CANADA EXCELLENCE RESEARCH CHAIR

DATA SCIENCE FOR REAL-TIME DECISION-MAKING

PREDICTING TACTICAL SOLUTIONS TO OPERATIONAL PLANNING PROBLEMS UNDER IMPERFECT INFORMATION

Eric Larsen
Sébastien Lachapelle
Yoshua Bengio
Emma Frejinger
Simon Lacoste-Julien
Andrea Lodi

January 2019

DS4DM-2019-003

POLYTECHNIQUE MONTRÉAL
DÉPARTEMENT DE MATHÉMATIQUES ET GÉNIE INDUSTRIEL
Pavillon André-Aisenstadt
Succursale Centre-Ville C.P. 6079
Montréal - Québec
H3C 3A7 - Canada
Téléphone: 514-340-5121 # 3314

Predicting Tactical Solutions to Operational Planning Problems under Imperfect Information

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

January 24, 2019

Abstract

This paper offers a methodological contribution at the intersection of machine learning and operations research. Namely, we propose a methodology to quickly predict tactical solutions to a given operational problem. In this context, the tactical solution is less detailed than the operational one but it has to be computed in very short time and under imperfect information. The problem is of importance in various applications where tactical and operational planning problems are interrelated and information about the operational problem is revealed over time. This is for instance the case in certain capacity planning and demand management systems.

We formulate the problem as a two-stage optimal prediction stochastic program whose solution we predict with a supervised machine learning algorithm. The training data set consists of a large number of deterministic (second stage) problems generated by controlled probabilistic sampling. The labels are computed based on solutions to the deterministic problems (solved independently and offline) employing appropriate aggregation and subselection methods to address uncertainty. Results on our motivating application in load planning for rail transportation show that deep learning algorithms produce highly accurate predictions in very short computing time (milliseconds or less). The prediction accuracy is comparable to solutions computed by sample average approximation of the stochastic program.

Keywords: stochastic programming, integer linear programming, supervised learning, deep learning

1 Introduction

Operations research (OR) has been successful in developing methodologies and algorithms 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 wealth of applications. Conversely, Machine learning (ML), and its subfield known as *deep learning* (Goodfellow et al., 2016), have 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. Building on top of those two respective strengths, we propose a methodology where OR and ML complement each other and are respectively applied to the tasks they are best suited for in order to address an overall decision problem that we could not solve otherwise.

The overall problem we address is that of achieving accurate and timely decision support at the tactical level for a given operational problem. We assume that we can compute solutions to the operational problem under full information using an existing deterministic optimization model and a solver. Ahead of the time at which the operational problem is solved, we wish to predict certain

[*]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, Polytechnique Montréal

characteristics of the operational solution based on currently available (partial) information. We call such a characterization a *tactical solution description*. The level of detail presented in a tactical solution description may be lesser than that required in the operational solution since its role is to support early planning decisions rather than to set out a detailed implementation. In comparison, a *strategic solution description* would comprise even fewer details, possibly only the predicted value of the optimization criterion.

Computing accurate tactical solution descriptions can pose difficult challenges due to restrictions in the computational time budget and to imperfect availability of information. Indeed, we assume that information about the decision problems under consideration is revealed progressively over time and that full information is only available at the time when the operational problem is solved. Furthermore, we attend to applications where tactical solutions have to be computed repeatedly and in very short time. In brief, we propose a novel methodology that makes it possible to compute solutions to a stochastic (tactical) optimization problem repeatedly and in very short computing time. This problem is of importance in various applications where tactical and operational planning problems are interrelated, for example, in capacity planning and demand management systems. We consider such an application where the tactical capacity planning solution depends on the solution to a kind of packing problem at the operational level. That particular setting occurs, e.g., in airline cargo and intermodal rail transportation as well as less than truckload (LTL) shipping.

The idea underlying the proposed methodology is simple and attractive: we predict the tactical solution descriptions using supervised ML, where the training data consists of a large number of deterministic operational problems that have been solved independently and offline. The methodology consists of four broad steps: First, operational problem instances under perfect information (the input) are sampled from the space relevant to the application at hand and solved using an existing deterministic optimization model and a solver. Second, the detailed operational solutions are synthesized according to the chosen tactical solution description (the output). Third, a ML approximator is trained based on the generated input-output data while accounting for imperfect information regarding certain features of the input. Fourth, this approximator is used to generate predictions for the tactical solutions of actual problem instances based on the available information. (Note that this paper employs the ML vocabulary. Hence, the ML approximator, i.e., predictor, acts as a heuristic and not as an approximation algorithm). 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 in order to achieve the goal of a low average cost for applications, where the predictions must be used with a high frequency.

Some important challenges arise in this undertaking. First, the structure of the output is tied to the chosen operational solution description and can be of fixed or variable size. Second, inputs and outputs defined in the chosen solution description may be related by a number of explicit constraints that must also be satisfied by the predictions. Third, uncertainty regarding a subset of the inputs needs to be addressed through appropriate sampling, subselection and aggregation methods. The designs of the ML models and algorithms depend on how these challenges are met.

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 deterministically 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 – on how many of a given set of containers of different types can be loaded on a given set of railcars of different types and how many of the railcars of each type are needed. The decisions 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 computed in very short time (fraction of a second). This problem is of practical importance and features characteristics that make it useful for illustrating the proposed methodology: Although an integer linear programming (ILP) formulation of the problem can be solved under full information by commercial ILP solvers in reasonable time[1] (Mantovani et al., 2018), this formulation cannot be used for the application because the container weights are not

[1] Seconds to minutes.

known at the time of booking. Moreover, for the purposes of booking, the operational solution (assignment of each container to positions on railcars) is unnecessarily detailed.

Short Literature Review and Pointers. 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, limited success was ultimately achieved and this area of research was left nearly inactive from the beginning of this century. A renewal of interest has been kindled by the successes witnessed in deep learning and the state of the art is advancing at an increasing pace.

The most successful locus of synergies between OR and ML, which has attracted a huge amount of attention in the last ten years, is the introduction of continuous optimization methods originating in OR to algorithms used in ML, most notably the stochastic gradient descent algorithm. The interested reader is referred to, e.g., Curtis and Scheinberg (2017) for a tutorial.

More recent is the exploratory bulk of work devoted to the use of ML for discrete optimization problems. On the one hand, ML is used as a tool for approximating complex and time-consuming tasks in OR algorithms, arguably the most important one being branching for enumerative approaches to NP-hard problems (see, e.g., Lodi and Zarpellon, 2017, for a survey). On the other hand, ML is used to more directly solve (of course, heuristically) discrete optimization problems. Different learning algorithms can be used for this purpose. Closest to our work are those based on supervised learning, like, for example, Vinyals et al. (2015) that focuses on predicting fully detailed solutions to the famous Traveling Salesman Problem. However, those supervised learning algorithms focus on deterministic problems. The interested reader is refereed to Bengio et al. (2018) for a very recent survey on all aspects of ML for discrete optimization.

Finally, although not strictly relevant for the topic of this paper, it is important to mention the extensive work in the use of discrete optimization techniques, especially integer programming, within classical ML tasks like regression, classification and support vector machine, see, for example, Bertsimas and Shioda (2017), Günlük et al. (2017), Belotti et al. (2016), just to mention a few.

Paper contribution. The paper offers four main contributions.

1. Although state-of-the-art stochastic programming methodologies are available to solve the stochastic optimization problem formally defined in the next section, in this paper we shift paradigm by proposing a novel methodology rooted in supervised learning for predicting tactical descriptions of optimal operational solutions where only partial information on the exact operational problem is available at the time of prediction.

2. With respect to the current literature using ML for OR, the proposed methodology combines discrete optimization and ML in an innovative way, i.e., by integrating an ML predictor / approximator to deal with the data uncertainty that is inherent to the strategic and tactical planning levels. Deterministic settings may be viewed as special cases. For example, Fischetti and Fraccaro (2017) who consider, at the strategic level, a deterministic wind farm layout optimization problem and use ML to predict the objective values achieved by candidate sites.

3. In contrast with the existing alternatives offered by approximate stochastic programming and the recently proposed reinforcement learning approach by Nair et al. (2018), our methodology built on ML anticipates calculations by generating, in advance, a *prediction function* instead of pointwise solutions on demand. Hence, it does not require to solve deterministic second-stage problems at prediction time. We illustrate through an extensive computational evaluation on our real-world application how this comparative advantage allows to build both fast and accurate predictors.

4. Our methodology relies on existing ML models and algorithms. This is a key advantage since we can benefit from the recent advances in deep learning. The methodology leads to state-of-the-art results, i.e., solves a problem that otherwise would have been unsolvable to that extent of accuracy in the allowed time budget, essentially online.

The remainder of the paper is structured as follows: Section 2 defines the prediction problem under consideration and discusses existing solution methods from the field of stochastic programming. Section 3 delineates the proposed methodology and Section 4 presents a detailed application of the methodology and reports the results. Finally, Section 5 summarizes the content of the paper, reviews outstanding issues and describes directions for future research.

2 The prediction problem

Let a particular instance of an operational (deterministic) optimization problem be represented by the input feature vector $\mathbf{x}$. The optimal operational solution (i.e., that containing values of all decision variables) is $\mathbf{y}^*(\mathbf{x}) :\equiv \arg\inf_{\mathbf{y}\in\mathcal{Y}(\mathbf{x})} C(\mathbf{x},\mathbf{y})$, where $C(\mathbf{x},\mathbf{y})$ and $\mathcal{Y}(\mathbf{x})$ denote the cost function and the admissible space, respectively. Ahead of the time at which the operational problem is solved, we wish to predict certain characteristics of the optimal operational solution, based on currently available information. We call such a characterization a *tactical solution description*. Information on a subset of the feature vector $\mathbf{x}$ may be unavailable or incomplete at the time of prediction and we define the partition $\mathbf{x} = [\mathbf{x}_\mathrm{a}, \mathbf{x}_\mathrm{u}]$ accordingly, where $\mathbf{x}_\mathrm{a}$ contains available features and $\mathbf{x}_\mathrm{u}$ unavailable or yet unobserved ones. Furthermore, we denote by $g(.)$ the mapping from the fully detailed operational solution to the tactical solution description featuring the level of detail relevant to the context at hand. Hence, $g(\mathbf{y})$ is the synthesis of the operational solution $\mathbf{y}$ according to the tactical solution description embedded in $g(.)$.

Our goal is to compute or at least approximate the solution $\bar{\mathbf{y}}^*(\mathbf{x}_\mathrm{a})$ to the following two-stage, optimal prediction stochastic programming (see, e.g., Birge and Louvaux, 2011, Kall and Wallace, 1994, Shapiro et al., 2009) problem:

$$\bar{\mathbf{y}}^*(\mathbf{x}_\mathrm{a}) :\equiv \arg \inf_{\bar{\mathbf{y}}(\mathbf{x}_\mathrm{a})\in\bar{\mathcal{Y}}(\mathbf{x}_\mathrm{a})} \Phi_{\mathbf{x}_\mathrm{u}}\{\|\bar{\mathbf{y}}(\mathbf{x}_\mathrm{a}) - g(\mathbf{y}^*(\mathbf{x}_\mathrm{a},\mathbf{x}_\mathrm{u}))\| \mid \mathbf{x}_\mathrm{a}\} \tag{1}$$

$$\mathbf{y}^*(\mathbf{x}_\mathrm{a},\mathbf{x}_\mathrm{u}) :\equiv \arg \inf_{\mathbf{y}\in\mathcal{Y}(\mathbf{x}_\mathrm{a},\mathbf{x}_\mathrm{u})} C(\mathbf{x}_\mathrm{a},\mathbf{x}_\mathrm{u},\mathbf{y}) \tag{2}$$

where $\|\|$ denotes a suitable norm (e.g. the L_1- or L_2-norm when the output has fixed size) and $\Phi_{\mathbf{x}_\mathrm{u}}\{\|.\| \mid \mathbf{x}_\mathrm{a}\}$ denotes either the expectation or a quantile (e.g. the median) operation over the distribution of $\mathbf{x}_\mathrm{u}$, conditional upon $\mathbf{x}_\mathrm{a}$. So, $\bar{\mathbf{y}}^*(\mathbf{x}_\mathrm{a})$ is the optimal prediction of the synthesis of the second stage optimizer $g(\mathbf{y}^*(\mathbf{x}_\mathrm{a},\mathbf{x}_\mathrm{u}))$, conditionally on information available at first stage. Finally, $\mathcal{Y}(\mathbf{x}_\mathrm{a},\mathbf{x}_\mathrm{u})$ is the admissible space defined by the set of constraints relevant to the operational context, whereas $\bar{\mathcal{Y}}(\mathbf{x}_\mathrm{a})$ is defined only by the set of constraints relevant to the tactical context.

In real-time or repeated applications, we need to generate solutions to (1) and (2) at a high speed for any value of $\mathbf{x}_\mathrm{a}$. Whenever closed-form solutions are unavailable, which usually occurs, it is generally prohibitive to compute a solution to (1) and (2) on demand for every particular value of $\mathbf{x}_\mathrm{a}$ encountered. As detailed in Section 3, our methodology generates a prediction function that can take any value of $\mathbf{x}_\mathrm{a}$ as input and outputs accurate predictions $\widehat{\mathbf{y}}^*(\mathbf{x}_\mathrm{a})$ of $\bar{\mathbf{y}}^*(\mathbf{x}_\mathrm{a})$. The predictions are given by $\widehat{\mathbf{y}}^*(\mathbf{x}_\mathrm{a}) \equiv f(\mathbf{x}_\mathrm{a};\boldsymbol{\theta})$ where $f(\cdot;\cdot)$ is a particular ML model and $\boldsymbol{\theta}$ is a vector of parameters.

Stochastic Programming. A number of alternative approaches and methods are available from the field of stochastic programming to address the problem defined by (1) and (2). For general specifications of $C(\mathbf{x}_\mathrm{a},\mathbf{x}_\mathrm{u},\mathbf{y})$ and $\mathcal{Y}(\mathbf{x}_\mathrm{a},\mathbf{x}_\mathrm{u})$, that is, essentially, whenever (1) and (2) depart from the extensively researched and documented case of linear programming, stochastic programming resorts to approximate methods involving sampling. These methods originate from two broad areas of research and perspectives: *Monte Carlo stochastic programming* (e.g., de Mello and Bayraksan, 2014, Shapiro, 2003) and *simulation optimization* (e.g., Fu, 2015). In the former, the solution methods may leverage available knowledge about the inner structure of (2). In the latter, (2) is viewed as a black box and the available knowledge consists solely in the ability to evaluate the solution $\mathbf{y}^*(\mathbf{x}_\mathrm{a},\mathbf{x}_\mathrm{u})$. (That is, $\mathbf{y}^*(\mathbf{x}_\mathrm{a},\mathbf{x}_\mathrm{u})$ may be computable with a standard OR solver without any assumption, for instance, about closed-form derivatives with respect to $\mathbf{x}_\mathrm{a}$ or $\mathbf{x}_\mathrm{u}$.) An approximate solution to the problem jointly defined by (1) and (2) may be obtained through one of the methods available from Monte Carlo stochastic programming or simulation optimization *for each particular value of* $\mathbf{x}_\mathrm{a}$.

Methods where sampling occurs once at the outset of the solution process to convert stochastic optimization into deterministic optimization and where sampling occurs throughout the optimization process are respectively said to involve *batch* or *external* sampling and *sequential* or *internal* sampling. Methods originating from the perspective of Monte Carlo stochastic programming that are in principle available to solve (1) and (2) include *sample average approximation* (external) described in (Shapiro et al., 2009, p. 155) and Kim et al. (2015) as well as versions of *stochastic approximation* (internal) where a knowledge of the inner structure of (2) is introduced (Shapiro et al., 2009, p. 230). Methods originating from the perspective of simulation optimization that are

in principle available to solve (1) and (2) include *response surface methods* (internal) (Kleijnen, 2015), *stochastic search* (internal) (Andradóttir, 2015, Hu, 2015, Zabinsky, 2015) and versions of *stochastic approximation* (internal) where knowledge of (2) is limited to the ability to perform evaluations (Chau and Fu, 2015).

Our attention is directed to real-time or high-repetition applications requiring the computation of solutions to (1) and (2) at a high speed. Now, it is typically considerably more time-consuming to solve (1) and (2) on demand for a particular value of $\mathbf{x}_\text{a}$ through one of the existing methods available from approximate stochastic programming than it is to solve a single instance of the deterministic problem (2). As a result, a methodology that succeeds in achieving on-demand prediction times of smaller order than the the time it takes to solve one instance of (2) presents a comparative advantage. The methodology based on ML that we propose satisfies this condition. For example, in the application presented in Section 4, the solution of a single instance of the deterministic problem (2) with a solver requires up to a minute whereas our methodology based on ML can yield predictions on demand within a millisecond. To illustrate the parallel between the proposed methodology and the existing alternatives offered by stochastic programming, Section 4.5 details the steps in the implementation of one such alternative and reports numerical results.

3 Methodology

In this section, the methodology outlined so far is discussed in great detail starting from Section 3.1 that describes the process to generate meaningful data. In Section 3.2, the ML approach is detailed while in Section 3.3, the input/output aggregation as well as the selection of the right level of solution description are discussed.

3.1 Data generation process

The data used for ML derives from operational (deterministic) problem instances and their corresponding solutions. These may either result from controlled probabilistic sampling or, under restrictive conditions, may be collected from historical observations. In our context, controlled probabilistic sampling is advantageous because: (i) the selected sampling distribution is deemed an accurate representation of the stochasticity in the unknown features $\mathbf{x}_\text{u}$ of the problem instances, (ii) it is possible to generate data at will according to a known sampling protocol and to evaluate the performance of ML training and model selection in any arbitrarily defined region in the feature space and (iii) it is possible to generate additional data for further training if the predictive performance is judged insufficient. In contrast, the use of a dataset consisting of historical problem instances is only appropriate when attempting to mimic the behavior reflected in such data. Otherwise, sampling from historical data would likely introduce uncontrollable distortions in the resulting dataset. Indeed, observed instances result from censoring/constraining the space of admissible instances and have resulted from decision processes that should be accounted for but are often unknown in practice. In view of these limitations, we concentrate our attention on data generation through controlled probabilistic sampling.

The first step in the probabilistic data generation process is to sample a set of operational problem instances $\{\mathbf{x}^{(i)},\ i = 1, \ldots, m\}$. Elements of $\mathbf{x}$ that are expected to vary in the intended application should be made to vary and covary in the dataset in commensurate ranges. We can generate problem instances through pseudo-random or quasi-random sampling. Data generation is meant to account for the actual uncertainty about the values of the elements of the input $\mathbf{x}$. In other words, the distributions from which we sample those values are viewed as describing this uncertainty and should be selected accordingly. Whereas simple pseudo-random sampling and stratified pseudo-random sampling are simple and easily applicable, it is also conceivable to apply alternative protocols in order to improve sample efficiency. For instance, importance sampling can artificially increase the abundance of data about infrequently observed characteristics of the problem instances. We refer to, for example, Asmussen and Glynn (2010) and Law (2014) for further details on data sampling and simulations.

We employ an existing solver to generate the operational solutions $\mathbf{y}^*(\mathbf{x}^{(i)})$, $i = 1, \ldots, m$ to the problem instances. We fix the parameters of the solver to ensure that its solution process is deterministic. The solver hence returns the same optimal solution for a given instance.

In order to make efficient use of the computational resources available for ML, the operational, fully detailed optimal solutions $\mathbf{y}^*(\mathbf{x}^{(i)})$, $i = 1, \ldots, m$ are synthesized as $\bar{\mathbf{y}}^*(\mathbf{x}^{(i)}) \equiv g(\mathbf{y}^*(\mathbf{x}^{(i)}))$,

$i = 1, \ldots, m$ according to a tactical description $g(.)$ whose level of detail accommodates without exceeding that required in the intended application. According to their level of detail, such descriptions vary in complexity. They may be highly structured and may feature a variable size. The complexity of the input vector $\mathbf{x}$ and the synthesized output vector $\bar{\mathbf{y}}^*(\mathbf{x})$, as well as the explicit constraints that may tie their elements impact the selection and performance of ML models and algorithms.

3.2 ML approximation

For clarity of exposition, we find it useful to disentangle at first the approximation of the tactical solutions with ML from the treatment of the stochasticity in some of its inputs with ML. We therefore suppose for the moment that the input vector available at the time of prediction is equal to the full vector of input features, that is $\mathbf{x} = \mathbf{x}_\mathrm{a}$ and $\mathbf{x}_\mathrm{u}$ is empty. Under this assumption, the aim of the ML approximation is simply to find the best possible prediction $\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta})$ of $\bar{\mathbf{y}}^*(\mathbf{x})$ where 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 $(\mathbf{x}, \bar{\mathbf{y}}^*(\mathbf{x}))$ 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 $\mathbf{x}$ and $\bar{\mathbf{y}}$ and must also uphold the constraints that may explicitly relate their individual elements. The choice of a model and an algorithm necessarily depends on the exact application at hand and there is in ML a range of classification and regression approximators available for these purposes. 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. Training and validation loss functions are selected.
3. Parameters of candidate models are tuned through minimization of average training loss, i.e. empirical risk, over training set.
4. Performances of trained candidate models are assessed and compared based on average validation loss, i.e. generalization error, measured over the validation set.
5. Additional data is generated and processed if the performance on the validation set is unsatisfactory.
6. The model achieving the lowest generalization error over the validation set is retained.
7. The model performance is finally evaluated based on the average validation loss, measured over the test set.
8. Provided the selected model demonstrates sufficient accuracy, it is used as a high speed, low marginal cost, on-demand predictor for the operational 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, whenever $\mathbf{x} \neq \mathbf{x}_\mathrm{a}$. Those issues are addressed in the next section. However, the ML algorithm above remains essentially unchanged since the treatment of the stochasticity in $\mathbf{x}_\mathrm{u}$ with ML hinges on the particular definition of the input-output data pairs that are passed to ML.

3.3 Aggregation and subselection

The treatment of stochasticity in $\mathbf{x}_\mathrm{u}$ with ML can proceed in a number of ways using *aggregation methods*. All ultimately translate into a particular definition for the input-output data pairs that are passed to ML and all leverage the probabilistic information embodied in the distribution of $\mathbf{x}_\mathrm{u}$ conditional on the σ-algebra generated by $\mathbf{x}_\mathrm{a}$, say $\sigma(\mathbf{x}_\mathrm{a})$, so as to "aggregate" the probability mass distributed over the support of $\mathbf{x}_\mathrm{u}$. We shall thus say that input features $\mathbf{x}_\mathrm{u}$ whose values are unavailable at the time of prediction (e.g. railcar capacities and container weights, in our application) are to be aggregated.

The more appealing aggregation methods are those involving the replacement of $\Phi_{\mathbf{x}_\mathrm{u}}\{\|.\| \mid \mathbf{x}_\mathrm{a}\}$ in (1) with a sample version or a closed-form approximation (*aggregation of outputs* for short) rather than the direct substitution of $\mathbf{x}_\mathrm{u}$ for a $\sigma(\mathbf{x}_\mathrm{a})$-measurable predictor (*aggregation over inputs*

for short). Aggregation over inputs is simple but only acceptable when the distribution of $\mathbf{x}_u$ is highly concentrated. The replacement of $\Phi_{\mathbf{x}_u}\{\|.\| \mid \mathbf{x}_a\}$ in aggregation over outputs can occur either (i) *before*, (ii) *through* or (iii) *after* ML approximation and gives rise to predictors for the tactical solution $\bar{\mathbf{y}}^*(\mathbf{x}_a, \mathbf{x}_u)$ that are also $\sigma(\mathbf{x}_a)$-measurable by construction. We focus on the treatment of stochasticity in $\mathbf{x}_u$ with aggregation *through* ML approximation in view of its simpler application, lesser computational demands and, as we show in Section 4, good empirical performance. This method proceeds implicitly, by passing the dataset $\{(\mathbf{x}_a^{(i)}, g(\mathbf{y}^*(\mathbf{x}_a^{(i)}, \mathbf{x}_u^{(i)})),\ i = 1, \ldots, m\}$ to machine learning. The prediction $\widehat{\mathbf{y}}^*(\mathbf{x}_a)$ of the tactical solution $\bar{\mathbf{y}}^*(\mathbf{x}_a, \mathbf{x}_u) \equiv g(\mathbf{y}^*(\mathbf{x}_a, \mathbf{x}_u))$ is obtained from the trained model through $\widehat{\mathbf{y}}^*(\mathbf{x}_a) \equiv f(\mathbf{x}_a; \boldsymbol{\theta})$, where $f(\cdot;\cdot)$ is the selected ML model and $\boldsymbol{\theta}$ is a vector of tuned parameters. The implementation of aggregation through ML approximation is straightforward since the required data is obtained directly from the sample of synthesized operational solutions.

In the following we motivate how the model $f(\mathbf{x}_a; \boldsymbol{\theta})$ resulting from aggregation through ML approximation can account for the stochasticity in $\mathbf{x}_u$. If a uniform law of large numbers holds so that the average validation loss converges stochastically towards the expectation of the validation loss 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 method indeed minimizes an approximation to the expected validation loss with respect to $\mathbf{x}_a$ *as well as* $\mathbf{x}_u$. In other words, ML could in this case be viewed as minimizing an approximation to the expected discrepancy between the exact tactical solution $\bar{\mathbf{y}}^*(\mathbf{x}_a, \mathbf{x}_u)$ resulting from knowledge of $[\mathbf{x}_a, \mathbf{x}_u]$ and the solution $f(\mathbf{x}_a; \boldsymbol{\theta})$ based solely on the knowledge of $\mathbf{x}_a$. Furthermore, if $\Phi_{\mathbf{x}_u}\{\|.\| \mid \mathbf{x}_a\}$ stands for the expectation of a particular loss function and if the latter agrees with the loss function applied in ML validation, then we may argue that the method of aggregation through ML produces indeed a *bona fide* approximator of $\bar{\mathbf{y}}^*(\mathbf{x}_a)$.

Selecting the scope of analysis, namely determining which variables of $\mathbf{x}$ and domains thereof to include in the operational problem, the level of detail of the tactical description and the partition $[\mathbf{x}_a, \mathbf{x}_u]$ achieves a compromise between statistical precision, accuracy and tractability. Thus, it may be acceptable to exclude low probability regions in the support of certain relevant variables or even some relevant but judged-less-critical variables altogether from consideration in order to gain statistical power and precision at the expense of some accuracy. For short, we shall call such a restriction in the scope of analysis *subselection*. For instance, in our application, it was judged useful to disregard infrequent railcar types and container lengths from the support of $\mathbf{x} = [\mathbf{x}_a, \mathbf{x}_u]$ and the scope of analysis in order to gain statistical leverage and tractability in ML.

The excluded support and/or excluded variables and their complement constitute a partition of the whole support/set of variables that gives rise to an informational σ-algebra, say $\mathcal{G}$. The probabilistic knowledge that embodies these exclusions may be represented by the joint probability distribution of $[\mathbf{x}_a, \mathbf{x}_u]$ conditional on $\mathcal{G}$. If useful, for instance for the purpose of drawing formal comparisons between results with and without exclusions, conditioning with $\mathcal{G}$ may be introduced explicitly, possibly in addition to conditioning with $\sigma(\mathbf{x}_a)$. Otherwise, the ML procedures can be blind to the exclusions. We proceed in this manner in our application.

4 Application

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 solution depends on individual characteristics of containers and railcars. 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. Crucially, railcars are characterized by the weight capacity and tare of each platform and by the specific loading capabilities 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 or in minutes, depending on the size of the problem, using a commercial solver.

Figure 1 depicts a small example of a problem instance along with four descriptions of the

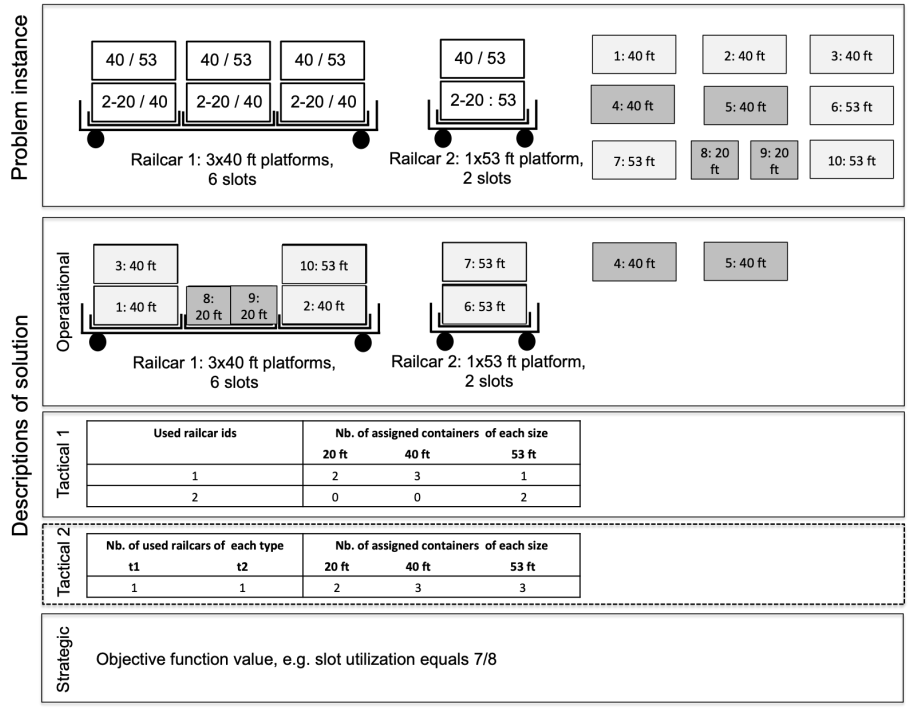

Figure 1: Example of the double-stack railcar LPP

optimal solution at different levels of detail. The instance contains ten containers of three different lengths: 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 operational solution is illustrated immediately 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 assigned. If the objective function is defined as the slot utilization, then its value is 7/8. The latter corresponds to the strategic solution, shown in the bottom part of the figure. Two alternative tactical descriptions of the solution whose levels of detail are intermediate are depicted in the figure. One (labelled Tactical 1) specifies for each individual railcar the number of containers of each size that are assigned to it. The other, less detailed (labelled Tactical 2), specifies 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 where the objective is to predict this tactical solution given a set of containers and a set of railcars without knowledge of the container weights. Since the predictions are part of an online booking system, they need to be computed at high speed.

We use our application to illustrate the notation: $\mathbf{x}$ contains the detailed information about the problem instance required to solve the LPP. Subvector $\mathbf{x}_\mathrm{a}$ reports features that are known at time of prediction: total numbers of available rail cars of each type and of available containers of each length. Subvector $\mathbf{x}_\mathrm{u}$ reports features that are unknown at time of prediction: individual gross weights of containers. The problem described in (2) corresponds to the ILP formulation of Mantovani et al. (2018). $\mathbf{y}^*(\mathbf{x}_\mathrm{a}, \mathbf{x}_\mathrm{u})$ represents the detailed assignment of containers to slots in the operational solution (top of Figure 1). The two tactical and the strategic solutions in Figure 1 constitute three examples of $\bar{\mathbf{y}}^*(\mathbf{x}_\mathrm{a}, \mathbf{x}_\mathrm{u})$ that can be obtained from the detailed solution through the synthesis $\bar{\mathbf{y}}^*(\mathbf{x}_\mathrm{a}, \mathbf{x}_\mathrm{u}) :\equiv g(\mathbf{y}^*(\mathbf{x}_\mathrm{a}, \mathbf{x}_\mathrm{u}))$.

4.1 Subselection of containers and railcars and aggregation

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 that amount to nearly 90% of the North American fleet.

As a result of the subsection of railcar types, the exact weight capacities and tares of the railcars are unknown at the time of prediction. We account for this fact through aggregation. In the case of the railcars, aggregation is straightforward because weight capacities and tares vary very little for each given standardized type. Hence, population estimates of median capacities and tares conditional on type are reasonable representative values. This amounts to perform aggregation over input values.

A key challenge in our application is related to the container weights that are unknown at the time of prediction. In contrast to the railcar weight capacities, container weights are highly variable, even conditionally upon the values of other container characteristics. We therefore perform aggregation over output values in view of its superior theoretical underpinnings compared to aggregation over input values (see Section 3.3).

4.2 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 and tested on either simpler (A, B, C) or 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 by randomly sampling and distributing the total number of platforms among railcars belonging to the 10 standardized types, by randomly sampling the number of containers of each length and by randomly sampling the weight of each container given its length. In detail, sampling the container gross weights proceeds as follows: First, we determine the empty/non-empty state of a container through a Bernouilli experiment where the probability that a container is empty conditionally upon its length has been estimated from container transportation data. Second, conditionally on the container not being empty, we sample its net weight from a uniform distribution over values ranging between 10% and 90% of its net capacity given length. Third, we equate the generated total weight of a non empty container to the sum of the generated net weight given length and the *a priori* estimate of median tare given length. Table 2 reports the number of examples for each 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 computing times per instance using three out of the six cores of an Intel Xeon X5650 Westmere 2.67 GHz processor. For instance, the median time required to solve an instance of class A down to an optimality gap of at most 5% is equal to 0.48 s.

Data class	# instances	Time percentiles (s)		
		P_5	P_{50}	P_{95}
A	20M	0.007	0.48	1.67
A	200K	0.011	0.64	2.87
B	200K	0.02	1.26	3.43
C	200K	0.72	2.59	6.03
D	100K	2.64	5.44	20.89

Table 2: Data generation

4.3 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, MLPs had demonstrated modest success in ML. However, 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 integer unit 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 number of railcars 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. This is done by computing the softmax over admissemble outputs only. Training is conducted through likelihood maximization where we treat output distributions as independent.

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 (Kingma and Ba, 2014). 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 (Bergstra and Bengio, 2012). 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. For each ML model and each dataset, we tried 50 hyperparameter combinations in a random search and retained the best performing models.

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. Algorithm 1 (HeurV) greedily accounts for the total number of slots available on each railcar but disregards all other constraints pertaining to the loading problem. In contrast, Algorithm 2 (HeurS) greedily 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

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

4.4 Results

In this section we summarize and compare the predictive performances achieved by the models with the sum of mean absolute prediction error (MAE) over the numbers of used slots (MAE_{slots}) and of mean absolute error over the numbers of loaded containers (MAE_{conts}) in output solution. They are defined as follows:

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

$$MAE_{slots} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{10}|\widehat{y}_j^{(i)} - y_j^{(i)}|s_j, \tag{4}$$

$$MAE_{conts} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=11}^{12}|\widehat{y}_j^{(i)} - y_j^{(i)}|, \tag{5}$$

where $s_j,\ j = 1, ..., 10$, equals the number of slots on railcar type j. Notice that $s_{11} = s_{12} = 1$ do not appear in (5). Clearly, the MAE provides highly useful but partial information that cannot by itself fully express the distribution of absolute errors (AE). To draw a more complete portray, we also calculated empirical quantiles of the set of absolute errors $\{AE^{(i)},\ i = 1, \ldots, n\}$, where the absolute error $AE^{(i)}$ associated with observation i is given by

$$AE^{(i)} = \sum_{j=1}^{12}|\widehat{y}_j^{(i)} - y_j^{(i)}|s_j. \tag{6}$$

Table 3 reports the MAE (3) incurred by each model over an independent test data set similar to that used for training and validation. The aggregation of the unknown container weights is performed with the method of *aggregation over output values through ML*, as discussed in Section 3.3. Standard deviations of the estimates are shown in parentheses. We report results for

models trained and validated based on 200K i.i.d. examples of class A (200K-A, second column in the table), 20M i.i.d. examples of class A (20M-A, third column in the table) and 600K i.i.d examples made up of the union of 200K examples from each of the A, B and C classes (600K-ABC, fourth column in the table). Note that each figure reported for ClassMLP or for RegMLP corresponds to the most favorable set of hyperparameters according to the validation set (found in the random search described in Section 4.3).

A number of findings emerge from examination of Table 3. For 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 20M-A 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 MAE for 200K-A and 20M-A data are only 1.304 and 0.985, respectively. The marginal value of using 100 times more observations is hence fairly small. The decrease in performance when enlarging the training-validation set from A to include the more difficult instances (from 200K-A to the union of data classes A, B and C in 600K-ABC) is quite modest for both ClassMLP and RegMLP. For example, MAE for RegMLP increases from 1.304 (200K-A) to 2.109 (600K-ABC).

The excellent predictive performance of the method is confirmed by the examination of the distribution of absolute errors in Table 4: For example, at least 95% of the absolute errors made by ClassMLP and RegMLP are smaller than or equal to 4. This is in stark contrast with the performances of the benchmark models whose distributions of absolute errors are highly skewed and whose median absolute error is either equal to 4 in the most favorable case (LogReg) or well beyond this figure elsewhere (LinReg, HeurV, HeurS).

Data	**200K-A**	**20M-A**	**600K-ABC**
# examples	200K	20M	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 3: Testing over data similar to that used in training-validation: mean absolute errors (MAE)

Data	**200K-A**							
# examples	200K							
Percentiles	P_{50}	P_{60}	P_{70}	P_{80}	P_{85}	P_{90}	P_{95}	P_{99}
ClassMLP	0	0	0	1	2	4	4	18
LogReg	4	6	7	10	12	14	17	26
RegMLP	0	0	0	1	2	4	4	18
LinReg	11	14	19	30	38	47	61	82
HeurV	10	12	16	24	30	36	46	68
HeurS	13	17	23	31	35	41	52	72

Table 4: Testing over data similar to that used in training-validation: distribution of absolute errors

Figure 2 displays the MAE in relation to the number of available slots and the number of

Data	**200K-A**		
# examples	200K		
Percentiles	P_5	P_{50}	P_{95}
ClassMLP	2.6	2.9	3.2
RegMLP	0.7	0.8	1.0
HeurV	0.3	0.4	0.8
HeurS	0.3	0.7	1.6

Table 5: Prediction time per instance (milliseconds)

available containers (input) for the RegMLP model and 20M-A data. It shows that errors occur mainly in conditions of excess supply or excess demand.

Table 5 provides information on the distribution of the GPU 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 when input belongs to class A is 0.8 milliseconds. As clearly shown 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 computing 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 operational 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 tactical solution with model RegMLP is expected to require a time close to the indicated median of 0.8 milliseconds.

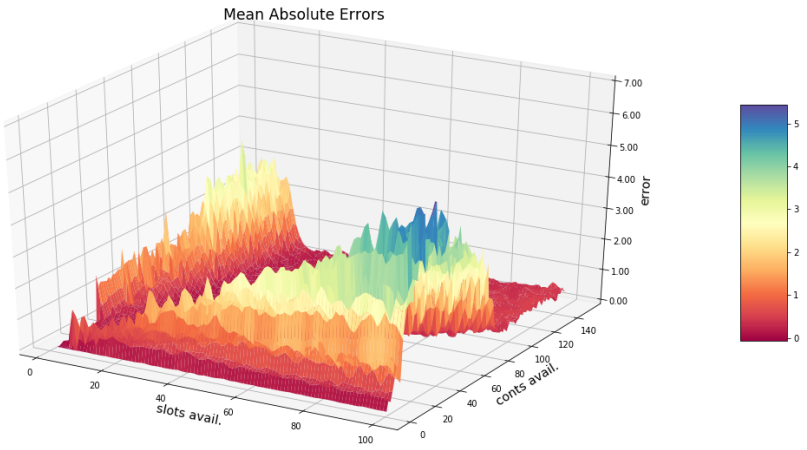

Figure 2: MAE over instances with specified numbers of available slots and containers, RegMLP model and 20M-A data

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 the previous results where testing was conducted on data similar to that used for training and validation, we now focus on 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. Standard

deviations of the estimates are shown in parentheses. 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 the 200K-A training-validation data set (NA in the first column of the table). In this respect, the regression models have a clear advantage.

The following findings emerge: The performance of the models whose testing results are reported in Table 3 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 decreases from 4.412 to 2.372.

The MAE values 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 20M-A and testing over 200K-D. 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 maximum optimality gap to a more lenient value. Training and validation could then be performed on these instances as well.

Table 7 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 markedly different.

Training-validation data # **examples**	**20M-A** 20M	**600K-ABC** 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 6: Testing on class D instances (not used for training-validation): mean absolute errors (MAE)

Data # **examples**	**20M-D** 10M		
Percentiles	P_5	P_{50}	P_{95}
RegMLP	0.6	1.3	1.9
HeurV	0.3	1.0	1.6
HeurS	1.9	3.9	4.1

Table 7: Prediction time in milliseconds per instance with extraneous data

4.5 An alternative solution from stochastic programming

To illustrate the parallel between the proposed methodology based on ML and the alternatives offered by approximate stochastic programming for the solution of (1) and (2), we detail the application of one such alternative in the context of the LPP. We select the method of sample average approximation (SAA) (Shapiro et al., 2009, p. 155) and Kim et al. (2015) in view of its position as a *de facto* standard in approximate stochastic programming: it is broadly applicable, commonly used, based on simple principles and supported by an abundant knowledge of its properties, both theoretical and empirical. We stress that an application of approximate stochastic programming to (1) and (2) seeks a solution for each particular value of $\mathbf{x}_\mathrm{a}$. Hence, unless the domain of $\mathbf{x}_\mathrm{a}$ is both finite and small, the solutions arising from approximate stochastic programming must be computed on demand, one at a time. In contrast, our proposed methodology outputs a prediction function defined over the domain of $\mathbf{x}_\mathrm{a}$. This function is computed in advance with ML and may later on yield solutions on demand.

The general optimal prediction problem of (1) and (2) is specialized to the LPP as follows:

$$\bar{\mathbf{y}}^*(\mathbf{x}_\mathrm{a}) :\equiv \arg \inf_{\bar{\mathbf{y}}(\mathbf{x}_\mathrm{a})\in\bar{\mathcal{Y}}(\mathbf{x}_\mathrm{a})} E_{\mathbf{x}_\mathrm{u}}\{\sum_{j=1}^{12}|\bar{\mathbf{y}}_j(\mathbf{x}_\mathrm{a}) - g_j(\mathbf{y}^*(\mathbf{x}_\mathrm{a},\mathbf{x}_\mathrm{u}))|s_j \mid \mathbf{x}_\mathrm{a}\} \tag{7}$$

$$\mathbf{y}^*(\mathbf{x}_\mathrm{a},\mathbf{x}_\mathrm{u}) :\equiv \arg \inf_{\mathbf{y}\in\mathcal{Y}(\mathbf{x}_\mathrm{a},\mathbf{x}_\mathrm{u})} C(\mathbf{x}_\mathrm{a},\mathbf{x}_\mathrm{u},\mathbf{y}) \tag{8}$$

The application of the SAA method proceeds through the following steps:

1. Fix a particular value for $\mathbf{x}_\mathrm{a}$.
2. Select a sampling distribution for $\mathbf{x}_\mathrm{u}$ and generate a sample of size N $\{\mathbf{x}_\mathrm{u}^{(i)},\ i=1,\ldots,N\}$.
3. Define sample versions for (7) and (8) as

$$\bar{\mathbf{y}}^*(\mathbf{x}_\mathrm{a}) :\equiv \arg \inf_{\bar{\mathbf{y}}(\mathbf{x}_\mathrm{a})\in\bar{\mathcal{Y}}(\mathbf{x}_\mathrm{a})} \sum_{i=1}^{N}\sum_{j=1}^{12}|\bar{\mathbf{y}}_j(\mathbf{x}_\mathrm{a}) - g_j(\mathbf{y}^{*(i)}(\mathbf{x}_\mathrm{a},\mathbf{x}_\mathrm{u}^{(i)}))|s_j \tag{9}$$

$$\mathbf{y}^{*(i)}(\mathbf{x}_\mathrm{a},\mathbf{x}_\mathrm{u}^{(i)}) :\equiv \arg \inf_{\mathbf{y}^{(i)}\in\mathcal{Y}(\mathbf{x}_\mathrm{a},\mathbf{x}_\mathrm{u}^{(i)})} C(\mathbf{x}_\mathrm{a},\mathbf{x}_\mathrm{u}^{(i)},\mathbf{y}^{(i)}),\ i=1,\ldots,N \tag{10}$$

4. Solve the N operational problems in (10).
5. Solve the ILP (9).

The fact that the set of solutions to (10) be finite for a given value of $\mathbf{x}_\mathrm{a}$ allows important simplifications in the analysis of the statistical properties of the SAA method. Hence, the distance between the SAA solution(s) to (9) and the solution(s) to (7) is shown to converge strongly to zero with respect to N (Kleywegt et al., 2002). Although some theoretical bounds are available, determining with sufficient precision the speed of this convergence and an appropriate value for N is essentially a problem- and data-specific issue that must be resolved empirically. Despite the lack of precise knowledge about the appropriate value for N, we can reach definite conclusions about the relative computational performance of the SAA method and our methodology: The application of the SAA method to LPP requires for each value of $\mathbf{x}_\mathrm{a}$ the solution of N operational (deterministic) load planning problems and the solution of an ILP program with $24N+12$ integral variables, $24N+12$ non-negativity constraints and $12N$ equality constraints. The time required for performing these calculations on demand for each value of $\mathbf{x}_\mathrm{a}$ exceeds by two orders of magnitude the requirement that the solutions must be available on demand in a small fraction of the time required to solve a single instance of the operational problem.

To illustrate this assessment, we conducted an experiment through the following steps:

1. Generate a dataset of class A from the sampling distributions described in Section 4.2 through a two-stage sampling process. In the first stage, randomly sample 100K values for $\mathbf{x}_\mathrm{a}$. In the second stage, for each one of the first stage values, randomly sample 100 values for $\mathbf{x}_\mathrm{u}$.

2. For each one of the 100K values of $\mathbf{x}_a$ resulting from first stage, compute the 100 deterministic load planning solutions corresponding to this value of $\mathbf{x}_a$ and to each one of the 100 values of $\mathbf{x}_u$, respectively. Compute the SAA solution with $N = 99$ from 99 out of the 100 load planning solutions, as in (9) and (10). Compute the absolute error between the SAA solution and the remaining deterministic load planning solution as well as the total time required to compute the SAA solution.

3. Compute the empirical distributions of the absolute errors and on-demand computing times incurred over the 100K repetitions resulting from the 100K values of $\mathbf{x}_a$ sampled in first stage. Compare with the empirical distributions associated with ClassMLP and RegMLP.

With $N = 99$, we find that the average absolute error incurred by SAA is equal to 0.82 (with a standard error of estimate of 0.0087) and comparable to that achieved by ClassMLP (0.965) and RegMLP (0.985). As expected, the average on-demand computing time is prohibitively higher, equal to 77,820 milliseconds (with a standard error of estimate of 258), in comparison with ClassMLP (2.9) and RegMLP (0.8). Tables 8 and 9 provide further evidence on the comparison.

Data	**100K-A**							
# examples	100K							
Percentiles	P_{50}	P_{60}	P_{70}	P_{80}	P_{85}	P_{90}	P_{95}	P_{99}
SAA	0	0	0	1	2	4	4	17

Table 8: SAA: distribution of absolute errors

Data	**100K-A**		
# examples	100K		
Percentiles	P_5	P_{50}	P_{95}
SAA	981	58,500	232,300

Table 9: SAA: computation time (milliseconds)

4.6 Feasibility of predicted tactical solutions at the operational level

The results in Section 4.4 showed that the predictions of the tactical solutions have a high accuracy. In this section we remove the focus from prediction accuracy to auxiliary results designed to numerically assess if there exists a feasible operational solution for a given predicted tactical one.

We designed an experiment whose setting is similar to the one conducted in the previous section. Namely, we predict tactical solutions for 100K randomly sampled values of the first stage variables $\mathbf{x}_a$. For each value of $\mathbf{x}_a$, we consider a random sample of 100 second-stage variables $\mathbf{x}_u$. We solve the 100 detailed instances of the LPP where the predicted tactical solution is imposed. We report in Table 10 the sample ratio of predicted solution descriptions that are feasible to the detailed problem.

	Sample ratio feasible	**Std err sample ratio (gaussian approx of binomial)**
ClassMLP	0.975	0.00035
LogReg	0.614	0.00109
RegMLP	0.966	0.00041
LinReg	0.742	0.00098
HeurV	0.324	0.0011
HeurS	0.400	0.0011

Table 10: Sample ratio of predictions that lead to feasible solutions to the detailed problem

The results show that, although the LPP can have many feasible / optimal solutions, the task of constructing one is not trivial because the low capacity models have way worse results

(satisfy constraints in 61.4% and 74.2% of the instances) than the high capacity ones that satisfy the constraints in 97.5% and 96.6% of the instances. The deterministic heuristics have very poor performance, which aligns with the fact that they are not trained on the data and hence are entirely blind to the weights and the associated constraints. The higher capacity models also have smaller standard errors than the lower capacity ones.

We end the section by stressing that these results are not relevant to analyze the accuracy of the predictor but rather to assess if the ML algorithm can learn the constraints implicitly by seeing many deterministic examples, which indeed seems to be the case. This assessment is not the aim of our methodology, otherwise, a chance constraint formulation would have been more appropriate than the optimal prediction one we propose. Nevertheless, we still believe that assessing the feasibility of the operational solutions conditional on predicted tactical ones is of interest because of the expressiveness one can expect from a ML approximator for discrete optimization problems.

5 Conclusion and future work

In this paper, we proposed a supervised learning approach for predicting tactical solutions to operational planning problems under imperfect information in short computing time. The problem is of relevance to various applications where tactical and operational planning problems are interrelated. We considered an application related to demand management and capacity planning at the tactical level (accept / reject booking requests) whose solution depends on a kind of packing problem at the operational level. A similar problem occurs in other freight transportation settings, for example, airline cargo and less than truckload (LTL).

We formulated the problem as a two-stage optimal prediction stochastic programming problem whose solutions we aimed to predict with machine learning. Key in the proposed methodology is the generation of labeled training data for supervised learning. We proposed to sample operational problem instances (perfect information) by controlled probabilistic sampling. The generated operational problem instances are solved independently and offline using an existing solver. We computed labels based on the solutions employing appropriate aggregation and subselection methods. Otherwise, our methodology relies on existing ML models and algorithms. This is a key advantage since we can benefit from the recent advances in machine learning, and in our case, deep learning.

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 prediction. The results showed 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 also showed 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, as expected, the variations over the hyperparameter sets considered during the validation step were large when the nature of the data was very dissimilar. Finally, we compared the machine learning predictions with solutions computed with a sample average approximation of the two-stage stochastic program. While the average absolute error was only slightly better, the computing time was prohibitively large.

In this work, we considered an input and output structure that was quite 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 the 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 promising avenue. Finally, 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 Jean-François Cordeau, Matteo Fischetti and Michael Hewitt.

References

Andradóttir, S. A review of random search methods. In Fu, M. C., editor, *Handbook of Simulation Optimization*, volume 216 of *International Series in Operations Research & Management Science*, chapter 10, pages 277–292. Springer, 2015.

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

Belotti, P., Bonami, P., Fischetti, M., Lodi, A., Monaci, M., Nogales-Gómez, A., and Salvagnin, D. On handling indicator constraints in mixed integer programming. *Computational Optimization and Applications*, 65:545–566, 2016.

Bengio, Y., Lodi, A., and Prouvost, A. Machine Learning for Combinatorial Optimization: A Methodological Tour d'Horizon. *ArXiv e-prints arXiv:1811.06128*, 2018.

Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

Bertsimas, D. and Shioda, R. Classification and regression via integer optimization. *Operations Research*, 55:252–271, 2017.

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.

Chau, M. and Fu, M. C. An overview of stochastic approximation. In Fu, M. C., editor, *Handbook of Simulation Optimization*, volume 216 of *International Series in Operations Research & Management Science*, chapter 6, pages 149–178. Springer, 2015.

Curtis, F. E. and Scheinberg, K. Optimization Methods for Supervised Machine Learning: From Linear Models to Deep Learning. *ArXiv e-prints arXiv:1706.10207*, 2017.

de Mello, T. H. and Bayraksan, G. Monte carlo sampling-based methods for stochastic optimization. *Surveys in Operations Research and Management Science*, 19(1):56 – 85, 2014.

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.

Fu, M. C. *Handbook of Simulation Optimization*, volume 216 of *International Series in Operations Research & Management Science*. Springer, 2015.

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

Günlük, O., Kalagnanam, J., Menickelly, M., and Scheinberg, K. Optimal Generalized Decision Trees via Integer Programming. *ArXiv e-prints arXiv:1612.03225*, 2017.

Hu, J. Model-based stochastic search methods. In Fu, M. C., editor, *Handbook of Simulation Optimization*, volume 216 of *International Series in Operations Research & Management Science*, chapter 12, pages 319–340. Springer, 2015.

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

Kim, S., Pasupathy, R., and Henderson, S. G. A guide to sample average approximation. In Fu, M. C., editor, *Handbook of Simulation Optimization*, volume 216 of *International Series in Operations Research & Management Science*, chapter 8, pages 207–243. Springer, 2015.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

Kleijnen, J. P. Response surface methodology. In Fu, M. C., editor, *Handbook of Simulation Optimization*, volume 216 of *International Series in Operations Research & Management Science*, chapter 4, pages 81–104. Springer, 2015.

Kleywegt, A. J., Shapiro, A., and Homem-de Mello, T. The sample average approximation method for stochastic discrete optimization. *SIAM J. on Optimization*, 12(2):479–502, February 2002.

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

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.

Nair, V., Dvijotham, K., Dunning, I., and Vinyals, O. Learning fast optimizers for contextual stochastic integer programs. In *Conference on Uncertainty in Artificial Intelligence*, 2018.

Shapiro, A. Monte carlo sampling methods. In Ruszczynski, A. and Shapiro, A., editors, *Handbooks in Operations Research and Management Science*, volume 10, chapter 6, pages 353–425. Elsevier, 2003.

Shapiro, A., Dentcheva, D., and Ruszczynski, A. *Lectures on Stochastic Programming: Modeling and Theory, Second Edition*. Society for Industrial and Applied Mathematics, 2009.

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.

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

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

Zabinsky, Z. B. Stochastic adaptive search methods: Theory and implementation. In Fu, M. C., editor, *Handbook of Simulation Optimization*, volume 216 of *International Series in Operations Research & Management Science*, chapter 11, pages 293–318. Springer, 2015.