives of the cost function using the back-propagation rule. This method works very well in practice and provides good out-of-sample performance with short training times.

References

Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X (2015) TensorFlow: Large-scale machine learning on heterogeneous systems. URL `https://www.tensorflow.org/`, software available from tensorflow.org

Alvarez AM, Louveaux Q, Wehenkel L (2017) A machine learning-based approximation of strong branching. INFORMS Journal on Computing 29(1):185–195

Bengio Y (2009) Learning deep architectures for AI. Foundations and Trends® in Machine Learning 2(1):1–127

Bengio Y, Lodi A, Prouvost A (2018) Machine learning for combinatorial optimization: a methodological tour d'horizon. arXiv:1811.0612

Bertsimas D, Dunn J (2017) Optimal classification trees. Machine Learning 106(7):1039–1082

Bertsimas D, Dunn J (2018) Machine Learning under a Modern Optimization Lens. Dynamic Ideas Press, to appear

Bertsimas D, Stellato B (2019) Online mixed-integer optimization in milliseconds. arXiv:1907.02206

Bertsimas D, Tsitsiklis JN (1997) Introduction to Linear Optimization. Athena Scientific

Bertsimas D, Mazumder R, Sobiesk M (2018) Optimal classification and regression trees with hyperplanes are as powerful as classification and regression neural networks. (submitted for publication)

Bojarski M, Del Testa D, Dworakowski D, Firner B, Flepp B, Goyal P, Jackel LD, Monfort M, Muller U, Zhang J, Zhang X, Zhao J, Zieba K (2016) End to end learning for self-driving cars. arXiv:1604.17316

Bonami P, Lodi A, Zarpellon G (2018) Learning a classification of mixed-integer quadratic programming problems. In: van Hoeve WJ (ed) Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Springer International Publishing, Cham, pp 595–604

Boyd S, Vandenberghe L (2004) Convex Optimization. Cambridge University Press

Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and regression trees. The Wadsworth statistics/probability series, Wadsworth & Brooks/Cole Advanced Books & Software, Monterey, CA

Calafiore GC (2010) Random convex programs. SIAM Journal on Optimization 20(6):3427–3464

Copeland J (2012) Alan Turing: The codebreaker who saved 'millions of lives'. BBC News URL `https://www.bbc.com/news/technology-18419691`, [Online; posted 19-June-2012]

Dai H, Dai B, Song L (2016) Discriminative embeddings of latent variable models for structured data. In: Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, JMLR.org, ICML'16, pp 2702–2711

Dai H, Khalil EB, Zhang Y, Dilkina B, Song L (2017) Learning combinatorial optimization algorithms over graphs. arXiv:1704.01665

Diamond S, Boyd S (2016) CVXPY: A Python-embedded modeling language for convex optimization. Journal of Machine Learning Research 17(83):1–5

Good IJ (1953) The population frequencies of species and the estimation of population parameters. Biometrika 40(3/4):237–264

Goodfellow I, Bengio Y, Courville A (2016) Deep Learning. MIT Press

Gurobi Optimization, Inc (2016) Gurobi optimizer reference manual. URL `http://www.gurobi.com`

Hastie T, Tibshirani R, Friedman J (2009) The Elements of Statistical Learning. No. 2 in Springer Series in Statistics, Springer-Verlag New York

Hinton G, Deng L, Yu D, Dahl GE, Mohamed A, Jaitly N, Senior A, Vanhoucke V, Nguyen P, Sainath TN, Kingsbury B (2012) Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Processing Magazine 29(6):82–97

Hoffman AJ (1979) BINDING CONSTRAINTS AND HELLY NUMBERS. Annals of the New York Academy of Sciences 319(1 Second Intern):284–288

Jia Y, Shelhamer E, Donahue J, Karayev S, Long J, Girshick R, Guadarrama S, Darrell T (2014) Caffe: Convolutional architecture for fast feature embedding. arXiv:1408.5093

Khalil EB, Bodic PL, Song L, Nemhauser G, Dilkina B (2016) Learning to branch in mixed integer programming. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI Press, AAAI'16, pp 724–731, URL `http://dl.acm.org/citation.cfm?id=3015812.3015920`

Klaučo M, Kalúz M, Kvasnica M (2019) Machine learning-based warm starting of active set methods in embedded model predictive control. Engineering Applications of Artificial Intelligence 77:1 – 8

Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, p 2012

Kruber M, Lübbecke ME, Parmentier A (2017) Learning when to use a decomposition. In: Salvagnin D, Lombardi M (eds) Integration of AI and OR Techniques in Constraint Programming, Springer International Publishing, Cham, pp 202–210

LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444

Markowitz H (1952) Portfolio selection. The Journal of Finance 7(1):77–91

McAllester DA, Schapire RE (2000) On the convergence rate of Good-Turing estimators. In: Proceedings of the 13th Annual Conference on Computational Learning Theory

McDiarmid C (1989) On the method of bounded differences, Cambridge University Press, pp 148–188. London Mathematical Society Lecture Note Series

Misra S, Roald L, Ng Y (2019) Learning for constrained optimization: Identifying optimal active constraint sets. arXiv:1802.09639v4

Paszke A, Gross S, Chintala S, Chanan G, Yang E, DeVito Z, Lin Z, Desmaison A, Antiga L, Lerer A (2017) Automatic differentiation in PyTorch. In: NIPS-W

Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, Hubert T, Baker L, Lai M, Bolton A, Chen Y, Lillicrap L, Hui F, Sifre L, van den Driessche G, Graepel T, Hassabis D (2017) Mastering the game of go without human knowledge. Nature 550(7676):354–359

Sutton RS, Barto AG (2018) Reinforcement Learning: An Introduction, 2nd edn. A Bradford Book

Takapoui R, Moehle N, Boyd S, Bemporad A (2017) A simple effective heuristic for embedded mixed-integer quadratic programming. International Journal of Control pp 1–11

Zheng YS, Federgruen A (1991) Finding optimal (s, S) policies is about as simple as evaluating a single policy. Operations Research 39(4):654–665, `https://doi.org/10.1287/opre.39.4.654`