Artificial Intelligence 244 (2017) 368–395

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Auction optimization using regression trees and linear models as integer programs

Sicco Verwer[a], Yingqian Zhang[b,*], Qing Chuan Ye[b]

[a] Intelligent Systems Department, Delft University of Technology, The Netherlands
[b] Department of Econometrics, Erasmus University Rotterdam, The Netherlands

ARTICLE INFO

Article history:
Received in revised form 14 May 2015
Accepted 21 May 2015
Available online 29 May 2015

Keywords:
Auction design
Machine learning
Optimization
Integer linear programming
Regression

ABSTRACT

In a sequential auction with multiple bidding agents, the problem of determining the ordering of the items to sell in order to maximize the expected revenue is highly challenging. The challenge is largely due to the fact that the autonomy and private information of the agents heavily influence the outcome of the auction.

The main contribution of this paper is two-fold. First, we demonstrate how to apply machine learning techniques to solve the optimal ordering problem in sequential auctions. We learn regression models from historical auctions, which are subsequently used to predict the expected value of orderings for new auctions. Given the learned models, we propose two types of optimization methods: a black-box best-first search approach, and a novel white-box approach that maps learned regression models to integer linear programs (ILP), which can then be solved by any ILP-solver. Although the studied auction design problem is hard, our proposed optimization methods obtain good orderings with high revenues.

Our second main contribution is the insight that the internal structure of regression models can be efficiently evaluated inside an ILP solver for optimization purposes. To this end, we provide efficient encodings of regression trees and linear regression models as ILP constraints. This new way of using learned models for optimization is promising. As the experimental results show, it significantly outperforms the black-box best-first search in nearly all settings.

1. Introduction

One of the main challenges of mathematical optimization is to construct a mathematical model describing the properties of a system. When the structure of a system cannot be fully determined from the knowledge at hand, machine learning and data mining techniques have been used in optimization instead of this knowledge. They have, for example, been used in order to obtain decision values [1], fitness functions [2], or model parameters [3]. Models that have been learned from data are frequently used in a black-box manner, e.g., using only the predictions of learned models but not their internal structure. It is also possible to use these models in a white-box manner, for instance in order to determine search space cuts and parameter bounds. Neural networks have in this way been used to model unknown relations in constraint programming [4]. In this paper, we develop such a white-box optimization method for regression models in integer linear programming, that is, we map these entire models to sets of variables and constraints and solve them using an off the shelf solver.

* Corresponding author.
E-mail addresses: S.E.Verwer@tudelft.nl (S. Verwer), yqzhang@ese.eur.nl (Y. Zhang), ye@ese.eur.nl (Q.C. Ye).

http://dx.doi.org/10.1016/j.artint.2015.05.004
0004-3702/© 2015 Elsevier B.V. All rights reserved.

This white-box method together with a proposed black-box method provides a solution to an optimization problem of key interest to the artificial intelligence and operations research communities: auction design. We briefly introduce this problem domain before going into the details of our methods.

1.1. Sequential auction design

Auctions are becoming increasingly popular for allocating resources or items in business-to-business and business-to-customer markets. Often sequential auctions [5] are adopted in practice, where items are sold consecutively to bidders. Sequential auctions are in particular desirable when the number of items for sale is large (e.g., flower auctions [6]), or when the buyers enter and leave the auction dynamically (e.g., online auctions [7]). In a sequential auction, an auctioneer may tune several auction parameters to influence the outcome of an auction, such as reserve prices for items and in which order to sell them. In other words, (s)he can design auctions for the purpose of achieving some predefined goal. In this paper, we solve one specific auction design problem, namely, deciding the optimal ordering of items to sell in a sequential auction in order to maximize the expected revenue (OOSA in short). We assume bidders in such auctions are budget constrained. This is a highly relevant problem in today's auctions since bidders almost always have limited budget, as seen for instance in industrial procurement [8]. Previous research has shown that with the presence of budget constraints, the revenue collected by the auctioneer is heavily dependent on the ordering of items to sell [9–11]. This holds already for a toy problem with 2 items. Let us use a simple example to illustrate the importance of ordering in such cases.

Example 1. Two agents A_1 and A_2 take part in a sequential auction of items. For sale are items r_1 and r_2. Suppose the items are sold by means of first-price, English auction.[1] Assume the reserve prices, which are the lowest prices at which the auctioneer is willing to sell the times, for both items are 1. The amount that agent A_1 and agent A_2 are willing to pay for two items are: $v_1(r_1) = 10$, $v_1(r_2) = 15$, $v_2(r_1) = 12$, $v_2(r_2) = 10$. Furthermore, the budgets of A_1 and A_2 are 15 and 25 respectively.

We assume a simple bidding strategy in this example. The agents bid myopically on each item, that is, their highest bid on one item is the lower value between the amount that they are willing to pay and their remaining budget. The auctioneer's goal is to maximize the total sold price of the items. Consider one situation where the auctioneer sells first r_2 and then r_1. A_1 will get r_2 when she just over-bids A_2 with 11, and then when r_1 is auctioned, A_1 bids maximally 4 due to her budget limit, and A_2 will win the item with the price of 5. The total revenue is 16. However, if the selling sequence is (r_1, r_2), A_2 will win r_1 with the bid 11, and then A_2 will win r_2 with price 11. The collected revenue is 22 in this case. □

Most of the current approaches to the ordering problem in sequential auctions assume a very restricted market environment. They either study the problem of ordering two items, see [11,12], or a market with homogeneous bidders [13]. To the best of our knowledge, we are the first to consider how to order items for realistic auction settings with many heterogeneous bidders competing for many different items. This problem is highly complex—a good design on ordering needs to take care of many uncertainties in the system. For instance, in order to evaluate the revenue given an ordering, the optimization algorithm needs to know the bidders' budgets and preferences on items, which are usually private and unshared. Furthermore, the large variety of possible bidding strategies that bidders may use in auctions are unknown. This auction design problem is a typical example where the mathematical optimization model cannot be fully determined, and hence, machine learning and data mining techniques can come into play. This is exactly what our approach builds upon.

1.2. Learning models for white-box and black-box optimization

Nowadays more and more auctions utilize information technology, which makes it possible to automatically store detailed information about previous auctions along with their selling sequences and the selling price per auctioned item. Our approach to solving the problem of optimal ordering for sequential auctions starts with the historical auction data. We define and compute several relevant features and then use them to learn regression trees and linear regression models for the expected revenue. Given the models, we propose two approaches to find the optimal ordering for a new set of items: (1) a best-first search that uses the models as a black-box to evaluate different orderings of the items; and (2) a novel white-box optimization method that translates the models and the set of items into a mixed-integer program (MIP) and runs this in an ILP-solver (CPLEX). Fig. 1 displays the general framework of our approaches using these two optimization methods.

Just like the traditional black-box optimization approach (see, e.g. [14,15]), our best-first search is ignorant of the internal structure of the models and only calls it to perform function evaluations, i.e., predicting the revenue of an ordering of the items. Optimization is possible by means of a search procedure that uses heuristics to produce new orderings depending on previously evaluated ones. Our best-first search makes use of dynamic programming cuts inspired by sequential decision making in order to reduce the search space.

[1] The English auction that we consider is the one where the starting price is the reserve price, and bidders bid openly against each other. Each subsequent bid should be higher than the previous bid, and the item is sold to the highest bidder at a price equal to her bid.

One of the main contributions of this paper is the realization that learned regression models can be evaluated efficiently inside modern mathematical optimization solvers. This evaluation includes the computation of feature values (the input to machine learning), the evaluation of these features using a learned model (the output from machine learning), and a possible feedback from such evaluations to new features. In this paper, we efficiently translate all of these steps for two types of learned models (regression trees and linear regression models) into mixed-integer constraints. The resulting mixed-integer program can then be evaluated in any modern integer linear programming (ILP) solver.

In this way, modern exact solvers can be used instead of a heuristic search. These solvers use (amongst others) advanced branch-and-bound methods to cut the search space, compute and optimize a dual solution, and can prove optimality without testing every possible solution. This is the main benefit of using the white-box method over a black-box one. The downside, however, is that when the learned model is complex, the white-box method may lead to a large mathematical model that is difficult to optimize. We compare these two approaches and investigate this trade-off by applying them to the OOSA problem.

Contributions and organization Although we use sequential auction design to illustrate our method, all of our constructions are general and can be applied to any optimization setting where unknown relations can be represented using regression models that have been learned from data. The only constraint for using the white-box method is that the feature values need to be computable using integer linear functions from intermediate solutions. Our approach can thus be applied to complex optimization settings where entire orders, schedules, or plans need to be constructed beforehand.

We list our main contributions as follows:

- We demonstrate how to apply regression methods from machine learning to OOSA.
- We give an efficient encoding of regression trees and linear regressors into MIP constraints.
- We prove OOSA with budget constrained bidders to be NP-hard, also when using these regression models.
- We provide the first method that tackles OOSA in realistic settings.
- We demonstrate experimentally that white-box methods outperform black-box methods when the models are not overly complex.

In Section 2, we formally introduce the problem of optimal ordering for sequential auctions (OOSA), and then we show how to learn regression models from historical auction data in Section 3 using standard machine learning methods. Based on the learned models, our white-box optimization method and a black-box optimization are introduced to find the optimal ordering for OOSA in Section 4. Extensive experiments are presented in Section 5 where we compare the performance of the two proposed optimization methods using both the learned models and the auction simulator. Before we conclude, we compare and discuss more related works in Section 6.

2. Optimal ordering for sequential auctions OOSA

We assume there is a finite set of bidders (or agents). Let $R = \{r_1, \dots, r_l\}$ denote the collection of the item types, and the quantity of each item type can be more than 1. When it is clear from the context, we will slightly abuse the notation and use $S = \{r_1, r_2 \dots, r_1, \dots\}$ to denote the multiset of all available items. Each bidder agent i has a valuation (or preference) for each type of item $v_i : R \rightarrow \mathbb{R}^+$. In addition, each agent has a budget b_i on purchasing items, and (s)he desires to win as many items as are being auctioned within the budget limit.

In one auction, a set of n items S with type set $R' \subseteq R$ will be auctioned sequentially using a predetermined order. We use $(s_1, s_2, \dots, s_n)$ to denote such an ordering. For example, given types r_1 and r_2 with quantities of 1 and 2 respectively, there are three possible orderings of items: $(s_1 = r_1, s_2 = r_2, s_3 = r_2)$, $(s_1 = r_2, s_2 = r_1, s_3 = r_2)$, and $(s_1 = r_2, s_2 = r_2, s_3 = r_1)$. For each r_j that is being auctioned, agent A_i puts a bid on r_j that is the minimum between the amount she is willing to pay for r_j and the remaining budget. We point out that in the case of unconstrained budget, the maximum amount an agent is willing to pay for r_j, defined as $\nu_i(r_j)$, is equal to her valuation $v_i(r_j)$.

Each item r_j comes with a reserve price that is the lowest price at which the auctioneer is willing to sell r_j. If the received bids are all below the reserve price of r_j, r_j is not sold. Otherwise, the agent who bids highest on r_j wins r_j. The winners of items transfer some payment to the auctioneer depending on the auction rule. For example, in a first-price auction, the winner pays an amount equal to her bid, and in a second-price auction, she pays the second highest bid (or the reserve price for the item if it is higher). The revenue of the auctioneer is the sum of the total payment on the sold items and the total reserve values of the unsold items. This sequential auction ends when all items have been auctioned, or when all agents run out of their respective budgets.

We assume that such an auction is repeated over time, and each auction sells possibly different items S. At the end of each auction, we have the following information at our disposal: (1) the ordering of auctioned items; and (2) the allocation of items to agents with their payments. The optimization problem we study is: *given a set of items and budget constrained bidders, finding an optimal ordering of items in sequential auctions such that the expected revenue is maximized*. We call the problem OOSA.

We now show that the decision version of this optimization problem is NP-hard, even if we have complete information on bidders' preferences and they are not strategic (i.e., they bid truthfully according to their preferences).

Theorem 1. *Given a set of items S, preferences $v_i : S \to \mathbb{R}^+$, and budgets b_i for every bidder i. The problem of deciding whether there exists an ordering that obtains a revenue of at least $K \in \mathbb{R}^+$ is NP-hard.*

Proof. By reduction from the well-known NP-hard partition problem [16]: Given a set of integers $I = \{i_1, \ldots, i_n\}$, is I dividable into two sets A and B such that $\sum A = \sum B$? We need two bidders with preferences such that $v_1(r_k) = 2 \cdot i_k$ and $v_2(r_k) = 2 \cdot i_k + 1$ for $1 \le k \le n$. The reserve price for each item k ($1 \le k \le n$) is i_k. The agents' budgets are $b_1 = \frac{1}{2}\sum I$ and $b_2 = \sum I$. There are n items in S and K is $\frac{3}{2}\sum I$. We claim that I is partitionable into two sets with equal sums if and only if there exists an ordering that obtains a revenue of K (or more).

($\Rightarrow$) Given a partition of I into sets A and B, we sell all items in A first, and those in B later. In this case, agent 2 will buy all items in A with price $2 \cdot i_k$ as it is the minimal bid to win the items from agent 1. After buying all items in A, agent 2 will have spent $2 \cdot \sum_{i_k \in A} i_k$, which makes $\sum I$ in total (since $\sum_{i_k \in A} i_k = \frac{1}{2}\sum I$). This is the entire budget of agent 2. All items in B are then sold to agent 1 with the reserve price i_k. Thus agent 1 pays $\sum_{i_k \in B} i_k = \frac{1}{2}\sum I$. This makes a total revenue of $\sum I + \frac{1}{2}\sum I = \frac{3}{2}\sum I = K$.

($\Leftarrow$) Suppose we have an ordering such that agent 1 and 2 spend all of their budget (K in total). This means that agent 2 wins the first set of items A, each costing $2 \cdot i_k$ till it uses all its budget. Thus we have $2 \cdot \sum_{k \in A} i_k = \sum I$, i.e., $\sum_{k \in A} i_k = \frac{1}{2}\sum I$. Agent 1 pays i_k for the remaining items in B, $B = I \setminus A$, and it uses all its money: $\sum_{k \in B} i_k = \frac{1}{2}\sum I$. Hence, we have a partition of I where $\sum A = \sum B$.

The construction is clearly polynomial time. □

Several related works deal with this type of ordering optimization problem. For example, the authors of [11] investigate the optimal ordering strategy for the case where the auctioneer has two items to sell. They show that when the items are different in value, the higher valued items should be auctioned first in order to increase the seller's revenue. Pitchik [12] points out that in the presence of budget constraints, in a sealed-bid sequential auction, if the bidder who wins the first good has a higher income than the other one, the expected revenue is maximized. These greatly simplified auction settings make it possible to derive bidders' equilibrium bidding strategies. With some assumptions on the distributions of bidders' budgets and preferences, the optimal ordering can be theoretically derived. However, as real-world auctions are much more complex and uncertain in terms of the sizes of items/bidders, agents' preferences and bidding strategies, these existing results cannot be applied. In this paper, we instead focus on learning the overall behaviors of the group of bidders from historical auction data by machine learning techniques, as the first step of solving OOSA.

In order to apply ML techniques, we assume in every sequential auction the set of participating bidders and their characteristics (preferences, budgets, bidding strategies) to be similar. This simplifies the problem of learning a good ordering. Instead of learning the individual valuations/budget/bidding strategies of agents, we can treat the agent population as a single entity for which we need to find a single global function. Obviously, such an approach will fail if the agents are radically different in every auction. However we consider this assumption sensible in many auctions such as industrial procurement auctions where the same companies repeatedly join the auctions with similar interests, and the Dutch flower auction where there can be different bidders every day, but it seldom occurs that one day bidders are only interested in roses and the next day they only want tulips. Although the different participants can be interested in different item types, the interests of the group of participants remain stable.

3. Learning predictive models for OOSA

At the end of each sequential auction, we have the following information at our disposal: (1) the ordering of auctioned items; and (2) the price of each sold item. Before we build the optimization model to solve the OOSA problem, we need to find a suitable way to model the expected revenue of given orderings of auctioned items.

An ordering can be thought of as a sequence of items. However, to the best of our knowledge none of the existing sequence models fit our auction setting, see also Section 6.2. In this work, we view the prediction of an auction's outcome as a regression problem. We split this problem into the subproblems of predicting the value of the auctioned items. We then sum these up to obtain the overall objective function, i.e., the expected revenue $P(S)$ given a set S ($|S| = n$) of items:

$$P(S) = \sum_{1 \le k \le n} G(s_k, \{s_j \mid j < k\}, \{s_l \mid k < l\}),$$

where $G(s_k, J, L)$ is a regression function that determines the expected value of s_k given that J was auctioned before and L will be auctioned afterwards. The main benefit of this representation is that modern machine learning methods can be used to learn this function G from data. In addition, since every item sold represents a single sample, every auction contains many samples that can be used for learning, further reducing the amount of required data. We study two popular regressions functions.

3.1. Two regression functions

In this paper, we use regression trees [17] and least absolute shrinkage and selection operator (LASSO) [18] as regression functions, and train them using features based on the items auctioned before and after the current item. We first briefly introduce these regressors.

Regression trees Regression trees are a form of decision trees where the predicted values are real numbers. A decision tree is one of the most popular predictive models for mapping feature values to a target value. It is a tree-shaped graph with a root node, interior nodes, and leaf nodes. The root and every interior node contains a Boolean test for a specific feature value f, such as $f > 5$. Every leaf node contains an output value p. It maps the feature values to an output by performing all the tests along a path from the root to a leaf. For every test performed, if the outcome is true ($f > 5$), the path is continued along the left branch, if the outcome is false ($f \leq 5$), the path is continued along the right branch. Once a leaf is reached, it outputs the value it contains p.

A regression tree learner aims to find a tree that minimizes the mean squared error of the predicted and the actual observed values. Most regression tree learning algorithms follow a greedy strategy that splits interior nodes as long as the decrease in error is significant. A split replaces one leaf node by an interior node connected with two new leaf nodes. The interior node receives as Boolean constraint one that minimizes the mean-squared error of the resulting tree, where the leaf nodes predict the mean value of all observed data values that end up in that leaf after mapping all data samples to leaf nodes.

LASSO LASSO is a method for constructing a linear regression function $p(f_1, \ldots, f_m) = c_1 f_1 + c_2 f_2 + \ldots + c_m f_m + d$, where p is the value to predict, c_i are constants, f_i are feature values, and d is the intercept. The standard approach to find such a function is to minimize the mean squared error, which is easy to compute. LASSO is a popular regularized version of this simple estimation that penalizes the absolute values of the regression coefficients $c_1, \ldots, c_m$. Formally, given a dataset of features f_i^d and target values p^d, with $1 \leq d \leq k$ where k is the number of samples, it uses convex optimization in order to find a regression function that solves the following problem[2]:

$$\min \sum_{1 \leq d \leq k} \frac{1}{2 \cdot k} (p(f_1^d, \ldots, f_m^d) - p^d)^2 + \alpha \cdot \sum_{1 \leq i \leq m} |c_i|$$

where $0 \leq \alpha \leq 1$ is a parameter for the effect of the regularization. Intuitively, the larger α, the larger the penalty of having large coefficients c_i. Consequently, a larger value of α will drive more coefficients to zero. LASSO is a useful method when there are several correlated feature values, which could make an ordinary least squares model overfit on these values. We use LASSO regression because more zero coefficients implies we need to compute less feature values in order to evaluate the learned model, which has a positive effect on the optimization performance that we will discuss in Section 4.

3.2. Learning regression functions for predicting revenues

We first give an overview in Fig. 1 of the connection between the regression models and the optimization models for solving OOSA. Given historical auction data, a regression tree (or a LASSO linear regression function) is learned for each item type. The regression tree (or LASSO) can be used to evaluate the values of selling different items based on the feature values that are computed on a given ordering of the items. The learned regression trees (or LASSO functions) are then used in two ways to model the optimization problem OOSA: (1) Black-box optimization. In this paper, we use a best-first search heuristic to come up orderings of the items, and then use the learned regression trees (or LASSO) to compute the expected revenue of these orderings; (2) White-box optimization. We formulate the optimization problem of finding an optimal ordering as a mixed integer linear program (MIP), which is shown to be automatically constructed from the learned regression tress (or LASSO functions).

We now present the details of learning regression trees and LASSO functions for item types. Currently, we provide the following features for these two regression models:

Feature 1: `sold` For every item type r, the amount of r items already auctioned.
Feature 2: `remain` For every item type r, the amount of r items still to be auctioned.
Feature 3: `diff` For every pair of item types r and r', the difference between the amount of r and r' items already auctioned.
Feature 4: `sum` For every item type r, the amount of value obtained from auctioning r items, and the overall sum.
Feature 5: `index` For every item, the index at which it was auctioned.

Other sequential features such as sliding windows and N-grams (see, e.g., [20]) can of course be added to the model. However, since our white-box method computes these values inside an ILP solver, the only requirement is that they can

[2] This is the version implemented in the scikitlearn Python package [19], which we use to learn the models.

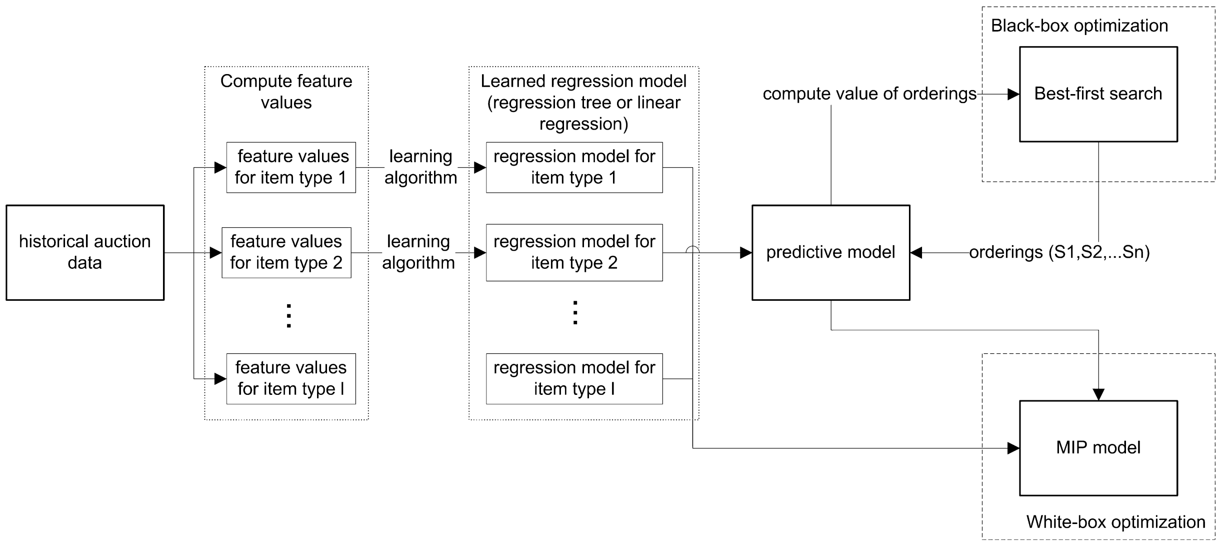

Black-box optimization
Compute feature values
Learned regression model (regression tree or linear regression)
compute value of orderings
Best-first search
feature values for item type 1
learning algorithm
regression model for item type 1
historical auction data
feature values for item type 2
learning algorithm
regression model for item type 2
predictive model
orderings (S1,S2,...Sn)
feature values for item type l
regression model for item type l
MIP model
White-box optimization

Fig. 1. Solving OOSA using white-box optimization and black-box optimization with learned models. Black-box optimization only calls the predictive model to evaluate possible orderings. White-box optimization translates the internal structure of the predictive model to MIP constraints.

Table 1
The data set created from the past two auctions $\{r_2, r_1\}$ and $\{r_1, r_2\}$ in Example 2.

Type	Value	soldr_1	soldr_2	remainr_1	remainr_2	diff$r_1 r_2$	sumr_1	sumr_2	sum	index
r_2	11	0	0	1	0	0	0	0	0	1
r_1	5	0	1	0	0	−1	0	11	11	2
r_1	11	0	0	0	1	0	0	0	0	1
r_2	11	1	0	0	0	1	11	0	11	2

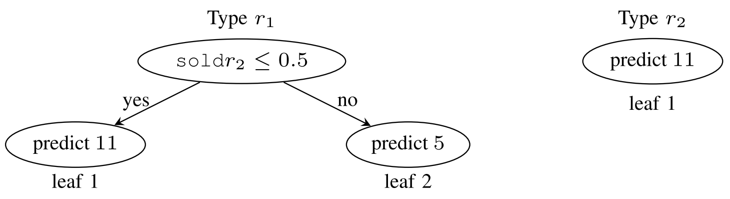


Fig. 2. Two learned regression trees for the two item types r_1 and r_2 from Example 2. The leaves of the left (right) tree output the predicted value of the item of type r_1 (r_2), determined by the feature value soldr_2.

be represented using an integer linear formulation. Although the diff feature can be determined using the first, we add it for convenience of learning a regression tree, which requires many nodes to represent such values. The influence of budget constraints is only directly modeled by the fourth feature: once the amount paid for r_1 items reaches a certain (to be learned) bound, we can expect all agents that only want r_1 items to be out of budget. Although, there only exists an indirect relation between the budget constraints and the first three features, including them can be beneficial and these are easier to compute. If used by the regression model, these features thus reduce the time needed to solve the auction design problem. For similar reasons, we add the last feature. Below we give an example of how an ordering and its obtained values is transformed into a data set using these 5 types of features.

Example 2. Consider the setting of Example 1. Assume two auctions have been carried out. One sold r_2 first and then r_1. The other reversed. As shown in Example 1, the first auction would obtain a revenue of 16, and the second auction would receive 22. We compute feature values from these two auctions as depicted in Table 1. Subsequently, we learn regression trees for both item types r_1 and r_2, as shown in Fig. 2.[3]

After learning these regression trees, we can optimize the ordering for a new (unseen) multiset of items $\{r_1, r_1, r_1, r_2\}$ by trying all orderings and choosing one with maximum expected revenue: (r_1, r_1, r_1, r_2) gives $11 + 11 + 11 + 11 = 44$, (r_1, r_1, r_2, r_1) gives $11+11+11+5=38$, (r_1, r_2, r_1, r_1) gives $11+11+5+5=32$, and (r_2, r_1, r_1, r_1) returns $11+5+5+5=26$. Hence, the optimizer will choose to schedule the r_2 item after all r_1 items. □

[3] The learned linear regression model is more straightforward. Hence we skip such an example here.

The example showed how to evaluate different orderings of items using the learned regression trees. In general, trying all possible orderings will be impossible: for a multiset of items $S = \{r_1, \ldots, r_m\}$ of m types, there are a total of $\frac{|S|!}{\prod_{1 \leq i \leq m} |\{r_i | r_i \in S\}|!}$ unique orderings, which blows up very quickly.

3.3. Modeling power and trade-off

Our method of regression modeling allows the use of any regression method from machine learning for predicting unknown quantities in optimization, such as objective values and parameters. In addition, since the regression function G uses other values in a (proposed) solution as input (J, L) instead of only external parameters/data, a learned regression model represents unknown relations between the different values in a solution. The model thus answers the question "What is the value of X given that we do Y?", as opposed to "What is the value of X?" that is answered by fitting only model parameters. Answering the first question allows for many more interesting possibilities. For instance, one could use stochastic optimization with fitted parameters to produce a schedule, use regression models to predict the effect of this schedule on the parameters, and use stochastic optimization again on the newly estimated parameters. This way, one can use machine learning tools to plan further ahead. Using our white-box method, this can even be done using a single call to the optimization software (see Section 4).

This loop-back functionality provides a lot of power to our method, but also comes with a risk. Every time the predictive models are used there is a probability that the predictions are inaccurate. When using a loop-back, these possible inaccuracies influence all future predictions that depend on it. These future predictions are thus more inaccurate and the predicted overall objective value can potentially diverge from the true value. These cascading inaccuracies are an issue, however, the added modeling power makes up for it. We make use of it in the `sum` feature, which relates the predicted value to the predictions of earlier auctioned items. This feature is very important for predicting budget constraints, and consequently is often used by the regression models to produce predictions.

4. White-box and black-box optimization for OOSA

Given the predictive models for the expected value per item, it is not straightforward to compute a good ordering as we already showed in Example 2. For a given ordering, we can predict the individual revenues of items using the regression model, and sum these up to obtain the revenue of the ordering. However, testing all possible orderings and choosing the one with the highest revenue will take a very long time. For instance, when we want to order 40 items of 8 types (the experimental setting in Section 5) with 5 of every type, we will need to test $\frac{40!}{5!^8} \approx 1.9 \cdot 10^{31}$ possible unique orderings.

In Appendix A, we also provide hardness results that demonstrate there is little hope (unless $\mathtt{P} = \mathtt{NP}$) of finding an efficient (polynomial-time) algorithm that gives the optimal ordering for any regression tree or linear regression predictor. In general, we cannot do better than performing a guided search through the space formed by all possible orderings. We present two such search-based optimization methods: (1) a novel "white-box" optimization (i.e., ILP model), and (2) a "black-box" heuristic (i.e., best-first search).

4.1. White-box optimization: an ILP model

Given regression tree and linear regression models for the expected value per item type, we automatically formulate the problem of finding an optimal ordering as a mixed integer linear program (MIP). We discuss the encoding of a sequential auction, feature values, objective function, and translating the learned models (regression tree and linear regression respectively) below.

Ordering an auction Given a multiset S of n items, each from a set of possible types R, we use the following free variables to encode any possible ordering of S: $x_{i,r} \in \{0, 1\}$. Item i is of type r if and only if $x_{i,r} = 1$. Thus, if x_{3,r_1} is equal to 1, it means that the third auctioned item is of type r_1. We require that at every index i at most one item type is auctioned, and that the total number of auctioned items of type r is equal to the number n_r of type r items in S.

$$\sum_{r \in R} x_{i,r} = 1 \text{ for all } 1 \leq i \leq n \tag{1}$$

$$\sum_{1 \leq i \leq n} x_{i,r} = n_r \text{ for all } r \in R \tag{2}$$

Any assignment of ones and zeros to the x variables that satisfies these two types of constraints corresponds to a valid ordering of the items in S. The value of such an ordering is determined by the learned regression models.

Translating feature values In order to compute the prediction of a regression model, we not only need to translate the models into ILP constraints, but also the values of the features used by these models.[4] Feature 1, 2, 3, and 5 can be computed using linear functions from the x variables:

$$\texttt{sold}_{i,r} = \sum_{j<i} x_{j,r} \quad \text{for all } 1 \le i \le n, r \in R \tag{3}$$

$$\texttt{diff}_{i,r,r'} = \texttt{sold}_{i,r} - \texttt{sold}_{i,r'} \quad \text{for all } 1 \le i \le n, r, r' \in R, r \neq r' \tag{4}$$

$$\texttt{remain}_{i,r} = \sum_{j>i} x_{j,r} \quad \text{for all } 1 \le i \le n, r \in R \tag{5}$$

$$\texttt{index}_{i} = i \quad \text{for all } 1 \le i \le n \tag{6}$$

For the fourth type of feature, we use an additional variable $p_{j,r}$, which encodes the expected value of the item auctioned at index j of type r. If the item at index j is not of type r, $p_{j,r}$ is equal to 0. Since the p variables are the predictions of the regression functions, we provide their definition after defining the regression models.

$$\texttt{sum}_{i,r} = \sum_{1\le j \le i} p_{j,r} \quad \text{for all } 1 \le i \le n, r \in R \tag{7}$$

To aid the ILP solver, we also pre-compute the minimum $m_{f,i}$ and maximum $M_{f,i}$ obtainable values of every feature f at every index i and provide these as bounds to the solver.

Constructing the objective function We aim to maximize the expected values $p_{i,r}$:

$$\max \sum_{1\le i\le n} \sum_{r\in R} p_{i,r} \tag{8}$$

Although it is also possible to compute both the objective function and the sum values as very large sums over the x and model variables (described next), specifying parts of these sums as intermediate continuous p variables significantly reduces both the encoding size and the computation time.

Finally, we discuss how to encode the learned regression tree and the linear regression model as the constraints in the ILP model.

Encoding regression trees We translate the regression tree models into ILP using carefully constructed linear functions. Our encoding only requires one new set of $\{0, 1\}$ variables $z_{i,l,r}$, representing whether a leaf node l is reached for item type r at index i. The internal (decision) nodes of the trees can be represented implicitly by the constraints on these new z variables. Intuitively, we encode that a z variable has to be false when the binary test of any of its parent nodes fails. By additionally requiring that exactly one z variable is true at every index, we fully encode the learned regression trees.

Let D_r be the set of all decision nodes in the regression tree for type r. Every decision node in D_r contains a boolean constraint $f \le c$, which is true if and only if feature f has a value less than or equal to a constant c. A key insight of our encoding is that every such boolean constraint directly influences the value of several z variables: if it is true (at index i), then all z variables representing leafs in the right subtree are false; if it is false, then all that represent leafs in the left subtree are false. In this way, we require only two constraints per boolean constraint in order to represent all possible paths to leaf nodes.

$$\texttt{fv}_{f,i} + (M_{f,i} - c) \cdot \sum_{l\in L} z_{i,l,r} \le M_{f,i} \quad \text{for all } 1 \le i \le n, r \in R, (f \le c) \in D_r \tag{9}$$

$$\texttt{fv}_{f,i} + (m_{f,i} - c) \cdot \sum_{l\in L'} z_{i,l,r} \ge m_{f,i} \quad \text{for all } 1 \le i \le n, r \in R, (f \le c) \in D_r \tag{10}$$

where $\texttt{fv}_{f,i}$ is a calculation of feature value f for index i, L and L' are the leaf nodes in the left and right subtrees of the decision node with constraint $(f \le c)$ in the regression tree for type r, and $M_{f,i}$ and $m_{f,i}$ are the maximum and minimum values of feature f at index i. For the feature calculation we simply replace $\texttt{fv}_{f,i}$ with the right-hand sides of the corresponding feature definitions.[5]

The above constraints ensure that when $z_{i,l,r}$ obtains a value of 1, all of the binary test in the parent nodes on the path to l in the tree for type r return true at index i. By construction of the regression trees, this ensures that at most one z

[4] Including any transformations applied to them.

[5] We ignore the possibility that a feature f is equal to c because, since features have a limited precision, we can always replace the constants in a decision node with one that cannot be obtained by f, without changing its behavior.

variable is true for every type r and index i. We require however that exactly one z variable is true at every index.[6] This z has to be of the same type as the item sold at index i, denoted by the x variables:

$$\sum_{l} z_{i,l,r} = x_{i,r} \quad \text{for all} 1 \leq i \leq n, r \in R \tag{11}$$

This completes our encoding of the regression trees. The predictions of the trees at every index i are given by the z variable that is true for index i. We multiply this z variable with the constant prediction in the leaf node it represents to obtain the prediction, and store it in the p variables used to compute the `sum` feature values.

$$p_{i,r} = \sum_{l \in L_r} c_{l,r} \cdot z_{i,l,r} \quad \text{for all } 1 \leq i \leq n, r \in R \tag{12}$$

where $c_{l,r}$ is the constant prediction of leaf l in the tree for type r.

Complexity Our translation of regression trees is very efficient. It requires only 2 constraints per decision node (D_r, Equations (9) and (10)) and 1 binary variable $z_{i,l,r}$ for every leaf node ($L + L'$, used in Equations (9), (10), and (11)) of the tree. To encode a complete depth k tree with $2^{k+1} - 1$ nodes, this thus requires only $2 \times (2^k - 1) = 2^{k+1} - 2$ constraints for the decision nodes and 2^k binary variables for the leaf nodes. In addition, we require 1 constraint to force exactly one leaf variable to be true (the same type as $x_{i,r}$, Equation (11)). All the other variables and constraints, used to compute the feature values and $p_{i,r}$ variables, can be computed directly without storing the result into a variable. Consequently, this adds zero variables and zero constraints to the translation.

In order to encode an entire OOSA problem, a new set of constraints representing a tree is constructed for every item type and every item index. In a problem with $|R|$ types and n items for sale, this creates $n \times |R| \times (2^{k+1} - 1)$ constraints and $n \times |R| \times (2^k)$ variables to encode a complete tree of depth k. The ordering problem itself requires $n \times |R|$ ($x_{i,r}$, Equations (1) and (2)) variables and $n + |R|$ constraints. This totals to: $n \times |R| \times (2^k + 1)$ variables and $n \times |R| \times (2^{k+1} - 1) + n + |R|$ constraints. Since a complete depth k tree has $2^{k+1} - 1$ nodes, this is linear in the number of nodes of the tree.

We now discuss how to encode a linear regression model.

Encoding linear regression model Due to its linear nature, implementing linear regression in ILP is very straightforward. We can directly compute the value of the p variables using the linear predictor function:

$$p_{i,r} = \sum_{f \in \mathtt{Feat}} c_{f,r} \cdot \mathtt{fv}_{f,i} \quad \text{for all } 1 \leq i \leq n, r \in R \tag{13}$$

where `Feat` is the set of all features, $\mathtt{fv}_{f,i}$ is feature f's values at index i, and $c_{f,r}$ is the constant coefficient for feature f in the regression function for type r. The only somewhat difficult part is that at every index, the used regression function can change depending on the auctioned item type r. We implemented this choice using indicator functions in CPLEX.[7] This changes the above formulation as follows:

$$x_{i,r} = 1 \rightarrow p_{i,r} = \sum_{f \in \mathtt{Feat}} c_{f,r} \cdot \mathtt{feat}_{f,i} \quad \text{for all } 1 \leq i \leq n, r \in R \tag{14}$$

$$x_{i,r} = 0 \rightarrow p_{i,r} = 0 \quad \text{for all } 1 \leq i \leq n, r \in R \tag{15}$$

It states that if $x_{i,r}$ is true, then the values of $p_{i,r}$ is determined using the regression function. Otherwise, its value is 0. These are the only constraints needed to fully implement a linear regression function. When using LASSO regularization, some coefficients will receive the value 0. These are removed from the encoding, making the models smaller and easier to evaluate in the solver.

Complexity The translation of regression functions is straightforward and only requires 2 constraints per item type for the indicator functions (Equations (14) and (15)). Because of these indicators, we do need to encode the $n \times |R| p_{i,r}$ variables. No additional constraints or variables are required. In order to encode an entire OOSA problem, we thus require only $n \times |R| \times 2$ constraints and $n \times |R| \times 2$ variables.

[6] Counterintuitively, it can occur that the objective function (discussed below) is maximized when every z variable is false at an index i. If a small `sum` is needed to reach a high revenue prediction, it can be beneficial to auction but not sell an item.

[7] Many other solvers have similar constructions. If not, these constraints can be implemented using a 'big-M' formulation, similar to the one we use to determine the value of the z variables in the regression tree formulation.

An example Now the two ILP models are complete and ready to solve the OOSA problem. We give the following example to illustrate how the formulation of ILP works given learned regression trees.

Example 3. Given the learned trees in Example 2, suppose we are asked to order a new multiset of items $\{r_1, r_2, r_2\}$. We translate this new set, together with the learned trees into the following integer linear program with the following $\{0, 1\}$ decision variables (for all $1 \le i \le 3$): $x_{i,r_1}, x_{i,r_2}, z_{i,1,r_1}, z_{i,2,r_1}, z_{i,1,r_2}$:

$$\max \sum_{1 \le i \le 3} p_{i,r_1} + p_{i,r_2}$$
$$\text{where } p_{i,r_1} = 11z_{i,1,r_1} + 5z_{i,2,r_1}$$
$$\text{and } p_{i,r_2} = 11z_{i,1,r_2}$$

subject to (for all $1 \le i \le 3$)

$$x_{1,r_1} + x_{2,r_1} + x_{3,r_1} = 1$$
$$x_{1,r_2} + x_{2,r_2} + x_{3,r_2} = 2$$
$$x_{i,r_1} + x_{i,r_2} = 1$$

This denotes that exactly one x variable is true at every index i, $2x$ variables are true for item type r_2, and 1 for type r_1. This encodes all possible orderings. From this we compute the feature values (for all $1 \le i \le 3$):

$$\texttt{sold}_{i,r_1} = x_{1,r_1} + \ldots + x_{i-1,r_1}$$
$$\texttt{sold}_{i,r_2} = x_{1,r_2} + \ldots + x_{i-1,r_2}$$

that are used in the constraints denoting the Boolean tests in the internal nodes:

$$\texttt{sold}_{i,r_2} + (100 - 0.5)z_{i,1,r_1} \le 100$$
$$\texttt{sold}_{i,r_2} + (-0.5)z_{i,2,r_1} \ge 0$$

where $M_{\texttt{sold},i} = 100$ and $m_{\texttt{sold},i} = 0$. The first two constraints encode that if $z_{i,1,r_1} = 1$, $\texttt{sold}_{i,r_2} \le 0.5$; and if $z_{i,2,r_1} = 1$, $\texttt{sold}_{i,r_2} \ge 0.5$. Thus, if a z variable is true for a leaf, then all the Boolean tests of internal nodes on the path from the root to that leaf have to succeed. At last, we require that exactly one z variable is true at every index:

$$z_{i,1,r_1} + z_{i,2,r_1} = x_{i,r_1},$$
$$z_{i,1,r_2} = x_{i,r_2}.$$

A satisfying assignment to the x variables is $x_{1,r_1}, x_{2,r_2}, x_{3,r_2}$ set to 1, the rest to 0, corresponding to the ordering (r_1, r_2, r_2). Since $\texttt{sold}_{1,r_2} = 0$, this leads to $99.5z_{1,1,r_1} \le 100$ and $-0.5z_{1,2,r_1} \ge 0$, forcing $z_{1,2,r_1} = 0$. Since $z_{1,1,r_1} + z_{1,2,r_1} = x_{1,r_1} = 1$, this implies $z_{1,1,r_1} = 1$. For the next index, since $x_{2,r_2} = 1$, it forces $z_{2,1,r_2} = 1$. Similarly, we obtain $z_{3,1,r_2} = x_{3,r_2} = 1$. This results in $p_{1,r_1} = 11$, $p_{2,r_2} = 11$, $p_{3,r_2} = 11$, and an objective value of 33. □

4.2. A black-box heuristic: best-first search algorithm

We also provide a black-box heuristic for solving the ordering problem, see also [21]. The traditional method to overcome the computational blowup caused by sequential decision making is to use a dynamic programming method. Although this lessens the computational load by combining the different paths that lead to the same sets of auctioned items, the search space is still too large and waiting for a solution will take too long. Instead, we therefore employ a best-first search strategy that can be terminated anytime in order to return the best found solution so far. We show how this best-first search strategy works in Algorithm 1.

The algorithm uses a hashtable and a priority queue. The hashtable is used to exclude the possibility of visiting the same nodes twice if the obtained value is less than before (just like a dynamic programming method). These dynamic programming cuts are sensible but lose optimality as on rare occasions it could be better to sell earlier items for less, leaving more budget for the remaining ones. The priority queue provides promising candidate nodes for the best-first strategy. By computing random orderings of the remaining items, the learned models can evaluate complete orderings of all items. The best one found is stored and returned if the algorithm is terminated. Unfortunately, this does not result in an admissible heuristic for an A* search procedure. Hence, even if the algorithm pops a solution from the queue, this is not necessarily optimal. In our experience, using random orderings of the remaining items in this heuristic provides a good spread over the search space. Although some nodes can be ‘unlucky’ and obtain a bad ordering of the remaining items, there are always multiple ways to reach nodes in the search space and it is very unlikely that all possibilities will be ‘unlucky’.

Algorithm 1 Black-box heuristic for solving OOSA: best-first search.

```
Require: A set of items $S$, historical data on orderings and their values $D$, a maximum number of iterations $m$
Ensure: Returned is a good (high expected value) ordering
  Transform $D$ into a data set
  for every item type $r$ do
    Learn a regression model from $D$ for predicting the value of item type $r$
  end for
  Initialize a hashtable $H$ and a priority queue $Q$
  Add the empty data row to $Q$
  while $Q$ is not empty and the size of $H$ is less than $m$ do
    Pop the row of features $F$ with highest value $p$ from $Q$
    if $H$ does not contain $F$ with a value $\geq p$ then
      Add $F$ with value $p$ to $H$
      Let $L$ be the set of remaining items in $F$
      for every item type $r$ of items in $L$ do
        Let $i_k$ be an item of Type $r$ in $L$
        Let $L'$ be a random ordering of $L - i_k$
        Use the models to evaluate the value $p'$ of auctioning the ordering $i_k L'$ after $F$
        Create new features $F'$ for auctioning $i_k$ after $F$
        Add $F'$ to $Q$ with value $p + p'$
      end for
    end if
  end while
  return The highest evaluated ordering
```

4.3. Discussion: white-box or black-box optimization?

The main difference between the two abovementioned approaches (see Fig. 1) is that the white-box method specifies the predictors entirely as constraints, which can be used to infer bounds on the predictions and cut the search space. The black-box method instead uses the predictors as oracles and is ignorant as to how the predictions are made, which are naturally more efficient to compute but cannot be used to infer search space cuts, i.e., to deduce that one ordering is better than another without testing both of them. Another key difference is that the white-box method results in a single optimization model that can be run in any modern solver, while the black-box method requires the use of executable code to produce the predictions. In the black-box setting, it is therefore much harder to use the powerful solving methods available in dedicated solvers for problems such as integer linear optimization (ILP), satisfiability (SAT), or constraint programming (CP). Instead, general search methods can be used such as best-first search, beam-search, meta-heuristics, genetic algorithms, etc.

Both black-box and white-box approaches have their advantages. The main advantage of black-box is that its performance is for a large part independent of the complexity of the used regression model. In contrast, by explicitly modeling the regression model as constraints in a white-box, more complex regression models lead to many more constraints, which can dramatically increase in the time needed to solve it. Another advantage of black-box optimization is that it is easy to include additional cuts such as the dynamic programming cuts discussed above. Such cuts can be added as constraints in an LP formulation, but this can lead to a blow-up in runtime.

The main benefit of using the white-box approach is the use of modern exact solvers instead of a heuristic search. These solvers use (amongst others) advanced branch-and-bound methods to cut the search space, compute and optimize a dual solution, and can prove optimality without testing every possible solution. Our white-box constructions can also be easily integrated into existing (I)LP formulations that have been used in a wide range of applications in for instance Operations Research. In this way, one can combine the vast amount of expert knowledge available in these applications with the knowledge in the readily available data.

The most important downside of white-box is that an evaluation of translated models likely requires more time than running the code as a black-box, especially when the models or features are somewhat complex. In our opinion, however, the advantages of white-box optimization largely outweigh those of black-box optimization and make it a very interesting topic for research in machine learning and optimization.

5. Experiments

Designing an optimal ordering for sequential auctions is difficult with heterogeneous bidders, as they may value items differently, have different budget constraints, and moreover, bid rationally or irrationally with various bidding strategies. To evaluate the performance of the proposed optimization methods, ideally, we should collect real auction data, build the optimization models, run real-world auctions with real bidders using different ordering of items produced by different methods, and then compare the resulting revenues. Since this evaluation method is not feasible for us, nor is it the main purpose of this paper, we opted for a widely accepted evaluation approach in research community, that is, we created

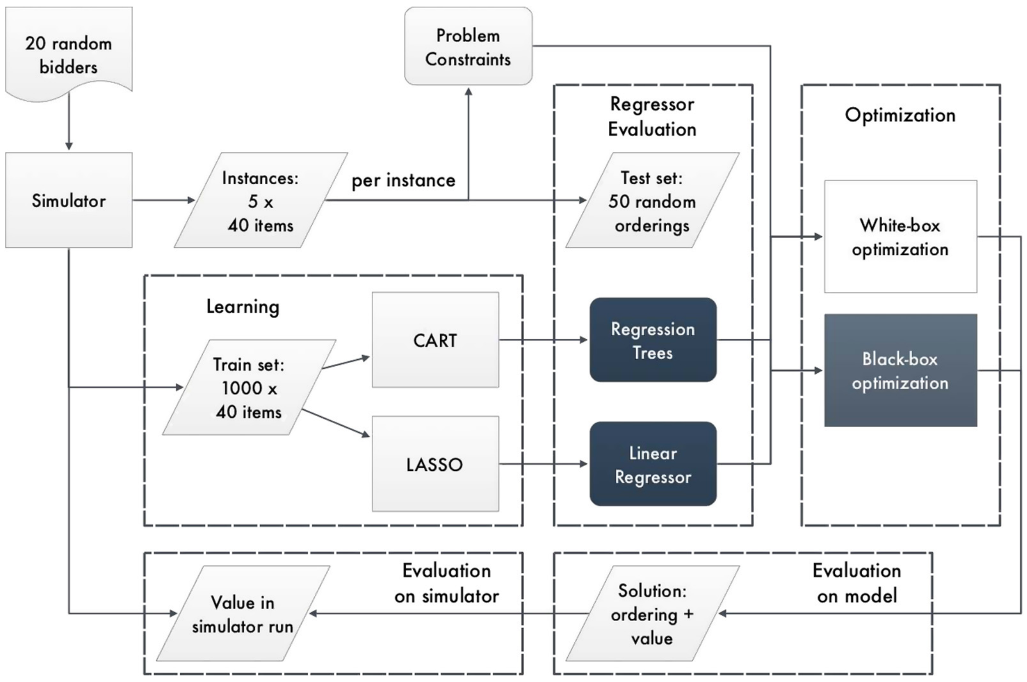


Fig. 3. Our framework for evaluating optimization using learned models for OOSA. There is a simulator that is used in two ways: to generate historic data and to evaluate the OOSA solutions. A train set and a few unseen problem instances are generated. The train set is used to learn the regression models. Random orderings of the problem instances are used to test them. The instances together with the learned models are provided as input to the ILP-based white-box optimization and the best-first black-box optimization. The resulting orderings are evaluated using the learned models, and using simulator runs.

an auction simulator which simulates auctions with agents. We used this simulator to generate auction data sets, and to evaluate the proposed method. An overview of this process is given in Fig. 3.

5.1. The simulator

Simulating auctions We simulate several sequential auction settings with the simulator: (i) first price auctions where agents bid the lower value between the amount they are willing to pay for the item, which is no higher than their valuation on the item, and their remaining budget, as in [11]; (ii) second price sealed bid auctions, i.e. Vickrey auctions [22]. Agents bid truthfully on each item in each round based on their valuations, or in case of insufficient budget, they bid their remaining budget, as in [7]. This is the best-response bidding strategy for myopic utility-maximizing agents who only consider the current round of the auction.[8] (iii) Vickrey auctions where agents bid smartly, i.e., they compare the utility obtained at the end of the auction when buying and not buying the item and place a bid based on the difference (see Section 5.4 for more details). On the last auctioned item, they bid truthfully if the budget allows.[9] Otherwise, they bid their remaining budget. Given all bids on an item, the highest bid wins. If multiple agents have the same highest bid, one of these is selected as winner uniformly at random. With these different auction settings, we intend to show our method is robust to the auction rules and bidding strategies. Below we explain how we generated agents and items for these settings and what parameters we used. At the end of this chapter, we will show how well our method scales when we use different parameters for the generator.

Item types We use a given set of 8 items to initialize the auction simulator. Every type r_i gets assigned a base value μ_i of $25 + (5 \cdot i)$, for $1 \leq i \leq 8$, and a reserve price $\rho_i = \frac{1}{2}(25 + (5 \cdot i))$. Every type is assigned popularity and sparsity values, denoted by γ_i and λ_i, drawn uniformly from $[2, 10]$. The popularity value measures the degree of desirability of the item type by the agents. The sparsity is a measure for the frequency that an item type is available in one auction. In every auction, 40 items are generated using a roulette wheel drawing scheme using the sparsity values.

Bidder agents The simulator starts with 20 randomly generated bidding agents. Every such agent A_j gets assigned a budget b_j between 25 and 150 uniformly at random. They may desire 1 to 5 of the 8 item types, where popular types have

[8] Note that in sequential Vickrey auctions with budget constrained agents, truth-telling is not an equilibrium bidding strategy (see [23]).
[9] Vickrey [22] showed that in a sequential auction with unlimited budget, it is a weakly dominant strategy for bidders to bid their true values for the last auctioned item.

a higher probability of being selected, drawn using a roulette wheel selection on the item types' popularity values. Every desired item type r_i assigned to an agent A_j is also given a value of $\nu_j(r_i) = \beta \cdot \mu_i$, where μ_i is the base value of type r_i, and β is a uniform random value between 0.5 and 2.0. The value $\nu_j(r_i)$ is used as the amount that agent A_j is willing to pay for r_i in the first-price auctions, and is used as the valuation of A_j on r_i in the second-price auctions. If this value is greater than the budget of the agent, that agent's budget is increased by another value between 25 and 150 (sampled uniformly), and adding this to its budget. This is repeated until the budget is sufficient for every item type.

Example 4. The following is an example of eight agents $A_1, \ldots, A_8$ and four item types r_1, r_2, r_3, r_4 generated for the small-scale experiments in Section 5.5.1, where for the four item types, the reserve prices of the auctioneer are: $\rho_1 = 12.5$, $\rho_2 = 15$, $\rho_3 = 17.5$, and $\rho_4 = 20$; the sparsity values of the four item types are: $\lambda_1 = 2$, $\lambda_2 = 7$, $\lambda_3 = 2$, and $\lambda_4 = 5$; and the popularity values are $\gamma_1 = 8$, $\gamma_2 = 8$, $\gamma_3 = 6$, and $\gamma_4 = 2$. Every agent's budget is sampled between 25 and 80.

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
budget b	78	37	80	60	119	103	46	63
$\nu(r_1)$	34	24	20	41		38	24	
$\nu(r_2)$	59	30		21		58	25	42
$\nu(r_3)$			61	30	53			
$\nu(r_4)$	74		22					

In the generated agents, we can clearly see that some item types are more popular than others. Item type r_4 is the least popular, being desired by only two agents, caused by the low popularity value of 2. The valuations are sampled uniformly using the base values. Agent A_5 has a budget greater than 80, due to the budget resampling. Five examples of the generated item sequences with corresponding revenues are:

index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
item	r_2	r_4	r_1	r_2	r_2	r_1	r_2	r_4	r_4	r_2	r_2	r_4	r_2	r_2	r_3
price	59	22	41	58	45	24	42	22	22	25	21	20	21	19	53
item	r_2	r_2	r_2	r_4	r_4	r_4	r_2	r_4	r_2	r_4	r_4	r_2	r_2	r_1	r_4
price	59	58	45	22	22	22	42	20	30	20	20	25	21	39	20
item	r_1	r_1	r_4	r_4	r_2	r_4	r_2	r_2	r_4	r_2	r_2	r_4	r_1	r_2	r_2
price	41	38	74	22	58	22	42	30	22	25	20	20	21	21	19
item	r_3	r_4	r_1	r_3	r_4	r_4	r_4	r_4	r_3	r_4	r_2	r_1	r_2	r_1	r_3
price	61	74	41	53	20	20	20	20	53	20	58	38	42	24	19
item	r_2	r_3	r_2	r_4	r_3	r_3	r_1	r_3	r_2	r_3	r_2	r_4	r_4	r_3	r_2
price	59	61	58	20	53	53	41	19	45	19	42	20	20	17.5	30

In the sequences, items of type r_1 or r_3 occur the least frequent, due to the low sparsity values. □

Training For a given set of 20 random bidders, the simulator generates 1000 historical auctions. The 40 items in these auctions are generated using the above scheme, and ordered randomly. These items are run in the simulator, where the agents use the above-mentioned bidding strategies to decide what value to bid, in order to determine the winners and the item selling prices. The total selling price of all items in one sequential auction is the collected revenue. For the 20 generated bidders, we first experiment the effect of different item orderings on the collected revenue by trying 100 random orderings and comparing the smallest, median, and largest collected revenue. If the difference between the largest and smallest is less than one tenth of the median revenue, 20 new bidders are generated. This process is repeated until we find a set of agents that passes this check, which typically occurs after a few iterations. By performing this check, we remove irrelevant problem instances. For the 20 agents that pass this check, we generate 1000 auctions of 40 items and simulate these auctions together with the agents. The resulting sequences of item-price pairs are then transformed to the features discussed in Section 3, and the resulting data set is used to train the regression trees and linear regressors.

Testing For the same set of 20 bidders, we generate 5 sets of 40 items, which are used for testing. First, the regressors are tested by comparing their predictions with the revenues generated by the simulator on 50 random orderings of each of these item sets. Second, we translate each of the item sets into constraints for both the black-box and white-box optimization solvers. The best ordering found by these solvers are compared based on their values on the regression model, and in the simulator.

5.2. Experimental setup

In each experiment, we generate agents and items as described above. We use an implementation of regression trees and LASSO from the scikit-learn machine learning module [19] in Python to learn (and evaluate) the regressors. We learn trees of different depths of 3, 5, 8 (we call them `tree3`, `tree5`, `tree8`), and we set the minimum number of samples required to split an internal node to 10. The LASSO regressor is run with 3 different values for α: 1.0, 0.1, and 0.000001 (`lasso1`, `lasso2`, `lasso3`), the tolerance threshold to test for convergence is set to 0.0001 and we use a maximum of 100 000 iterations. The resulting trees and linear models then get translated to ILP, which in turn gets solved by an ILP-solver (CPLEX [24]). In addition, we provide the solver with an initial solution (the best of 1000 random orderings) in order to start the search, and set the focus of the solver to finding integer feasible solutions. We set a time limit on the ILP solver of 15 minutes for each instance using a single thread on an Intel core i-5 with 8 GB RAM and record the best ordering of items that the ILP solver has obtained. The last minute is spent on solution polishing (a local search procedure in CPLEX). We apply our best-first search method on the same problem instances with the same running time limit.

Evaluations There are two levels of evaluations involved in our problem. Firstly, we determine the quality of the learned regressors, as they influence the quality of the solution after optimization. For this, we tested different regression trees with different maximum depths and linear regression models with different parameters.

Next, the optimization methods are evaluated in terms of the quality of the produced ordering. The optimization methods that we compare include the proposed white-box ILP model which finds a solution based on the abovementioned 6 regression models, the proposed black-box best-first search which evaluates a solution based on the 6 regression models, and in addition, two other simple ordering methods: (i) auctioning the most valuable item first (i.e., `mvf`), as suggested in [11][10]; and (ii) a random ordering strategy (i.e., `mean5000`), as seen in many real-world auctions for the purpose of fairness.

It is not feasible for us to compute the best solution given the problem size. Thus, we obtain a lower bound on the optimal solution as follows: given a set of items, we generate 5000 random orderings, and we use the *true* model (i.e., the simulator) to evaluate them and pick the one returning the highest revenue. We use the mean value of these 5000 random orderings as the output of the random ordering strategy.

We evaluate the 15 ordering methods in two ways:

- Model evaluation: we use the learned regression models to evaluate the solutions returned by the ordering methods to compute the predicted revenues.
- Actual evaluation: we run auctions with the solutions (i.e., orderings) returned by the ordering methods in our simulator to obtain the corresponding revenues. Note that such an evaluation is possible only when a simulator is available.

There are in total three sets of main experiments, as well as additional experiments, presented in this paper.

Experiment 1 We simulate a first-price auction where agents bid the minimum value between their budget and the amount that they are willing to pay. The winner pays her winning bid.

Experiment 2 The simulator runs the Vickrey auction with myopic agents, where agents bid the minimum between their true values on each item and their remaining budget. The winner pays the higher value between the second highest bid and the reserve price of the winning item.

Experiment 3 The simulator runs Vickrey auctions with smart agents, where agents bid smartly based on their expected utility at the end of the auction (see Section 5.4).

Experiment 4 In addition, we investigate the scalability of our approach using the following experiments: (1) first, we test the small instances to show how close the orderings found by the learned models are to the actual optimum; (2) second, we generate a randomized population of bidders to test whether our method can handle the cases where the bidders in different auction vary a lot; (3) third, we use a larger set of items and item types to demonstrate how well our approach can be expected to perform in larger auctions with many more items of greater diversity.

5.3. Experiment 1: first-price auctions

We run 60 sets of experiments. For each set of experiment, we generate a set of agents, and generate and run new sets of items 5 times. This results in 300 models for each learning method.

Prediction accuracy We first report the performance of the learned trees and linear models in terms of prediction accuracy. Given the same set of items as used during learning, we randomly generate 50 permutations of these items as orderings and compute the predicted values of these orderings using the learned models. Thus, for every set of agents, there are $50 \times 5 \times 40 = 10\,000$ bids to predict. These predictions are compared with the evaluated values of the orderings by the simulator. We report the coefficients of determination, or R^2 scores, in Fig. 4. This coefficient is a standard measure for comparing regression models and is defined as:

[10] We simply ordered the items according to their base values. We also tested ordering them based on their mean value in the data, but this performed worse in the test.

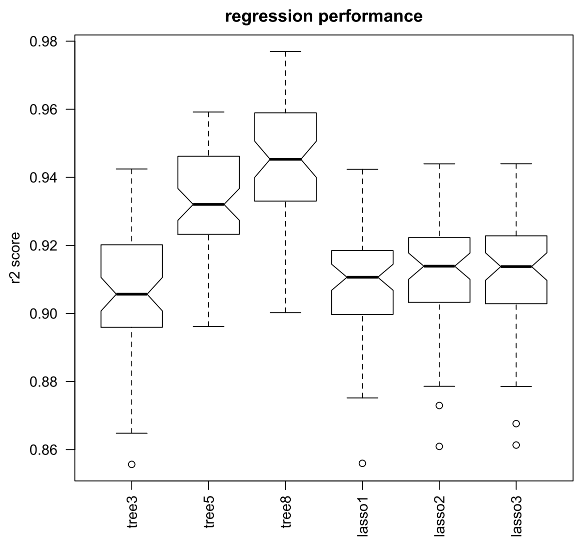


Fig. 4. Prediction performance of different learning models. The y-axis depicts for each learning model, the obtained prediction accuracy, i.e., R^2 scores (see Equation (16)). Each box contains 60 values from the 60 different sets of experiments.

Table 2
The frequencies of wins of 60 runs for each method against others (row method vs column method) in terms of R^2 scores.

	tree3	tree5	tree8	lasso1	lasso2	lasso3	Total wins
tree3	0	0	0	27	21	22	70
tree5	60	0	2	60	58	58	238
tree8	60	58	0	60	60	60	298
lasso1	33	0	0	0	7	9	49
lasso2	39	2	0	53	0	28	122
lasso3	38	2	0	51	32	0	123

$$R^2 = 1 - \frac{\sum_{1\le d\le k}(p^d - p(f_1^d, \ldots, f_m^d))^2}{\sum_{1\le d\le k}(p^d - \mathtt{mean}(p))^2}, \tag{16}$$

where k is the number of samples, p^d is the dth data value (i.e., the revenues returned by the simulator), $p(f_1^d, \ldots, f_m^d)$ the predicted values (i.e., the predicted revenues returned by the learning models: `tree3lp`, `tree5lp`, `tree8lp`, `lasso1`, `lasso2`, `lasso3`), and `mean`(p) the mean of all data values. Each score is computed from 10000 values in Fig. 4. A large value (close to 1.0) means the regressor is an almost perfect predictor, and smaller values indicate worse performance. Fig. 4 shows that all regression models lead to good prediction, with the lowest R^2 score over 0.86. The learned trees with depth 8 give the best performance, followed by the trees with depth 5. Intuitively a larger tree may give a better prediction. The scores of `lasso2` and `lasso3` are very similar, and slightly better than `lasso1`. This result makes sense since LASSO with a higher regularization parameter α (i.e., `lasso1`) implies the use of less features, and hence, may have less prediction power. The tree with depth 3 shows much worse performance than the larger trees, and it is on average worse than the three linear regression models. This is confirmed by the frequencies of wins by comparing the R^2 scores in pairs in Table 2.

Performance of the ordering methods We now discuss the actual performance of the different ordering methods. After every ordering method returns its best ordering, these orderings are then evaluated in the simulator to get corresponding revenues. As we ran 300 different instances (60 sets of different agents, each with 5 sets of different items), each method has 300 such revenues. We calculate the frequencies of wins by comparing the revenues in pairs in Table 3. One obvious conclusion from the table is that the ordering heuristic `mvf` (i.e., most valuable item first) performs worst, regardless of which method it compared to. In fact, this heuristic performed even worse than the random ordering strategy `mean5000` (123 wins vs. 174).[11] This result contradicts the theoretical finding that was concluded using much simpler auction settings. Another observation is that given the learned models, the developed white-box methods win over the black-box methods more than half of the time. This holds consistently for all 12 proposed methods (wins lp vs. bf: 171, 197, 178, 185, 195, 189). It shows that our new way of utilizing the internal structure of the learned models for optimization is promising.

[11] Notice that there is one instance where `mean5000` is better than `best5000`. This is due to the random scheme that we used in selecting winners who give two identical bids. Consequently, evaluating the same ordering twice in the simulator may result in two different revenues. We want to point out that this does not happen often and the effect is often negligible.

Table 3
The frequencies of wins of 300 runs for each method against others (row method vs column method), evaluated in the simulator.

	tree3lp	tree3bf	tree5lp	tree5bf	tree8lp	tree8bf	lasso1lp	lasso1bf	lasso2lp	lasso2bf	lasso3lp	lasso3bf	mvf	best5000	mean5000	total wins
tree3lp	0	171	105	152	107	125	102	130	71	127	82	119	231	17	240	2068
tree3bf	125	0	83	129	81	105	71	116	61	94	67	86	226	15	230	1770
tree5lp	192	213	0	197	149	168	144	169	111	156	128	164	249	24	273	2632
tree5bf	142	163	97	0	105	122	94	134	74	118	90	115	238	16	241	2037
tree8lp	188	217	142	193	0	178	137	181	113	160	128	162	260	23	287	2668
tree8bf	170	192	130	168	121	0	112	157	91	133	106	138	251	28	269	2361
lasso1lp	193	226	150	202	160	183	0	185	113	178	136	170	263	38	269	2756
lasso1bf	168	181	127	158	113	139	100	0	88	135	91	131	233	18	251	2219
lasso2lp	224	236	184	225	182	205	174	204	0	195	159	199	265	40	286	3073
lasso2bf	171	200	141	177	134	163	115	155	94	0	107	124	243	17	273	2407
lasso3lp	214	231	169	204	167	188	155	201	126	184	0	189	257	36	281	2899
lasso3bf	176	210	131	182	134	161	124	154	89	156	96	0	245	15	276	2439
mvf	68	73	51	61	38	48	36	65	34	55	41	54	0	9	123	982
best5000	282	284	268	284	277	271	257	280	255	278	252	280	291	0	299	4158
mean5000	57	65	27	58	13	30	31	48	12	26	18	24	174	1	0	864

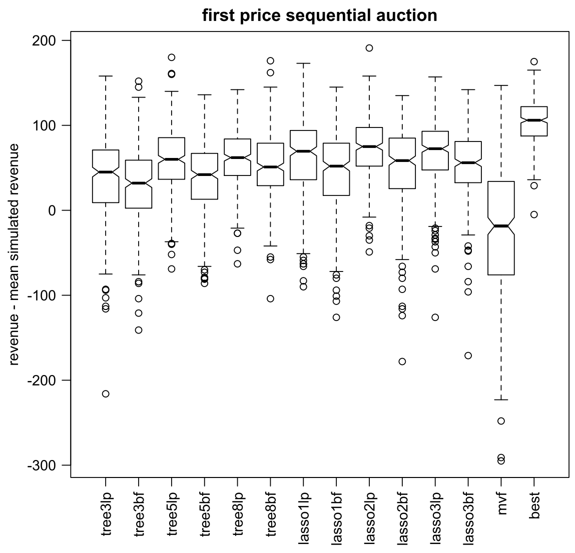


Fig. 5. The performance of the different ordering methods evaluated by the simulator, compared to `mean5000`. The simulated auctions are first-price auctions. Each box contains 300 values.

If we look at the results of the white-box methods built from the learned regression trees, i.e., `tree3lp`, `tree5lp`, `tree8lp`, we notice that `tree5lp` and `tree8lp` performed similarly and they are slightly better than `tree3lp`. This result is consistent with the higher R^2 scores of the larger trees. Interestingly, despite their lower R^2 scores, the linear regression LP methods return good orderings especially `lasso2lp` and `lasso3lp` which are on average better than the three regression tree LP models. These are further confirmed with Fig. 5, which depicts the revenue differences in the simulator between the ordering methods and `mean5000`. It is more obvious from this figure that the proposed linear regression LP models are the better optimization methods than the tree LP models in practice. We believe that this is due to the cascading inaccuracies, caused by the `sum` feature that relates the predicted value to the predictions of earlier auctioned items (see discussions in Section 3.3). Due to the crisp boundaries in regression trees, the effect of these errors on the solution evaluation is much greater than using linear regressors. We believe the effect is smaller for larger trees because they are more accurate.

Fig. 6 shows the performance difference between the LP model and the best-first search, which are built upon the same learned regression model. The solutions are evaluated using the predictive models. The value differences between the white-box and the black-box methods are more significant on smaller trees than on bigger trees (`tree3` vs `tree5` vs `tree8`), and on linear models with higher regularization parameter than on lower regularization parameter (`lasso1` vs `lasso2` vs `lasso3`). This trend shown in the results is somehow expected. The smaller tree leads to a smaller LP model, which is easier to optimize by the solver and consequently gives a much better performance than the best-first search. Similarly, the linear regression with a higher regularization parameter α implies less feature values to compute

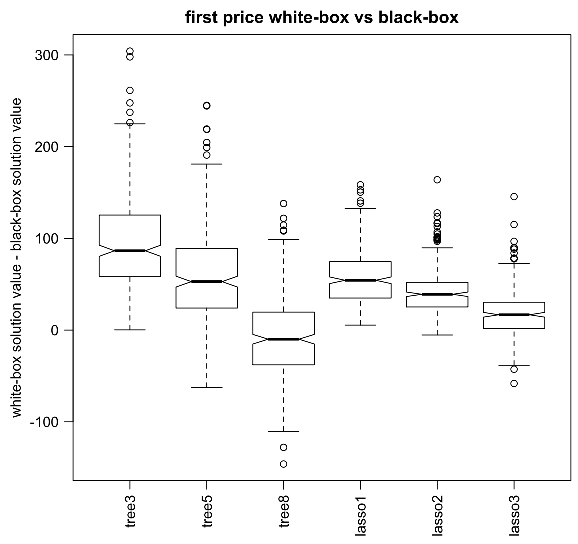


Fig. 6. The performance difference between the LP model and the best-first search, evaluated by the predictive model. Each box contains 300 values. The simulated auctions are first-price auctions.

during white-box optimization, and therefore, its advantage over the black-box method is more obvious than `lasso2` and `lasso3` which are with smaller values of α.

We observe that all white-box LP methods are better than the black-box methods, except the LP models resulting from the trees with depth 8. The depth 8 regression trees perform better for best-first search when evaluated on the model (Fig. 6), but better for LP when evaluated in the simulator (see Fig. 5). The most likely reason for the strange behavior of the depth 8 trees is that the best-first search outperforms LP on harder to predict instances.[12] Intuitively, because harder-to-predict instances typically result in larger models, they are harder to optimize in CPLEX. We checked this cause by investigating whether the R^2 scores of the depth 8 regression tree are correlated with which method performing better in the model evaluation. The mean of these R^2 scores are 0.950 when the LP performs better, and 0.942 when the best-first performs better. Although this difference seems small, it is significant.

Moreover, the small difference in R^2 scores between trees with depth 5 and depth 8 also has a significant effect on the difference between their model and simulator evaluations (e.g., due to cascading errors). To demonstrate this, we report in Fig. 7 the solution differences of the same ordering methods when being evaluated by the model and the simulator. The purpose of this comparison is to test whether the predicted outcome (using learned models) corresponds to the actual outcome (using simulator). This test is important as in general, there are no simulators available to evaluate the solutions. The figure demonstrates that the linear regression based optimization methods return more reliable solutions, i.e., their solutions evaluated on the learned linear models are closer to the solution values returned by the simulator. Note that it is logical that most values are over estimated because we the optimization tries to solve a maximization problem. The trees with depth 5 and 8 show a significant difference in this evaluation. The depth 3 trees end up with the highest evaluation difference, and overestimate the solution values the most.

5.4. Experiments 2 and 3: Vickrey auction with myopic and smart agents

In order to demonstrate that our method of auction optimization using learned models is robust to the used auction rule or the bidding strategies, we test it in a second-price auction in Experiment 2, and with smart agents in Experiment 3. We generate 10 sets of agents using the settings of the main experiments. For each set of agents we run new sets of items 5 times. Figs. 8 and 9 show the same plots for these settings as Figs. 5 and 6 for the setting in the first experiment.

In the second-price experiment, we test with truth-telling bidders in second-price auctions. The only differences with the setting in the first experiment are the bidding values and the payments. As Fig. 8 shows, the results are very similar to those in the first experiments: (1) all methods outperform the naive ordering strategies from the literature, (2) the white-box outperforms the black-box methods, and (3) the linear regression models perform best. However, the difference between the best performing methods `tree5` and `lasso2` is no longer significant.

[12] Another possible cause is that the LP solver makes small rounding errors. Such errors are unavoidable because using both very small and very large coefficients and/or precision in one problem formulation can cause numerical instability. We therefore round the mean values of the leaf predictions to two digits after the decimal point in the regression tree models before translating it to LP. Unfortunately, due to the crisp decision boundaries in regression trees, these small errors can sometimes have a large effect.

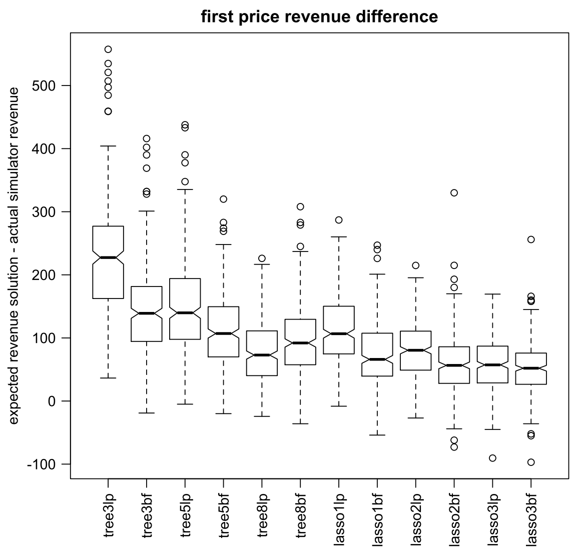
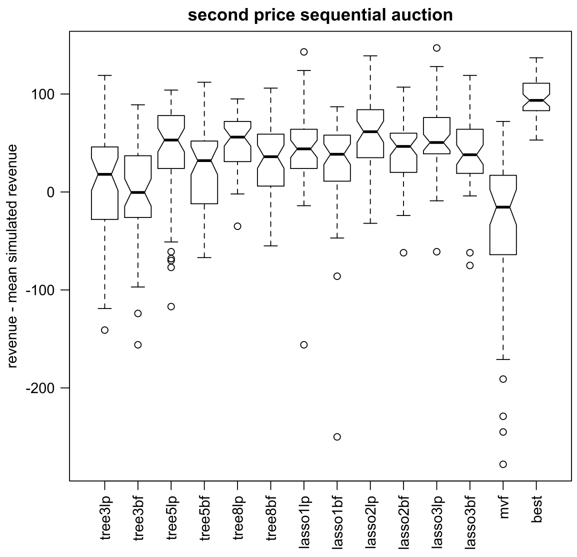


Fig. 7. The performance difference between the values evaluated by the model and the values evaluated by the simulator. Each box contains 300 values. The simulated auctions are first-price auctions.



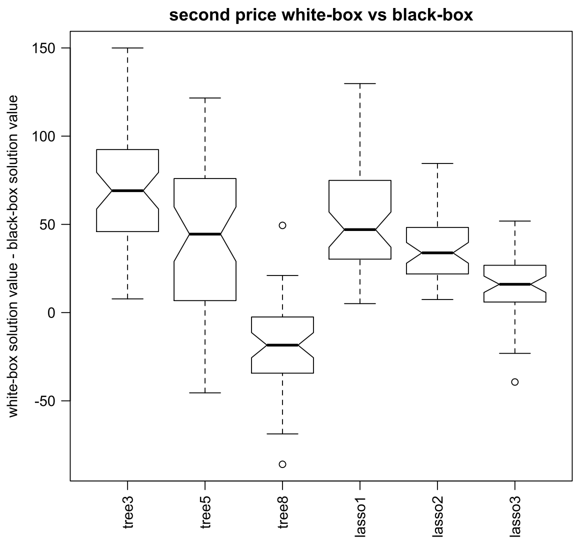


Fig. 8. Performance of the different ordering methods with second-price auctions and truth-telling, myopic agents. The two figures show the performance evaluated by the simulator and the model respectively. Each box contains 50 values.

In the third experiment, we test with smart agents that aim to maximize their final utility in a second-price auction. Specifically, when deciding what value to bid on the current item r_i, they have access to the auction simulator and use it to run the remaining items $I' \setminus r_i$. For computational reasons, when running the remaining items $I' \setminus r_i$, it is assumed that all agents bid truthfully and pay according to the second-price rule in these runs. For every bid r_i, they run the simulator twice: once in the situation where they bid the item r_i with their valuations (`run1`), and once where they do not buy the item (`run2`). They then decide what value to bid according to the following rules:

- If after `run2` the agent has a remaining budget greater than its value for the item, it bids truthfully. The intuition is that it is better to buy an item than to have budget left at the end of auction.
- Else, if the total utility after `run2` is less than after `run1`, it also bids truthfully. When it is better to buy an item, try to obtain it.
- Else, it bids its true value minus the difference in utility after `run2` and `run1`. When it is x monetary units better not to buy an item, try to obtain it for the value minus x.

Using these rules, the agents bid the highest value that they expect will give them an increase in utility.

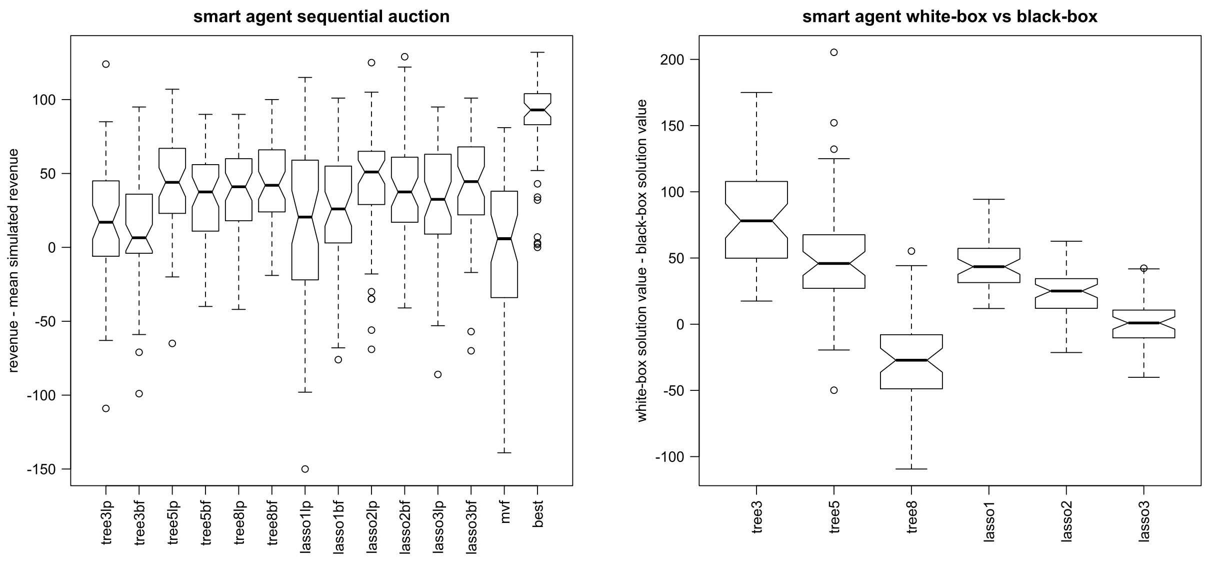


Fig. 9. Performance of the different ordering methods with second-price auctions and smart agents. The two figures show the performance evaluated by the simulator and the model respectively. Each box contains 50 values.

As can be seen in Fig. 9, also in this challenging setting, our method performs significantly better than the naive ordering rules from the literature. There is however a much larger variance in the performance of the different methods, causing the difference between black-box and white-box to be insignificant in the simulator. When evaluated on the model, however, white-box is still better than black-box in all cases except `tree8`. An interesting final observation is that, with these smart agents, the `mvf` method does seem to perform slightly better than `mean5000` (although not significantly), while in the other experiments it performed consistently worse.

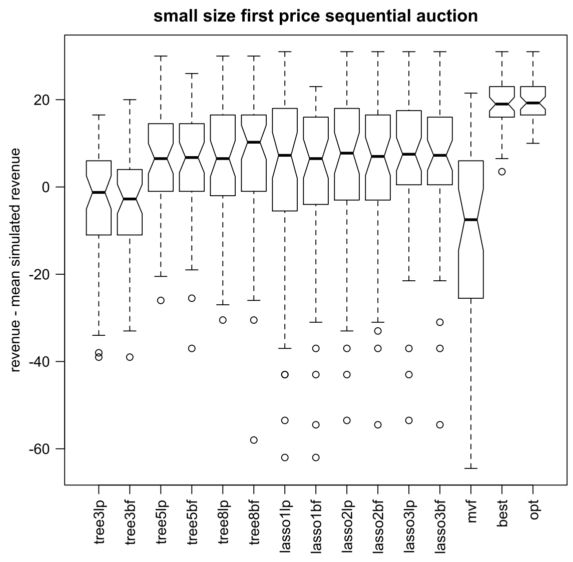
5.5. Experiment 4: practical issues

Although optimizing the orderings in sequential auctions is a hard problem, the above experiments demonstrate that high revenues can be obtained, significantly outperforming the naive methods proposed in the literature. This holds even in the presence of smart bidders. They also show the advantage of using the white-box method for optimization, an interesting trade-off between modeling and optimization power, and that better predictors are not necessarily better optimizers. In this section, we investigate several practical issues of our approach. We start with the easier instances, which will be used to show how close the orderings found by the learned models are to the actual optimum. Afterwards, we test our method using two new settings: a randomized population of bidders and a larger set of items and item types. The randomized population mimics a feature of a type of real-world auctions: the agents bidding in an auction are not always the same as often seen in online auctions. The larger instances aim to demonstrate how well our approach can be expected to perform in large-scale auctions with many more items of greater diversity.

5.5.1. Smaller auctions

The smaller problem instances are generated with: 1) a maximum budget of 80, 2) a maximum number of 3 desired items, 3) 4 item types, 4) 15 items per auction, and 5) 8 agents. All other settings are the same as those used in the main experiments. In addition, we added a brute-force method that uses the simulator as a black-box and tests all possible ordering of the items. Although a brute-force method is infeasible for 40 items, in the case of 15 items, there are only a few million possible orderings to consider. The results of 10 sets of agents, each tested with 5 sets of items, are shown in Fig. 10. The corresponding plots for the main experiments are Figs. 5 and 6.

The R^2 scores show a behavior similar to the main experiments, except that they range between 0.8 and 0.9. Outliers for the depth 3 trees can get as low as 0.5. This score decrease is likely caused by the decrease in data size (15 instead of 40 data rows per auction). The left plot if Fig. 10 shows the results obtained using the simulator. Any difference in performance between the different methods is insignificant except for depth 3 trees, which are significantly worse at estimating the obtained revenues and therefore result in lesser quality solutions. The improvement in performance of the other methods over random is approximately 10 on average, whilst about 20 could be obtained with perfect models in theory. Frequently, the black-box and white-box methods find an optimal solution for the learned models, so this difference is entirely due to the prediction quality of the models. It is good to see that the performance of the `mean5000` method is almost indistinguishable from optimal (the last boxplot in the figure), giving confidence that this is a good approximation of the upper bound. As can be seen in the right plot, the difference between the white-box and black-box methods are negligible for these small instances.



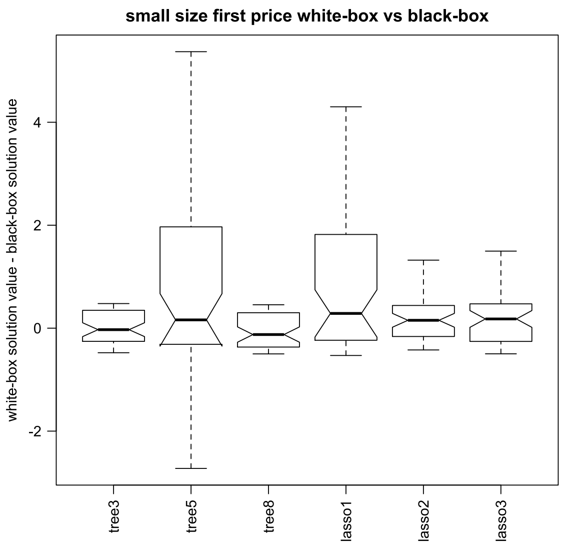
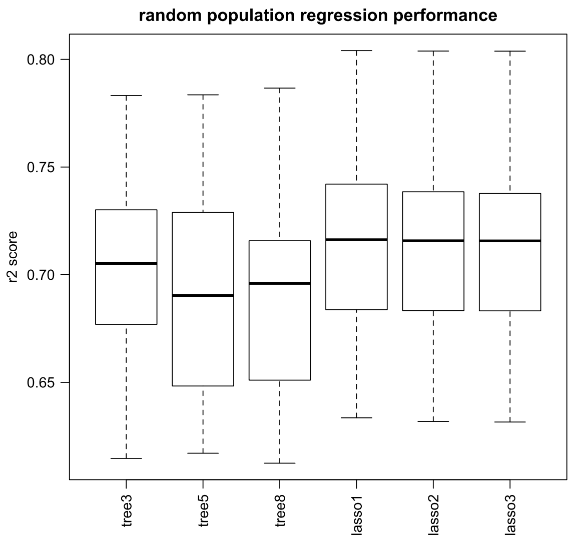


Fig. 10. Performance of the different ordering methods and a brute-force search (opt) for 15 items from 4 types. The figures show the performance evaluated by the simulator and the model respectively. Each box contains 50 values.



	tree3	tree5	tree8	lasso1	lasso2	lasso3	Total wins
tree3	0	9	17	6	6	6	44
tree5	10	0	15	4	5	5	39
tree8	2	4	0	0	0	0	6
lasso1	13	15	19	0	18	18	83
lasso2	13	14	19	1	0	17	64
lasso3	13	14	19	1	2	0	49

Fig. 11. Prediction performance of different learning models on the randomized population, including the number of wins table. Each box contains 19 values from the 19 different sets of experiments.

5.5.2. *Randomized population*

In this set of experiments, we obtain different sets of agents by creating a large population of size 60, and drawing 20 of them at random for every auction. In addition, we generate a random set of 20 to 60 items for every auction, sampled uniformly at random. Other than that, we use the exact same settings as the main experiments. We show the results of 19 sets of agents (we increased the set to reduce noise in the results), each being tested with 5 sets of items, see Fig. 12. The R^2-scores obtained in these experiments are shown in Fig. 11.

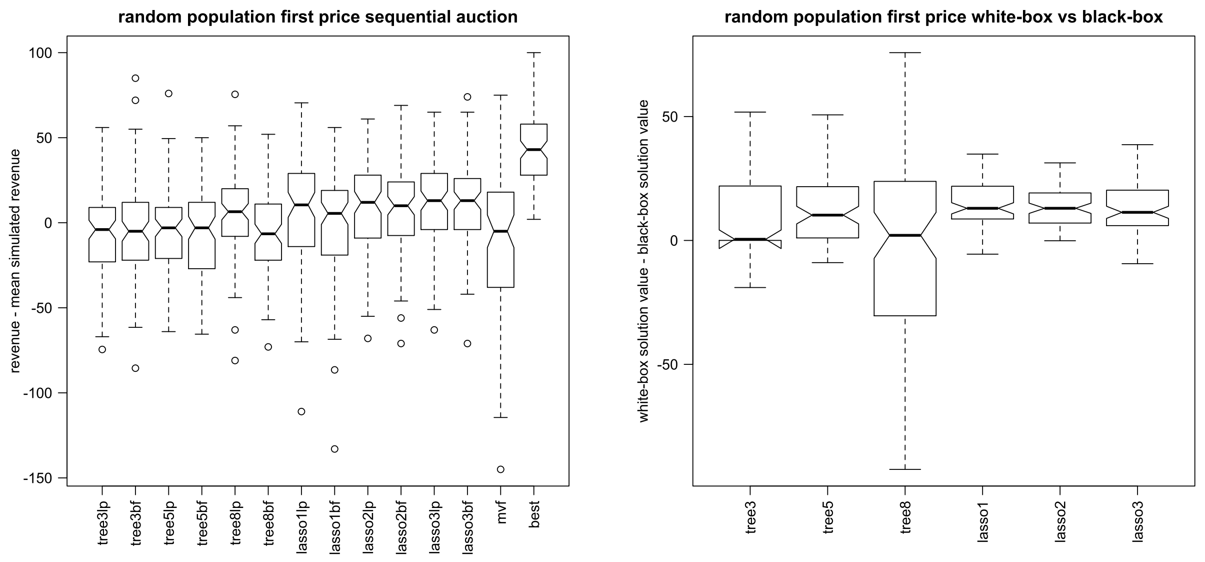


Fig. 12. Performance of the different ordering methods for 20 to 60 items from 8 types for a random population of 20 agents selected from a pool of 60. The two figures show the performance evaluated by the simulator and the model respectively. Each box contains 95 values.

From Fig. 11, we make two observations. Firstly, the performance of the predictors is much worse than in the main experiments, with median values around 0.7. As one may expect, the revenue is very hard to predict in this randomized setting, because it is heavily dependent on which agents take part in the auction. Learning larger trees does not increase this score. In fact, as the wins table shows, the depth 8 trees are clearly the worst predictor. Secondly, in contrast to the main experiments, linear regression appears to be a better estimator than a regression tree, although the difference is very small.

The simulator and model results in Fig. 10 confirm that the regression functions perform much worse in this randomized population setting. The linear regression models still perform better than random. The difference with the expected value of a random solution is so small that in cases with very uncertain populations it will probably not be worthwhile to model and optimize the problem. The model results show that also in these cases, white-box methods outperform black-box ones, with the only exception for depth 8 regression trees where the performance of white-box and black-box is similar.

5.5.3. *Larger problem instances*

The larger problem instances are generated with the same settings as those used in the main experiments, expect for: 1) 12 item types, 2) 80 items per auction, and 3) 40 agents. In addition, because the optimization problems are harder, we used a timeout of 30 minutes instead of 15. The results of 10 sets of agents, each tested with 5 sets of items, are shown in Figs. 13 and 14.

Notice that the performance for depth 8 trees is missing in Fig. 13. The reason for this is that the CPLEX runs for these trees ran out of memory. The encoding requires $n \times |R| \times (2^k + 1)$ variables, and $n \times |R| \times (2^(k + 1) - 1) + n + |R|$ constraints. If we fill in depth 8, 12 types, and 80 items, we end up with 46 720 variables and 490 652 constraints. This proved too much for our version of CPLEX (v12.5.1). Although the smaller trees did run, the depth 5 trees are now outperformed by the black-box approach. For depth 3 trees, it is still better to use white-box. A similar phenomenon is visible in Lasso results. In all previous results, white-box always outperforms black-box for the Lasso regressors. Suddenly, this is only the case for the smallest linear models, in those with more coefficients black-box performs better. These results match our intuition that using a white-box approach is beneficial when the models are not overly complex.

In the simulator results, we see that smaller models perform worse than larger ones, in spite of them being much harder to optimize. We believe this is due to them being worse predictors, as can be seen in the wins table in Fig. 14. Interestingly, for the larger linear models, the performance difference between black-box and white-box is no longer there. For the depth 5 tree, the difference is inverted, which we believe to be due to chance. However, it is also possible that this effect is caused by over estimation. For the black-box method, the average over estimation of solutions is 237, while it is 111 for the white-box method. This can be caused by the fact that CPLEX has problems finding solutions for these large instances, making it also harder to find very specific solutions that over estimate the solution value. Investigating this behavior further is left as future work.

Fig. 14 also shows an increase in R^2 scores: all median values are above the 0.92 mark. This is most likely cause by an increase in training data (twice as many items are sold per auction). The tree models are much better regressors for the large instances, although their optimization performance in the simulator is much worse. In fact, their performance is close to that of a random ordering. The larger linear models perform better than random, but are still far from optimal.

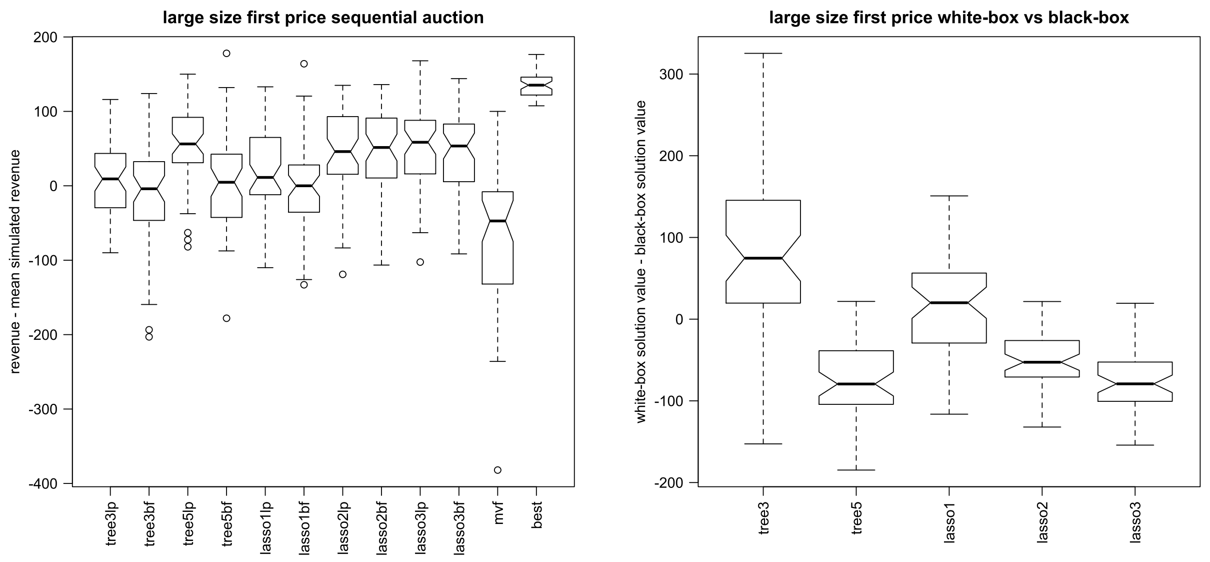
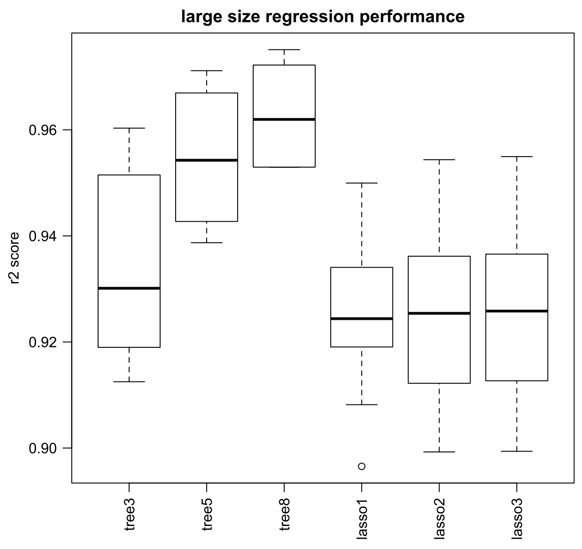


Fig. 13. Performance of the different ordering methods for 80 items from 12 types and 40 bidders. The two figures show the performance evaluated by the simulator and the model respectively. Each box contains 50 values.



	tree3	tree5	tree8	lasso1	lasso2	lasso3	Total wins
tree3	0	0	0	9	8	8	25
tree5	10	0	0	10	10	10	40
tree8	10	10	0	10	10	10	50
lasso1	1	0	0	0	1	1	3
lasso2	2	0	0	9	0	2	13
lasso3	2	0	0	9	8	0	19

Fig. 14. Prediction performance of different learning models for 80 items from 12 types and 40 bidders. The y-axis depicts for each learning model, the obtained prediction accuracy, i.e., R^2 scores (see Equation (16)). Each box contains 10 values from the 10 different sets of experiments.

6. Related work and discussion

We discuss related works and how our work contributes to and from several related research communities.

6.1. Interplay between mathematical optimization and machine learning

Many studies have investigated the interplay of data mining and machine learning with mathematical modeling techniques, see overview in e.g. [25–27]. Most of these investigate how to use data mining to estimate the value of parameters in decision making models or to replace decision model structure when it cannot be fully determined from the hypotheses at hand. For instance, Brijs et al. [28] build a decision model as an integer program that maximizes product assortment of a retail store. The decision model is then refined by incorporating additional decision attributes that are the learned patterns from recorded sales data. Li and Olafsson [3] use a decision tree to learn dispatching rules that are then used to decide which job should be dispatched first. These dispatching rules are previously unknown, and it is assumed that it is worthwhile to capture the current practices from previous data. Gabel and Riedmiller [1] model production scheduling problem as multi-agent reinforcement learning where each agent makes its dispatching decisions using a reinforcement learning algorithm based on a neural network function approximation.

Another line of work investigates how to use learning techniques during optimization in order to learn properties of good solutions. For instance, Defourny et al. [29] combine the estimation of statistical models for returning a decision rule given a state with scenario tree techniques from multi-stage stochastic programming. This line of work shares similarities with the field of black-box optimization, see, e.g., [14,15,30]. In black-box optimization, methods are used to approximate a function with unknown analytical form and which typically is expensive to execute. In contrast, in multi-stage stochastic programming this form is known but stochastic. An often applied technique for black-box optimization is the use of surrogate methods, see, e.g., [31]. Surrogates are approximations of the black-box function that are less expensive to execute. Typical examples include linear/polynomial regression, neural networks, and other methods from machine learning. These functions are trained during optimization from (as few as possible) black-box function calls.

As learning tasks can lead to challenging optimization problems, researchers have also applied mathematical optimization methods in order to increase learning efficiency. For instance, Bennett et al. [32] use linear programming for determining linear combination splits within two-class decision trees. Chang et al. [33] propose a Constrained Conditional Model (CCM) framework to incorporate domain knowledge into a conditional model for structured learning, in the form of declarative constraints. CCMs solve prediction problems. In [34], the authors build a mixed integer program for multi-class data classification. A comprehensive overview of optimization techniques used in learning is given in [35]. Researchers are also interested in using mathematical optimization methods in order to find entire models and rules, see e.g., [36–38].

Our approach fits in the first line of research of this interplay. The proposed best-first search method uses regression models to learn good orderings, which is then applied during search to evaluate the solutions of OOSA. Hence, similar to the works mentioned above, the models learned from data are used in a black-box fashion. This approach shares similarities with surrogate methods for black-box optimization. An important difference is that the (surrogate) models here are learned from data. Furthermore, our proposed white-box optimization method makes all the properties of the learned models visible to the optimization solver. Bartolini et al. [4] propose a similar method by translating neural networks into constraint programming (CP) models. Their approach is simple yet effective and allows to model complex relations (such as recurrent neural networks) between any pair of decision variables based on data. This is very powerful as it allows for multiple trained neural networks to be plugged into an existing CP model that is constructed by traditional means. It is up to the system designer which relations between variables to model traditionally and which to model based on data. In contrast, we use our translation to construct (parts of) the objective function of an existing MIP model using regression functions. To the best of our knowledge, we are the first to combine regression functions that are learned from data with a MIP model in this way.

Other closely related work from the CP literature considers the problem of constraint acquisition [39–42]. Given a large set of possible constraints L and some training data, the goal of constraint acquisition is to compose a constraint network (a graphical representation of a CP model, see, e.g., [43]) using the constraints in L such that it classifies all of the training data correctly, i.e., it should specify a language that includes all positive and excludes all negative training examples. Although this is a hard problem, several effective approaches have been proposed based on supervised [39], unsupervised [41], and active [40,42] learning. These approaches are based on their own new learning algorithms, some with proven performance bounds (e.g., [42]). Once learned, the constraints are embedded into an existing CP model and fully integrated into the optimization process. From a modeling perspective, this method has both benefits and downside. On one hand, constraint acquisition allows the modeler to define his or her own dedicated constraint set L instead of only the features/variables, potentially resulting in more sensible models for the problem at hand. On the other hand, the learning problem itself is arguably less understood than those tackled by existing algorithms from machine learning, potentially resulting less quality classifiers.

In addition, in probabilistic inference, an interesting combination of machine learning and optimization has also been proposed. First, a probabilistic model is learned in the traditional way. Then, when computing the posterior distribution over some target variables given new input data, additional constraints are added in order to limit the possible assignments to the targets. While the original probabilistic inference problem can often be solved using dynamic programming methods, the additional constraints make it much harder to solve, and it is therefore translated into a MIP. This approach has successfully been applied in order to add expert knowledge to conditional random fields for semantic role labeling by Roth and Yih in [44]. Later, the same principle was used to model the dependencies among textual tokens in text-based documents for entity recognition by Fersini et al. [45]. Interestingly, in this last work, the added constraints themselves were also learned from data and added as soft constraints to the MIP model.

6.2. Sequence models

As an auction ordering is essentially a sequence of items, our work is also related to the many machine learning approaches for sequence modeling. To the best of our knowledge none of the existing sequence models fits our auction setting. Language models such as deterministic automata [46] are too powerful since they can model every possible sequence independently and therefore require too much data to learn accurately. Short sequence models such as hidden Markov models or N-grams [47] do not model the dependence on items sold a long time (more than the sliding window length) before.

Markov decision processes (MDPs) (see, e.g., [48]) may be closest to our auction setting, as they can directly model the expected price per item and come with methods that can be used to optimize the expected total reward (revenue). However, we notice that a straightforward implementation of the auction design problem as an MDP is not possible. Let us try to model auction design as an MDP. Because of the Markov assumption, every state in this MDP has to contain all the relevant information for the auctioneer's decision on which item to auction: the set of available items, the bidders' valuation functions, budgets and strategies, and for every bidder the items (s)he already possesses. In every state q of this process, the auctioneer can choose what item i to put to auction from a multiset of available items I. The next state q' resulting from auctioning item i depends on the bidders and their valuations. These are unknown to the auctioneer, but probabilities can be used to estimate them. These probabilities $P_i(q, q')$ provide a distribution over the possible next states and the corresponding rewards $R_i(q, q')$, given i is auctioned. In every possible next state q', the set of available items is equal to $I - \{i\}$, i.e., equal to the items in q minus the sold item i. The goal of the auctioneer is to maximize the expected rewards (revenue) for a given set of items I. In every state q, (s)he thus has to take an action (choosing an item) that maximizes the sum of the expected rewards $V(q, I)$ of items in I starting in state q:

$$V(q, I) = \arg\max_{i \in I} \left(\sum_{q'} P_i(q, q') R_i(q, q') + V(q', I - \{i\}) \right).$$

In this equation, we separated I from q to highlight the major hurdle that needs to be overcome in order to represent the auction design problem as an MDP. I needs to be included in the MDP since it determines the set of available actions in every state. However, since the set of items is finite this makes the MDP acyclic and at least as large as the number of possible subsets of items from I (assuming the effect of their ordering is represented differently), i.e., at least $2^{|I|}$. In order to learn the rewards and transition probabilities, an auctioneer would therefore need an extremely large data sample.

This intuitively shows why it is difficult to represent the auction design problem as an MDP. However, with a suitable factored representation of the states and/or function approximation [48] of the rewards, it could be possible to represent our auction problem as an MDP. In this case, a major hurdle will be to find a representation that results in Markovian states, which is needed to apply the dynamic programming methods. Since the problem of deciding whether good auction ordering exists is NP-complete (Theorem 1), and these methods run in polynomial time, this is impossible without an exponentially large state space unless $\mathtt{P} = \mathtt{NP}$. Our method relies on solvers and search methods for NP-complete problems, making a polynomial state space possible, and therefore requiring much less data to estimate the model parameters.

6.3. Auction design

In the auction literature, a few existing papers investigate the impact of ordering on the performance of sequential auctions. One line of related research focuses on theoretical analysis. In the economics literature (see [9,12,11]), such theoretical studies were typically carried out under very restricted markets. The main research focus there is to analyze equilibrium bidding strategies of bidders who compete for (usually) two items (heterogeneous or homogeneous), and then to gain insights on the impact of ordering on the auction outcome based on derived bidding behaviors. For instance, Elmaghraby [9] studies the influence of ordering on the efficiency of the sequential second price procurement auctions, where a buyer outsources two heterogeneous jobs to suppliers with capacity constraints. Suppliers can only win 1 job in this setting. The author shows that specific sequences lower procurement costs and identifies a class of bidders' cost functions where the efficient orderings (i.e. the auction rewards the jobs to the suppliers with the lowest total costs) and equilibrium bidding strategies exist. Pitchik [12] points out that in the presence of budget constraints, a sealed-bid sequential auction with two bidders and two goods may have multiple symmetric equilibrium bidding functions, and the ordering of sale affects the expected revenue. If the bidder who wins the first good has a higher income than the other one, the expected revenue is maximized. Subramaniam and Venkatesh [11] investigate the optimal auctioning strategy of a revenue-maximizing seller, who auctions two items, which could be complements or substitutes. They show that when the items are different in value, the higher valued item (among the two) should be auctioned first in order to increase the seller's revenue. A similar revenue-maximizing strategy is proposed by Benoit and Krishna [49] in a complete information auction setting. The authors conclude that in such a setting, when selling two items to budget constrained bidders, it is always better to sell the more valued item first. However, this strategy does not optimize the revenue anymore when more than two items are to be auctioned.

Empirical research has been conducted to test the theoretical findings in the economics community. Grether et al. [10] report on a field experiment that tests the ordering strategies of a seller in sequential, ascending automobile auctions. They conclude that the worst performing ordering in terms of revenue is for the seller to auction vehicles from highest to lowest values.

In the computer science literature, Elkind and Fatima [13] study how to maximize revenue in sequential auctions with second-price sealed-bid rules, where bidders are homogeneous, i.e., all their valuations are drawn from public known uniform distributions, and they want to win only one item (but they can bid any of items). In this setting, the authors analyze the equilibrium bids, and develop an algorithm that finds an optimal agenda (i.e., ordering). Vetsikas et al. [50] study a similar auction setting, but unlike [13], they assume the valuations are known to the bidders at the beginning of the auction. The focus of their work was to compute the equilibrium strategies for bidders. Later, Vetsikas [23] analyzes the bidding strategies for budget constrained bidders in sequential Vickrey auctions. However, it is a challenge to compute the equilibrium strategies in practice.

We are not the first who consider learning from the previous auctions. However, the difference lies in the fact that the most existing work study how bidders learn from the past information, and update their bids. Boutilier et al. [51] propose a learning model for bidders to update their bidding policies in sequential auctions for resources with complementarities. The bidding strategies are computed based on the estimated distribution over prices, that is modeled by dynamic programming. Goes et al. [52] present an empirical study of real sequential online auctions. They analyze the data from an online auction retailer, and show that bidders learn and update their willingness to pay in repeated auctions of the same item. In [7], the authors show the benefits of using earlier auction data for the management of sequential, multi-unit auctions, where the seller needs to split its entire inventory into sequential auctions of smaller lots in order to increase its profit. In their work, an auction feedback mechanism is developed based on a Bayesian model, and it is used to update the auctioneer's beliefs about the bidders' valuation distribution.

Our contribution to the auction literature lies on the fact that our approach can be applied to design optimal auctions based on historical auction data. The advantage of using machine learning and data mining methods is that they are robust to the uncertainty (or noise), and hence have high potential to be used for real-world auction design. Moreover, the approach itself is general and can be applied to many different auction optimization problems, such as finding best reserve price for items for sale, or maximizing social welfare instead of revenue. The necessary changes may include the selection of the features for learning regression models and the encoding in the white-box optimization model.

7. Conclusions

Mathematical optimization relies on the availability of knowledge that can be used to construct a mathematical model for the problem at hand. This knowledge is not always available. For instance, in multiagent problems, agents are autonomous and often unwilling to share their local information. Frequently, this autonomy and private information influence the outcome of the optimization, making finding an optimal solution very difficult. In this paper, we adopt the idea of using machine learning techniques to estimate these influences for an optimization problem with many unknowns: the optimal ordering for sequential auctions (OOSA) problem.

We have demonstrated our approach by transforming historical auctions into data sets for learning regression trees and linear regression models, which subsequently are used to predict the expected value of orderings for new auctions. We proposed two types of optimization methods with learned models, a black-box best-first search approach, and a novel white-box approach that maps learned models to integer linear programs (ILP). We built an auction simulator with a set of bidder agents to simulate an auction environment. The simulator was used for generating historical auction data, and for evaluating the orderings of items returned by our methods. We ran an extensive set of experiments with different agents and bidding strategies. Although optimizing the orderings in sequential auctions is a hard problem, our proposed methods obtained very high values, significantly outperforming the naive methods proposed in the literature. The experimental results also demonstrate the advantage of using the white-box method for optimization, which significantly outperforms the black-box approach in nearly all settings. In addition, they indicate that when the learned model becomes more complex, it potentially results in more constraints and consequently, an increase in the time needed to solve the problem in a white-box fashion. Since more complex models are (potentially) better predictors, this shows a clear trade-off between modeling and optimization power in white-box optimization. In our opinion, the benefits of the white-box approach largely outweigh the benefits of using black-box optimization.

Finally, the extended experiments demonstrate that although our encodings are efficient, the regression tree breaks down when the data becomes too noisy. An intriguing extension would therefore be to use regression forests instead of individual trees. These are known to handle noisy data much better because of the crisp boundaries in individual trees. The same experiments also show that our method does not yet scale very well with the number of items, most likely due to the increase in the number of trees that need to be evaluated. We expect that a method based on regression forests will therefore require several simplifications or optimizations in order to be feasible.

Besides an improved performance, a very big benefit of the white-box formulation is that it provides a new way of obtaining traditional mathematical models. Our method therefore has many other potential application areas, especially in problems where more and more data is being collected. Even in cases where there already exists a handcrafted optimization model, a model that is learned and translated using our method can easily be integrated into existing (I)LP formulations in order to determine part of the objective function based on data. In this way, one can combine the vast amount of expert

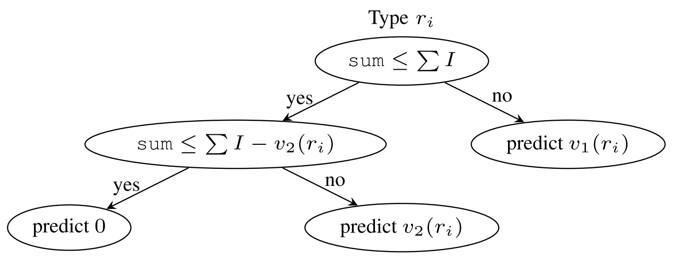


Fig. 15. The regression tree for every item type r_i used to model a partitioning problem.

knowledge available in these domains with the knowledge in the readily available data. We would like to investigate this combination in the future.

We chose a relatively simple auction model for ease of explanation in this paper. However, our approach works whenever regression models are able to provide reliable predictions of the bidding values. Hence we believe it can be applied to other auction formats with more complex valuation functions (i.e. combinatorial preferences [53]) and more complex bidding strategies. The results of our method on the larger experiments with 80 items shows that scaling the approach up to large real-world auctions will require several non-trivial simplifications. Moreover, in this paper, we learned regression trees and linear models from simulated data in order to test the optimization performance. When applying our approach to real-world data, it is important to test whether the regressors assumptions are satisfied. If not, it may be needed to transform or filter them. We plan to discover the simplifications and test our approach with real auction data in the near future.

Our experiments highlight some interesting properties of the white-box method. Firstly, they show an improvement in performance when the number of features is reduced and/or the models are less complex. It would therefore be very interesting to investigate the effect of pruning and feature selection or reduction on the performance of our methods. Secondly, they show a tendency of the regression tree optimizer to overestimate, i.e., find orderings that have a much higher expected revenue than their revenue in practice. Intuitively, the solver abuses the crisp nature of the regression tree in order to find a solution that satisfies exactly the right constraints. Part of the problem is that, although these constraints are learned from data, and therefore uncertain, the solver treats them as exact. Fortunately, there exists a long history of methods that try to optimize in the presence of such uncertainties in the area of robust optimization, see, e.g., [54]. As future work, we will investigate the potential uses of these techniques for learned models.

Recently, regression tree models with linear models in the leaf nodes have also been successfully used as black-box surrogate functions [55]. Since it is also straightforward to translate these trees given our two encodings (replace the leaf variables by indicators for which linear function to use), it would be very interesting to investigate the possibility of a white-box alternative.

Acknowledgement

Sicco Verwer is supported by Technologiestichting STW VENI project 13136 (MANTA). We thank the reviewers for their valuable suggestions and comments, which have significantly improved the quality of the paper.

Appendix A. Hardness of auction design using learned predictors

We show that using predictive models instead of agents with utility functions does not reduce the complexity of the problem: it remains NP-complete for both regression trees and linear regression predictors.

Lemma 1. *Using regression trees, the problem of whether there exists an ordering that has a total predicted value of at least K is NP-complete.*

Proof. The proof follows from the fact that we can use simple regression trees to model the preferences of the two agents from Theorem 1, and evaluating an ordering using these trees can be done in polynomial time. The regression tree for every item type r_i is shown in Fig. 15. □

Lemma 2. *Using linear regression predictors, the problem of whether there exists an ordering that has a total predicted value of at least K is NP-complete.*

Proof. We prove the lemma using a construction for computing the value of a quadratic function using only linear functions, the ordering problem, and our feature values. The maximum value of this quadratic function is then forced to coincide with the solution of a partition problem instance: Given a set of integers $I = \{v_1, \ldots, v_n\}$, is I dividable into two sets A and B such that $\sum A = \sum B$?

From the partition instance, let $k = \frac{1}{2}\sum_{1 \le i \le n} v_i$, we construct the following items and linear regression predictors (functions $v()$):

- n items of type $x_1 \dots x_n$, with $v(x_i) = v_i - \frac{v_i \cdot \mathtt{sum}(y)}{2k}$, and
- 1 item of type y, with $v(y) = 2k - \sum_{1 \le i \le n} \mathtt{sum}(x_i)$.

The objective is to maximize $\sum_{1 \le i \le n} v(x_i) + v(y)$. The corresponding decision problem is to ask whether there exists an ordering that achieves a value of $2\frac{1}{2}k$.

($\Rightarrow$) Let A, B be a partition of I such that $\sum A = \sum B$, and let $a_1, \dots, a_{|A|}$ and $b_1, \dots, b_{|B|}$ be the corresponding items of type $x_1 \dots x_n$. The following ordering then gives a value of $\sum_{1 \le i \le n} v_i$:

$$a_1 \ \dots \ a_{|A|} \ y \ b_1 \ \dots \ b_{|B|}$$

In this ordering, $\sum_{1 \le i \le |A|} v(a_i) = k$ by definition of A and since $\mathtt{sum}(y) = 0$ before item y is auctioned. Consequently, $\sum_{1 \le i \le n} \mathtt{sum}(x_i) = \sum_{1 \le i \le |A|} v(a_i) = k$, giving $v(y) = 2k - k = k$. $\sum_{1 \le i \le |A|} v(a_1) + v(y)$ thus already obtains the objective value of $2k$, and therefore $\sum_{1 \le i \le |B|} v(b_1)$ should be equal to $\frac{1}{2}k$. By definition of B, $\sum_{1 \le i \le |B|} v(b_i) = k - \frac{k \cdot \mathtt{sum}(y)}{2k}$. Since $v(y) = k$, $\mathtt{sum}(y) = k$, and thus $\sum_{1 \le i \le |B|} v(b_i) = k - \frac{k \cdot k}{2k} = k - \frac{k}{2} = \frac{1}{2}k$, proving that the ordering obtains a value of $2\frac{1}{2}k$.

($\Leftarrow$) To prove the other direction, let us further analyze the relation between the objective function $\sum_{1 \le i \le n} v(x_i) + v(y)$ and the auction ordering. The only term in the $v(x_i)$ predictors that depends on the ordering is $\mathtt{sum}(y)$, all other terms are constants. This term is equal to zero for the x_i items auctioned before y, and equal to $v(y)$ for the items auctioned after y. Similar to the ($\Rightarrow$) part, let $a_1, \dots, a_{|A|}$ denote the x_i items before y, $b_1, \dots, b_{|B|}$ those after y, and A, B the corresponding partition of items in I. The objective value is then given by $\sum_{1 \le i \le |A|} v(a_i) + v(y) + \sum_{1 \le i \le |B|} v(b_i)$. We analyze these three parts in turn.

- $\sum_{1 \le i \le |A|} v(a_i) = \sum_{v_i \in A} v_i$ since $\mathtt{sum}(y) = 0$ for these items.
- $v(y) = 2k - \sum_{1 \le i \le |A|} v(a_i) = 2k - \sum_{v_i \in A} v_i = \sum_{v_i \in B} v_i$.
- $\sum_{1 \le i \le |B|} v(b_i) = \sum_{v_i \in B} v_i - \sum_{v_i \in B} \frac{v_i \cdot v(y)}{2k}$, since $v(y) = \sum_{v_i \in B} v_i$, this becomes $\sum_{v_i \in B} v_i - \frac{v(y)^2}{2k}$.

Since $\sum_{v_i \in A} v_i + \sum_{v_i \in B} v_i = 2k$, the overall objective function is given by:

$$2k + v(y) - \frac{v(y)^2}{2k}$$

which is maximized when $v(y) = k$ (for $k > 0$) with value $2k + k - \frac{k^2}{2k} = 2\frac{1}{2}k$. This exact value of $v(y) = k$ is obtained when $\sum_{v_i \in B} v_i = k$. The sets A and B thus give a partition of I. $\square$

Remarks. We proved the NP-completeness for the general case of Lemma 2. However, we do not know whether the complexity holds for more realistic valuation functions that bidders have.

References

[1] T. Gabel, M. Riedmiller, Adaptive reactive job-shop scheduling with learning agents, Int. J. Inf. Technol. Intell. Comput. 2 (4) (2008) 1–30.
[2] A. Huyet, Optimization and analysis aid via data-mining for simulated production systems, Eur. J. Oper. Res. 173 (3) (2006) 827–838.
[3] X. Li, S. Ólafsson, Discovering dispatching rules using data mining, J. Sched. 8 (6) (2005) 515–527.
[4] A. Bartolini, M. Lombardi, M. Milano, L. Benini, Neuron constraints to model complex real-world problems, in: J.H.-M. Lee (Ed.), Principles and Practice of Constraint Programming, in: Lecture Notes in Computer Science, vol. 6876, Springer, 2011, pp. 115–129.
[5] D. Bernhardt, D. Scoones, A note on sequential auctions, Am. Econ. Rev. 84 (3) (1994) 653–657.
[6] E. van Heck, P.M.A. Ribbers, Experiences with electronic auctions in the Dutch flower industry, EM 7 (4) (1997) 29–34.
[7] E.J. Pinker, A. Seidmann, Y. Vakrat, Using bid data for the management of sequential, multi-unit, online auctions with uniformly distributed bidder valuations, Eur. J. Oper. Res. 202 (2) (2010) 574–583.
[8] J. Gallien, L.M. Wein, A smart market for industrial procurement with capacity constraints, Manag. Sci. 51 (2005) 76–91.
[9] W. Elmaghraby, The importance of ordering in sequential auctions, Manag. Sci. 49 (2003) 673–682.
[10] D.M. Grether, C.R. Plott, Sequencing strategies in large, competitive, ascending price automobile auctions: an experimental examination, J. Econ. Behav. Organ. 71 (2) (2009) 75–88.
[11] R. Subramaniam, R. Venkatesh, Optimal bundling strategies in multiobject auctions of complements or substitutes, Mark. Sci. 28 (2009) 264–273, http://dx.doi.org/10.1287/mksc.1080.0394.
[12] C. Pitchik, Budget-constrained sequential auctions with incomplete information, Games Econ. Behav. 66 (2) (2009) 928–949.
[13] E. Elkind, S. Fatima, Maximizing revenue in sequential auctions, in: Proceedings of the 3rd International Conference on Internet and Network Economics, WINE'07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 491–502.
[14] D.R. Jones, M. Schonlau, W.J. Welch, Efficient global optimization of expensive black-box functions, J. Glob. Optim. 13 (4) (1998) 455–492.
[15] S. Shan, G.G. Wang, Survey of modeling and optimization strategies to solve high-dimensional design problems with computationally-expensive black-box functions, Struct. Multidiscip. Optim. 41 (2) (2010) 219–241.
[16] M.R. Garey, D.S. Johnson, Computers and Intractability – A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, 1979.
[17] L. Breiman, J. Friedman, R. Olshen, C. Stone, Classification and Regression Trees, Wadsworth and Brooks, Monterey, CA, 1984.

[18] R. Tibshirani, Regression shrinkage and selection via the lasso, J. R. Stat. Soc. B 58 (1994) 267–288.
[19] scikit-learn: machine learning in Python, http://scikit-learn.org/.
[20] T.G. Dietterich, Machine learning for sequential data: a review, in: Structural, Syntactic, and Statistical Pattern Recognition, Springer, 2002, pp. 15–30.
[21] S. Verwer, Y. Zhang, Revenue prediction in budget-constrained sequential auctions with complementarities, in: AAMAS'12, 2012, pp. 1399–1400.
[22] W. Vickrey, Counterspeculation, auctions, and competitive sealed tenders, J. Finance 16 (1) (1961) 8–37.
[23] I.A. Vetsikas, Sequential auctions with budget-constrained bidders, in: IEEE 10th International Conference on e-Business Engineering, IEEE, 2013, pp. 17–24.
[24] IBM ILOG CPLEX Optimizer, http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/.
[25] K.P. Bennett, E. Parrado-Hernández, The interplay of optimization and machine learning research, J. Mach. Learn. Res. 7 (2006) 1265–1281.
[26] S. Meisel, D. Mattfeld, Synergies of operations research and data mining, Eur. J. Oper. Res. 206 (1) (2010) 1–10.
[27] D. Corne, C. Dhaenens, L. Jourdan, Synergies between operations research and data mining: the emerging use of multi-objective approaches, Eur. J. Oper. Res. 221 (3) (2012) 469–479.
[28] T. Brijs, G. Swinnen, K. Vanhoof, G. Wets, Building an association rules framework to improve product assortment decisions, Data Min. Knowl. Discov. 8 (1) (2004) 7–23.
[29] B. Defourny, D. Ernst, L. Wehenkel, Scenario trees and policy selection for multistage stochastic programming using machine learning, INFORMS J. Comput. 25 (3) (2013) 488–501.
[30] L.M. Rios, N.V. Sahinidis, Derivative-free optimization: a review of algorithms and comparison of software implementations, J. Glob. Optim. 56 (3) (2013) 1247–1293.
[31] S. Koziel, D.E. Ciaurri, L. Leifsson, Surrogate-based methods, in: Computational Optimization, Methods and Algorithms, Springer, 2011, pp. 33–59.
[32] K.P. Bennett, O.L. Mangasarian, Bilinear separation of two sets in n-space, Comput. Optim. Appl. 2 (1993) 207–227.
[33] M. Chang, L. Ratinov, D. Roth, Structured learning with constrained conditional models, Mach. Learn. 88 (3) (2012) 399–431.
[34] F. Uney, M. Turkay, A mixed-integer programming approach to multi-class data classification problem, Eur. J. Oper. Res. 173 (3) (2006) 910–920.
[35] S. Sra, S. Nowozin, S.J. Wright, Optimization for Machine Learning, MIT Press, 2012.
[36] E. Carrizosa, D. Romero Morales, Supervised classification and mathematical optimization, Comput. Oper. Res. 40 (1) (2013) 150–165.
[37] L.D. Raedt, T. Guns, S. Nijssen, Constraint programming for data mining and machine learning, in: AAAI, 2010, pp. 1671–1675.
[38] M.J. Heule, S. Verwer, Exact DFA identification using SAT solvers, in: Grammatical Inference: Theoretical Results and Applications, in: Lecture Notes in Computer Science, vol. 6339, Springer, Berlin, Heidelberg, 2010, pp. 66–79.
[39] C. Bessiere, R. Coletta, F. Koriche, B. O'Sullivan, A sat-based version space algorithm for acquiring constraint satisfaction problems, in: Machine Learning: ECML 2005, Springer, 2005, pp. 23–34.
[40] C. Bessière, R. Coletta, B. O'Sullivan, M. Paulin, Query-driven constraint acquisition, in: M.M. Veloso (Ed.), Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007, 2007, pp. 50–55.
[41] N. Beldiceanu, H. Simonis, A model seeker: extracting global constraint models from positive examples, in: M. Milano (Ed.), Principles and Practice of Constraint Programming, in: Lecture Notes in Computer Science, vol. 7514, Springer, 2012, pp. 141–157.
[42] C. Bessiere, R. Coletta, E. Hebrard, G. Katsirelos, N. Lazaar, N. Narodytska, C.-G. Quimper, T. Walsh, Constraint acquisition via partial queries, in: International Joint Conference on Artificial Intelligence, 2013.
[43] R. Dechter, Constraint Processing, Morgan Kaufmann, 2003.
[44] D. Roth, W.-T. Yih, Integer linear programming inference for conditional random fields, in: Proceedings of the 22nd International Conference on Machine Learning, ICML '05, ACM, New York, NY, USA, 2005, pp. 736–743.
[45] E. Fersini, E. Messina, G. Felici, D. Roth, Soft-constrained inference for named entity recognition, Inf. Process. Manag. 50 (5) (2014) 807–819, http://dx.doi.org/10.1016/j.ipm.2014.04.005.
[46] C. de la Higuera, Grammatical Inference: Learning Automata and Grammars, Cambridge University Press, New York, NY, USA, 2010.
[47] C.M. Bishop, Pattern Recognition and Machine Learning, Information Science and Statistics, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
[48] M.L. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming, vol. 414, John Wiley & Sons, 2009.
[49] J.-P. Benoit, V. Krishna, Multiple-object auctions with budget constrained bidders, Rev. Econ. Stud. 68 (1) (2001) 155–179.
[50] I.A. Vetsikas, N.R. Jennings, Sequential Auctions with Partially Substitutable Goods, Lecture Notes in Business Information Processing, vol. 59, 2009, pp. 242–258.
[51] C. Boutilier, M. Goldszmidt, B. Sabata, Sequential auctions for the allocation of resources with complementaritie, in: Proceedings of the 16th International Joint Conference on Artifical Intelligence, vol. 1, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999, pp. 527–534.
[52] P.B. Goes, G.G. Karuga, A.K. Tripathi, Understanding willingness-to-pay formation of repeat bidders in sequential online auctions, Inf. Syst. Res. 21 (2010) 907–924.
[53] P. Cramton, Y. Shoham, R. Steinberg, Combinatorial Auctions, MIT Press, 2006.
[54] A. Ben-Tal, L. El Ghaoui, A. Nemirovski, Robust Optimization, Princeton Series in Applied Mathematics, Princeton University Press, 2009.
[55] D. Verbeeck, F. Maes, K. De Grave, H. Blockeel, Multi-objective optimization with surrogate trees, in: Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference, ACM, 2013, pp. 679–686.