Two-Stage Predict+Optimize for Mixed Integer Linear Programs with Unknown Parameters in Constraints

Xinyi Hu[1], **Jasper C.H. Lee**[2], **Jimmy H.M. Lee**[1]
[1]Department of Computer Science and Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
[2]Department of Computer Sciences & Institute for Foundations of Data Science
University of Wisconsin–Madison, WI, USA
{xyhu,jlee}@cse.cuhk.edu.hk, jasper.lee@wisc.edu

Abstract

Consider the setting of constrained optimization, with some parameters unknown at solving time and requiring prediction from relevant features. Predict+Optimize is a recent framework for end-to-end training supervised learning models for such predictions, incorporating information about the optimization problem in the training process in order to yield better predictions in terms of the quality of the predicted solution under the true parameters. Almost all prior works have focused on the special case where the unknowns appear only in the optimization objective and not the constraints. Hu et al. proposed the first adaptation of Predict+Optimize to handle unknowns appearing in constraints, but the framework has somewhat ad-hoc elements, and they provided a training algorithm only for covering and packing linear programs. In this work, we give a new *simpler* and *more powerful* framework called *Two-Stage Predict+Optimize*, which we believe should be the canonical framework for the Predict+Optimize setting. We also give a training algorithm usable for all mixed integer linear programs, vastly generalizing the applicability of the framework. Experimental results demonstrate the superior prediction performance of our training framework over all classical and state-of-the-art methods.

1 Introduction

Optimization problems are prevalent in modern society, and yet the problem parameters are not always available at the time of solving. For example, consider the real-world application scenario of stocking a store: as store managers, we need to place monthly orders for products to stock in the store. We want to stock products that sell fast and yield high profits, as much of them as possible, subject to the hard constraint of limited storage space. However, orders need to be placed two weeks in advance of the monthly delivery, and the customer demand next month cannot be known exactly at the time of order placement. In this paper, we consider the supervised learning setting, where the unknown parameters can be predicted from relevant features, and there are sufficient historical (features, parameters) pairs as training data for a prediction model. The goal, then, is to learn a prediction model from the training data such that, if we plug in the estimated parameters into the optimization problem and solve for an *estimated solution*, the estimated solution remains a good solution even after the true parameters are revealed.

The classic approach to the problem would be to train a simple regression model, based on standard losses such as (regularized) ℓ_2 loss, to predict parameters from the features. It is shown, however, that having a small prediction error in the parameter space does not necessarily mean that the estimated solution performs well under the true parameters. The recent framework of Predict+Optimize, by

Elmachtoub and Grigas [7], instead proposes the more effective *regret* loss for training, which compares the solution qualities of the true optimal solution and the estimated solution under the true parameters. Subsequent works [6, 8, 10, 13, 17, 19, 27] have since appeared in the literature, applying the framework to more and wider classes of optimization problems as well as focusing on speed-vs-prediction accuracy tradeoffs.

However, all these prior works focus only on the case where the unknown parameters appear in the optimization objective, and not in the constraints. The technical challenge for the generalization is immediate: if there were unknown parameters in the constraints, the estimated solution might not even be feasible under the true parameters revealed afterwards! Thus, in order to tackle the Predict+Optimize setting with unknowns in constraints, the recent work of Hu et al. [12] presents the first such adaptation on the framework: they view the estimated solution as representing a *soft commitment*. Once the true parameters are revealed, corrective action can be taken to ensure feasibility, potentially at a penalty corresponding to the real-life cost of (partially) reneging on a soft commitment. Their framework captures application scenarios whenever such correction is possible, and requires the practitioner to specify both the correction mechanism and the penalty function. These data can be determined and derived from the specific application scenario. As an example, in the product-stocking problem, an additional unknown parameter is the storage space, because it depends on how the current products in the store sell before the new order arrives. We need to place orders two weeks ahead based on predicted storage space. The night before the order arrives, we know the precise available space, meaning that the unknown parameter is revealed. A possible correction mechanism then is to throw away excess products that the store cannot keep, while incurring the penalty that is the retail price of the products, as well as disposal fees.

While the Hu et al. [12] framework does capture many application scenarios, there are important shortcomings. In their framework, they require the practitioner to specify a correction function that amends an infeasible solution into a feasible solution. However, the derivation of a correction function can be rather ad-hoc in nature. In particular, given an infeasible estimated solution, there may be many ways to transform the solution into a feasible one, and yet their framework requires the practitioner to pick one particular way. This leads to the second downside: it is difficult to give a *general* algorithmic framework that applies to a wide variety of optimization problems. Hu et al. had to restrict their attention only to packing and covering linear programs, for which they could propose a generic correction function. In this work, we aim to *vastly generalize* the kinds of optimization problems that Predict+Optimize can tackle under uncertainty in the constraints. In addition, the approach of Hu et al. fails to handle the interesting situation in which post-hoc correction is still desirable when the estimated solution is feasible but not good under the true parameters.

Our contributions are three-fold:

• To mitigate the shortcomings of the prior work, we propose and advocate a new framework, which we call *Two-Stage Predict+Optimize*[1], that is both *conceptually simpler* and *more expressive* in terms of the class of optimization problems it can tackle. The key idea for the new framework is that the correction function is unnecessary. All that is required is a penalty function that captures the cost of modifying one solution to another. A penalty function is sufficient for defining a correction process: we formulate the correction process itself as a “Stage 2” optimization problem, taking the originally estimated solution as well as the penalty function into account.

• Under this framework, we further propose a general end-to-end training algorithm that applies not only to packing and covering linear programs, but also to all mixed integer linear programs (MILPs). We adapt the approach of Mandi and Guns [18] to give a gradient method for training neural networks to predict parameters from features.

• We apply the proposed method to three benchmarks to demonstrate the superior empirical performance over classical and state-of-the-art training methods.

2 Background

In this section, we give basic definitions for optimization problems and the Predict+Optimize setting [7], and describe the state-of-the-art framework [12] for Predict+Optimize with unknown parameters

[1]The literature sometimes uses “two-stage" to mean approaches where the prediction is agnostic to the optimization problem. Here, “two-stage" refers to the soft commitment and the correction.

in constraints. The theory is stated in terms of minimization but applies of course also to maximization, upon appropriate negation. Without loss of generality, an *optimization problem* (OP) P can be defined as finding:

$$x^* = \arg\min_x obj(x) \quad \text{s.t. } C(x)$$

where $x \in \mathbb{R}^d$ is a vector of decision variables, $obj : \mathbb{R}^d \to \mathbb{R}$ is a function mapping x to a real objective value that is to be minimized, and C is a set of constraints that must be satisfied over x. We call x^* an *optimal solution* and $obj(x^*)$ the *optimal value*. A *parameterized optimization problem (Para-OP)* $P(\theta)$ is an extension of an OP P:

$$x^*(\theta) = \arg\min_x obj(x, \theta) \quad \text{s.t. } C(x, \theta)$$

where $\theta \in \mathbb{R}^t$ is a vector of parameters. The objective $obj(x, \theta)$ and constraints $C(x, \theta)$ can both depend on θ. When the parameters are known, a Para-OP is just an OP.

In the *Predict+Optimize* setting [7], the true parameters $\theta \in \mathbb{R}^t$ for a Para-OP are not known at solving time, and *estimated parameters* $\hat{\theta}$ are used instead. Suppose each parameter is estimated by m features. The estimation will rely on a machine learning model trained over n observations of a training data set $\{(A^1, \theta^1), \ldots, (A^n, \theta^n)\}$ where $A^i \in \mathbb{R}^{t \times m}$ is a *feature matrix* for θ^i, in order to yield a *prediction function* $f : \mathbb{R}^{t \times m} \to \mathbb{R}^t$ predicting parameters $\hat{\theta} = f(A)$.

Solving the Para-OP using the estimated parameters, we obtain an *estimated solution* $x^*(\hat{\theta})$. When the unknown parameters appear in constraints, one major challenge is that the feasible region is only approximated at solving time, and hence the estimated solution may be infeasible under the true parameters. Fortunately, in certain applications, the estimated solution is not a hard commitment, but only represents a soft commitment that can be modified once the true parameters are revealed. Hu et al. [12] propose a Predict+Optimize framework for such applications. The framework is as follows: i) the unknown parameters are estimated as $\hat{\theta}$, and an estimated solution $x^*(\hat{\theta})$ is solved using the estimated parameters, ii) the true parameters θ are revealed, and if $x^*(\hat{\theta})$ is infeasible under θ, it is *amended* into a *corrected solution* $x^*_{\text{corr}}(\hat{\theta}, \theta)$ while potentially incurring some *penalty*, and finally iii) the solution $x^*_{\text{corr}}(\hat{\theta}, \theta)$ is evaluated according to the sum of both the objective, under the true parameters θ, and the incurred penalty from correction.

More formally, a *correction function* takes an estimated solution $x^*(\hat{\theta})$ and true parameters θ and returns a *corrected solution* $x^*_{corr}(\hat{\theta}, \theta)$ that is feasible under θ. A *penalty function* $Pen(x^*(\hat{\theta}) \to x^*_{corr}(\hat{\theta}, \theta))$ takes an estimated solution $x^*(\hat{\theta})$ and the corrected solution $x^*_{corr}(\hat{\theta}, \theta)$ and returns a non-negative penalty. Both the correction function and the penalty function should be chosen according to the precise application scenario at hand. The final corrected solution $x^*_{corr}(\hat{\theta}, \theta)$ is evaluated using the *post-hoc regret*, which is defined with respect to the corrected solution $x^*_{corr}(\hat{\theta}, \theta)$ and the penalty function $Pen(x^*(\hat{\theta}) \to x^*_{corr}(\hat{\theta}, \theta))$. The post-hoc regret is the sum of two terms: (a) the difference in objective between the *true optimal solution* $x^*(\theta)$ and the corrected solution $x^*_{corr}(\hat{\theta}, \theta)$ under the true parameters θ, and (b) the penalty that the correction process incurs. Mathematically, the post-hoc regret function $PReg(\hat{\theta}, \theta) : \mathbb{R}^t \times \mathbb{R}^t \to \mathbb{R}_{\geq 0}$ (for minimization problems) is:

$$PReg(\hat{\theta}, \theta) = obj(x^*_{corr}(\hat{\theta}, \theta), \theta) - obj(x^*(\theta), \theta) + Pen(x^*(\hat{\theta}) \to x^*_{corr}(\hat{\theta}, \theta)) \tag{1}$$

where $obj(x^*_{corr}(\hat{\theta}, \theta), \theta)$ is the *corrected optimal value* and $obj(x^*(\theta), \theta)$ is the *true optimal value*.

Given the post-hoc regret as a loss function, the empirical risk minimization principle dictates that we choose the prediction function to be the function f from the set of models $\mathcal{F}$ attaining the smallest average post-hoc regret over the training data:

$$f^* = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} PReg(f(A^i), \theta^i) \tag{2}$$

3 Two-stage Predict+Optimize Framework

While the prior work by Hu et al. [12] is the first Predict+Optimize framework for unknowns in constraints, and is indeed applicable to a good range of applications, it has several shortcomings.

First, the framework requires mathematically formalizing both a penalty function and a correction function from the application scenario, and essentially imposes differentiability assumptions on the correction function for the framework to be usable. The ad-hoc nature of writing down a correction function limits the practical applicability of the framework. Second, as a result of needing a single (differentiable) correction function, Hu et al. [12] needed to restrict their attention to only packing and covering linear programs, in order to derive a general correction function that is applicable to all the instances. This also significantly limits the immediate applicability of their framework. Third, their framework only corrects an estimated solution when it is infeasible under the true parameters. Yet, there are applications where corrections are possible even when the estimated solution were feasible, but just not very good under the true parameters.

In this paper, we advocate using a simpler yet more powerful framework, which we call *Two-Stage Predict+Optimize*, addressing all of the above shortcomings. The simplified perspective will allow us to discuss more easily how to handle the entire class of mixed integer linear programs (MILPs) instead of being restricted to just packing and covering linear programs. Since MILPs include all optimization problems in NP (under a reasonable definition of NP for optimization problems), our framework is significantly more applicable in practice. We will describe the Two-Stage Predict+Optimize framework below, and discuss its application to MILPs in the next section.

Our framework is simple: we forgo the idea of a correction function and treat correction itself as an optimization problem, based on the penalty function, the estimated solution and the revealed true parameters. Recall the Hu et al. view of Predict+Optimize under uncertainties in constraints: the estimated solution is a form of soft commitment, which can be modified at a cost once the true parameters are revealed. The penalty function describes the cost of changing from an estimated solution to a final solution. The main observation is that, given an estimated solution and the revealed parameters, we should in fact solve a *new* optimization problem, formed by applying the true parameters to the original optimization, and adding the penalty function to the objective. The final solution from this new optimization thus takes the penalty of correction into account. This approach yields three immediate advantages. First, the practitioner no longer needs to specify a correction function, thus reducing the ad-hoc nature of the framework. Second, even feasible solutions are allowed to be modified after the true parameters are revealed if the penalty of doing so is not infinity. Third, conditioned on the same penalty function, the solution quality from our two-stage optimization approach is always at least as good as that from using any correction function. The last advantage is presented as Proposition A.1.

Now we formally define the Two-Stage Predict+Optimize framework.

I. In Stage 1, the unknown parameters are estimated as $\hat{\theta}$ from features. The practitioner then solves the *Stage 1* optimization, which is the Para-OP using the estimated parameters, to obtain the *Stage 1 solution* x_1^*. The Stage 1 solution should be interpreted as some form of soft commitment, that we get to modify in Stage 2 at extra cost/penalty. Assuming the notation of the Para-OP in Section 2, the Stage 1 OP can be formulated as:

$$x_1^* = \arg\min_x \; obj(x, \hat{\theta}) \quad \text{s.t. } C(x, \hat{\theta})$$

II. At the beginning of Stage 2, the true parameters θ are revealed. The Stage 2 optimization problem augments the original Stage 1 problem by adding a penalty term $Pen(x_1^* \to x_2^*, \theta)$ to the objective, which accounts for the penalty (modelled from the application scenario) for changing from the softly-committed Stage 1 solution x_1^* to the new *Stage 2* and *final* solution x_2^*. The Stage 2 OP can then be formulated as:

$$x_2^* = \arg\min_x \; obj(x, \theta) + Pen(x_1^* \to x, \theta) \quad \text{s.t. } C(x, \theta)$$

Solving the Stage 2 problem yields the final Stage 2 "corrected" solution x_2^*.

III. The Stage 2 solution x_2^* is evaluated according to the analogous post-hoc regret, as follows:

$$PReg(\hat{\theta}, \theta) = \; obj(x_2^*, \theta) + Pen(x_1^* \to x_2^*, \theta) - obj(x^*(\theta), \theta)$$

where again, $x^*(\theta)$ is an optimal solution of the Para-OP under the true parameters θ. Note that the post-hoc regret depends on *all* of a) the predicted parameters, b) the induced Stage 1 solution, c) the true parameters and d) the final Stage 2 solution.

To see this new framework applies in practice, the following example expands on the product-stocking problem in the introduction.

Example 1. *Consider the product-stocking problem again, where regular orders have to be placed two weeks ahead of monthly deliveries. Since the available space at the time of delivery is unknown when we place the regular orders, depending on the sales over the next two weeks, we need to make a prediction on the available space to make a corresponding order. We learn the predictor using historical sales records from features such as time-of-year and price. Then, we use the predicted available space to optimize for the regular order we place. This is the Stage 1 solution.*

The night before the order arrives, the unknown constraint parameter, i.e. the precise available space, is revealed. We can then check if we have over-ordered or under-ordered. In the case of over-ordering, we would have to call and ask the wholesale company to drop some items from the order. The company would perhaps allow taking the items off the final bill, but naturally they have a surcharge for last-minute changes. Similarly, if we under-ordered, we might request the wholesale company to send us more products, again naturally with a surcharge for last-minute ordering. The updated order is the Stage 2 decision. The incurred wholesaler surcharges induce the penalty function.

A reader who is familiar with the literature on two-stage optimization problems may note that the above framework is phrased slightly differently from some other two-stage problem formulations. In particular, some two-stage frameworks phrase Stage 1 solutions as *hard* commitments, and include *recourse variables* in both stages of optimization to model what changes are made in Stage 2. We show in Appendix A.1 how our framework can capture this other perspective, and in general discuss how problem modelling can be done in our new framework.

The reader may also wonder: what about application scenarios where the (Stage 1) estimated solution is a hard commitment, and there is absolutely no correction/recourse available? In Appendix A.2, we discuss how our framework is *still* useful and applicable for learning in these situations.

We also give a more detailed comparison, in Appendix A.3, between our new Two-Stage Predict+Optimize framework and the prior Hu et al. framework. Technically, if we *ignored* differentiability issues, the two frameworks are mathematically equivalent in expressiveness. However, we stress that our new framework is both *conceptually simpler* and *easier to apply* to a *far wider* class of optimization problems. We show concretely in the next section how to end-to-end train a neural network for this framework for all MILPs, vastly generalizing the method of Hu et al. which is restricted to packing and covering (non-integer) linear programs. In addition, Appendix A.3 also states and proves Proposition A.1, that if we fix an optimization problem, a prediction model and a penalty function, then the solution quality from our two-stage approach is always at least as good as using the correction function approach.

4 Two-Stage Predict+Optimize on MILPs

In this section, we describe how to give an end-to-end training method for neural networks to predict unknown parameters from features, under the Two-Stage Predict+Optimize framework. The following algorithmic method is applicable whenever *both* stages of optimization are expressible as MILPs. Due to the page limit, the discussion in this section is high-level and brief, with all the calculation details deferred to Appendix B.

The standard way to train a neural network is to use a gradient-based method. In the Two-Stage Predict+Optimize framework, we use the post-hoc regret $PReg$ as the loss function. Therefore, for each edge weight w_e in the neural network, we need to compute the derivative $\frac{\mathrm{d}PReg}{\mathrm{d}w_e}$. Using the law of total derivative, we get

$$\frac{\mathrm{d}PReg(\hat{\theta},\theta)}{\mathrm{d}w_e} = \left.\frac{\partial PReg(\hat{\theta},\theta)}{\partial x_2^*}\right|_{x_1^*} \frac{\partial x_2^*}{\partial x_1^*}\frac{\partial x_1^*}{\partial \hat{\theta}}\frac{\partial \hat{\theta}}{\partial w_e} + \left.\frac{\partial PReg(\hat{\theta},\theta)}{\partial x_1^*}\right|_{x_2^*} \frac{\partial x_1^*}{\partial \hat{\theta}}\frac{\partial \hat{\theta}}{\partial w_e} \tag{3}$$

As such, we wish to calculate each term on the right hand side.

The easiest term to handle is $\frac{\partial \hat{\theta}}{\partial w_e}$, since $\hat{\theta}$ is the neural network output, and so the derivatives can be directly calculated by standard backpropagation [25]. As for the terms $\left.\frac{\partial PReg(\hat{\theta},\theta)}{\partial x_2^*}\right|_{x_1^*}$ and $\left.\frac{\partial PReg(\hat{\theta},\theta)}{\partial x_1^*}\right|_{x_2^*}$, they are easily calculable whenever both the optimization objective and penalty function are smooth, and in fact linear as in the case of MILPs. What remains are the terms $\frac{\partial x_2^*}{\partial x_1^*}$ and

$\frac{\partial x_1^*}{\partial \hat{\theta}}$. The challenge is that x_2^* is the solution of a MILP optimization (Stage 2) that uses x_1^* as its parameters, i.e., differentiate through a MILP. Similarly, x_1^* depends on $\hat{\theta}$ through a MILP (Stage 1). Since MILP optima may not change under minor parameter perturbations, the gradients can be either 0 or non-existent, which are uninformative. We thus need to compute some approximation in order to get useful training signals.

Our approach, inspired by the work of Mandi and Guns [18], is to define a new surrogate loss function $\widetilde{PReg}$ that is differentiable and produces informative gradients. Prior works related to learning unknowns in constraints [1, 2, 27] give ways of differentiating through LPs or LPs with regularizations. These works can be used in place of the proposed approach. However, experiments in Appendix E demonstrate that the proposed approach performs at least as well in post-hoc regret performance as the others, while being faster. We show the construction of the proposed approach below, and note that it does not have a simple closed form. Nonetheless, we can compute its gradients.

The rest of the section assumes that both stages of optimization are expressible as a MILP in the following standard form:

$$x^* = \arg\min_{x} c^\top x \text{ s.t. } Ax = b, Gx \geq h, x \geq 0, x_S \in \mathbb{Z} \tag{4}$$

with decision variables $x \in \mathbb{R}^d$ and problem parameters $c \in \mathbb{R}^d$, $A \in \mathbb{R}^{p\times d}$, $b \in \mathbb{R}^p$, $G \in \mathbb{R}^{q\times d}$, $h \in \mathbb{R}^q$. The subset of indices S denotes the set of variables that are under integrality constraints. Since the unknown parameters may appear in any combination of c, A, b, G and h in the Stage 1 optimization for x_1^*, the surrogate loss function construction needs computable and informative gradients for all of $\frac{\partial x^*}{\partial c}$, $\frac{\partial x^*}{\partial A}$, $\frac{\partial x^*}{\partial b}$, $\frac{\partial x^*}{\partial G}$ and $\frac{\partial x^*}{\partial h}$.

We follow the interior-point based approach of Mandi and Guns [18], used also by Hu et al. [12]. Consider the following convex relaxation of (4), for a *fixed* value of $\mu \geq 0$:

$$x^* = \arg\min_{x,s} c^\top x - \mu \sum_{i=1}^{d} \ln(x_i) - \mu \sum_{i=1}^{q} \ln(s_i) \text{ s.t. } Ax = b, Gx - s = h \tag{5}$$

This is a relaxation of (4) by i) dropping all integrality constraints, ii) introducing slack variables $s \geq 0$ to turn $Gx \geq h$ into $Gx - s = h$ and iii) replacing both the $x \geq 0$ and $s \geq 0$ constraints with the logarithm barrier terms in the objective, with multiplier $\mu \geq 0$. The observation is that the gradients $\frac{\partial x}{\partial c}$, $\frac{\partial x}{\partial A}$, $\frac{\partial x}{\partial b}$, $\frac{\partial x}{\partial G}$ and $\frac{\partial x}{\partial h}$ for (5) are all well-defined, computable and informative for a *fixed* value of $\mu \geq 0$: Slater's condition holds for (5), and so the KKT conditions must be satisfied at the optimum (x^*, s^*) of (5). We can thus compute all the relevant gradients via differentiating the KKT conditions, using the implicit function theorem. We give all the calculation details in Appendix B.

Given the above observation, we then aim to construct the surrogate loss function by replacing the x_1^* and x_2^*, which are supposed to solved using MILP (4), with a) $\widetilde{x}_1$ that is solved from program (5) relaxation of the Stage 1 optimization problem, using the predicted parameters $\hat{\theta}$ and b) $\widetilde{x}_2$ that is solved from the program (5) relaxed version of Stage 2 optimization, using $\widetilde{x}_1$ and the true parameters θ. The only remaining question then, is, which values of μ do we use for the two relaxed problems?

Given a MILP in the form of (4), the interior-point based solver of Mandi and Guns [18] generates and solves (5) for a sequence of decreasing non-negative μ, with a termination condition that μ cannot be smaller than some cutoff value. Thus, we simply choose the cutoff value to use as "μ" in (5), which then completes the definition of the surrogate loss $\widetilde{PReg}$.

Algorithmically, we train the neural network on the surrogate loss $\widetilde{PReg}$ as follows: given predicted parameters, we run the Mandi and Guns solver to get the optimal solution (x^*, s^*) for the final value of μ. We can then compute the gradient of the output solution with respect to any of the problem parameters using the calculations in Appendix B, combined with backpropagation, to yield $\frac{\mathrm{d}\widetilde{PReg}}{\mathrm{d}w_e}$ according to Equation (3).

In Appendix C, we give three example application scenarios, along with their penalty functions, that our training approach can handle. These problems are: a) an alloy production problem, for factory trying to source ores under uncertainty in chemical compositions in the raw materials, b) a variant of the classic 0-1 knapsack with unknown weights and rewards, and c) a nurse roster scheduling problem with unknown patient load. We show explicitly in Appendix C how both stages of optimization

Table 1: Relevant problem sizes of the three benchmarks.

Problem name	Brass alloy production	Titanium-alloy production	0-1 knapsack	Nurse scheduling problem
Dimension of x	10	10	10	315
Number of constraints	12	14	21	846
Number of unknown parameters	20	40	10	21
Number of features (per parameter)	4096	4096	4096	8

can be formulated as MILPs for these applications, and apply the Appendix B calculations to yield gradient computation formulas for the surrogate loss $\widetilde{PReg}$ for these problems.

A limitation of our approach is the requirement that both stages must be expressible as MILPs, constraining the optimization objectives to be linear in the MILP decision variables. This contrasts the Hu et al. framework [12] which handles non-linear penalties. We point out that even MILPs can handle some non-linearity by using extra decision variables: for example, the absolute-value function. Moreover, the Appendix B gradient calculations can be adapted to handle general differentiable non-linear objectives. We present only MILPs as a main overarching application for this paper because of their widespread use in discrete optimization, with readily available solvers.

5 Experimental Evaluation

We evaluate the proposed method[2] on three benchmarks described in Section 4 and Appendix C. Table 1 reports the relevant benchmark problem sizes. We compare our method (2S) with the state of the art Predict+Optimize method, IntOpt-C [12], and 5 classical regression methods [9]: ridge regression (Ridge), k-nearest neighbors (k-NN), classification and regression tree (CART), random forest (RF), and neural network (NN). All of these methods use their classic loss function to train the prediction models. At test time, to ensure the feasibility of the solutions when computing the post-hoc regret, we perform Stage 2 optimization on the estimated solutions for these classical regression methods before evaluating the final solution. Additionally, CombOptNet [23] is a different method focusing on learning unknowns in constraints, but with a different goal and loss function. We experimentally compare our proposed method with CombOptNet on the 0-1 knapsack benchmark—the only with available CombOptNet code. We also present a qualitative comparison in Section 6.

In the following experiments, we will need to take care to distinguish two-stage optimization as a training technique (Section 4) and as an evaluation framework (Section 3). We will denote our training method as "2S" in the experiments, and when we say "Two-Stage Predict+Optimize" framework, we always mean it as an evaluation framework. 2S is always evaluated according to the Two-Stage Predict+Optimize framework. As explained above, we will also evaluate all the classical training methods using the Two-Stage Predict+Optimize framework. For our comparison with the prior work of Hu et al. [12], we will also distinguish their training method and evaluation framework. The name "IntOpt-C" always refers to their training method using their correction function. We will simply call their evaluation framework the "Hu et al. framework" or with similar phrasing (see Section 2 to recall details). IntOpt-C will sometimes be evaluated using our new Two-Stage Predict+Optimize framework, and sometimes the prior framework of Hu et al. [12] using their correction function.

The methods of k-NN, RF, NN, and IntOpt-C as well as 2S have hyperparameters, which we tune via cross-validation. We include the hyperparameter types and chosen values in Appendix D. In the main paper we only report the prediction performances. See Appendix H for runtime comparisons.

Alloy Production Problem The alloy production problem is a covering LP, see Appendix C.1 for the practical motivation and LP model. Since Hu et al. [12] also experimented on this problem, we use it to compare our 2S method with IntOpt-C [12], using the same dataset and experimental setting.

We conduct experiments on the production of two real alloys: brass and an alloy blend for strengthening Titanium. For brass, 2 kinds of metal materials, Cu and Zn, are required [14]. The blend of the two materials are, proportionally, $req = [627.54, 369.72]$. For the titanium-strengthening alloy, 4 kinds of metal materials, C, Al, V, and Fe, are required [15]. The blend of the four materials are proportional to $req = [0.8, 60, 40, 2.5]$. We use the same real data as that used in IntOpt-C [12] as numerical values in our experiment instances. In this dataset [23], each unknown metal concentration

[2]Our implementation is available at https://github.com/Elizabethxyhu/NeurIPS_Two_Stage_Predict-Optimize

Table 2: Comparison of the Two-Stage Predict+Optimize framework and the Hu et al. framework on the alloy production problem.

PReg		Two-Stage Predict	Hu et al.
Alloy	Penalty factor	+Optimize Framework	Framework
Brass	0.25±0.015	**43.87±2.73**	68.16±6.26
	0.5±0.015	**65.71±4.81**	82.91±5.45
	1±0.015	**88.75±5.91**	107.64±6.85
	2±0.015	**123.90±6.84**	150.47±12.99
	4±0.015	**161.86±8.49**	178.69±10.09
	8±0.015	**194.06±13.09**	206.84±12.51
Titanium-alloy	0.25±0.015	**4.52±0.47**	6.45±0.81
	0.5±0.015	**6.03±0.62**	7.90±0.56
	1±0.015	**8.58±0.74**	10.73±0.81
	2±0.015	**12.17±1.24**	14.17±1.31
	4±0.015	**16.10±1.06**	17.48±0.99
	8±0.015	**19.69±0.91**	21.08±1.91

Table 3: Mean post-hoc regrets and standard deviations for the alloy production problem using the Two-Stage Predict+Optimize framework.

PReg		2S	IntOpt-C	Ridge	k-NN	CART	RF	NN	TOV
Alloy	Penalty factor								
Brass	0.25±0.015	**43.87±2.73**	45.27±3.35	60.80±2.55	63.32±4.39	77.80±6.37	60.85±2.35	64.96±3.58	312.02±6.94
	0.5±0.015	**65.71±4.81**	67.69±4.25	71.12±3.48	74.36±5.69	93.67±7.03	70.86±3.29	74.32±2.90	
	1±0.015	**88.75±5.91**	89.83±4.79	91.82±6.41	96.52±8.90	125.50±9.49	90.97±6.14	93.12±4.24	
	2±0.015	**123.90±6.84**	125.46±9.26	133.18±12.98	140.77±16.02	189.12±16.10	131.12±12.48	130.67±10.52	
	4±0.015	**161.86±8.49**	164.94±10.33	215.87±26.54	229.22±30.74	316.31±30.95	211.40±25.56	205.76±24.33	
	8±0.015	**194.06±13.09**	200.42±8.51	381.30±53.75	406.19±60.42	570.75±61.42	372.01±51.82	355.96±52.25	
Titanium-alloy	0.25±0.015	**4.52±0.47**	4.72±0.58	6.43±0.39	6.13±0.34	7.07±0.45	5.75±0.48	6.56±0.59	30.27±0.54
	0.5±0.015	**6.03±0.62**	6.23±0.64	7.71±0.45	7.27±0.39	8.57±0.45	6.76±0.55	7.38±0.67	
	1±0.015	**8.58±0.74**	8.71±0.95	10.26±0.62	9.55±0.52	11.57±0.52	8.76±0.72	9.03±0.84	
	2±0.015	**12.17±1.24**	12.31±1.31	15.37±1.03	14.11±0.84	17.57±0.80	12.78±1.11	12.34±1.21	
	4±0.015	**16.10±1.06**	16.97±1.70	25.60±1.89	23.24±1.56	29.57±1.53	20.81±1.93	18.95±2.00	
	8±0.015	**19.69±0.91**	20.80±1.74	46.04±3.65	41.49±3.03	53.57±3.10	36.88±3.63	32.16±3.60	

is related to 4096 features. For experiments on both alloys, 350 instances are used for training and 150 instances for testing the model performance. For NN, IntOpt-C, and 2S, we use a 5-layer fully connected network with 512 neurons per hidden layer.

In the penalty function described in Appendix C.1, we need to choose a penalty factor/multiplier for each supplier. We conduct experiments on 6 types of penalty factor (σ) settings: 6 vectors where each entry is i.i.d. uniformly sampled from $[0.25 \pm 0.015]$, $[0.5 \pm 0.015]$, $[1.0 \pm 0.015]$, $[2.0 \pm 0.015]$, $[4.0 \pm 0.015]$, and $[8.0 \pm 0.015]$ respectively. This random sampling of σ ensures that the penalty factor for each supplier is different, but remains roughly on the same scale.

The first experiment we run compares 2S+Two-Stage Predict+Optimize framework with IntOpt-C+Hu et al. framework. Specifically, we compare a) using 2S for training and evaluating using the Two-Stage Predict+Optimize framework in Section 3, versus b) using IntOpt-C for training and evaluating using the same correction function from training, according to the Hu et al. framework described in Section 2. Table 2 compares the mean post-hoc regret and standard deviations for the alloy production problem for the two different frameworks. The table shows that Two-Stage Predict+Optimize framework always achieves smaller mean post-hoc regret than the Hu et al. framework. Compared with the Hu et al. framework, our framework obtains 6.18%-35.63% smaller mean post-hoc regret in brass production, and 6.59%-29.89% smaller mean post-hoc regret in titanium-alloy production.

We present a further comparison in Appendix F with a variant of the Hu et al. framework—the ℓ_2 projection idea in [3], which performs even worse than the Hu et al. framework.

The second experiment compares various training approaches *all* evaluated under the Two-Stage Predict+Optimize framework. That is, the models are trained differently, but at test time, we always use Stage 2 optimization to give a final solution and evaluate post-hoc regret on it. Table 3 reports the mean post-hoc regrets and standard deviations across 10 runs for each training method on the alloy production problem. The table shows that our method, 2S, achieves the best performance, compared with IntOpt-C achieving the second best performance, beating all the classical training approaches. Compared with IntOpt-C, 2S obtains 1.20%-3.18% smaller mean post-hoc regrets in brass production, and 1.18%-5.33% smaller mean post-hoc regret in titanium-alloy production. Compared with the classical approaches, the improvements are much more significant. 2S obtains at least 2.44%-45.48% smaller mean post-hoc regrets in brass production, and at least 1.39%-38.78% smaller mean post-hoc regret in titanium-alloy production. The average True Optimal Values (TOV) are reported in the last column of Table 3 for reference, although the reader should take care to *not* over-interpret the ratio

Table 4: Mean post-hoc regrets and standard deviations for 0-1 knapsack problem using the Two-Stage Predict+Optimize framework.

PReg	Penalty factor	2S	CombOptNet	Ridge	k-NN	CART	RF	NN	TOV
100	0.21	**1.26±0.01**	9.45±0.19	9.46±0.19	9.38±0.21	8.67±0.13	9.50±0.26	9.81±0.20	29.68±0.14
	0.25	**6.28±0.05**	9.60±0.22	9.77±0.19	9.70±0.19	9.19±0.12	9.82±0.27	10.11±0.20	
	0.3	**9.22±0.10**	10.45±0.34	10.16±0.19	10.10±0.18	9.85±0.11	10.22±0.28	10.49±0.21	
150	0.21	**0.73±0.01**	8.90±8.97	9.12±0.22	8.91±0.20	8.46±0.18	9.20±0.27	9.66±0.47	40.23±0.19
	0.25	**3.64±0.04**	9.11±9.41	9.40±0.21	9.19±0.20	8.88±0.17	9.47±0.26	9.92±0.43	
	0.3	**7.27±0.06**	9.34±9.38	9.76±0.22	9.53±0.19	9.41±0.17	9.81±0.24	10.23±0.38	
200	0.21	**0.33±0.01**	15.16±0.21	6.57±0.21	6.38±0.29	6.26±0.21	6.59±0.23	7.08±0.95	48.13±0.24
	0.25	**1.67±0.03**	15.20±0.27	6.80±0.20	6.62±0.29	6.57±0.19	6.82±0.21	7.27±0.88	
	0.3	**3.33±0.06**	15.25±0.22	7.09±0.19	6.91±0.28	6.95±0.19	7.10±0.18	7.52±0.80	
250	0.21	**0.07±0.00**	20.42±0.25	2.39±0.22	2.18±0.20	2.45±0.20	2.34±0.32	2.70±1.34	53.43±0.26
	0.25	**0.34±0.02**	20.47±0.13	2.53±0.21	2.34±0.19	2.60±0.19	2.49±0.30	2.82±1.26	
	0.3	**0.69±0.04**	20.54±0.32	2.71±0.20	2.54±0.18	2.79±0.18	2.67±0.28	2.97±1.16	

Table 5: Mean post-hoc regrets and standard deviations for the NSP using the Two-Stage Predict+Optimize framework.

Penalty factor	2S	Ridge	k-NN	CART	RF	NN	TOV
0.25±0.015	**3.94±1.91**	6.45±4.68	15.20±5.76	26.20±8.96	19.47±7.19	4.27±2.22	190.21±26.17
0.5±0.015	**6.92±2.26**	12.68±9.35	30.29±11.53	52.47±17.96	38.93±14.42	8.20±4.40	
1.0±0.015	**13.12±3.15**	25.12±18.71	60.43±23.07	105.01±36.00	77.86±28.99	16.00±8.78	
2.0±0.015	**25.04±9.29**	49.95±37.39	120.62±46.08	210.02±72.06	155.64±58.06	31.51±17.40	
4.0±0.015	**33.29±9.53**	99.61±74.78	241.01±92.14	420.04±144.18	311.19±116.23	62.52±34.64	
8.0±0.015	**46.72±14.80**	198.91±149.54	481.79±184.27	840.10±288.45	622.32±232.56	124.54±69.14	

between the post-hoc regret and the true optimal value, since the post-hoc regret also includes the penalty term which increases with the penalty factors.

0-1 knapsack In the second example, we showcase our framework on a packing integer programming problem, a variant of the 0-1 knapsack problem, with unknown item prices p_i and sizes s_i. See Appendix C.2 for details of an application in running a "proxy buyer" business. Here, the unknown parameters appear in both the objective and constraints. The proposed 2S method can handle this MILP straightforwardly, but the IntOpt-C method cannot be applied. Thus, we only experiment with the Two-Stage Predict+Optimize framework for evaluation, and compare the proposed 2S method with classical approaches and CombOptNet. Again, all approaches are evaluated at test time using the Stage 2 optimization to yield the final solution, on which the post-hoc regret is computed.

The MILP formulation of the two stages and the penalty function are described in Appendix C.2. We use the dataset of Paulus et al. [23], in which each 0-1 knapsack instance consists of 10 items and each item has 4096 features related to its price and size. For both NN and our method, we use a 5-layer fully-connected network with 512 neurons per hidden layer. We conduct experiments on 4 different knapsack capacities: 100, 150, 200, and 250. We use 700 instances for training and 300 instances for testing the model performance. Considering the real-life setting, we use 3 scales of the penalty factor for the penalty function in Appendix C.2: $\sigma = 0.05, 0.25$, or 0.5.

Table 4 reports the mean post-hoc regrets and standard deviations across 10 runs for each approach on this 0-1 knapsack problem. Due to the space limitation and the fact that larger penalty factors are unrealistic in this problem setting, we present penalty factors ≥ 1 in Appendix G. The average True Optimal Values (TOV) are reported in the last column, again for reference. As shown in the table, our proposed 2S method has significantly better results. In addition, we observe that across all approaches, the post-hoc regrets decrease as the knapsack capacity increases: this is due to the fact that as the capacity increases, more and more items can be selected, and so minor inaccuracies in predicted values/weights do not affect the selected set of items as much. On the other hand, the advantage of our 2S method over other approaches actually becomes more significant as the capacity increases, demonstrating the superior accuracy of our approach.

Nurse Scheduling Problem Our last experiment is on the nurse scheduling problem (NSP) with unknown patients needs, with the goal of scheduling a nurse roster satisfying unknown patient load demands while minimizing mismatched nurse-shift preferences as the objective. See Appendix C.3 for a description of the application scenario, the MILP formulations of the two stages, as well as the associated penalty function. Given that NSP is not an LP, IntOpt-C again does not apply, and so we only compare the proposed 2S training method with the classical approaches, using the Two-Stage Predict+Optimize framework for evaluation. Each NSP instance consists of 15 nurses, 7 days, and 3 shifts per day. The nurse preferences are obtained from the NSPLib dataset [26], which is widely used for NSP [16, 20]. The number of patients that each nurse can serve in one shift is randomly generated from [10,20], representing the fact that each nurse has different capabilities. Given that we are unable to find datasets specifically for the patient load demands and relevant prediction features, we follow the experimental approach of Demirovic et al. [4, 5, 6] and use real data from a different problem (the ICON scheduling competition) as the numerical values required for our experiment instances. In this dataset, the unknown number of patients per shift is predicted by 8 features.

Since there are far fewer features than the previous experiments, for both NN and 2S we use a smaller network structure: a 4-layer fully-connected network with 16 neurons per hidden layer. We use 210 instances for training and 90 instances for testing. Just like the first experiment, we use 6 scales of penalty factors (see Appendix C.3 for the penalty function): γ with i.i.d. entries drawn uniformly from $[0.25 \pm 0.015]$, $[0.5 \pm 0.015]$, $[1.0 \pm 0.015]$, $[2.0 \pm 0.015]$, $[4.0 \pm 0.015]$, and $[8.0 \pm 0.015]$.

Table 5 reports the mean post-hoc regrets and standard deviations across 10 runs for each approach on the NSP. The table shows that the proposed 2S method again has the best performance among all the training approaches. Our 2S method obtains at least 7.61%, 15.65%, 17.99%, 20.51%, 46.76%, and 62.49% smaller post-hoc regret than other classical methods when the penalty factor is $[0.25 \pm 0.015]$, $[0.5 \pm 0.015]$, $[1.0 \pm 0.015]$, $[2.0 \pm 0.015]$, $[4.0 \pm 0.015]$, and $[8.0 \pm 0.015]$ respectively.

Runtime Analysis Appendix H gives the training times for each method. Most classical approaches are faster than our 2S method, although as shown their post-hoc regrets are much worse. In alloy production, the only setting where IntOpt-C applies, its running time is shorter but comparable with 2S. In 0-1 knapsack, the only problem with public CombOptNet code, the 2S method is much faster.

6 Literature Review

Section 1 already summarized prior works in Predict+Optimize, most of which focus on learning unknowns only in the objective. Only the Hu et al. [12] framework considers unknowns in constraints.

Here we summarize other works related to learning unknowns in optimization problem constraints, particularly those outside of Predict+Optimize. These works can be placed into two categories.

One category also considers learning unknowns in constraints, but with very different goals and measures of loss. For example, CombOptNet [23] and Nandwani et al. [21] focus on learning parameters so as to make the predicted optimal solution (first-stage solution in our proposed framework) as close to the true optimal solution x^* as possible in the solution space/metric. By contrast, our proposed framework explicitly formulates the two-stage framework and post-hoc regret in order to directly capture rewards and costs in application scenarios. Experiments on 0-1 knapsack in Section 5 show that these other methods yield worse predictive performance when evaluated on the post-hoc regret, under the proposed two-stage framework.

Another category gives ways to differentiate through LPs or LPs with regularizations, as a technical component in a gradient-based training algorithm. As mentioned in Section 4, these works can indeed be used in place of our proposed approach in Section 4/Appendix B. However, we point out that: (i) these other technical tools are essentially orthogonal to our primary contribution, which is the two-stage framework (Section 3), and (ii) nonetheless, experiments on the 0-1 knapsack in Appendix E demonstrate that our gradient calculation approach performs at least as well in post-hoc regret performance as other works, while being faster.

7 Summary

We proposed Two-Stage Predict+Optimize: a new, conceptually simpler and more powerful framework for the Predict+Optimize setting where unknown parameters can appear both in the objective and in constraints. We showed how the simpler perspective offered by the framework allows us to give a general training framework for all MILPs, contrasting prior work which apply only to covering and packing LPs. Experimental results demonstrate that our training method offers significantly better prediction performance over other classical and state-of-the-art approaches.

Acknowledgments

We thank the anonymous referees for their constructive comments. In addition, Xinyi Hu and Jimmy H.M. Lee acknowledge the financial support of a General Research Fund (RGC Ref. No. CUHK 14206321) by the University Grants Committee, Hong Kong. Jasper C.H. Lee was supported in part by the generous funding of a Croucher Fellowship for Postdoctoral Research, NSF award DMS-2023239, NSF Medium Award CCF-2107079 and NSF AiTF Award CCF-2006206.

References

[1] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019.

[2] B. Amos and J. Z. Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pages 136–145. PMLR, 2017.

[3] B. Chen, P. L. Donti, K. Baker, J. Z. Kolter, and M. Bergés. Enforcing policy feasibility constraints through differentiable projection for energy optimization. In *Proceedings of the Twelfth ACM International Conference on Future Energy Systems*, pages 199–210, 2021.

[4] E. Demirović, P. J. Stuckey, J. Bailey, J. Chan, C. Leckie, K. Ramamohanarao, and T. Guns. An investigation into Prediction+Optimisation for the knapsack problem. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 241–257. Springer, 2019.

[5] E. Demirović, P. J. Stuckey, J. Bailey, J. Chan, C. Leckie, K. Ramamohanarao, and T. Guns. Predict+Optimise with ranking objectives: Exhaustively learning linear functions. *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 1078–1085, 2019.

[6] E. Demirović, P. J. Stuckey, T. Guns, J. Bailey, C. Leckie, K. Ramamohanarao, and J. Chan. Dynamic programming for Predict+Optimise. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*, pages 1444–1451, 2020.

[7] A. N. Elmachtoub and P. Grigas. Smart "Predict, then Optimize". *Management Science*, 68(1):9–26, 2022.

[8] A. N. Elmachtoub, J. C. N. Liang, and R. McNellis. Decision trees for decision-making under the predict-then-optimize framework. In *Proceedings of the 37th International Conference on Machine Learning*, pages 2858–2867, 2020.

[9] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*. Springer series in statistics New York, 2001. Volume 1, Number 10.

[10] A. U. Guler, E. Demirović, J. Chan, J. Bailey, C. Leckie, and P. J. Stuckey. A divide and conquer algorithm for Predict+Optimize with non-convex problems. In *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence*, 2022.

[11] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.

[12] X. Hu, J. C. H. Lee, and J. H. M. Lee. Predict+Optimize for packing and covering LPs with unknown parameters in constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

[13] X. Hu, J. C. H. Lee, J. H. M. Lee, and A. Z. Zhong. Branch & Learn for recursively and iteratively solvable problems in Predict+Optimize. In *Advances in Neural Information Processing Systems*, 2022.

[14] K. B. Kabir and I. Mahmud. Study of erosion-corrosion of stainless steel, brass and aluminum by open circuit potential measurements. *Journal of Chemical Engineering*, pages 13–17, 2010.

[15] N. Kahraman, B. Gülenç, and F. Findik. Joining of titanium/stainless steel by explosive welding and effect on interface. *Journal of Materials Processing Technology*, 169(2):127–133, 2005.

[16] B. Maenhout and M. Vanhoucke. Branching strategies in a Branch-and-Price approach for a multiple objective nurse scheduling problem. *Journal of scheduling*, 13(1):77–93, 2010.

[17] J. Mandi, V. Bucarey, M. M. K. Tchomba, and T. Guns. Decision-focused learning: Through the lens of learning to rank. In *International Conference on Machine Learning*, pages 14935–14947. PMLR, 2022.

[18] J. Mandi and T. Guns. Interior point solving for LP-based Prediction+Optimisation. *Advances in Neural Information Processing Systems*, 33:7272–7282, 2020.

[19] M. Mulamba, J. Mandi, M. Diligenti, M. Lombardi, V. Bucarey, and T. Guns. Contrastive losses and solution caching for Predict-and-Optimize. *arXiv preprint arXiv:2011.05354*, 2020.

[20] R. Muniyan, R. Ramalingam, S. S. Alshamrani, D. Gangodkar, A. Dumka, R. Singh, A. Gehlot, and M. Rashid. Artificial bee colony algorithm with Nelder–Mead method to solve nurse scheduling problem. *Mathematics*, 10(15):2576, 2022.

[21] Y. Nandwani, R. Ranjan, P. Singla, et al. A solver-free framework for scalable learning in neural ilp architectures. *Advances in Neural Information Processing Systems*, 35:7972–7986, 2022.

[22] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. 2019.

[23] A. Paulus, M. Rolínek, V. Musil, B. Amos, and G. Martius. Comboptnet: Fit the right NP-hard problem by learning integer programming constraints. In *International Conference on Machine Learning*, pages 8443–8453. PMLR, 2021.

[24] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[25] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[26] M. Vanhoucke and B. Maenhout. Nsplib–a nurse scheduling problem library: A tool to evaluate (meta-) heuristic procedures. In *Operational research for health policy: making better decisions, proceedings of the 31st annual meeting of the working group on operations research applied to health services*, pages 151–165, 2007.

[27] B. Wilder, B. Dilkina, and M. Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*, pages 1658–1665, 2019.