Predicting Solution Summaries to Integer Linear Programs under Imperfect Information with Machine Learning

Eric Larsen [*] Sébastien Lachapelle [†] Yoshua Bengio [†‡]
Emma Frejinger [*§] Simon Lacoste-Julien [†¶] Andrea Lodi[‖]

March 1, 2018

Abstract

The paper provides a methodological contribution at the intersection of machine learning and operations research. Namely, we propose a methodology to quickly predict solution summaries (i.e., solution descriptions at a given level of detail) to discrete stochastic optimization problems. We approximate the solutions based on supervised learning and the training dataset consists of a large number of deterministic problems that have been solved independently and offline. Uncertainty regarding a missing subset of the inputs is addressed through sampling and aggregation methods.

Our motivating application concerns booking decisions of intermodal containers on double-stack trains. Under perfect information, this is the so-called *load planning problem* and it can be formulated by means of integer linear programming. However, the formulation cannot be used for the application at hand because of the restricted computational budget and unknown container weights. The results show that standard deep learning algorithms allow one to predict descriptions of solutions with high accuracy in very short time (milliseconds or less).

Keywords: integer linear programming, deep learning, supervised learning

1 Introduction

Operations research (OR) has been successful in developing methodologies and algorithms allowing to solve efficiently various types of decision problems that can be formalized but are nevertheless too complex or time consuming for humans to process. These methodologies and algorithms are crucial to a huge variety of applications. Machine learning (ML), on the other hand, and deep learning in particular, has had remarkable success in automating tasks that are easy to accomplish but difficult to formalize by humans, for example, image analysis, natural language processing, voice and face recognition. Through this undertaking, ML has developed an array of powerful classification and regression methods that can be used to approximate generic input-output maps. By leveraging this apparatus, ML has also achieved major breakthroughs in solving certain decision problems. A prominent example is the successful combination of deep and reinforcement learning for the game of Go (Silver et al., 2016, 2017).

The paper provides a methodological contribution at the intersection of ML and OR. Namely, we propose a methodology to predict descriptions of solutions to discrete stochastic optimization problems. A solution can be described at different levels of detail. The value of the objective function is the most global one and the detailed solution – the set of values taken by all decision variables – is the most detailed. We concentrate on solution descriptions that lie between these two extremes.

The proposed methodology concerns problems for which a deterministic optimization model is available and nevertheless cannot be used in certain applications due to both computational limits

[*]Department of Computer Science and Operations Research and CIRRELT, Université de Montréal
[†]Department of Computer Science and Operations Research and Mila, Université de Montréal
[‡]CIFAR Senior Fellow
[§]Corresponding author. Email: emma.frejinger@cirrelt.ca
[¶]CIFAR Fellow
[‖]Canada Excellence Research Chair, École Polytechnique de Montréal

and imperfect knowledge of the exact problem instances. Our approach is based on supervised learning and consists of four main steps: First, problem instances under perfect information (input) are sampled from the space relevant to the application at hand and solved using an existing deterministic optimization model and a solver. Second, solutions are synthesized according to the chosen solution description (output). Third, a ML approximator is trained based on the generated input-output data. Fourth, this approximator is used to generate predictions for the solutions of actual problem instances. (Note that in this case we use the ML vocabulary and the ML approximator acts as a heuristic and not an approximation algorithm which is the topic of, e.g., Vazirani, 2003).

These predictions are expected to be delivered with high speed, high accuracy, as well as low marginal cost in terms of data, memory and computing requirements. Given the relatively high fixed cost associated with data generation and training of the ML approximator, the latter is important to achieve the goal of a low average cost for applications where the predictions must be used with a high frequency.

The idea behind the methodology is simple: We predict a description of the solution to a discrete stochastic optimization problem using supervised learning where the training data consists of a large number of deterministic problems that have been solved independently (and offline). Nevertheless, some important challenges arise. First, the structure of the output is tied to the chosen solution description and can be of fixed or variable size. Second, inputs and outputs may be related by a number of constraints that must be satisfied by the predictions. Third, uncertainty regarding a possibly missing subset of the inputs needs to be addressed through appropriate sampling, subselection and aggregation methods. The design and training of the ML algorithm depend on how these challenges are addressed.

Our motivating application concerns booking decisions of intermodal containers on double-stack trains. Similar to passengers who need a flight reservation, containers must have a train reservation. Whereas a person occupies exactly one seat and he/she can hence be assigned to a seat at the time of booking, the assignment of containers to slots on railcars is a combinatorial optimization problem – called *load planning problem* (LPP) – that cannot be solved at the time of booking due to imperfect information: the LPP depends on characteristics of both railcars (e.g., weight holding capacity, length of slots) and containers (e.g., weight and size). In this paper, we focus on the problem of deciding – in real time – (i) how many of a given set of containers of different types can be loaded on a given set of railcars of different types and (ii) how many of the railcars of each type are needed. The decision must be made without knowledge of the container weights since this information is not available at the time of booking. Given the real-time application, the solution must be obtained in very short computing time (fraction of a second).

This problem is of practical importance and features characteristics that make it useful for illustrating the proposed methodology. An integer linear programming (ILP) formulation of the problem can be solved (at least up to a certain precision) by commercial ILP solvers in reasonable time (seconds to minutes) *under perfect information* (Mantovani et al., 2018). However, the formulation cannot be used for the application because of the restricted computational budget (fraction of a second) and, even more importantly, because of the uncertainty over the yet unknown container weights at the time of booking.

The application of ML to discrete optimization problems was the focus of an important research effort in the 1980's and 1990's (Smith, 1999). However, very limited success was ultimately achieved and the area of research was left nearly inactive from the beginning of this century until very recently. The important advances that have meanwhile taken place in ML justify a current renewal of interest. Closest to our work is Vinyals et al. (2015) who, like us, adopt a supervised learning approach and do not construct optimal policies through dynamic programming or reinforcement learning. Their aim is to predict detailed descriptions of solutions for a particular class of deterministic discrete optimization problems (including, e.g., the Traveling Salesman Problem). In contrast with us, they assume perfect knowledge of the problem instances at the time of prediction. The main challenge they attempt to address resides in handling high-dimensional and variable-size input and output structures. Our immediate objectives involve the prediction of less detailed descriptions of solutions, which in our application result in short, fixed-size input and output vectors. The key challenges we address relate to imperfect information regarding the problem instances at the time of prediction and achieving the computation of predictions in very short time. In fact, our aim is to generate predictions considerably faster than the time required to solve a fully informed/deterministic version of the optimization problem.

We model our problem both as a classification problem and as a regression problem. We compare the predictive performance of high-capacity regression/classification deep feedforward neural networks with that of low-capacity logistic/linear regression models and two simple heuristic solutions.

Our main contribution is a methodology that can predict descriptions of optimal solutions where only partial information on the problem instance is available at the time of prediction. We demonstrate through our application how this methodology allows to build accurate and fast predictors for descriptions of optimal solutions.

The remainder of the paper is structured as follows: Section 2 examines the prediction problem under consideration and Section 3 reviews the relevant literature in ML and OR. Section 4 expounds the proposed methodology. Section 5 describes a detailed application of the methodology and reports the results. Finally, Section 6 summarizes the content of the paper, reviews outstanding issues and describes directions for future research.

2 The prediction problem

Our objective is to alleviate the computational challenges presented by certain discrete stochastic optimization problems. We attend to problems for which (*i*) a deterministic formulation exists, (*ii*) computing solutions with a solver is possible, (*iii*) computational budget or insufficient speed prevent the use of the solver in repeated or real-time applications, (*iv*) a subset of the problem inputs is uncertain at the time of predicting the output. We propose to apply ML to find solution descriptions that are computable with sufficiently high accuracy, low marginal cost and high speed. After opening with a presentation of the basic concepts, we state the prediction problem.

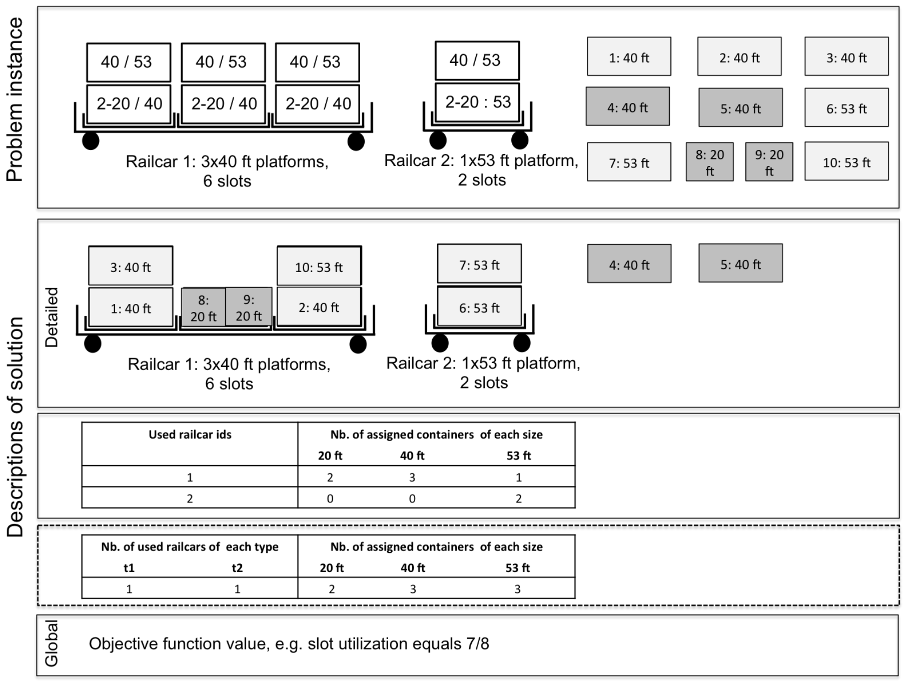
2.1 Descriptions of solutions

The solution to a discrete optimization problem can be described according to various levels of detail. The value of the single objective function and the set of values taken by all decision variables included in the problem stand respectively at the less detailed, we shall say the most *global*, and the most *detailed* positions in the range of possible descriptions of a solution. Due to its discrete nature and high dimensionality, predicting the detailed solution (a structured, classification or discrete-valued regression problem) is a considerably more complex task than predicting the value of the objective function (an unstructured, real-valued regression problem). The latter is sufficient for instance when approximating the cost-to-go/payoff function in reinforcement learning or approximate/neuro-dynamic programming. Whereas a prediction is in principle possible for any choice of description, in general the level of detail should be adjusted to the application at hand and should not exceed its intended use.

Throughout the paper, we use the double-stack intermodal railcar LPP (Mantovani et al., 2018) to illustrate the proposed methodology. The LPP can be briefly described as follows. Given a set of containers and a sequence of railcars, determine the subset of containers to load and the exact way of loading them on a subset of railcars. The objective consists in minimizing the total cost of containers left behind and partly filled railcars. The problem depends on individual characteristics of containers and railcars: Containers are characterized by their length, height, standardized type, content and weight. Railcars comprise different numbers of platforms and each platform has a lower and an upper slot where containers may be loaded. Crucially, railcars are characterized by the weight capacity and tare of each platform and by the specific loading capabilities associated with their standardized type.

Figure 1 depicts a small example of a problem instance along with four descriptions of the optimal solution at different levels of detail. The instance contains ten containers of three different sizes: 20 feet (ft), 40 ft and 53 ft. For the sake of simplicity, we do not report exact weights but note that containers drawn in dark gray are considerably heavier than those drawn in light gray. The instance contains two railcars: one with three 40 ft platforms and another with one 53 ft platform. The numbers in each slot indicate the feasible assignments with respect to container sizes: Containers exceeding the length of the platform cannot be loaded in a bottom slot and 20 ft containers cannot be loaded in a top slot.

The detailed optimal solution is illustrated right below the problem instance in Figure 1. The solution makes use of the two railcars and a subset of the containers are assigned to slots. In this example, one top slot is not used because of weight constraints and two heavy containers are not



Used railcar ids	Nb. of assigned containers of each size		
	20 ft	40 ft	53 ft
1	2	3	1
2	0	0	2

Nb. of used railcars of each type		Nb. of assigned containers of each size		
t1	t2	20 ft	40 ft	53 ft
1	1	2	3	3

Figure 1: Example of the double-stack railcar LPP

assigned. If the objective function is defined by the slot utilization, then its value is 7/8. This corresponds to the global description of the solution, shown in the bottom part of the figure. Two alternative descriptions whose levels of detail lie between the detailed and global are depicted in the figure. One defines for each individual railcar the number of each container size that is assigned to it. The other, less detailed (dashed frame), defines the number of railcars of each type that are used in the solution and the number of containers of each size that are assigned. The latter description is used in our application.

2.2 Predicting descriptions of solutions

Let a particular instance of a deterministic discrete optimization problem be represented by the feature vector $\widetilde{\mathbf{x}}$ and let the fully detailed solution (i.e. containing values of all decision variables) produced by the optimization solver $\widetilde{f}^*(\cdot)$ be represented by the vector $\widetilde{\mathbf{y}} = \widetilde{f}^*(\widetilde{\mathbf{x}})$. Information on a subset of the feature vector $\widetilde{\mathbf{x}}$ may be unavailable or incomplete at the time of prediction. We therefore define a partition $\widetilde{\mathbf{x}} = [\widetilde{\mathbf{x}}_{\text{av}}, \widetilde{\mathbf{x}}_{\text{unav}}]$ accordingly, where $\widetilde{\mathbf{x}}_{\text{av}}$ contains available features and $\widetilde{\mathbf{x}}_{\text{unav}}$ unavailable or yet unobserved ones.

We denote by $\bar{\mathbf{y}} = g(\widetilde{\mathbf{y}})$ the synthesis of $\widetilde{\mathbf{y}}$ according to the desired description. Our objective is to find the best possible prediction $\mathbf{y} = f(\widetilde{\mathbf{x}}_{\text{av}}; \boldsymbol{\theta})$ of $\bar{\mathbf{y}} = \bar{f}^*(\widetilde{\mathbf{x}})$ where $\bar{f}^* = g \circ \widetilde{f}^*$. The approximator $f(\cdot;\cdot)$ is a particular ML model and $\boldsymbol{\theta}$ is a vector of parameters. $f(\cdot;\cdot)$ and $\boldsymbol{\theta}$ are selected through a ML algorithm based on the available input-output training data made up of $(\widetilde{\mathbf{x}}, \bar{\mathbf{y}})$ pairs. If complete information on the full feature vector were available at the time of prediction, then $\widetilde{\mathbf{x}} = \widetilde{\mathbf{x}}_{\text{av}}$ would hold. Otherwise, $f(\cdot;\cdot)$ must properly account for the unavailable features $\widetilde{\mathbf{x}}_{\text{unav}}$. This issue is examined in Section 4.3.

We use our application to illustrate the notation: $\widetilde{\mathbf{x}}$ contains the detailed information about the problem instance required to solve the LPP. The subvector $\widetilde{\mathbf{x}}_{\text{av}}$ contains features that are known at time of prediction: the total numbers of available rail cars of each type and available containers of each length. The subvector $\widetilde{\mathbf{x}}_{\text{unav}}$ contains features that are unknown at time of prediction: the individual gross weights of the containers. The problem is formalized as a deterministic ILP and can be solved by a commercial solver. The application of the solver then corresponds to $\widetilde{f}^*(\cdot)$ and the detailed assignment of containers to slots (solution at the top in Figure 1) to $\widetilde{\mathbf{y}} = \widetilde{f}^*(\widetilde{\mathbf{x}})$. We stress that the input-output data used in constructing the approximator is not obtained by restricting the original optimization model according to $\bar{\mathbf{y}}$. We rather compute the full solution under perfect information $\widetilde{\mathbf{y}}$ and then produce the synthesis $\bar{\mathbf{y}}$. In Figure 1 we provide three different examples of $\bar{\mathbf{y}}$ that can be obtained by synthesis of the detailed solution $\widetilde{\mathbf{y}}$.

3 Literature review

This section surveys the relevant literature found in OR and ML. We open with a brief background presentation of the standard ML methodology underlying our research. We then review the existing work found at the intersection of OR and ML. We consider in turn the two following standpoints so as to locate the position of our own research: (i) the application of ML methods by the OR community, (ii) the ML community addressing problems typically attended to by the OR community.

3.1 ML, deep learning and statistics

The two following simple but consequential principles are put forth and implemented by ML (e.g., Hastie et al., 2009, Murphy, 2012). Both stem from a strongly pragmatic stand. First, models are merely viewed as useful representations of the data. No claim to exact representation of an underlying data generating process is made and every available scientific resource (e.g. from statistics, information theory, connexionism, statistical physics, econometrics) can be used in specifying representations, provided it is effective. Second, models are to be evaluated and compared based on their performance on sets of data distinct from those used in their calibration. Whereas these ideas are known to statistics, they are developed and applied systematically in ML. The successes of ML and most notably the remarkable advances that have occurred in its subfield known as deep learning are rooted in applications of these principles.

In accordance with these principles, the model selection procedures applied in ML inherit from three sources: frequentist statistics, bayesian statistics and information theory, and are selected based on the efficacy and convenience of their application. The frequentist algorithms that we shall apply may be broadly summarized as follows: We specify a *loss function* that singles out relevant aspects of the performance achieved in representing the observable data (e.g. quadratic or absolute error in regression problems, pseudo-likelihood in regression or classification problems). (Notice that a loss function may either have interest on its own or act as a surrogate for the loss function of interest whose introduction would lead to difficult or intractable computations. For example, cross-entropy is viewed as a differentiable surrogate for the 0-1 loss in classification problems.) Model selection is aimed at minimizing expected loss (i.e. risk) incurred over alternative models under consideration, with respect to the actual but unknown distribution of the data. Models generally comprise parameters and their estimation is typically conducted through minimization of average loss (empirical risk) over a training (sample) data set.

Models under consideration may differ in structure and/or in capacity, that is, with respect to their flexibility in adapting to data. Families of models are indexed by *hyperparameters.* Minimization of empirical risk over a common data sample cannot be used to discriminate between families of models with finite data, for it is biased towards selection of models with higher capacities. Models with excess capacity are said to *overfit* the training data and tend to generalize poorly to new data. ML instead defines and aims to estimate and minimize an expected or limiting so-called *generalization error* incurred when trained models are applied to independent data. This might be interpreted as an attempt to minimize a proper estimate of the true expected loss, accounting for parameter estimation. The loss functions underlying the empirical risk used for parameter estimation and the mean generalization error used for model selection may differ however, either for convenience or to align the model in relation to the data in a particular direction.

Model selection is performed through the operation known as *validation.* Simple validation consists in minimizing the average generalization error incurred when the estimated models under consideration are applied to a distinct data sample known as the validation or hold-out set. In statistics, empirical risk and average generalization error would be respectively said to be calculated *in-sample* and *out-of-sample.* Important theoretical justifications for minimization of average generalization error include the following: (i) Provided sufficient conditions are satisfied, a uniform law of large numbers holds and may be applied to demonstrate the stochastic convergence of the average generalization error arising from validation towards a non-stochastic limit (Vapnik, 1999). (ii) Provided sufficient conditions are satisfied, a so-called *oracle inequality* holds and may be applied to demonstrate the stochastic convergence to 1 of the ratio of the two following expressions:

on the one hand, the difference between the generalization error evaluated at the optimal model resulting from training and validation with the current finite data sample and the expected error resulting from the actual minimum risk model; on the other hand, the difference between the error associated with an oracle based on the available data sample and the expected error resulting from the minimum risk model (Arlot and Celisse, 2010, van der Laan et al., 2003). In brief, minimization of average generalization error (i) approximates the minimization of risk and (ii) is asymptotically comparable to an oracle.

Ultimately, a third data set, the test set, which has been set aside initially and excluded from any of the learning or validation operations is used to perform a final assessment of the generalization error estimates and a final confirmation of the outcomes of the model selection process.

The models and algorithms that we shall implement belong to the subfield of ML known as deep learning (Goodfellow et al., 2016). The latter concentrates on the simultaneous development of (i) a broad array of high-capacity data representations based on multilayer neural networks and (ii) effective and efficient algorithms for their training and selection. Deep learning has tailored and adapted these representations and algorithms to perform with remarkable success a diverse and growing range of data processing and generation tasks. These tasks include generic prediction and classification, analysis and generation of text, sound and image.

3.2 The OR-ML nexus and this research project

Changes are occurring at the intersection of OR and ML at an accelerating pace. ML, especially deep learning, has become an important contributor and, especially in complex decision making problems, the interplay between ML and OR looks extremely promising. The most traditional locus of synergies between OR and ML, which has attracted a huge amount of attention in the last five to ten years, is the introduction of optimization, especially continuous, methods originating in OR to algorithms used in ML. More recently, in OR, there are growing interest and success in introducing ML techniques in existing solution methodologies. Finally, in ML, there are growing interest and initial glimpses of success in independently addressing optimization problems typically attended to by OR. For an overview of the current relations and commonalities between OR and ML from the standpoint of OR, see, e.g., Lodi (2017). The following two sections take a closer look to ML within combinatorial optimization and discuss the relation between our work and some relevant literature.

3.2.1 ML as a contributor to OR

ML is a source of generic predictors and classifiers for OR, for example, used to produce demand forecasts when optimizing supply. There is a large body of literature on this classic topic and we mention this straightforward application of ML for completeness.

ML can be employed to perform specific tasks that are integral to solution processes originating in OR. For example, Lodi and Zarpellon (2017) and Khalil et al. (2016) examine the guidance provided by ML for the branching decisions involved in solving mixed integer programming problems. Furthermore, Oroojlooyjadid et al. (2016) and Ahmadian, Sara (2017) introduce ML in the solution processes of inventory and location problems and Kruber et al. (2017) perform automatic Dantzig-Wolfe decompositions with ML.

Fischetti and Fraccaro (2017) use ML in a setting that shares some similarities with ours. Namely, they seek to gain information about optimal solutions to a problem they cannot solve due to restrictions on computing time. They apply ML to predict the objective function value of optimal solutions. They consider the offshore wind farm layout optimization problem and use ML to predict the potential value of a new site. In contrast to Fischetti and Fraccaro (2017), we consider the approximation of solutions to decision problems at a more detailed level. In addition, we attend to a setting where there is imperfect knowledge of the problem instances, namely, uncertainty in the demand.

Using ML to approximate the objective function value is a well-researched topic in approximate dynamic programming, in its applications to either deterministic or stochastic problems. In this context, the approximations concern the cost-to-go as well as the policy functions. The general methodology is described, e.g., in Bertsekas and Tsitsiklis (1996) and Powell (2011).

In the stochastic programming context, Bertsimas and Kallus (2014) consider the problem of optimizing the expectation of an objective defined in terms of a vector of variables of interest, with respect to a vector of decision variables, conditionally to a vector of information variables.

The core proposal is (i) to approximate the objective as a weighted sum by applying one among a number of alternative supervised ML algorithms that account for the information contained in the conditioning variables and (ii) to optimize the simpler approximate objective with respect to the decision variables, conditionally to the conditioning variables. They illustrate the tractability of the methodology through several applications (see also Bertsimas et al., 2016) and examine the asymptotic optimality of the resulting solutions.

3.2.2 ML as an alternative to OR

The application of ML to discrete optimization problems (with a particular focus on the Traveling Salesman Problem) was the focus of an important research effort in the 1980's and 1990's. The relevant literature is extensively surveyed in Smith (1999). However, the levels of performance resulting from that effort turned out to be disappointing in comparison with both integer programming (in the sense of exact methods) and heuristics available in the OR literature (e.g., LaMaire and Mladenov, 2012, Sarwar and Bhatti, 2012). This small return in comparison with the amount of research invested and the lack of significant progress were possibly responsible for the lesser attention received from ML in this area of research from the beginning of this century until recently.

Nonetheless, the situation in ML has, since then, advanced considerably: ranges of the available methodologies, algorithms and architectures have been greatly broadened, computational power has increased tremendously (chiefly due to the introduction of the GPUs), and the new regularization procedures introduced by deep learning have been demonstrated to be highly effective. These advances justify the renewed interest in the application of ML to optimization problems that is testified by a number of recent studies. See Vinyals et al. (2015), Bello et al. (2016), Dai et al. (2017), Kaempfer and Wolf (2018) and Wouter et al. (2018).

Currently, methodologies in ML for decision problems that are typically addressed by OR are mainly found in the areas of reinforcement learning, learning to search and multi-armed bandits. Our research is based on supervised learning and does not involve the construction of policies through dynamic programming/reinforcement learning. It is closest to the recent research on pointer networks (Vinyals et al., 2015). The following provides a brief overview of each one of the three areas before providing more details on pointer networks.

Reinforcement learning (e.g., Sutton and Barto, 1998, Szepesvári, 2010) shares a common theoretical framework with approximate dynamic programming: both techniques are aimed to address a very wide range of sequential decision problems and have evolved in close parallel and mutual awareness. Essentially, they differ only with respect to the particular algorithms that they have contributed to. Bello et al. (2016), Dai et al. (2017), Han and E (2016), Li and Malik (2016), Wouter et al. (2018) illustrate the recent successes achieved by reinforcement learning with respect to problems typically addressed by OR.

Learning to search algorithms (Chang et al., 2015, Daumé et al., 2009) aim to solve structured prediction problems. They consist of three main steps: convert a structured prediction problem into a sequential search problem, construct a reference policy based on training data, progressively learn a policy imitating the reference policy.

Multi-armed bandits algorithms (Zhou, 2015) attend to a class of sequential decisions problems under uncertainty where the agent must learn online the rewards associated with the decisions and where the state of the world is assumed unaffected by the decisions.

Pointer network (Vinyals et al., 2015) designates a particular instance of neural network tailored to the prediction of solutions to discrete optimization problems where the input consists of a sequence of points in $\mathbb{R}^n$ and the output may be expressed as a sequence of indexes referring to elements of the input sequence. Perfect information on the problem at hand is assumed available at the time of prediction. Pointer networks are aimed to predict detailed descriptions of solutions and present their main challenges in connection with the handling of high-dimensional and variable-size input and output structures.

Summing up, the distinctive features of our research project are as follows: we predict descriptions of solutions to discrete stochastic optimization problems by applying a supervised learning approach, while avoiding the construction of policies through dynamic programming/reinforcement learning. The key challenges we meet consist in addressing the imperfect knowledge of the problem instances at the time of prediction and generating predictions in very short computing time.

4 Methodology

In this section we delineate a methodology for predicting descriptions of solutions to discrete optimization problems through supervised ML algorithms. We overview its four main components: data generation, ML approximation, aggregation and subselection procedures, treatment of uncertain or missing inputs. The discussion is cast from a general perspective as it is intended to cover a broad variety of applications. Section 5 describes in detail how the methodology can be implemented in the context of our application.

4.1 Data generation

The first step in the data generation process is to sample a set of problem instances $\{\widetilde{\mathbf{x}}^{(i)},\ i = 1,\dots,m\}$. Elements of $\widetilde{\mathbf{x}}$ that are expected to vary in the intended application should be made to vary and covary in the dataset in commensurate ranges. The ability to generate data at will according to a known sampling protocol has important advantages since the performance of ML training and model selection can be evaluated in any arbitrarily defined region in the feature space. Moreover, if the performance is not sufficiently good, more data can be generated.

Problem instances can be generated through pseudo-random or quasi-random sampling. Simple pseudo-random sampling and stratified pseudo-random sampling are simple and easily applicable protocols. It is also conceivable to apply alternative protocols, such as importance sampling, in order to improve sample efficiency. Uncertainty regarding the values of elements of the input $\widetilde{\mathbf{x}}$ should be accounted for in data generation. Indeed, the distributions from which we sample those values are viewed as a description of this uncertainty and should be selected accordingly. We refer to, for example, Asmussen and Glynn (2010) and Law (2014) for further details on data sampling and simulations.

Supervised learning requires labeled data and we employ an existing solver to generate the fully detailed solutions of each problem instance $\widetilde{\mathbf{y}}^{(i)} = \widetilde{f}^*(\widetilde{\mathbf{x}}^{(i)})$, $i = 1,\dots,m$. For the approximation performed later on to be more precise, the optimal solution of each instance should be uniquely defined. This can be achieved by setting the parameters of the solver such that it always returns the same solution for a given instance.

In order to make efficient use of the computational resources available for ML, the fully detailed optimal solutions $\widetilde{\mathbf{y}}^{(i)}$, $i = 1,\dots,m$ are synthesized as $\bar{\mathbf{y}}^{(i)}$ $i = 1,\dots,m$ according to a description whose level of detail accommodates without exceeding that required in intended applications. As illustrated in the introductory example (Figure 1), such descriptions vary in complexity. They may be highly structured and may also feature variable size. The resulting dataset, available for ML approximation, is $\{(\widetilde{\mathbf{x}}^{(i)},\bar{\mathbf{y}}^{(i)}),\ i = 1,\dots,m\}$. A number of constraints may tie elements of the input vector $\widetilde{\mathbf{x}}$ and the synthesized output vector $\bar{\mathbf{y}}$. Such constraints, as well as the exact definitions of the input and output vectors impact the selection and performance of ML models and algorithms.

The use of a data set consisting of historically observed problem instances should be subject to an important caveat: in brief, sampling from historical data is only appropriate when attempting to mimic a behavior reflected in such data. This is not our objective since we aim to predict solutions resulting from a solver. Hence, sampling from historical data would be likely to introduce uncontrollable distortions in the resulting dataset for two main reasons. First, observed solutions lie in a subspace of the space of admissible problem instances as they result in effect from censoring/constraining the instances to a given context. Second, historical solutions may have resulted from extraneous decision processes that in this case should be accounted for. This introduces additional difficulties and it is only possible if the extraneous decision process in known.

4.2 ML approximation

Whenever the input vector available at the time of prediction is equal to the full vector of input features, that is $\widetilde{\mathbf{x}} = \widetilde{\mathbf{x}}_{\text{av}}$, we seek to find the best possible prediction $\mathbf{y} = f(\widetilde{\mathbf{x}};\boldsymbol{\theta})$ of $\bar{\mathbf{y}} = \bar{f}^*(\widetilde{\mathbf{x}})$. The approximator $f(\cdot;\cdot)$ is a ML model and $\boldsymbol{\theta}$ is a vector of parameters. $f(\cdot;\cdot)$ and $\boldsymbol{\theta}$ are selected through a ML algorithm, based on the available input-output data made up of $(\widetilde{\mathbf{x}},\bar{\mathbf{y}})$ pairs. The models under consideration and the algorithms used in their training and selection must conform with the structure embodied in the input and output vectors $\widetilde{\mathbf{x}}$ and $\bar{\mathbf{y}}$ and must also uphold the constraints that may relate their individual elements. Hence, the choice of a model and an algorithm necessarily depends on the exact application at hand. There is in ML a range of classification and regression approximators that can be used for this purpose. The ML algorithm that we apply is standard and can be broadly summarized as follows:

1. The full dataset is divided at random between training, validation and test sets.
2. Parameters of all candidate models are tuned on the training set.
3. Performances of trained models are assessed and compared based on the validation set.
4. Additional data is generated and processed if performance on validation set is unsatisfactory.
5. The model achieving the lowest error measured on the validation set is retained.
6. The model performance is finally evaluated on the testing set.
7. Provided the selected model demonstrates sufficient accuracy, it is used as a high speed, low marginal cost predictor for the synthesized solution of any problem instance.

Additional challenges arise when the input vector available at the time of prediction is not equal to the full vector of input features, that is, $\widetilde{\mathbf{x}} \neq \widetilde{\mathbf{x}}_{\mathrm{av}}$. This is the case when dealing with inputs whose values are imperfectly known at the time of prediction. This issue is addressed in the next sections.

4.3 Aggregation and subselection

The operations of aggregation and subselection can address a lack of information as well as reduce the complexity of information collection and processing in regard to the input features. On the one hand, it is critical to account for missing or insufficient information, in our application, regarding container weights at the time of prediction. On the other hand, it can be useful to reduce the scope of analysis, either for theoretical (e.g., reduce the complexity of the original optimization problem or of the ML approximators) or computational (reduce the resources required for computing the exact solutions of the optimization problem or training the ML approximators) convenience. In our application, the latter is achieved by restricting the scope of analysis to subsets of railcar types and container lengths and by limiting the numbers of containers and railcars.

The main solution we propose is basically to train the ML model using only as input $\widetilde{\mathbf{x}}_{\mathrm{av}}$, and the targets being obtained from solving the OR problem with the full $\widetilde{\mathbf{x}}$ obtained from its distribution, with $\widetilde{\mathbf{x}}_{\mathrm{av}}$ extracted from $\widetilde{\mathbf{x}}$. Some aggregation operation is then necessary to combine the different targets associated with different draws of the missing input. To formalize and justify such procedures, we therefore rely on the operations of aggregation and subselection with respect to a conditional probability distribution (e.g., Billingsley, 1995). Consider the joint probability distribution of $\widetilde{\mathbf{x}} = [\widetilde{\mathbf{x}}_{\mathrm{av}}, \widetilde{\mathbf{x}}_{\mathrm{unav}}]$ that, in turn, induces the probability distribution of $\bar{\mathbf{y}}$ through $\bar{\mathbf{y}} = \bar{f}^*(\widetilde{\mathbf{x}})$. Since $\widetilde{\mathbf{x}}_{\mathrm{av}}$ is known exactly at prediction time, the probabilistic information available at that moment is embodied in the distribution of $\widetilde{\mathbf{x}}_{\mathrm{unav}}$ conditional on the σ-algebra generated by $\widetilde{\mathbf{x}}_{\mathrm{av}}$, say $\sigma(\widetilde{\mathbf{x}}_{\mathrm{av}})$. We distinguish two manners in which this conditional distribution may be leveraged through aggregation to predict $\bar{\mathbf{y}}$:

(i) Select a loss function measuring the discrepancy between $\widetilde{\mathbf{x}}_{\mathrm{unav}}$ and a predictor $\mathbf{x}_{\mathrm{unav}}$ of the latter and minimize the expectation of this loss with respect to the distribution of $\widetilde{\mathbf{x}}_{\mathrm{unav}}$ conditional on $\sigma(\widetilde{\mathbf{x}}_{\mathrm{av}})$. For instance, when the loss function is given by L_1 or L_2, the predictor is respectively equal to the median or the mean of $\widetilde{\mathbf{x}}_{\mathrm{unav}}$, conditional on $\sigma(\widetilde{\mathbf{x}}_{\mathrm{av}})$. Next, substitute this predicted value for $\widetilde{\mathbf{x}}_{\mathrm{unav}}$ along with $\widetilde{\mathbf{x}}_{\mathrm{av}}$ in $\bar{\mathbf{y}} = \bar{f}^*(\widetilde{\mathbf{x}})$ to obtain a prediction for $\bar{\mathbf{y}}$. This approach is explained in sections 4.4.1 and 4.4.2 where it is compounded with a ML approximation for $\bar{\mathbf{y}} = \bar{f}^*(\widetilde{\mathbf{x}})$.

(ii) Select a loss function measuring the discrepancy between $\bar{\mathbf{y}}$ and a predictor $\mathbf{y}$ of the latter and minimize the expectation of this loss with respect to the distribution of $\widetilde{\mathbf{x}}_{\mathrm{unav}}$ conditional on $\sigma(\widetilde{\mathbf{x}}_{\mathrm{av}})$. Once again, when the loss function is given by L_1 or L_2, the predictor $\mathbf{y}$ is respectively equal to the median or the mean of $\bar{\mathbf{y}}$, conditional on $\sigma(\widetilde{\mathbf{x}}_{\mathrm{av}})$. This approach is explained in sections 4.4.3, 4.4.4 and 4.4.5 where it is also compounded with a ML approximation for $\bar{\mathbf{y}} = \bar{f}^*(\widetilde{\mathbf{x}})$.

Evidently, $\widetilde{\mathbf{x}}_{\text{unav}}$ and $\bar{\mathbf{y}}$ are not $\sigma(\widetilde{\mathbf{x}}_{\text{av}})$-measurable. However, through integration with respect to the distribution of $\widetilde{\mathbf{x}}_{\text{unav}}$ conditional on $\sigma(\widetilde{\mathbf{x}}_{\text{av}})$, both approaches in effect “aggregate" in a single point the probability mass distributed over the support of $\widetilde{\mathbf{x}}_{\text{unav}}$, resulting into predictors for $\widetilde{\mathbf{x}}_{\text{unav}}$ and $\bar{\mathbf{y}}$ that are by construction measurable with respect to $\sigma(\widetilde{\mathbf{x}}_{\text{av}})$.

Subselection restricts the scope of analysis to a subset of the support of $\widetilde{\mathbf{x}} = [\widetilde{\mathbf{x}}_{\text{av}}, \widetilde{\mathbf{x}}_{\text{unav}}]$. Such a subset and its complement constitute a partition of the support and give rise to a corresponding informational σ-algebra, say $\mathcal{G}$. The probabilistic knowledge that embodies this restriction may be represented by the joint probability distribution of $[\widetilde{\mathbf{x}}_{\text{av}}, \widetilde{\mathbf{x}}_{\text{unav}}]$ conditional on $\mathcal{G}$. If useful, for instance for the purpose of drawing formal comparisons, conditioning with $\mathcal{G}$ may be introduced explicitly, possibly in addition to conditioning with $\sigma(\widetilde{\mathbf{x}}_{\text{av}})$.

4.4 Treatment of uncertain or missing inputs

Information regarding some elements of the input feature vector may be missing or incomplete at the time when predictions of solutions are required. We consider the case where information regarding the value of some features is unavailable at this time and we partition the input vector accordingly as $\widetilde{\mathbf{x}} = [\widetilde{\mathbf{x}}_{\text{av}}, \widetilde{\mathbf{x}}_{\text{unav}}]$. The subvectors $\widetilde{\mathbf{x}}_{\text{av}}$ and $\widetilde{\mathbf{x}}_{\text{unav}}$ describe available and unavailable features and can be viewed respectively as a deterministic and a stochastic vector (or both may be viewed as stochastic with the distribution of $\widetilde{\mathbf{x}}_{\text{av}}$ being degenerate). Faced with uncertainty regarding the value taken by $\widetilde{\mathbf{x}}_{\text{unav}}$, either one of these two alternative paths might be followed, independently of the choice to involve or not ML:

(i) Define approximate predictors $\mathbf{y} = f(\widetilde{\mathbf{x}}_{\text{av}}; \boldsymbol{\theta})$ of $\bar{\mathbf{y}} = \bar{f}^*(\widetilde{\mathbf{x}})$ that nonetheless account for the stochasticity in $\widetilde{\mathbf{x}}_{\text{unav}}$. Those involve aggregation, that is the replacement of the distribution of non-degenerate stochastic variables by surrogate values concentrating all probability mass.

(ii) Cast the prediction of $\bar{\mathbf{y}} = \bar{f}^*(\widetilde{\mathbf{x}})$ inside the framework of stochastic programming (see e.g., Birge and Louvaux, 2011, Kall and Wallace, 1994, Shapiro et al., 2014) where the stochasticity arises from the uncertainty in $\widetilde{\mathbf{x}}_{\text{unav}}$ and $\widetilde{\mathbf{x}}_{\text{av}}$ may possibly be a conditioning vector.

Sections 4.4.1 to 4.4.5 describe alternative (i). The preference between the two ultimately depends on their respective trade-off between tractability and performance. Typically, it is more costly to solve a stochastic programming problem than solving the corresponding deterministic problem. In this paper, we focus on an application that requires to predict solutions in very short time (fraction of a second). In fact, the available computational budget is less than the time required to solve the deterministic problem. We therefore focus on (i). A comparison with (ii) is left for future research.

In the following sections, we outline five approximation-aggregation methods that are based on ML. They differ with respect to the manner in which a value representative of the output description $\bar{\mathbf{y}}$ resulting from $[\widetilde{\mathbf{x}}_{\text{av}}, \widetilde{\mathbf{x}}_{\text{unav}}]$ may be computed only from knowledge of $\widetilde{\mathbf{x}}_{\text{av}}$. All the methods do not predict the same output, moreover, they have different computational cost. Therefore, the choice of method depends on the application at hand. In this work we consider two methods that perform aggregation over output values (Sections 4.4.3 and 4.4.4).

4.4.1 Aggregation over input values before ML approximation

The dataset passed to the ML algorithm is $\{(\widetilde{\mathbf{x}}^{(i)}_{\text{av}}, \hat{\mathbf{y}}^{(i)})\ i = 1, \ldots, m\}$ where $\hat{\mathbf{y}}^{(i)} = \bar{f}^*(\widetilde{\mathbf{x}}^{(i)}_{\text{av}}, \widehat{\mathbf{x}}^{(i)}_{\text{unav}})$. A direct replacement $\widehat{\mathbf{x}}^{(i)}_{\text{unav}}$ of $\widetilde{\mathbf{x}}^{(i)}_{\text{unav}}$ is calculated according to $\widehat{\mathbf{x}}^{(i)}_{\text{unav}} = \int \widetilde{\mathbf{x}}^{(i)}_{\text{unav}} g_0(\widetilde{\mathbf{x}}^{(i)}_{\text{av}}, \widetilde{\mathbf{x}}^{(i)}_{\text{unav}}) dF_{\widetilde{\mathbf{x}}^{(i)}_{\text{unav}}}$ where $g_0(\cdot,\cdot)$ is a generalized measurement function that might itself be trained and $F_{\widetilde{\mathbf{x}}^{(i)}_{\text{unav}}}$ is the joint cumulative distribution function of $\widetilde{\mathbf{x}}^{(i)}_{\text{unav}}$ conditional or not on knowledge of $\widetilde{\mathbf{x}}^{(i)}_{\text{av}}$. Solution prediction $\mathbf{y}$ is obtained from the trained approximator f through $\mathbf{y} = f(\widetilde{\mathbf{x}}_{\text{av}}; \boldsymbol{\theta})$.

As an example one can select g_0 so that $\widehat{\mathbf{x}}_{\text{unav}}$ is the mean or median of $\widetilde{\mathbf{x}}_{\text{unav}}$, either unconditional or conditional on the value of $\widetilde{\mathbf{x}}_{\text{av}}$. A predictor obtained by substituting a deterministic value for a stochastic input variable is sometimes said to be a "certainty-equivalent" prediction. With respect to the prediction of outputs, this aggregation method is less interesting than those presented in Sections 4.4.3, 4.4.4 and 4.4.5 since output predictions are based on representative input values rather than being defined directly. Nevertheless, this type of aggregation method is appropriate when streamlining the inputs of an overly complex ILP problem or even when information on certain inputs is missing, provided their distribution is highly concentrated. An application is described in section 5.1.

4.4.2 Aggregation over input values after ML approximation

The dataset passed to the ML algorithm is $\{(\widetilde{\mathbf{x}}^{(i)}, \bar{\mathbf{y}}^{(i)})\ i = 1, \ldots, m\}$. The prediction $\mathbf{y}$ is obtained from the trained approximator f through $\mathbf{y} = f(\widetilde{\mathbf{x}}_{\text{av}}, \widehat{\mathbf{x}}_{\text{unav}}; \boldsymbol{\theta})$ where a direct replacement $\widehat{\mathbf{x}}_{\text{unav}}$ of $\widetilde{\mathbf{x}}_{\text{unav}}$ is calculated according to $\widehat{\mathbf{x}}_{\text{unav}} = \int \widetilde{\mathbf{x}}_{\text{unav}} g_1(\widetilde{\mathbf{x}}_{\text{av}}, \widetilde{\mathbf{x}}_{\text{unav}}) dF_{\widetilde{\mathbf{x}}_{\text{unav}}}$ and $g_1(\cdot, \cdot)$ is a measurement function. Remarks are similar to those made in regard to the method of section 4.4.1.

4.4.3 Aggregation over output values before ML approximation

The dataset passed to the ML algorithm is $\{(\widetilde{\mathbf{x}}_{\text{av}}^{(i)}, \hat{\mathbf{y}}^{(i)})\ i = 1, \ldots, m\}$ where

$$\hat{\mathbf{y}}^{(i)} = \int \bar{f}^*(\widetilde{\mathbf{x}}_{\text{av}}^{(i)}, \widetilde{\mathbf{x}}_{\text{unav}}^{(i)}) g_2(\widetilde{\mathbf{x}}_{\text{av}}^{(i)}, \widetilde{\mathbf{x}}_{\text{unav}}^{(i)}) dF_{\widetilde{\mathbf{x}}_{\text{unav}}^{(i)}}$$

and $g_2(\cdot, \cdot)$ is a measurement function. Solution prediction $\mathbf{y}$ is obtained from the trained approximator f through $\mathbf{y} = f(\widetilde{\mathbf{x}}_{\text{av}}; \boldsymbol{\theta})$.

For instance, one could select as a representative solution $\hat{\mathbf{y}}^{(i)}$ the one achieving the mean value or a particular quantile of the objective criterion (or the closest thereof), given $\widetilde{\mathbf{x}}_{\text{av}}^{(i)}$, over the support of $\widetilde{\mathbf{x}}_{\text{unav}}^{(i)}$. A greater risk aversion towards the uncertainty in $\widetilde{\mathbf{x}}_{\text{unav}}^{(i)}$ would lead to the selection of larger quantiles. This type of procedure can be implemented if the dataset is generated through a two-stage sampling process. The first stage concerns elements in $\widetilde{\mathbf{x}}_{\text{av}}^{(i)}$. The second stage concerns elements in $\widetilde{\mathbf{x}}_{\text{unav}}^{(i)}$ and their sampling distribution is viewed as a proper representation of their uncertainty. As an example, consider the case where for each sample of the first stage variables, the instance achieving the median value of the optimality criterion over the second stage sample, conditionally on the values taken by first stage variables, is selected as the representative instance. This amounts to performing Monte Carlo integration to calculate the integral above. The map between first stage variables and median solutions may then be approximated with ML based on the resulting dataset. Efficient Monte Carlo estimation of expectations has been extensively researched (see e.g., Asmussen and Glynn, 2010, Law, 2014). The Monte Carlo estimation of quantiles has been researched by Avramidis and Wilson (1998), Hesterberg and Nelson (1998), Hsu and Nelson (1990).

4.4.4 Aggregation over output values through ML approximation

The dataset passed to the ML algorithm is $\{(\widetilde{\mathbf{x}}_{\text{av}}^{(i)}, \bar{\mathbf{y}}^{(i)})\ i = 1, \ldots, m\}$. Solution prediction $\mathbf{y}$ is obtained from the trained approximator f through $\mathbf{y} = f(\widetilde{\mathbf{x}}_{\text{av}}; \boldsymbol{\theta})$.

The implementation of this procedure is straightforward: The required data is generated by sampling all input variables, whether or not they will be available at the time of prediction and this data is fed directly to ML, without any additional transformation. Clearly, the ML model $f(\widetilde{\mathbf{x}}_{\text{av}}; \boldsymbol{\theta})$ cannot directly account for $\widetilde{\mathbf{x}}_{\text{unav}}$. If we can invoke a uniform law of large numbers to claim stochastic convergence of average generalization error arising from validation towards expected error (i.e. risk) with respect to the distribution of the variables that are sampled in the data (e.g., Vapnik, 1999). Then we can argue that this aggregation procedure in effect minimizes an approximation to the expected error with respect to $\widetilde{\mathbf{x}}_{\text{av}}$ *as well as* $\widetilde{\mathbf{x}}_{\text{unav}}$. In other words, ML could in this case be viewed as minimizing an approximation to the expected discrepancy between the exact solution $\bar{\mathbf{y}}$ resulting from knowledge of $[\widetilde{\mathbf{x}}_{\text{av}}, \widetilde{\mathbf{x}}_{\text{unav}}]$ and the solution $f(\widetilde{\mathbf{x}}_{\text{av}}; \boldsymbol{\theta})$ based solely on knowledge of $\widetilde{\mathbf{x}}_{\text{av}}$.

4.4.5 Aggregation over output after ML approximation

The dataset passed to the ML algorithm is $\{(\widetilde{\mathbf{x}}^{(i)}, \bar{\mathbf{y}}^{(i)})\ i = 1, \ldots, m\}$. Solution prediction $\mathbf{y}$ is obtained indirectly from the trained approximator through $\mathbf{y} = \int f(\widetilde{\mathbf{x}}_{\text{av}}, \widetilde{\mathbf{x}}_{\text{unav}}) g_3(\widetilde{\mathbf{x}}_{\text{av}}, \widetilde{\mathbf{x}}_{\text{unav}}) dF_{\widetilde{\mathbf{x}}_{\text{unav}}}$ and $g_3(\cdot, \cdot)$ is a measurement function. The ML model $f(\cdot, \cdot)$ and the measurement function $g_3(\cdot, \cdot)$ should be selected so that the integral may be computed analytically. Otherwise, the requirement to compute the integral through numerical integration for each prediction would defeat the purpose of the procedure. This aggregation procedure is closely related to a topic known as modeling with missing inputs or classification with missing inputs in the deep learning literature (e.g., Goodfellow et al., 2016). This approach consists in learning a probability distribution over all variables and integrating analytically at prediction time with respect to missing variables.

5 Application

We consider the LPP of double-stacked intermodal trains introduced in Section 2. The assignment of containers to slots on the railcars depends on individual characteristics of both. Intermodal containers are characterized by their length, height, standardized type, content and weight. In North America, double-stack intermodal railcars comprise one to five platforms and each platform has a lower and an upper slot where containers may be loaded. In addition, railcars are characterized by the weight capacity and tare of each platform and by the loading capabilities uniquely associated with their standardized type. We can express the loading capabilities with a set of loading patterns enumerating all possible ways in which containers of diverse lengths can be placed in the lower and upper slots of each platform. In general, the loading capabilities cannot be decomposed by platform, which leads to sets of loading patterns of high cardinality.

Mantovani et al. (2018) propose an ILP formulation that can be solved in seconds using a commercial solver. The objective seeks to minimize the total cost of containers left on the ground and the cost of railcars used to load at least one container. Assignment constraints ensure that exactly one loading pattern is associated with each railcar and that each container is assigned to at most one slot. Weight capacity constraints ensure that the weight holding capabilities of the railcars are not exceeded. Other weight related constraints guarantee that the center of mass lies below a given threshold. In addition, there are a number of technical loading constraints pertaining to, for example, specific types of goods (such as refrigerated or hazardous materials) and specific types of containers (such as open-top containers that lack a roof).

The ILP formulation can be used to compute actual load plans in intermodal railway terminals. In this context, the containers and the railcars are physically in the terminal and their characteristics are fully known. The application we focus on here is different. Given a set of containers and a set of railcars, the objective is to predict which containers are to be assigned to railcars and which railcars are to be used in the optimal solution. We wish to compute these predictions at a time when container weights are unavailable. As mentioned in Section 1, such predictions are of interest when making booking decisions. In this case, container weights are unavailable at the time of prediction and predictions must be computed in very short time.

5.1 Aggregation and subselection of containers and railcars

Container data may be transformed by subselecting lengths, heights, standardized types and contents and by aggregating weights. Similarly, railcar data may be transformed by subselecting standardized types and by aggregating weight capacities and tares. Hence, in order to ensure that the LPP remains manageable, container lengths, heights, standardized types, and contents have been subselected to retain only the two most relevant lengths, 40 ft and 53 ft, a single height, a single standardized type and a single content. Similarly, railcar types have been subselected to retain the 10 most numerous ones. The latter comprise close to 90% of the North American fleet.

Exact weight capacities and tares of railcars as well as exact gross weights of containers are unknown at the time of prediction and are therefore aggregated. Aggregating weight capacities and tares of railcars is straightforward since those vary very little for each given standardized type. Hence, population estimates of median capacities and tares conditional on type may be viewed as reasonable representative values. This amounts to performing their aggregation according to the method of Section 4.4.1. In contrast, container weights are highly variable, even conditionally upon the values of other container characteristics, and aggregation is more delicate. Hence, we model uncertainty over container weights explicitly in the data generation so that the output aggregation methods of Section 4.4 may be applied. These methods are preferred in our context over the input aggregation methods of Sections 4.4.1 and 4.4.2 in view of their superior theoretical underpinnings.

Since the tare and the net capacity of a container are closely tied to its length, statistical estimates of median tare and median net capacity conditional on container length are viewed as reasonable representative values. Conditionally on the container not being empty, we model its net weight by a uniform distribution ranging between 10% and 90% of its net capacity given length. We estimate the probability that a container is empty conditionally upon length from container transportation data. Hence, we generate the total weight of a container conditionally on its length as the sum of the median tare and the generated net weight, both conditionally on length.

Section 5.4 reports two parallel sets of predictions in relation with the following aggregation methods: aggregation over output values before ML approximation, henceforth OBefML (Section 4.4.3) and aggregation over output values through ML approximation, henceforth OThrML

(Section 4.4.4).

5.2 Data generation

We partition the available data into four classes, as reported in Table 1. This partition facilitates experiments where models are trained and validated on simpler instances (A, B, C) and tested on harder ones (D).

Class name	Description	# of containers	# of platforms
A	Simple ILP instances	[1, 150]	[1, 50]
B	More containers than A (excess demand)	[151, 300]	[1, 50]
C	More platforms than A (excess supply)	[1, 150]	[51, 100]
D	Larger and harder instances	[151, 300]	[51, 100]

Table 1: Data classes

We generated data according to two different sampling procedures: *One-stage* (1S), with random sampling of number of platforms of railcars belonging to 10 standardized types, random sampling of number of containers of each length (2) and random sampling of weight of each container. *Two-stage* (2S), where in the first stage the number of railcars of each type and the number of containers of each length are selected and in the second stage 100 alternative sets of container weights are selected for each specific sample resulting from the first stage. The objective of 2S sampling is to use the second stage to compute a raw Monte Carlo estimate of the ILP solution achieving median objective value with respect to the distribution of weights, conditionally on values drawn in first stage. Container gross weights are sampled similarly in 1S and 2S: First, the proportion of empty containers is randomly generated. Second, the load of each non-empty container is randomly generated between a lower and an upper ratio of its nominal capacity given its length. Third, the gross weight of each container is equated to the sum of its load and its tare given its length. Table 2 reports the number of examples for each combination of sampling procedure and data class. We randomly divided each dataset into training (64%), validation (16%) and test (20%) sets.

Each generated instance of the ILP loading problem was solved with IBM ILOG CPLEX 12.6 down to an optimality gap of at most 5%. The solutions in the resulting problem instance-solution pairs were described with a limited subset of features: number of railcars of each type and number of containers of each length used in the loading solution. The objective of the loading ILP problem was set so as to enforce the following priorities in lexicographic order: maximize total number of containers loaded, minimize total length of railcars used, maximize total length of containers loaded. Table 2 reports the percentiles of computation times per instance using three out of the six cores of an Intel Xeon X5650 Westmere 2,67 GHz processor.

Sampling procedure	Data class	# instances	Percentiles time (s) P_5	P_{50}	P_{95}
1S	A	20M	0.007	0.48	1.67
2S	A	20M	0.011	0.64	2.87
2S	B	20M	0.02	1.26	3.43
2S	C	20M	0.72	2.59	6.03
2S	D	10M	2.64	5.44	20.89

Table 2: Data generation

5.3 ML approximation

The predictive models are based on feedforward neural networks, a.k.a. multilayer perceptrons (MLP). From their introduction several decades ago until recently, MLP's had demonstrated modest success in ML. However, as highlighted in Section 3, through the recent algorithmic advances that have occurred in the subfield of ML known as deep learning, they have become simple but powerful generic approximators. They are useful in real applications whenever input and output vectors have short fixed lengths and do not feature complex structures.

The mapping between inputs and outputs could be interpreted as a classification or as a regression problem and we implemented the two corresponding architectures. The resulting families of models are hereafter named ClassMLP and RegMLP, respectively. Both families feature 12 units in their input layer (one for each railcar type and for each container length) and rectified linear units (ReLU) are used as activation functions in their hidden layers. The two families differ with respect to their output layer, the manner in which input-output constraints are upheld and the loss function used in their training. On the one hand, ClassMLP outputs 12 discrete probability distributions (one for each railcar type and for each container length) that are each modeled with a softmax operator. The supports of these distributions are the sets of possible numbers of railcar of each type and of containers of each length. Thus, concatenation of the 12 distributions yields an output layer of size 812 when the numbers of railcar platforms of each type and of containers of each length used in the solution may respectively vary from 1 to 50 and from 1 to 150. The following constraints are enforced at training, validation and testing times: for each type of railcar and length of container, the number in output cannot exceed the number in input. Training is conducted through pseudo-likelihood maximization. On the other hand, RegMLP outputs 12 scalars that are rounded to the nearest integer, input-output inequalities are enforced only at validation and testing times and training is conducted through minimization of the sum of absolute errors incurred in predicting output numbers for railcars of each type and containers of each length. For both families, the assumption that outputs are conditionally mutually independent given inputs is implicit to their architecture. Training of both ClassMLP and RegMLP was performed with mini-batch stochastic gradient descent and the learning rate adaptation was governed by the Adam (adaptive moment estimation) method. Regularization was ensured by early stopping. Hyperparameter selection included number and width of hidden layers as well as L_1 and L_2 regularization terms coefficients. Sets of hyperparameter values were generated randomly and the preferred set was determined according to validation results. The ranges of values considered were as follows: [3, 13] for the number of hidden layers, [300, 1000] for the number of units per hidden layer, [0, .001] for the L_1 and L_2 regularization coefficients.

In order to establish benchmark results, we also implemented a logistic regression (classMLP without hidden layers), a linear regression (RegMLP without hidden layers) and two simple greedy algorithms. Greedy algorithm 1 (HeurV) accounts for the total number of slots available on each railcar but disregards all other constraints pertaining to the loading problem. In contrast, greedy algorithm 2 (HeurS) accounts for all constraints relevant to the loading problem, namely: (i) 53 ft containers can only be assigned to 53 ft slots whereas 40 ft containers can be assigned to any slot, (ii) each railcar features known numbers of 53 ft and 40 ft slots, for some railcars, (iii) some 53 ft slots are available above only provided 40 ft containers are loaded below. This algorithm also attempts to account for the lexicographic objective.

Algorithm 1 Very simple greedy heuristic (HeurV)

```
while unassigned cont. and avail. car do
  for all car type in car types do
    for all car matching car type do
      assign avail. cont(s) to car, alternating if possible between 40' and 53' cont(s);
    end for
  end for
end while
```

5.4 Results

Once the models have been trained, computing the predicted output values for a particular set of input values requires negligible time.

We measure the predictive performance with the sum (MAE) of mean absolute prediction error over the numbers of used slots (MAE_{slots}) and of mean absolute error over the numbers of loaded

Algorithm 2 Simple greedy heuristic (HeurS)

```
while unassigned 53' cont. and avail. car with usable 53' slot(s) do
    choose shortest among cars with greatest number of usable 53' slots not exceeding number of
    53' conts still to assign;
    otherwise, choose shortest among cars with smallest number of usable 53' slots;
    assign as many 53' conts as possible to usable 53' slots on car;
    assign as many 40' conts as possible to remaining available slots of car;
end while
while unassigned 40' cont. and avail. car do
    choose shortest among cars with greatest number of slots not exceeding number of 40' conts
    still to assign;
    otherwise, choose shortest among cars with smallest number of slots;
    assign as many 40' conts as possible to available slots of car;
end while
```

containers (MAE_{conts}) in output solution. They are defined as follows,

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{12}|\widehat{y}_j^{(i)} - y_j^{(i)}|s_j, \tag{1}$$

$$MAE_{slots} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{10}|\widehat{y}_j^{(i)} - y_j^{(i)}|s_j, \tag{2}$$

$$MAE_{conts} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=11}^{12}|\widehat{y}_j^{(i)} - y_j^{(i)}|, \tag{3}$$

where $s_j,\ j = 1, ..., 10$, equals the number of slots on railcar type j. $s_{11} = s_{12} = 1$ and do not appear in (3). We report parallel sets of results for two aggregation methods. On the one hand, the OBefML method that predicts the solution featuring the median objective value conditionally on available inputs. On the other hand, the OThrML method that predicts the solution minimizing the expected discrepancy between the predicted and exact solutions. We stress that selecting one of the two methods should not be based solely on their predictive performances and computing costs but first and foremost on the interpretation of the predictions. In this context, OThrML provides the predictions that are the most relevant to our application. Nevertheless, we include OBefML for the sake of illustration.

5.4.1 Testing errors

Table 3 reports the MAE (1) incurred by each model over an independent data set similar to that used for training and validation, when aggregation of the unknown container weights is performed with method OBefML (Section 4.4.3). Standard deviations of the estimates are shown in parentheses. The OBefML aggregation method is based on the 2S sampling procedure. We train and validate the models on class A data only which corresponds to 200K i.i.d. examples (2S-Abef, second column in the table) and on the union of A, B and C data which corresponds to 600K i.i.d. examples (2S-ABCbef, third column in the table). Note that each figure reported for ClassMLP or for RegMLP is attached to the most favorable set of hyperparameters according to validation step.

Similarly, Table 4 reports the MAE (1) incurred over an independent data set similar to that used for training and validation, when aggregation of unknown container weights is performed with OThrML (Section 4.4.4). In this case we can use data generated with 2S or 1S sampling procedure. With the 2S data, instead of selecting the instance with median value of the objective function (OBefML), we choose an instance at random. We report results for models trained and validated using 2S class A data that corresponds to 200K i.i.d. examples (2S-Athr, second column in the table), 1S class A data that corresponds to 20M i.i.d. examples (1S-Athr, third column in the table) and finally the union of 2S A, B and C data which corresponds to 600K i.i.d. examples (2S-ABCthr, fourth column in the table).

A number of findings emerge from examination of Tables 3 and 4. For both aggregation methods and all datasets, the average performances of both feedforward neural network models –

ClassMLP and RegMLP – are very good and considerably better than those of the benchmarks (logistic regression, linear regression and the two heuristics). RegMLP features a slightly better average performance than ClassMLP, except for the 1S-Athr data. A possible explanation for this is that the pseudo-likelihood function used as a surrogate for MAE in training ClassMLP does not account for the magnitude of the prediction errors.

The OThrML aggregation method performs very well. The MAE for 2S-Athr and 1S-Athr data are only 1.304 and 0.985, respectively. In this context, we also note that the marginal value of using 100 times more observations with OThrML is fairly small since MAE for RegMLP modestly increases from 0.985 to 1.304. Moreover, the decrease in performance when enlarging the training-validation set from A to include the more difficult instances (union of data classes A, B and C) is quite modest for both aggregation methods. For example, MAE increases from from 0.967 to 1.597 for 2S-Abef compared to 2S-ABCbef. An increase of similar magnitude can be seen for the other aggregation method (OThrML).

Data **# examples**	**2S-Abef** 200K	**2S-ABCbef** 600K
ClassMLP	1.181 (0.016)	1.807 (0.012)
LogReg	5.911 (0.029)	8.944 (0.027)
RegMLP	0.967 (0.016)	1.597 (0.012)
LinReg	18.301 (0.094)	40.014 (0.084)
HeurV	14.744 (0.074)	27.269 (0.083)
HeurS	17.808 (0.083)	31.465 (0.089)

Table 3: Testing over data similar to that used in training-validation: OBefML aggregation

Data **# examples**	**2S-Athr** 200K	**1S-Athr** 20M	**2S-ABCthr** 600K
ClassMLP	1.481 (0.018)	0.965 (0.002)	2.312 (0.014)
LogReg	5.956 (0.029)	5.887 (0.003)	9.051 (0.027)
RegMLP	1.304 (0.017)	0.985 (0.002)	2.109 (0.014)
LinReg	18.306 (0.094)	18.372 (0.009)	39.907 (0.084)
HeurV	14.733 (0.075)	14.753 (0.008)	27.24 (0.083)
HeurS	17.841 (0.083)	17.842 (0.008)	31.448 (0.089)

Table 4: Testing over data similar to that used in training-validation: OThrML aggregation

Figure 2 displays the MAE in relation to the number of available slots and the number of available containers (input) for the OBefML aggregation method. Figure 3 displays the same information for the OthrML aggregation method. Both figures relate to RegMLP model and class A data: 2S-Abef and 1S-Athr, respectively. The results show that errors occur mainly in conditions of excess supply or excess demand and confirm the lower standard deviation of 1S-Athr.

Table 5 provides information on the distribution of the CPU time required to compute a prediction based on input data similar to that used for training and validation of the predictor. For example, the median time required to compute a prediction based on model RegMLP and aggre-

Data	**2S-Abef**			**2S-Athr**		
# examples	200K			200K		
Percentiles	P_5	P_{50}	P_{95}	P_5	P_{50}	P_{95}
ClassMLP	2.6	2.9	3.2	2.3	2.4	2.5
RegMLP	0.7	0.8	1.0	0.5	0.6	0.8
HeurV	0.3	0.4	0.8	0.3	0.4	0.8
HeurS	0.3	0.7	1.6	0.3	0.7	1.5

Table 5: Prediction time per instance (milliseconds)

gation method OThrML when input belongs to class A is 0.6 milliseconds. As evidenced by the closeness of the 5th, 50th and 95th percentiles, the distribution of the prediction time is highly concentrated and we ought to expect very little variations among computation times around the median value. Furthermore, it is expected that the figures of Table 5 will vary little across input classes with a similar model. Computational speed should instead depend on model complexity (in our case number and width of hidden layers). Hence, whereas the exact solution of a loading problem may require a median time ranging from 0.48 to 5.44 seconds according to the exact class of the input (Table 2), the prediction of the solution with model RegMLP and aggregation method OThrML is expected to require a time close to the indicated median of 0.6 milliseconds.

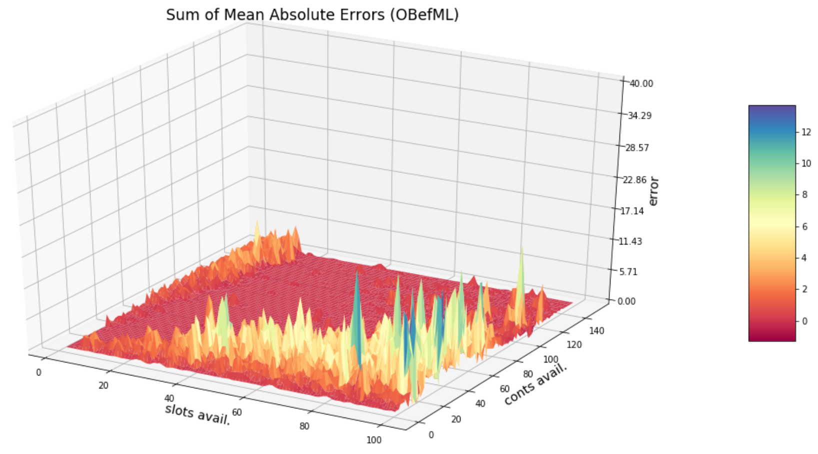


Figure 2: MAE over instances with specified numbers of available slots and containers, RegMLP model and 2S-Abef data

5.4.2 Extraneous errors

In view of the higher costs of generating harder instances, it is desirable that models that are trained and validated on simpler instances generalize to harder instances without specific training and validation. In contrast with Section 5.4.1 where testing was conducted on data similar to that used for training and validation, this section reports the testing performance over a set of class D data containing the largest instances. We emphasize that instances of this nature have not been used for training-validation.

Table 6 reports the MAE for the exact same models as in the previous section and the OBefML aggregation method. Standard deviations of the estimates are shown in parentheses. Moreover, we report the range in performance achieved over all hyperparameter sets between brackets. Since the classification models cannot produce predictions for instances that are of sizes differing from the ones used for training-validation, we do not report any values for 2S-Abef data (NA in the first

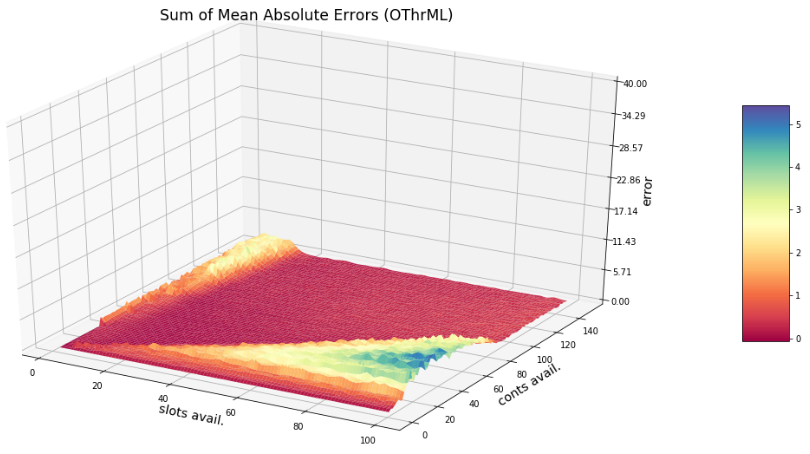


Figure 3: MAE over instances with specified numbers of available slots and containers, RegMLP mode and 1S-Athr data

column of the table). In this respect, the regression models have a clear advantage. Similarly, we report the results for the OThrML aggregation method in Table 7.

The following findings emerge: The performance of the models whose testing results are reported in Tables 3 and 4 is still good when measured on the harder, extraneous problem instances of class D. It is clearly beneficial to train and validate on classes B and C in addition to class A. For example, MAE of RegMLP with aggregation method OBefML decreases from 2.852 to 0.323, and for the OThrML aggregation method it decreases from 4.412 to 2.372.

The MAE reported between brackets indicate that the range of the performances achieved on D over all hyperparameter sets considered at validation is wide. For example, it varies between 2.481 and 24.702 for RegMLP when training and validating over 1S-Athr and testing over 2S-Dthr. We note that some hyperparameter sets achieving close to best validation performance perform poorly on D.

The range of performances achieved on D over all hyperparameter sets considered at validation is reduced when training and validating on B and C in addition to A. Hence, whereas there appears to be a high risk associated with the application of a model on extraneous data (data with characteristics that have not been encountered during training-validation), this risk may possibly be mitigated by reducing the extent of the dissimilarities between training-validation and extraneous data. This opens up for questions concerning alternative data generation procedures. For example, less expensive generation of the hardest instances could be accomplished by setting the minimum optimality gap to a more lenient value. Training and validation could then be performed on the instances as well.

Table 8 provides information on the distribution of the CPU time required to compute a prediction based on extraneous data. The remarks made in relation to Table 5 are still relevant and the figures reported here are not not markedly different.

6 Conclusion and future work

In this paper, we proposed a supervised machine learning approach for predicting descriptions of solutions to discrete stochastic optimization problems in very short computing time. The required data is generated by independently sampling and solving a large number of deterministic problems. Moreover, we presented aggregation methods to address situations where there is imperfect knowledge of the problem instances at the time of prediction.

We illustrated the methodology with a train load planning problem, where some features of the inputs (problem instances) – in this case container weights – are unavailable at the time of

Training-validation data **# examples**	**2S-Abef** 200K	**2S-ABCbef** 600K
ClassMLP	NA	14.823 [9.532, 23.782] (0.061)
LogReg	NA	28.171 (0.048)
RegMLP	2.852 [0.741, 9.052] (0.011)	0.323 [0.323, 1.109] (0.052)
LinReg	22.94 (0.047)	71.322 (0.054)
HeurV	32.098 (0.069)	32.098 (0.069)
HeurS	41.792 (0.077)	41.792 (0.077)

Table 6: Testing on class D instances (not used for training-validation), OBefML aggregation

Training-validation data **# examples**	**1S-Athr** 20M	**2S-ABCthr** 600K
ClassMLP	NA	14.831 [13.161, 23.892] (0.072)
LogReg	NA	29.568 (0.065)
RegMLP	4.412 [2.481, 24.702] (0.050)	2.372 [2.355, 3.305] (0.051)
LinReg	24.560 (0.064)	72.847 (0.060)
HeurV	33.737 (0.085)	33.737 (0.085)
HeurS	43.303 (0.089)	43.303 (0.089)

Table 7: Testing on class D instances (not used for training-validation), OThrML aggregation

Data	**2S-Dbef**			**2S-Dthr**		
# examples	10M			10M		
Percentiles	P_5	P_{50}	P_{95}	P_5	P_{50}	P_{95}
RegMLP	0.7	1.5	2.3	0.6	1.3	1.9
HeurV	0.3	1.0	1.6	0.3	1.0	1.6
HeurS	1.9	2.9	4.0	1.9	3.9	4.1

Table 8: Prediction time in milliseconds per instance with extraneous data

prediction. The input and output vectors were by design of small and fixed size, therefore making it possible to apply standard deep learning models and algorithms. We modeled the problem as both a classification and a regression problem.

The results have shown that a regression feedforward neural network had the best performance overall. Remarkably, the solutions could be predicted with a high accuracy in very short computing time (in the order of a millisecond or less). In fact, the time required to predict the solution descriptions under imperfect information using ML is much shorter than the time required to solve a single deterministic instance with an ILP solver. The results have also shown that the regression feedforward neural network model that was trained and validated on simpler instances could generalize reasonably well to harder instances without specific training and validation. However, the variations over the hyperparameter sets considered during the validation step were large when the nature of the data was very dissimilar.

We believe the methodology successfully developed in this paper can have a relevant impact for real-time decision problems under imperfect information where a fully detailed solution is not required. Of course, many interesting questions remain to be addressed and, among the areas for future research (both related to our specific application and in general), we mention:

- Compare the results to those obtained with a stochastic programming approach.
- Among the five different aggregation methods described in the paper only two were used in the current investigation. We believe that the third method involving aggregation over output, i.e. "aggregation over output after ML", known as "classification with missing inputs" in the deep learning literature, could also be a promising direction to investigate. It was not implemented in our application since it is costlier and the performance of the simpler methods was already very good.
- The input and output structures considered were designed to be small and of fixed size. A direction for future research is to predict more detailed solutions where the input and output structures would be of large and variable size and would possibly feature additional constraints. The trade-off between level of detail and uncertainty of the input is a question by itself. In this context, an approach related to pointer networks (Vinyals et al., 2015) is a potential avenue.
- Data generation is the costliest part of the methodology. Future research should investigate active learning, where the trade-off between the cost of generating data and the predictive performance can be controlled.

Acknowledgements

This research was funded by the Canadian National Railway Company (CN) Chair in Optimization of Railway Operations at Université de Montréal and a Collaborative Research and Development Grant from the Natural Sciences and Engineering Research Council of Canada (CRD-477938-14). Computations were made on the supercomputers Briarée and Guillimin, managed by Calcul Québec and Compute Canada. The operation of these supercomputers is funded by the Canada Foundation for Innovation (CFI), the Ministère de l'Économie, de la Science et de l'Innovation du Québec (MESI) and the Fonds de recherche du Québec - Nature et technologies (FRQ-NT). The research is also partially funded by the "IVADO Fundamental Research Project Grants" under project entitled "Machine Learning for (Discrete) Optimization". We are also grateful for important insights obtained through discussions with Michael Hewitt and Matteo Fischetti.

References

Ahmadian, Sara. *Approximation Algorithms for Clustering and Facility Location Problems*. PhD thesis, University of Waterloo, 2017.

Arlot, S. and Celisse, A. A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4:40–79, 2010.

Asmussen, S. and Glynn, P. W. *Stochastic simulation : algorithms and analysis*. Stochastic modelling and applied probability 57. Springer, New York, 2010.

Avramidis, A. N. and Wilson, J. R. Correlation-induction techniques for estimating quantiles in simulation experiments. *Operations Research*, 46(4):574–591, 1998.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. Neural combinatorial optimization with reinforcement learning. *CoRR*, abs/1611.09940, 2016.

Bertsekas, D. and Tsitsiklis, J. *Neuro-Dynamic Programming.* Athena Scientific, 1996.

Bertsimas, D. and Kallus, N. From Predictive to Prescriptive Analytics. *ArXiv e-prints arXiv:1402.5481*, February 2014.

Bertsimas, D., Kallus, N., and Hussain, A. Inventory management in the era of big data. *Production and Operations Management*, 25(12):2006–2009, 2016.

Billingsley, P. *Probability and Measure.* Wiley Series in Probability and Statistics. John Wiley and Sons, third edition, 1995.

Birge, J. R. and Louvaux, F. *Introduction to Stochastic Programming.* Springer Series in Operations Research and Financial Engineering. Springer New York, New York, NY, 2011.

Chang, K., Krishnamurthy, A., Agarwal, A., III, H. D., and Langford, J. Learning to search better than your teacher. In *ICML*, 2015.

Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B., and Song, L. Learning combinatorial optimization algorithms over graphs. In *NIPS*, 2017.

Daumé, H., Langford, J., and Marcu, D. Search-based structured prediction. *Machine Learning*, 75(3):297–325, 2009.

Fischetti, M. and Fraccaro, M. Using OR + AI to predict the optimal production of offshore wind parks: A preliminary study. In *Optimization and Decision Science: Methodologies and Applications*, volume 217, pages 203–211. Springer, 2017.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning.* MIT Press, 2016.

Han, J. and E, W. Deep learning approximation for stochastic control problems. *ArXiv e-prints arXiv:1611.07422*, November 2016.

Hastie, T., Tibshirani, R., and Friedman, J. *The elements of statistical learning: data mining, inference and prediction.* Springer, 2nd edition, 2009.

Hesterberg, T. C. and Nelson, B. L. Control variates for probability and quantile estimation. *Manage. Sci.*, 44(9):1295–1312, September 1998.

Hsu, J. C. and Nelson, B. L. Control variates for quantile estimation. *Management Science*, 36(7): 835–851, 1990.

Kaempfer, Y. and Wolf, L. Learning the Multiple Traveling Salesmen Problem with Permutation Invariant Pooling Networks. *ArXiv e-prints arXiv:1402.5481*, April 2018.

Kall, P. and Wallace, S. W. *Stochastic Programming.* John Wiley & Sons, 1994.

Khalil, E. B., Le Bodic, P., Song, L., Nemhauser, G. L., and Dilkina, B. N. Learning to branch in mixed integer programming. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pages 724–731, 2016.

Kruber, M., Lubbecke, M., and Parmentier, A. Learning when to use a decomposition. In Salvagni, n. D. and Lombardi, M., editors, *Integration of AI and OR Techniques in Constraint Programming*, volume 10335 of *Lecture Notes in Computer Science*, 2017.

LaMaire, B. F. J. and Mladenov, V. M. Comparison of neural networks for solving the travelling salesman problem. In *11th Symposium on Neural Network Applications in Electrical Engineering*, pages 21–24, 2012.

Law, A. M. *Simulation modeling and analysis.* McGraw-Hill series in industrial engineering and management science. McGraw-Hill, Boston, 5th ed.. edition, 2014.

Li, K. and Malik, J. Learning to optimize. *ArXiv e-prints arXiv:1606.01885*, June 2016.

Lodi, A. On the role of (machine) learning in (mathematical) optimization. In Salvagnin, D. and Lombardi, M., editors, *Integration of AI and OR Techniques in Constraint Programming*, 2017.

Lodi, A. and Zarpellon, G. On learning and branching: A survey. *TOP*, 25(2):207–236, 2017.

Mantovani, S., Morganti, G., Umang, N., Crainic, T. G., Frejinger, E., and Larsen, E. The load planning problem for double-stack intermodal trains. *European Journal of Operational Research*, 267(1):107–119, 2018.

Murphy, K. P. *Machine Learning: A Probabilistic Perspective.* The MIT Press, 2012.

Oroojlooyjadid, A., Snyder, L. V., and Takác, M. Applying deep learning to the newsvendor problem. *ArXiv e-prints arXiv:1607.02177*, July 2016.

Powell, W. B. *Approximate Dynamic Programming Solving the Curses of Dimensionality.* Wiley, 2011.

Sarwar, F. and Bhatti, A. A. Critical analysis of hopfield's neural network model for tsp and its comparison with heuristic algorithm for shortest path computation. In *Proceedings of 2012 9th International Bhurban Conference on Applied Sciences Technology (IBCAST)*, pages 111–114, 2012.

Shapiro, A., Dentcheva, D., and Ruszczynski, A. *Lectures on Stochastic Programming: Modeling and Theory, Second Edition.* Society for Industrial and Applied Mathematics, 2014.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

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

Smith, K. A. Neural networks for combinatorial optimization: A review of more than a decade of research. *INFORMS Journal on Computing*, 11(1):15–34, 1999.

Sutton, R. and Barto, A. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

Szepesvári, C. Algorithms for reinforcement learning. In Brachman, R. J. and Dietterich, T., editors, *Synthesis Lectures on Artificial Intelligence and Machine Learning.* Morgan & Claypool, 2010.

van der Laan, M. J., Dudoit, S., and Keles, S. Asymptotic optimality of likelihood-based cross-validation. *Statistical Applications in Genetics and Molecular Biology*, 3(1):1–23, 2003.

Vapnik, V. *The Nature of Statistical Learning Theory.* Information Science and Statistics. Springer New York, 1999.

Vazirani, V. V. *Approximation Algorithms.* Springer, 2003.

Vinyals, O., Fortunato, M., and Jaitly, N. Pointer Networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

Wouter, K., van Hoof, H., and Welling, M. Attention Solves Your TSP, Approximately. *ArXiv e-prints arXiv:1803.08475v2*, June 2018.

Zhou, L. A survey on contextual multi-armed bandits. *CoRR*, abs/1508.03326, 2015.