Predict+Optimize for Packing and Covering LPs 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

Predict+Optimize is a recently proposed framework which combines machine learning and constrained optimization, tackling optimization problems that contain parameters that are unknown at solving time. The goal is to predict the unknown parameters and use the estimates to solve for an estimated optimal solution to the optimization problem. However, all prior works have focused on the case where unknown parameters appear only in the optimization objective and not the constraints, for the simple reason that if the constraints were not known exactly, the estimated optimal solution might not even be feasible under the true parameters. The contributions of this paper are two-fold. First, we propose a novel and practically relevant framework for the Predict+Optimize setting, but with unknown parameters in both the objective and the constraints. We introduce the notion of a correction function, and an additional penalty term in the loss function, modelling practical scenarios where an estimated optimal solution can be modified into a feasible solution after the true parameters are revealed, but at an additional cost. Second, we propose a corresponding algorithmic approach for our framework, which handles all packing and covering linear programs. Our approach is inspired by the prior work of Mandi and Guns, though with crucial modifications and re-derivations for our very different setting. Experimentation demonstrates the superior empirical performance of our method over classical approaches.

Introduction

Constrained optimization problems are ubiquitous in daily life, yet, they often contain parameters that are unknown at solving time. As an example, retail merchants wish to optimize their stocking of products in terms of revenue and cost, and yet the precise demands for each product are not known ahead of time. The goal, then, is to 1) predict the unknown parameters and 2) solve the optimization problem using these predicted parameters, in the hopes that the estimated solution is good even under the true parameters revealed later on. The classical approaches would learn a predictor for these unknown parameters using losses like the mean squared error, which are independent of the optimization at hand. However, a small error for the predicted parameters in the parameter space does not necessarily guarantee a high solution quality evaluated under the true parameters. The recent framework of Predict+Optimize by Elmachtoub and Grigas (2017; 2022) proposes to instead use the more effective *regret function* as the loss function, capturing the difference in objective between the estimated and true optimal solutions, both evaluated using the true parameters.

A number of prior works (Wilder, Dilkina, and Tambe 2019; Elmachtoub, Liang, and McNellis 2020; Guler et al. 2022) have developed algorithmic implementations of this framework on a variety of classes of optimization problems. Yet, all the prior works have focused on the case where only the optimization objective contains unknown parameters, and never the constraints. This is for a simple technical reason: if we had used some predicted parameters to solve for an estimated solution, the solution might not even be feasible under the true parameters! On the other hand, some application scenarios allow for post-hoc correction of an estimated solution into a feasible solution after the true parameters are revealed, potentially at additional cost or penalty. Using the product stocking example again, a hard constraint is the available warehouse space, which needs to be predicted, depending on how well the already-bought products sell. If a merchant buys in excess of the available space, they always have the option to throw away some of the newly-bought products, which would involve 1) paying a disposal company as well as 2) losing out on the profit of the thrown-away products as a "penalty".

The contributions of this paper are two-fold. First, we capture the above intuition and significantly generalize the Predict+Optimize framework, allowing us to address optimization problems with unknown parameters in both the objective and the constraints. Specifically, we introduce the process of post-hoc correction, which makes use of a correction function and a penalty function. The definition of regret is enhanced to take into account the post-hoc correction of a solution, and the associated cost and penalty. Second, we propose an algorithmic implementation for this novel framework as applied to packing and covering linear programs (LPs), a well-studied and significant class of practically relevant optimization problems. We give a general correction function for packing and covering LPs, and demonstrate how to learn a predictor in this setting using an approach inspired by the work of Mandi and Guns (2020). We also apply our approach on 3 benchmarks to demonstrate the superior em-

pirical performance of our method over classic learning algorithms[1].

Background

In this section, we describe the formulation of Predict+Optimize as it appears in prior works, on problems with unknown parameters appearing only in the objective. The theory is stated in terms of minimization but applies of course also to maximization, upon appropriate negation.

An *optimization problem* P is defined as finding

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

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* which is to be minimized, and $C(x)$ is a set of constraints over x. We say x^* is an *optimal solution* and $obj(x^*)$ is the *optimal value*.

In prior works, a *parameterized optimization problem (Para-OP)* $P(\theta)$ extends an *optimization problem* P as:

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

where $\theta \in \mathbb{R}^t$ is a vector of parameters. The objective depends on θ, and note that the constraints do not (in prior works). When the parameters are known, a Para-OP is just an optimization problem.

In Predict+Optimize (Elmachtoub and Grigas 2017, 2022), the true parameters $\theta \in \mathbb{R}^t$ for a Para-OP are unknown at solving time, and *estimated parameters* $\hat{\theta}$ are used instead. Suppose that for each parameter, there are m relevant features. A learner is given n observations forming 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, and the task is to learn a *prediction function* $f : \mathbb{R}^{t \times m} \to \mathbb{R}^t$ predicting parameters $\hat{\theta} = f(A)$ from any feature matrix A.

The key aspect of Predict+Optimize is to measure quality of the estimated parameters $\hat{\theta}$ using the *regret function* as the loss function. The regret is the objective difference between the *true optimal solution* $x^*(\theta)$ and the *estimated solution* $x^*(\hat{\theta})$ under the true parameters θ. Formally, the regret function $Regret(\hat{\theta}, \theta) : \mathbb{R}^t \times \mathbb{R}^t \to \mathbb{R}_{\geq 0}$ is:

$$Regret(\hat{\theta}, \theta) = obj(x^*(\hat{\theta}), \theta) - obj(x^*(\theta), \theta)$$

where $obj(x^*(\hat{\theta}), \theta)$ is the *estimated optimal value* and $obj(x^*(\theta), \theta)$ is the *true optimal value*. Following the empirical risk minimization principle, prior learning methods (Elmachtoub, Liang, and McNellis 2020) aim to return the prediction function to be the function f from the set of models $\mathcal{F}$ attaining the smallest average regret over the training data:

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

Mandi and Guns (2020) proposed to use a (feedforward) neural network to predict the unknown parameters from features. The standard approach to training neural networks is via gradient descent using the backpropagation algorithm, in order to learn the weight on each edge of the network. Concretely, fixing a training feature matrix A and a corresponding true parameter vector θ, for each edge e on the network with weight w_e, we need to compute the derivative $\frac{\mathrm{d}Regret}{\mathrm{d}w_e}$. Using the multivariate chain rule, the derivative can be decomposed as follows:

$$\frac{\mathrm{d}Regret(\hat{\theta}, \theta)}{\mathrm{d}w_e} = \frac{\partial Regret(\hat{\theta}, \theta)}{\partial x^*(\hat{\theta})} \frac{\partial x^*(\hat{\theta})}{\partial \hat{\theta}} \frac{\partial \hat{\theta}}{\partial w_e} \qquad (2)$$

where $\frac{\partial Regret(\hat{\theta}, \theta)}{\partial x^*(\hat{\theta})}$ is a vector with the same length as the decision variable vector x^*, $\frac{\partial x^*(\hat{\theta})}{\partial \hat{\theta}}$ is a matrix, and $\frac{\partial \hat{\theta}}{\partial w_e}$ is a vector with the same length as the number of unknown parameters. The right hand side of the Equation 2 is to be interpreted as a matrix product.

On the right hand side, the first term is the gradient of the regret with respect to the estimated optimal solution. In the context of linear programs, this is trivial to compute since the objective function is linear in x^*. The third term, on the other hand, is the gradient of the estimated parameters with respect to the neural network edge weight, which can be computed efficiently using the standard backpropagation algorithm (Rumelhart, Hinton, and Williams 1986). What remains is the second term $\frac{\partial x^*}{\partial \hat{\theta}}$: the derivative of each decision variable with respect to each predicted parameter. In general, these derivatives do not exist for linear programs. Mandi and Guns (2020) thus proposed to use an interior-point LP solver: it generates a sequence of modified programs, with *logarithmic barrier terms* of decreasing weights introduced into the objective. Upon termination of the solver at an approximate optimum of the LP, the interior-point solver returns the approximate optimum as well as auxiliary information such as the weight of the barrier term at termination, all of which are used by Mandi and Guns to extract some gradient information related to the original problem.

Post-hoc Correction

We now generalize the framework in the previous section to include unknown parameters also in constraints.

The notion of a Para-OP can be easily extended to allow unknown parameters in both the objective and constraints:

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

Note that in this extension, both the objective and constraints depend on the unknown parameters θ.

When constraints contain unknown parameters, the feasible region is only approximated at solving time, and the estimated solution may be infeasible under the true parameters. Fortunately, in some applications, once the true parameters are revealed, there might be possible ways for us to correct an infeasible solution into a feasible one. This can be formalized as a *correction function*, which 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. The choice of correction function will be problem and application-specific;

[1]We allow estimated solutions to be corrected also for these classic learning algorithms, but the training itself just uses the original loss function, which is oblivious to any potential correction.

indeed, the space of correction functions depends on the situation. The goal then is to choose a correction function that generally loses the least amount in the objective from the correction.

Example 1. *Consider a simplified version of the product stocking problem. There are 4 divisible products (e.g. oil and rice). Each product i has a per-unit revenue r_i and a per-unit weight w_i, and there is a maximum of M_i units available for sourcing. The goal is to make an order of x_i units of item i, so as to maximize $\sum_{i=1}^{4} r_i \cdot x_i$ subject to the constraint $\sum_{i=1}^{4} w_i \cdot x_i \leq C$, where $r = [13, 14, 10, 11]$ and $w = [5, 3, 4, 9]$ are two arrays representing the per-unit revenues and weights of the products, as well as the constraint that $x_i \leq M_i$ for all i. However, the available capacity C when the products arrive is unknown at solving time, depending on the volume of sales between the orders being made and the arrival of the products.*

In Example 1, the products are selected based on an estimated warehouse capacity, but the prediction might be an overestimate. One trivial correction function is to throw out the entire order, which is not useful. A more useful correction function is to throw out some of each product to fit them into the actually available capacity.

While application scenarios may allow for post-hoc correction of an estimated solution, such correction may incur a penalty. A *penalty function* $Pen(x^*(\hat{\theta}) \rightarrow x^*_{corr}(\hat{\theta}, \theta))$ takes an estimated solution $x^*(\hat{\theta})$ and the corrected solution $x^*_{corr}(\hat{\theta}, \theta)$ and returns a non-negative penalty. In Example 1, the correction incurs both 1) logistical costs for removing items and 2) costs of having paid for these products.

We are now ready to define the notion of *post-hoc regret* $PReg(\hat{\theta}, \theta)$ with respect to correction function $x^*_{corr}(\hat{\theta}, \theta)$ and penalty function Pen:

$$\begin{aligned} PReg(\hat{\theta}, \theta) = obj(x^*_{corr}(\hat{\theta}, \theta), \theta) - obj(x^*(\theta), \theta) \\ + \ Pen(x^*(\hat{\theta}) \rightarrow x^*_{corr}(\hat{\theta}, \theta)) \end{aligned} \tag{3}$$

Learning a prediction function using different correction functions and penalty functions on the same Para-OP can yield vastly different prediction behaviors. For example, increasing the penalty will yield *conservative* predictions that induce estimated solutions that are more likely to be feasible under the true parameters, since an infeasible estimated solution would have to incur a large penalty. Similarly, in the above product stocking example, the trivial correction function of "throwing everything out" would encourage predictions (much) more conservative than a correction that throws out just enough items to fit the true available capacity.

Since the prediction behavior depends on the chosen correction and penalty functions, it is important for the practitioner to choose them *carefully*. The practitioner should make sure that 1) both accurately reflect the application scenario at hand and that 2) the correction function should be as efficient as the application setting allows.

Given a correction function and a penalty, we will follow Mandi and Guns (2020) and train a neural network to minimize the empirical post-hoc regret. In the rest of the paper, we will study the application of this framework to packing and covering linear programs. We will propose a generic correction function that should be applicable generally, and show how we can learn a neural network that performs well under the post-hoc regret.

Predict+Optimize on Packing LPs

In this section, we derive how we can train a neural network to predict unknown parameters in both the objective and constraints of a packing LP, under the new Predict+Optimize framework proposed in the last section.

Consider a packing LP in the standard form:

$$x^* = \arg\max_x c^\top x \ \text{ s.t. } Gx \leq h, x \geq 0 \tag{4}$$

with decision variables $x \in \mathbb{R}^d$ and problem parameters $c \in \mathbb{R}^d$, $G \in \mathbb{R}^{p \times d}_{\geq 0}$, $h \in \mathbb{R}^p_{\geq 0}$. Here, we consider the most general setting where all the problem parameters c, G, and h can be unknown.

We stated in the last section that the choice of a correction function generally depends on the specific problem and application. On the other hand, packing LPs have a lot of structure we can exploit. For example, the all 0s solution is always feasible. We propose the following generic correction function, which is generally applicable for packing LPs: given an uncorrected solution x^*, find the largest $\lambda \in [0, 1]$ such that λx^* satisfies the constraints under the true parameters. This can be formalized as follows:

$$\begin{aligned} x^*_{corr}(\hat{\theta}, \theta = (c, G, h)) = \lambda x^*(\hat{\theta}) \\ \text{where } \lambda = \max\{\lambda \in [0, 1] \,|\, G(\lambda x^*(\hat{\theta})) \leq h\} \end{aligned} \tag{5}$$

We also need to decide on a penalty function, which again is generally problem and application-specific. For simplicity and for wide applicability, in the rest of the paper we will assume that the penalty function is *linear*, in the sense that the penalty for the correction is the dot product between 1) the difference between the corrected and uncorrected solution vectors and 2) a vector of penalty factors. Due to scaling reasons, we express this vector of penalty factors in units of the objective c, that is, the penalty vector is $\sigma \circ c$ where $\circ$ is the Hadamard/entrywise product, and $\sigma \geq 0$ is a non-negative tunable vector. Then, the penalty function Pen is formally defined as $Pen(x^*(\hat{\theta}) \rightarrow x^*_{corr}(\hat{\theta}, \theta)) = (\sigma \circ c)^\top (x^* - x^*_{corr})$.

With the above choices of correction and penalty, we can now write down the simplified form of post-hoc regret for packing LPs. Note that, since packing LPs are maximization problems instead of minimization, the following has some sign differences from Equation 3.

$$\begin{aligned} PReg(\hat{\theta}, \theta) = c^\top (x^*(\theta) - x^*_{corr}(\hat{\theta}, \theta)) + \\ (\sigma \circ c)^\top (x^*(\hat{\theta}) - x^*_{corr}(\hat{\theta}, \theta)) \end{aligned} \tag{6}$$

where $\sigma \in \mathbb{R}^d_{\geq 0}$.

Following the approach of Mandi and Guns (2020), briefly described in in the Background section, we use a neural network (of various architectures depending on the precise problem) to predict the parameters, before feeding the parameters into the interior-point LP solver of Mandi and

Guns. This interior point solver iteratively generates a sequence of relaxations to the LP, into problems of the form

$$x^* = \arg\max_x c^\top x + \mu \left[\sum_{i=1}^{d} \ln(x_i) + \sum_{i=1}^{p} \ln(h_i - G_i^\top x) \right] \tag{7}$$

for a sequence of decreasing non-negative μ. Upon termination, we retrieve a solution x which is approximately the optimum of the original LP, as well as the value of μ last used.

We derive how, using the solution x and the barrier weight μ, we can compute the relevant (approximations of) derivatives in order to train the neural network via gradient descent. Using the law of total derivative, we get

$$\frac{\mathrm{d}PReg(\hat{\theta},\theta)}{\mathrm{d}w_e} = \left.\frac{\partial PReg(\hat{\theta},\theta)}{\partial x^*_{corr}}\right|_{x^*} \frac{\partial x^*_{corr}}{\partial x^*} \frac{\partial x^*(\hat{\theta})}{\partial \hat{\theta}} \frac{\partial \hat{\theta}}{\partial w_e} + \left.\frac{\partial PReg(\hat{\theta},\theta)}{\partial x^*}\right|_{x^*_{corr}} \frac{\partial x^*(\hat{\theta})}{\partial \hat{\theta}} \frac{\partial \hat{\theta}}{\partial w_e} \tag{8}$$

On the right hand side, the terms $\left.\frac{\partial PReg(\hat{\theta},\theta)}{\partial x^*_{corr}}\right|_{x^*}$ and $\left.\frac{\partial PReg(\hat{\theta},\theta)}{\partial x^*}\right|_{x^*_{corr}}$ are straightforward from (6): $\left.\frac{\partial PReg(\hat{\theta},\theta)}{\partial x^*_{corr}}\right|_{x^*} = -(1+\sigma)\circ c$ and $\left.\frac{\partial PReg(\hat{\theta},\theta)}{\partial x^*}\right|_{x^*_{corr}} = \sigma \circ c$. The term $\frac{\partial \hat{\theta}}{\partial w_e}$ relates only to the neural network and is handled directly by the standard backpropagation algorithm (Rumelhart, Hinton, and Williams 1986). Therefore, in the remainder of this section, we show how to compute (approximations of) $\frac{\partial x^*_{corr}}{\partial x^*}$ and $\frac{\partial x^*(\hat{\theta})}{\partial \hat{\theta}}$.

Computing $\frac{\partial x^*_{corr}}{\partial x^*}$. The term $\frac{\partial x^*_{corr}}{\partial x^*}$ is determined solely by the correction function (5), and has nothing to do with the LP solver. We use the law of total derivative again to decompose the term:

$$\frac{\partial x^*_c(\hat{\theta},\theta)}{\partial x^*(\hat{\theta})} = \left.\frac{\partial x^*_c(\hat{\theta},\theta)}{\partial \lambda}\right|_{x^*} \frac{\partial \lambda}{\partial x^*(\hat{\theta})} + \left.\frac{\partial x^*_c(\hat{\theta},\theta)}{\partial x^*(\hat{\theta})}\right|_{\lambda}$$

Observe that $\left.\frac{\partial x^*_c(\hat{\theta},\theta)}{\partial \lambda}\right|_{x^*} = x^*(\hat{\theta})$ and $\left.\frac{\partial x^*_c(\hat{\theta},\theta)}{\partial x^*(\hat{\theta})}\right|_{\lambda} = \lambda I$ (I is an identity matrix). It remains to derive $\frac{\partial \lambda}{\partial x^*(\hat{\theta})}$, captured in the following lemma.

Lemma 1. *Let $x^*(\hat{\theta})$ denote the estimated optimal solution of the packing LP shown in (4), $x^*_{corr}(\hat{\theta},\theta) = \lambda x^*(\hat{\theta})$ be the correction function shown in (5). Suppose that at the optimal λ of (5), the i^{th} inequality constraint G_i is tight, namely $G_i^\top(\lambda x^*(\hat{\theta})) = h_i$. Then, we have*

$$\frac{\partial \lambda}{\partial x^*(\hat{\theta})} = -\frac{\lambda}{G_i^\top x^*(\hat{\theta})} G_i^\top.$$

As a corollary, we have

$$\frac{\partial x^*_{corr}(\hat{\theta},\theta)}{\partial x^*(\hat{\theta})} = \frac{-\lambda}{G_i^\top x^*(\hat{\theta})} x^*(\hat{\theta}) G_i^\top + \lambda I.$$

The lemma follows from an implicit differentiation of the tight constraint and the proof is in Appendix A.1.

Approximating $\frac{\partial x^*(\hat{\theta})}{\partial \hat{\theta}}$. Recall that the interior point solver of Mandi and Guns (2020) solves a sequence of relaxations of the form given in Equation (7). The term $\mu[\sum_{i=1}^{d} \ln(x_i) + \sum_{i=1}^{p} \ln(h_i - G_i^\top x)]$ is also known as a logarithmic barrier term, which is commonly used in interior-point based solving methods (Boyd and Vandenberghe 2004). At termination, we get the values of x^* and μ. We will use these values, as well as Equation (7), to approximate the gradient information $\frac{\partial x^*(\hat{\theta})}{\partial \hat{\theta}}$.

In the context of the packing LP, the unknown parameter $\hat{\theta}$ may either be c, G or h. The case of c has already been derived by Mandi and Guns (2020) (see Appendix A.1 and A.2 in their paper). The following two lemmas captures the other two cases.

Define the notation $f(x,c,G,h) = c^\top x + \mu(\sum_{i=1}^{d} \ln(x_i)) + \mu(\sum_{i=1}^{p} \ln(h_i - G_i x))$. Then, Problem (7) can be expressed as finding $x^* = \arg\max_x f(x,c,G,h)$. Using this notation, we write down the following two lemmas on computing $\frac{\partial x^*}{\partial h}$ and $\frac{\partial x^*}{\partial G}$ approximately, with proofs in Appendix A.1.

Lemma 2. *Consider the LP relaxation (7), defining x^* as a function of c, G and h. Then, under this definition of x^*,*

$$\frac{\partial x^*}{\partial h} = -f_{xx}(x^*)^{-1} f_{hx}(x^*)$$

where f_{xx} denotes the matrix of second derivatives of f with respect to different coordinates of x, and similarly for other subscripts, and explicitly:

$$f_{x_k x_j}(x) = \begin{cases} -\mu x_j^{-2} - \mu \sum_{i=1}^{p} G_{ij}^2/(h_i - G_i^\top x)^2 & j = k \\ -\mu \sum_{i=1}^{p} G_{ij} G_{ik}/(h_i - G_i^\top x)^2 & j \neq k \end{cases}$$

and

$$f_{h_\ell x_j}(x) = \mu G_{\ell j}/(h_\ell - G_\ell^\top x)^2$$

Lemma 3. *Consider the LP relaxation (7), defining x^* as a function of c, G and h. Then, under this definition of x^*,*

$$\frac{\partial x^*}{\partial G} = -f_{xx}(x^*)^{-1} f_{Gx}(x^*)$$

where f_{xx} is defined as in Lemma 2 and

$$f_{G_{\ell q} x_j}(x) = \begin{cases} -\mu G_{\ell j} x_q/(h_\ell - G_\ell^\top x)^2 - \mu/(h_\ell - G_\ell^\top x) & q = j \\ -\mu G_{\ell j} x_q/(h_\ell - G_\ell^\top x)^2 & q \neq j \end{cases}$$

We end this section with a remark that the LP solver of Mandi and Guns (2020) in fact returns more information than just x and μ. In their work, they start not with (7) but with the homogeneous self-dual (HSD) formulation of the original LP, involving the extra information returned by the solver, and perform derivative calculations similar in spirit to our lemmas in this section. However, in our context of unknown G and h, if we also tried using the HSD formulation for gradient calculations, we would end up with derivatives that are degenerate. For this reason, we have opted to use the simpler Equation (7) which, as we demonstrate in the Experimental Evaluation section, appears to work well in practice.

Predict+Optimize on Covering LPs

Covering LPs are closely related to packing LPs—in fact, they are the duals of each other. Consider a covering LP in standard form:

$$x^* = \arg\min_x c^\top x \text{ s.t. } Gx \geq h, x \geq 0 \tag{9}$$

with decision variables $x \in \mathbb{R}^d$ and problem parameters $c \in \mathbb{R}^d$, $G \in \mathbb{R}^{p \times d}$, $h \in \mathbb{R}^p$. We are again in the general setting where all the problem parameters c, G, and h can be unknown.

Performing Predict+Optimize on covering LPs is essentially the same as in the previous section, up to some sign changes to account for changed inequality directions and minimization vs maximization. The only non-trivial difference is the need to change the correction function. Instead of scaling down an uncorrected solution for feasibility, we will scale up in covering LPs, defined formally as follows:

$$\begin{gathered} x^*_{corr}(\hat{\theta}, \theta = (c, G, h)) = \lambda x^*(\hat{\theta}) \\ \text{where } \lambda = \min\{\lambda \geq 1 \,|\, G(\lambda x^*(\hat{\theta})) \geq h\} \end{gathered} \tag{10}$$

We use the same penalty function as in the packing LP case. The differentiation calculations from the last section apply essentially verbatim to covering LPs apart from minor sign differences. We therefore defer the details to Appendix A.2.

Experimental Evaluation

We evaluate the proposed interior point based method with post-hoc correction (IntOpt-C) on 3 benchmarks: a maximum flow transportation problem with unknown edge capacities, an alloy production problem with unknown chemical composition in the raw materials, and a fractional knapsack problem with unknown rewards and weights. We compare our method with 5 classical regression methods (Friedman, Hastie, and Tibshirani 2001) including ridge regression (Ridge), k-nearest neighbors (k-NN), classification and regression tree (CART), random forest (RF), and neural network (NN). All of these methods train the prediction models with their classic loss function. We also apply the chosen correction function of each problem to the estimated solutions for these classical regression methods, in order to ensure feasibility, to compute the post-hoc regret. However, the correction function has nothing to do with the training of these classic methods. The methods of k-NN, RF and NN as well as our method have hyperparameters, which we tune via cross-validation: for k-NN, we tried $k \in \{1, 3, 5\}$; for RF, we try different numbers of trees in the forest $\{10, 50, 100\}$; for both NN and our method, we treat the learning rate, epochs and weight decay as hyperparameters. We include the final hyperparameter choices in Appendix B.[2]

Ridge, k-NN, CART and RF are implemented using *scikit-learn* (Pedregosa et al. 2011). The neural network is implemented using *PyTorch* (Paszke et al. 2019). All models are trained with Intel(R) Xeon(R) CPU @ 2.20GHz. To compute the optimal solution of an LP under the true parameters, we use the LP solver from *Gurobi* (Gurobi Optimization, LLC 2023) instead of the solver of Mandi and Guns.

We first focus on the solution quality comparisons and then make the runtime analysis.

Solution Quality

We compare the solution quality of the proposed IntOpt-C method and 5 classical regression methods on 3 benchmarks: a maximum flow transportation problem with unknown edge capacities, an alloy production problem with unknown chemical composition in the raw materials, and a fractional knapsack problem with unknown rewards and weights.

A maximum flow transportation problem with unknown capacities. In our first experiment, we formulate a transportation problem as a single-source-single-sink maximum flow problem (MFP). To formulate it as a packing LP, we use the formulation where the decision variables each correspond to a simple path from the source to the sink. In this experiment, the unknown parameters are the edge capacities, which is the h vector in the packing LP. We experiment in a setting where the goal is to use Predict+Optimize to learn which paths we will be using for transport, and proportionally how much flow we will be sending along each path—for example, the prediction is used to apply for permits from a city council for sending a lot of traffic along particular routes. Given that we are less concerned about predicting the actual flow magnitudes, in this experiment we set the penalty factor σ to the all-0s vector.

We conduct experiments on 3 real-life graphs: POLSKA (Orlowski et al. 2007) with 12 vertices and 18 edges, USANet (Lucerna et al. 2009) with 24 vertices and 43 edges, and GÉANT (LLC 2018) with 40 vertices and 61 edges. Given that we are unable to find datasets specifically for this max-flow problem, we follow the experimental approach of Demirovic et al. (2019; 2019; 2020) and use real data from a different problem (the ICON scheduling competition) as numerical values required for our experiment instances. In this dataset, each unknown edge capacity has 8 features. For experiments on POLSKA and USANet, out of the available 789 instances, 610 are used for training and 179 for testing the model performance, while for experiments on GÉANT, out of the available 620 instances, 490 are used for training and 130 for testing the model performance.

For both NN and our method, we use a 3-layer fully-connected network with 16 neurons per hidden layer.

Figure 1 shows a box plot of the post-hoc regrets of the various methods on the three different graphs. We observe that our IntOpt-C method (in red) achieves the smallest post-hoc regret in all cases. Compared with other methods, IntOpt-C obtains at least 10.71% smaller post-hoc regret on POLSKA, 10.67% smaller on USANet, and 10.02% smaller on GÉANT.

For comparison, we also show a box plot of the mean squared error (MSE), i.e. squared ℓ_2 error, of the predicted parameters, across different methods and graphs, in Figure 2 (in log scale). Even though the goal is to minimize post-hoc

[2]Our implementation is available at https://github.com/dadahxy/AAAI_PostHocRegret.

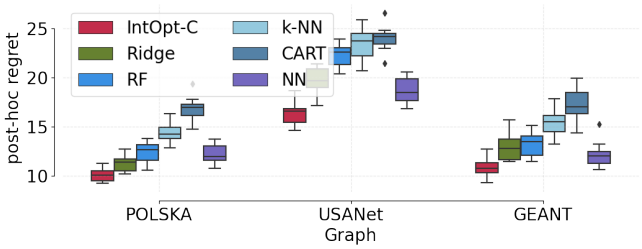

Figure 1: Post-hoc regrets for max-flow transportation.

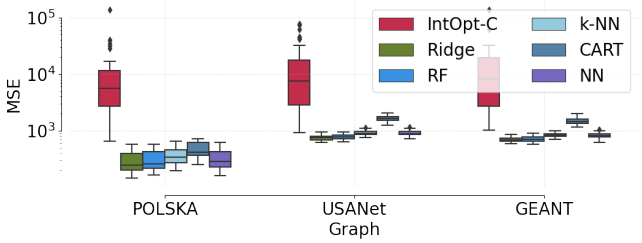

Figure 2: MSE for max-flow transportation.

regret and it is unreasonable to evaluate our method on the MSE, we present these results anyway for all our experiments, as they help illustrate the behavior of our method. In this experiment, the MSE of IntOpt-C is drastically worse than all the other methods. We argue that this is by design: our method optimizes for learning in terms of post-hoc regret, while all the other classical methods learn to minimize in MSE. There does remain the question of why our method has that bad of an MSE. From investigating the data (see Appendix C.1), this is due to our method predicting the unknown parameters at several orders of magnitude larger than the true parameters. The reason for this phenomenon lies in the problem formulation, where we are trying to predict which paths to send flows through, and are less concerned with predicting the precise amount of flow. As a modelling choice, therefore, we picked the penalty factor $\sigma = 0$. Note that, since the unknown parameters are the "h" vector in the packing LP, if we scale up the h vector, then the corresponding solutions x are scaled up by the same factor. Thus, the phenomenon is equivalent to the estimated solution being much larger than the true optimal solution. This is fine from the Predict+Optimize perspective: the correction function scales down an over-capacity estimated solution, and so the predictor only needs to predict the direction of the solution vector; even if it gives a far-too-large norm, the correction function will fix the magnitude at no cost. Our learning algorithm appears to have learnt to exploit this correction function, and nonetheless, the estimated solution still gives the desired information—which paths to send flow along in the graph.

If we did care about predicting the actual flow values, then we would set the penalty factor to a non-zero value. In the next couple of experiments, we explore how the penalty factor affects the performance of our method, in terms of both the post-hoc regret and the MSE of the predicted parameters. We note again that the penalty factor is a property of the application, and not an algorithmic choice we make.

An alloy production problem with unknown chemical compositions in raw materials. In our second experiment, we consider an alloy production problem that is expressible as a covering LP. An alloy production plant needs to produce a certain amount of a particular alloy, requiring a mixture of M kinds of metals. To that end, it must acquire at least req_m tons of each of the $m \in [M]$ metals. The raw materials are to be obtained from K suppliers, each supplying a different type of ore. The ore supplied by site $k \in [K]$ contains a $con_{km} \in [0, 1]$ fraction of material m at a price of $cost_k$ per ton. The objective is to meet the minimum material requirements for each metal, at the minimum cost. The decision variables x_k are the number of tons of ores to order from each site k. Affected by the uncertainty in the mining process, the metal concentration (% of the $m \in M$ material per ton) of each ore is unknown, i.e. con_{km} is unknown, which is the G matrix in the covering LP.

Following the correction function and penalty described in Sections and respectively, if the estimated solution does not meet the minimum tonnage requirements of any metal, the alloy production plant will scale up its order by a factor of $\lambda \geq 1$ (from Equation 10) across all the suppliers. On the other hand, for this after-the-fact order, each supplier k will charge a new cost of $(1 + \sigma_k)cost_k$ per ton of its ore, instead of the previous cost of $cost_k$. We experimented on various values of penalty factors σ_k and we will report and discuss how the value of σ_k affects the performance of the prediction pipeline. We stress again that the value of σ_k is from the application, and not an algorithmic choice.

We conduct experiments on two real alloys: brass and an alloy blend for strengthening Titanium. For brass, 2 kinds of metal materials, Cu and Zn, are required (Kabir and Mahmud 2010), that is $M = 2$. The requirements 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 (Kahraman, Gülenç, and Findik 2005), i.e., $M = 4$. The requirements of the four materials are $req = [0.8, 60, 40, 2.5]$. Since we could not find any real data on the concentration of metals in ores, we again use real data from a different problem (a knapsack problem (Paulus et al. 2021)) as numerical values in our experiment instances. In this dataset, each unknown metal concentration is related to 4096 features. For experiments on both of the two alloys productions, 350 instances are used for training and 150 instances for testing the model performance.

For NN and our method, we use a 5-layer fully connected network with 512 neurons per hidden layer.

We conduct experiments on 5 types of penalty factor (σ) settings: the all-0s vector, and then 4 vectors where each entry is i.i.d. uniformly sampled from $[0.25 \pm 0.015]$, $[0.5 \pm 0.015]$, $[1.0 \pm 0.015]$, and $[2.0 \pm 0.015]$ respectively. This random sampling of σ ensures that the penalty factor for each supplier is different, but remain roughly in the same scale.

For space reasons, we only report the experimental results for the brass alloy, and defer the qualitatively-similar titanium-alloy results to Appendix C.2. Figure 3 shows the

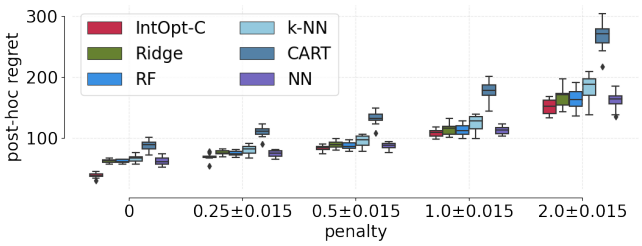

Figure 3: Post-hoc regrets for alloy production.

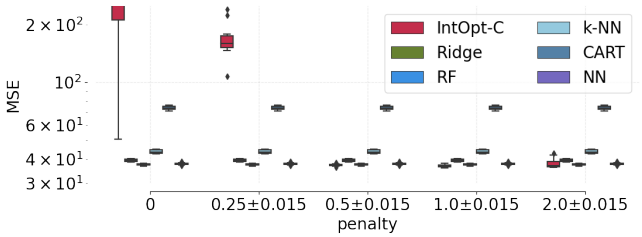

Figure 4: MSE for alloy production.

post-hoc regrets of the different learning methods, across the different scales of penalty factor σ. When the penalty factor is 0, our method improves the solution quality substantially, obtaining at least 38.67% smaller post-hoc regret than the other methods. When the penalty factor is non-zero as given in the last paragraph, our method obtains at least 7.80%, 3.99%, 3.24%, and 6.56% smaller post-hoc regret respectively. The results suggest that the advantages of our IntOpt-C method on solution quality first decreases and then increases as the penalty factor σ grows.

We show also the MSE of the predicted parameters in Figure 4, across different methods, in log scale. As discussed in the previous max-flow experiment, when the penalty is 0, the MSE for our method can be very large as the lack of penalty gets exploited by the method. Interestingly, as we observe in Figure 4, the MSE for our method first decreases and then increases as σ grows (the growth at the end is slightly difficult to read on this plot). Here we explain why the MSE values of IntOpt-C may increase when the penalty term grows too large. When the penalty is non-zero but somewhat small, it acts as a regularizer to prevent our method from exploiting the correction function as in the previous experiment. On the other hand, as the penalty increases, the post-hoc regret becomes dominated by the penalty term. As such, our method is strongly disincentivized to use *any* correction whatsoever. Therefore, when σ is large, our method tends to be conservative and always predicts parameters that make the estimated solution a bit too large. This explains why the MSE of the predicted parameters gets bigger again (albeit not by much) as σ increases.

Fractional knapsack problem with unknown prices and weights. The last experiment is on the fractional knapsack problem with unknown rewards and weights. The unknown parameters appear in both objective "c" and constraints "G" of the packing LP. In our setting, word descriptions of a collection of M infinitely-divisible items is presented to the algorithm, from which the weight w_i and reward c_i of each item i need to be predicted. The player's goal is to maximize the total reward of (fractionally) selected items without exceeding a known fixed capacity of the knapsack. We use the dataset of Paulus et al. (2021), in which each fractional knapsack instance consists of 10 items and each item has 4096 features related to its reward and weight.

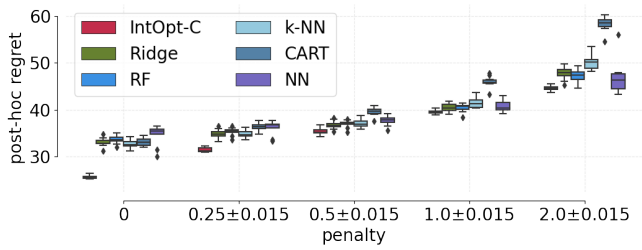

Figure 5: Post-hoc regrets for fractional knapsack.

For both NN and our method, we use a 5-layer fully-connected network with 512 neurons per hidden layer.

In line with the choice of correction function and penalty in the section for Packing LPs, if the estimated solution violates the capacity constraint, items will need to be removed at a penalty and in a proportional manner (i.e. the over-capacity knapsack is scaled down). If the change in the amount of item i is Δ_i, then the penalty for this removal is $\sigma_i c_i \Delta_i$.

We conduct experiments on 4 different capacities: 50, 100, 150, and 200. In the main paper, we report only the results for capacity 200—the rest are qualitatively similar, and they can be found in Appendix C.3. We use 700 instances for training and 300 instances for testing the model performance. Identically to the second experiment, we use 5 scales of penalty factors: all-0s penalty, and penalty factor σ with i.i.d. entries drawn uniformly from $[0.25 \pm 0.015]$, $[0.5 \pm 0.015]$, $[1.0 \pm 0.015]$, and $[2.0 \pm 0.015]$.

Figure 5 shows the post-hoc regrets of the different methods across the different scales of penalty factors. Our method again performs the best across all the evaluated algorithms. Observing a similar trend as in the alloy production experiment, the improvements of our method over other classical methods, in terms of the post-hoc regret, first decreases and then increases as the penalty factor σ grows. When the penalty factor is 0, our method obtains at least 21.48% smaller post-hoc regret than other methods. When the penalty factor is non-zero as given in the last paragraph, the post-hoc regret of our method is at least 9.78%, 4.57%, 2.18%, and 4.83% smaller.

As in the previous experiments, we also compare the MSE of the parameters predicted by our method against the other methods, as shown in Figure 6 in log scale. Similar to the other experiments, when the penalty term is zero, the predicted parameters of IntOpt-C are shifted by several orders of magnitude from the true parameters (the post-hoc regret is small but the MSE value is large). Then, as σ grows, the MSE of our method decreases to roughly the same as the

Runtime(s)	Maximum flow transportation			Alloy production		Fractional knapsack			
	POLSKA	USANet	GÉANT	Brass	Titanium-alloy	Capacity=50	Capacity=100	Capacity=150	Capacity=200
IntOpt-C	18.65	132.22	15.48	228.00	331.38	131.49	132.89	139.44	132.37
Ridge	<1	<1	<1	20.22	56.89	22.33			
k-NN	<1	<1	<1	25.14	70.22	26.00			
CART	<1	<1	<1	30.33	94.89	34.83			
RF	4.11	11.00	11.89	959.50	2552.25	1034.07			
NN	10.33	12.82	13.89	212.22	321.11	135.80			

Table 1: Average runtime (in seconds) for the maximum flow transportation, alloy production, and fractional knapsack problems.

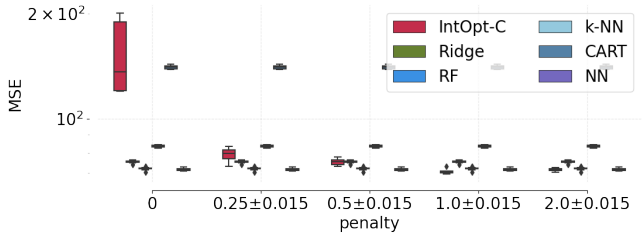

Figure 6: MSE for fractional knapsack.

other methods, before growing slightly again as σ becomes large and the predictor learnt from our method becomes conservative.

Runtime Analysis

Table 1 shows the average runtime across 10 simulations for different optimization problems. Here, the runtime refers to only the training time of the prediction model, and does not include the testing time which is relatively small and insignificant for our benchmarks. At training time, only our method solves the LP. Training for the usual NN does not involve the LP at all, and so training is much faster (but gives worse results).

In the alloy production problem and the fractional knapsack problem, the runtimes of our IntOpt-C method are comparable to NN, and are much better than RF. In the maximum flow transportation problem, the runtimes of IntOpt-C are comparable to NN in POLSKA and GÉANT, but the runtimes of IntOpt-C are large in USANet. The reason is that we use the formulation where each of the decision variables corresponds to a simple path from the source to the sink. Thus, when the number of paths is large (the number of paths in USANet is 242), the number of decision variables of the LP is large and the LP takes more time to solve.

Summary

We proposed the first Predict+Optimize framework addressing the scenario where the constraints may contain unknown parameters. Specifically, we introduced the novel notions of correction function, penalty function and post-hoc regret into the framework. Algorithmically, we focused on packing and covering linear programs—a large and widely-studied class of problems—and presented a method to train parameter predictors in our novel framework. Empirical results in 3 benchmarks demonstrate better prediction performance of our method over 5 classical methods which do not take the correction function into account during training. An interesting piece of future work is to extend the use of post-hoc regret for a wider class of problems.

We also note that our framework is designed only for applications that allow post-hoc correction of an infeasible solution into a feasible one, after the true parameters are revealed. In addition, the correction function and the penalty function are problem and application specific, and thus require human input for their formulation. Another interesting research direction is to explore automatic suggestions of useful correction functions and penalty functions based on the problem models.

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

Boyd, S.; and Vandenberghe, L. 2004. *Convex optimization*. Cambridge University Press.

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

Demirović, E.; Stuckey, P. J.; Bailey, J.; Chan, J.; Leckie, C.; Ramamohanarao, K.; and Guns, T. 2019. Predict+Optimise with Ranking Objectives: Exhaustively Learning Linear Functions. *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, 1078–1085.

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

Elmachtoub, A. N.; and Grigas, P. 2017. Smart "Predict, then Optimize". *Technical report. https://arxiv.org/pdf/1710.08005.pdf.*

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

Elmachtoub, A. N.; Liang, J. C. N.; and McNellis, R. 2020. Decision Trees for Decision-Making under the Predict-then-Optimize Framework. In *Proceedings of the 37th International Conference on Machine Learning*, 2858–2867.

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

Guler, A. U.; Demirović, E.; Chan, J.; Bailey, J.; Leckie, C.; and Stuckey, P. J. 2022. A Divide and Conquer Algorithm for Predict+Optimize with Non-Convex Problems. In *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence*.

Gurobi Optimization, LLC. 2023. Gurobi Optimizer Reference Manual. https://www.gurobi.com. Accessed: 2022-08-13.

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

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

LLC, M. 2018. Geant Topology Map dec2018 copy. https://www.geant.org/Resources/Documents/GEANT_Topology_Map_December_2018.pdf. Accessed: 2020-09-10.

Lucerna, D.; Gatti, N.; Maier, G.; and Pattavina, A. 2009. On the efficiency of a game theoretic approach to sparse regenerator placement in WDM networks. In *GLOBECOM 2009-2009 IEEE Global Telecommunications Conference*, 1–6. IEEE.

Mandi, J.; and Guns, T. 2020. Interior Point Solving for LP-based Prediction+Optimisation. *Advances in Neural Information Processing Systems*, 33: 7272–7282.

Orlowski, S.; Pióro, M.; Tomaszewski, A.; and Wessäly, R. 2007. SNDlib 1.0–Survivable Network Design Library. In *Proceedings of the 3rd International Network Optimization Conference*. http://sndlib.zib.de, extended version accepted in Networks, 2009.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, 8024–8035. Curran Associates, Inc.

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

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

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

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