Branch & Learn with Post-hoc Correction for Predict+Optimize with Unknown Parameters in Constraints

Xinyi Hu[1(✉)], Jasper C. H. Lee[2(✉)], and Jimmy H. M. Lee[1(✉)]

[1] Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
xyhu@cse.cuhk.edu.hk, jasper.lee@wisc.edu
[2] Department of Computer Sciences & Institute for Foundations of Data Science, University of Wisconsin-Madison, Madison, WI, USA
jlee@cse.cuhk.edu.hk

Abstract. Combining machine learning and constrained optimization, Predict+Optimize tackles optimization problems containing parameters that are unknown at the time of solving. Prior works focus on cases with unknowns only in the objectives. A new framework was recently proposed to cater for unknowns also in constraints by introducing a loss function, called Post-hoc Regret, that takes into account the cost of correcting an unsatisfiable prediction. Since Post-hoc Regret is non-differentiable, the previous work computes only its approximation. While the notion of Post-hoc Regret is general, its specific implementation is applicable to only packing and covering linear programming problems. In this paper, we first show how to compute Post-hoc Regret exactly for any optimization problem solvable by a recursive algorithm satisfying simple conditions. Experimentation demonstrates substantial improvement in the quality of solutions as compared to the earlier approximation approach. Furthermore, we show experimentally the empirical behavior of different combinations of correction and penalty functions used in the Post-hoc Regret of the same benchmarks. Results provide insights for defining the appropriate Post-hoc Regret in different application scenarios.

Keywords: Constraint Optimization · Machine Learning · Predict+Optimize

1 Introduction

Constraint optimization problems are ubiquitous and occur in many daily and industrial applications [2,7,8]. In practice, constraint optimization problems can contain certain parameters which are unknown at the time of solving and require prediction based on some historical records. For example, a train company needs to schedule a minimal number of trains while meeting the passenger demand, but the precise demand is unknown ahead of time and needs to be predicted. The task is to 1) predict the unknown parameters, then 2) solve the optimization problem using the predicted parameters, such that the resulting solutions are good even under true parameters. Traditionally, in the prediction stage, machine learning models are trained with error metrics independent of

optimization problems, such as mean squared error. However, this kind of error metric does not necessarily represent the performance of the resulted solutions. The predicted parameters may in fact lead to a low-quality solution for the (true) optimization problem despite being "high-quality" for the error metric. Predict+Optimize instead trains the prediction model with the more effective *regret* function, capturing the difference in objective between the estimated and true optimal solutions, both evaluated using the true parameters. The challenge is, regret, the new error metric, is piecewise constant and non-differentiable [4], thus gradient-based methods do not apply.

A number of prior works [3–5, 10, 20] propose methods to overcome the non-differentiability of regret, and they can be roughly divided into *approximation* and *exact* methods. Approximation methods [5, 20] compute the (approximate) gradients of (approximations of) the regret function. They work not with the regret loss itself, but an approximation of it. While novel, they are not always reliable. On the other hand, exact methods [3, 4, 10] work directly with the regret to find a good prediction model, even if the method cannot always find the global-optimum model for the training data (e.g. if the method uses a local optimization method to find the output model). To overcome the nondifferentiability of the regret, they exploit the structure of optimization problems to train models without computing gradients, and can be applied to dynamic programming solvable problems [4] and recursively solvable problems [10].

Despite the variety of approaches, most of the previous works [3–5, 10, 20] on Predict+Optimize handle problems with unknowns only in the objective. When constraints contain also unknown parameters, one major challenge is that the estimated solution may end up being *infeasible* under the true parameters—an issue inherent with uncertainty in constraints. The *regret* function designed for fixed solution space is not applicable in this situation. Hu *et al.* [9] propose a more general loss function called *post-hoc regret*, in which an infeasible estimated solution is first corrected into a feasible one (with respect to true parameters), and then the error of prediction is the sum of 1) the objective difference between the true optimal solution and the feasible solution, and 2) potential penalty incurred by correction. When unknown parameters only appear in the objective, the post-hoc regret degenerates into the regret. The post-hoc regret is also nondifferentiable, and Hu *et al.* further propose an approximation approach for packing and covering linear programs [9]. However, exact approaches considering the post-hoc regret remain uncovered.

The contributions of this paper are threefold. First, we propose the first exact approach for Predict+Optimize with unknown parameters in both the objective and constraints. The proposed method is extended from Branch & Learn [10] and handles recursively solvable problems with unknown constraints. Second, extensive experiments are conducted to investigate the performance of post-hoc regret. We experimentally compare the proposed method with the state-of-the-art approximation method [9] and investigate the performance of post-hoc regret on more general problems. Third, we empirically study different combinations of the two key components of post-hoc regret, i.e., the correction function and the penalty function to gain insights for defining post-hoc regret in different scenarios. Due to the page limit, we opted to present only part of the experiment results. For the complete experiment results and analysis, as well as future work, please refer to the extended version on arXiv with the same title.

2   Background

Without loss of generality, we define an *optimization problem* (OP) P 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 real-valued objective function in x which is to be minimized, and C is a set of constraints over x. We say x^* is an *optimal solution* and $obj(x^*)$ is 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 the constraints $C(x, \theta)$ all depend on θ. An OP is a degenerated case of a Para-OP when there are no unknowns.

The true parameters $\theta \in \mathbb{R}^t$ for a Para-OP are hidden at the time of solving in the *Predict+Optimize* (P+O) setting [3], and *estimated parameters* $\hat{\theta}$ are utilized in their places. 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 under the estimated parameters, we can obtain an *estimated solution* $x^*(\hat{\theta})$. When constraints contain unknown parameters, a big challenge is that the feasible region is only approximated at solving time, and thus the estimated solution may be infeasible under the true parameters. Fortunately, some applications allow us to correct an infeasible solution into a feasible one, after the true parameters are revealed. Under these applications, Predict+Optimize can use a novel error measurement, called *Post-hoc Regret* [9], to evaluate the quality of the estimated parameters $\hat{\theta}$. The correction process 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. Although some scenarios may allow for post-hoc correct of an estimated solution, some penalties may incur from such correction. 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. The choice of both the correction function and the penalty function are problem and application-specific.

With respect to the corrected solution $x^*_{corr}(\hat{\theta}, \theta)$ and penalty function Pen, we are now ready to define the *Post-hoc Regret*. The post-hoc regret contains two parts, one is the objective difference between the *true optimal solution* $x^*(\theta)$ and the corrected solution $x^*_{corr}(\hat{\theta}, \theta)$ under the true parameters θ, another one is the penalty that changing from the estimated solution $x^*(\hat{\theta})$ to the corrected solution $x^*_{corr}(\hat{\theta}, \theta)$ will incur. The *Post-hoc Regret* $PReg(\hat{\theta}, \theta)$ can be formally defined as:

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

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

When only the objective contains unknown parameters, Post-hoc Regret degenerates into the *Regret* function [5], which compares the difference between the objective value of the *true optimal solution* $x^*(\theta)$ and the *estimated solution* $x^*(\hat{\theta})$ under true parameters θ. The regret function can be defined as: $Reg(\hat{\theta}, \theta) = obj(x^*(\hat{\theta}), \theta) - obj(x^*(\theta), \theta)$, where $obj(x^*(\hat{\theta}), \theta)$ is the *estimated optimal value*.

Following the empirical risk minimization principle, Hu *et al.* [9] 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:

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

For discrete OPs and linear programs, the Post-hoc Regret is non-differentiable. Hence, traditional machine learning algorithms that rely on gradients are not applicable.

Branch & Learn (B&L) [10] is a Predict+Optimize framework for Para-OPs with unknown parameters only in the objective, which can compute the Regret exactly. B&L can handle optimization problems solvable with a recursive algorithm (under some restrictions). In B&L, Hu *et al.* study the class $\mathcal{F}$ of linear prediction functions and represent the solution structure of a Para-OP using (continuous) piecewise linear functions. A *piecewise linear function* h is a real-valued function defined on a finite set of (closed) intervals $\mathbb{I}(h)$ partitioning $\mathbb{R}$. Each interval $I \in \mathbb{I}(h)$ is associated with a linear function $h[I]$ of the form $h[I](r) = a_I r + b_I$, and the value of $h(r)$ for a real number $r \in \mathbb{R}$ is given by $h[I](r)$ where $r \in I$. An algebra can be canonically defined on piecewise linear functions [18]. For piecewise linear functions h and g, we define pointwise addition as $(h + g)(r) = h(r) + g(r)$ for all $r \in \mathbb{R}$. Pointwise subtraction, max/min and scalar products are similarly defined. All five operations can be computed efficiently by iterating over intervals of the operands [4].

This work is extended from B&L and in the rest of the paper, following the assumption in B&L, we assume that the prediction function f is a *linear mapping of the form* $f(A) = A\alpha$ for some m-dimensional vector of *coefficients* $\alpha \in \mathbb{R}^m$.

3   Branch & Learn with Post-hoc Correction

In this section, we extend B&L to cater for unknown parameters also in constraints, and call the extended framework *Branch & Learn with post-hoc correction (B&L-C)*. Post-hoc regret is used as the error metric.

To solve Problem 2, following the approach of Hu *et al.* [10], we update coefficients α of f iteratively via coordinate descent (Algorithm 1). The algorithm starts with an arbitrary initialization of α, and updates each coefficient in a round-robin fashion. Each iteration (Lines 3–11) contains four functions. `Construct` constructs a Para-OP as a function of the *free coefficient*, fixing the other coefficients in α, with an initial domain I_0. `Convert` returns a piecewise function of the free coefficient from the Para-OP, and each interval of the function corresponds to one or a set of estimated solution(s). `Correct` takes the returned piecewise function from `Convert` and the true parameters as inputs. Then makes the post-hoc correction, and returns a piecewise function of

---

Algorithm 1: Branch & Learn with Post-hoc Correction

Input: A Para-COP $P(\theta)$ and a training data set $\{(A^1, \theta^1), \dots, (A^n, \theta^n)\}$
Output: a coefficient vector $\alpha \in \mathbb{R}^m$

1   Initialize α arbitrarily and $k \leftarrow 0$;
2   **while** *not converged* $\wedge$ *resources remain* **do**
3      $k \leftarrow (k \bmod m) + 1$;
4   Initialize L to be the zero constant function;
5   **for** $i \in [1, 2, \dots, n]$ **do**
6          $(P_\gamma^i, I_0) \leftarrow \texttt{Construct}(P(\theta), k, A^i)$;
7          $E^i(\gamma) \leftarrow \texttt{Convert}(P_\gamma^i, I_0)$;
8          $C^i(\gamma) \leftarrow \texttt{Correct}(E^i, \theta^i, I_0)$;
9          $L^i(\gamma) \leftarrow \texttt{Evaluate}(E^i, C^i, \theta^i, I_0)$;
10         $L(\gamma) \leftarrow L(\gamma) + L^i(\gamma)$;
11      $\alpha_k \leftarrow \arg\min_{\gamma \in \mathbb{R}} L(\gamma)^*$;
12   return α;

---

the free coefficient from the Para-OP. Each interval of the function corresponds to a data structure representing one or a set of corrected solution(s). $\texttt{Evaluate}$ takes the two returned functions from $\texttt{Convert}$ and $\texttt{Correct}$, and the true parameters as inputs. Then computes the corrected optimal value and the penalty, and obtains the post-hoc regret as a piecewise function of the free coefficient.

Let us describe lines 3–11 in Algorithm 1 in more detail. In each iteration (Lines 3–11), a coefficient α_k is updated. Iterating over index $k \in \{1, \dots, m\}$, we replace α_k in α with a variable $\gamma \in \mathbb{R}$ by constructing $\alpha + (\gamma - \alpha_k)e_k$, where e_k is a unit vector for coordinate k. In Lines 5–11, we wish to update α_k as:

$$\alpha_k \leftarrow \arg\min_{\gamma \in \mathbb{R}} \sum_{i=1}^n PReg(A^i e_k \gamma + A^i(\alpha - \alpha_k e_k), \theta^i)$$

For notational convenience, let $a^i = A^i e_k \in \mathbb{R}^m$ and $b^i = A^i(\alpha - \alpha_k e_k) \in \mathbb{R}^m$, which are vectors independent of the free variable γ.

$\texttt{Construct}$ synthesizes the Para-OP

$$P_\gamma^i \equiv x^*(a^i \gamma + b^i) = \arg\min_x obj(x, a^i \gamma + b^i) \text{ s.t. } C(x, a^i \gamma + b^i)$$

Sometimes, the Para-OP can also have an initial domain $I_0 \neq \mathbb{R}$ for γ.

$\texttt{Convert}$ takes P_γ^i to create a function E^i mapping γ to the estimated objective $E^i(\gamma) = obj(x^*(a^i \gamma + b^i), a^i \gamma + b^i)$. Associated with each interval $I \in \mathbb{I}(E^i(\gamma))$, a linear function maps γ to the objective computed with the estimated parameters $a^i \gamma + b^i$. When the unknown parameters only appear in the objective, the estimated solution $x^*(a^i \gamma + b^i)$ remains the same in each interval I [3,4], i.e., each interval corresponds to one estimated solution. However, when the unknown parameters appear in constraints, the estimated solution may not remain the same in each interval I. If the estimated solution changes in one interval, one interval corresponds to a set of estimated solutions.

Whether the estimated solution will remain the same in each interval depends on the optimization problem and the positions of the unknown parameters. In Sect. 4, we show two examples that the estimated solution will remain the same in each interval and one example that the estimated solution will change in each interval.

`Correct` implements the correction function in the post-hoc regret. It takes the returned piecewise function $E^i(\gamma)$ from `Convert` and the true parameters θ^i as inputs. For each interval $I \in \mathbb{I}(E^i(\gamma))$, one or a set of estimated solution(s) $x^*(a^i\gamma + b^i)$ can be obtained. `Correct` makes the post-hoc correction with the true parameters θ^i, and creates a function C^i mapping γ to the corrected optimal value $C^i(\gamma) = obj(x^*_{corr}(a^i\gamma + b^i, \theta^i), \theta^i)$. Each interval $I \in \mathbb{I}(C^i(\gamma))$ corresponds to one or a set of corrected solution(s) $x^*_{corr}(a^i\gamma + b^i, \theta^i)$. Whether the corrected solution will remain the same in each interval I depends on the correction function. Besides, the form of the returned function $C^i(\gamma)$ depends on the correction function.

`Evaluate` takes the returned function from `Convert`, the returned function from `Correct`, and the true parameters as inputs. It computes the corrected optimal value and the penalty, and obtains the post-hoc regret L^i for each γ, i.e.

$$
\begin{aligned}
L^i[I] = PReg(a^i\gamma + b^i, \theta^i) = {}& obj(x^*_{corr}(a^i\gamma + b^i, \theta^i), \theta^i) - obj(x^*(\theta^i), \theta^i) \\
& + Pen(x^*(a^i\gamma + b^i) \rightarrow x^*_{corr}(a^i\gamma + b^i, \theta))
\end{aligned}
$$

When the unknown parameters only appear in objectives, the post-hoc regret function L^i returned from `Evaluate` is always a piecewise constant function of the free coefficient [3,4,10]. It is straightforward to compute the sum of two piecewise constant functions and update the coefficient (Lines 10–11 in Algorithm 1). However, when the unknown parameters appear in constraints, the form of the post-hoc regret function L^i returned from `Evaluate` depends on the correction function and the penalty function, which are both problem and application specific. Under different scenarios, the post-hoc regret function may even be a piecewise nonlinear function, which leads to a technical obstacle: how to sum up two piecewise nonlinear functions and find the minimum of the resulted function efficiently. We will discuss this obstacle in Sect. 4.

While coordinate descent is a standard technique, a challenge of using this framework is how to construct `Convert` for an algorithm. B&L presents a standard template for recursive algorithms and shows how to cleanly adapt a recursive algorithm to `Convert`. We use their template to construct `Convert` here. Therefore, the proposed B&L-C framework has the same restrictions on the optimization problems as the B&L.

4   Case Studies

4.1   Maximum Flow with Unknown Edge Capacities

We first demonstrate, using the example of the maximum flow problem (MFP), how our framework can solve problems solvable by a state-of-the-art approximation method (IntOpt-C) [9]. The problem aims to find the largest possible flow sent from a source s to a terminal t in a directed graph, under the constraints that the flow sent on each edge cannot exceed the edge capacity.

Using the template proposed in B&L [10], we adapt the Edmonds-Karp algorithm [19] to `Convert`, which recursively finds an unblocking path with remaining

capacity and sends a flow such that at least one edge along the path is saturated. The estimated solution $x^*(a^i\gamma + b^i)$ of MFP is the flows sent through each path and therefore will change with the capacities of saturated edges. In each interval of $E^i(\gamma)$ returned by `Convert`, the saturated edges remain the same, but the estimated solution will change when γ changes. When the edge capacities are unknown, we need to consider a case where the flow computed with the estimated capacities might exceed the true capacities of some edges. We consider two possible correction functions:

- **Correction Function A**: given an infeasible estimated solution x^*, find the largest $\lambda \in [0, 1]$ such that λx^* satisfies the constraints under the true parameters.
  Note that Correction Function A is the same as the one used in IntOpt-C [9], which is designed for packing linear programs. By using the same correction function, we can investigate the performance difference between B&L-C and IntOpt-C.
- **Correction Function B**: re-compute the blocking flows of the chosen paths in the infeasible estimated solution with the true capacities, and then augment the paths one by one with the re-computed blocking flows. The ordering of path augmentation is important but computing the best order requires $O(n!)$ time. We can adopt an approximate method: the paths are augmented according to the order of the path augmentation of the Edmonds-Karp algorithm.

Using Correction Function B, augmenting the chosen paths by their blocking flows one by one may lead to a situation that, since some edges are shared by several paths, they may be blocked before some paths are used. Thus the blocking flows of some chosen paths in the estimated solution may be zero and these chosen paths are wasted. Therefore, we propose a penalty function:

- **Penalty Function I**: whenever a chosen path in the estimated solution is wasted, deduct K units of flow.

Penalty Function I is not needed if Correction Function A is used, since the true capacities are all positive, λ will not be zero in this problem and no chosen paths in the estimated solution will be wasted.

Using Correction Function A, the corrected solution $x^*_{corr}(a^i\gamma + b^i, \theta^i)$ will not remain the same in each interval $I \in \mathbb{I}(C^i(\gamma))$ either, where $C^i(\gamma)$ is the piecewise rational linear function returned from `Convert`. Since the true optimal value $obj(x^*(\theta^i), \theta^i)$ is a constant value, the post-hoc regret function $L(\gamma)$ returned from `Evaluate` is a piecewise rational linear function. This will lead to the technical obstacle mentioned in Sect. 3: how to sum up two piecewise rational linear functions and find the minimum of the resulting piecewise rational linear function efficiently. In this work, we deal with this obstacle by using grid search.

Using Correction Function B, the corrected solution $x^*_{corr}(a^i\gamma + b^i, \theta^i)$ remains the same in each interval $I \in \mathbb{I}(C^i(\gamma))$, where $C^i(\gamma)$ is the piecewise constant function returned from `Convert`. Using Penalty Function I, $Pen(x^*(a^i\gamma + b^i) \rightarrow x^*_{corr}(a^i\gamma + b^i, \theta^i))$ is also a piecewise constant function. Therefore, the post-hoc regret function $L(\gamma)$ returned from `Evaluate` is a piecewise constant function, and we can easily sum up two piecewise constant functions and minimizes $L(\gamma)$ in Lines 9 and 10 respectively in Algorithm 1.

4.2 0-1 Knapsack with Unknown Weights

In the second example, we showcase our framework on a packing integer programming problem, the 0-1 knapsack problem, which can be handled by our framework straightforwardly but not by IntOpt-C. Given a set of items, each with a weight w_i and a value v_i, and a knapsack with a maximum capacity C. The aim is to maximize the total value of the selected items under the constraint that the total weight of the selected items is less than or equal to the maximum capacity. Using the template proposed in B&L [10], we adapt the branching algorithm for the 0-1 knapsack problem to `Convert`. The estimated solution $x^*(a^i\gamma + b^i)$ is a set of the selected items. In each interval of $E^i(\gamma)$ returned by `Convert`, the set of the selected items, i.e., the estimated solution, remains the same. When the weights are unknown, we need to consider a case where the items are selected with the estimated weights, but the total true weights might exceed the capacity. We propose three correction functions here:

– **Correction Function A**: remove the selected items in the estimated solution one by one in increasing order of the ratios of value/weight until the capacity is sufficient.
– **Correction Function B**: remove the selected items in the estimated solution one by one in decreasing order of the weights until the capacity is sufficient.
– **Correction Function C**: remove all the selected items in the estimated solution when it is infeasible.

Removing selected items from the knapsack may incur some removal fees, which are formulated as the penalty function here. We consider two possible penalty functions:

– **Penalty Function I**: when the ith item is removed from the estimated solution, $\sigma_i v_i$ units of value is deducted, where $\sigma \geq 0$ is a non-negative tunable vector.
– **Penalty Function II**: whenever a selected item in the estimated solution is removed, K units of value is deducted, where K is a constant.

Since the solution set is discrete and finite, the estimated solution $x^*(a^i\gamma + b^i)$ remains the same in each interval $I \in \mathbb{I}(E^i(\gamma))$, where $E^i(\gamma)$ is the piecewise linear function returned from `Convert`. Using the above three correction functions, the corrected solution $x^*_{corr}(a^i\gamma + b^i, \theta^i)$ remains the same in each interval $I \in \mathbb{I}(C^i(\gamma))$, where $C^i(\gamma)$ is the piecewise constant function returned from `Convert`. Using the above two penalty functions, $Pen(x^*(a^i\gamma + b^i) \rightarrow x^*_{corr}(a^i\gamma + b^i, \theta^i))$ is also a piecewise constant function. Therefore, the post-hoc regret function $L(\gamma)$ returned from `Evaluate` is a piecewise constant function, and we can easily sum up two piecewise constant functions and minimizes $L(\gamma)$ in Lines 9 and 10 respectively in Algorithm 1.

4.3 Minimum Cost Vertex Cover with Unknown Costs and Edge Values

Our last example is a variant of the minimum cost vertex cover (MCVC) problem, where we show how to apply our framework to an optimization problem that has unknown parameters in both the objective and the constraints. This problem is also not solvable by IntOpt-C. Given a graph $G = (V, E)$, there is an associated *cost* $c \in \mathbb{R}^{|V|}$ denoting the cost of picking each vertex, as well as edge values $\ell \in \mathbb{R}^{|E|}$, one real value for each

edge. Both the costs and edge values are unknown parameters. The goal is to pick a subset of vertices, minimizing the total cost, subject to the constraint that for all edges *except the one with the smallest edge value*, the edge needs to be covered, namely at least one of the two vertices on the edge needs to be picked. This problem is relevant in applications such as building public facilities. Consider, for example, the graph being a road network with edge values being traffic flow, and we wish to build speed cameras at intersections with minimum cost, while covering all the roads except the one with the least traffic.

Using the template proposed in B&L [10], we adapt the branching algorithm for the MCVC to Convert. The estimated solution $x^*(a^i\gamma + b^i)$ is a set of the picked vertices. In each interval of $E^i(\gamma)$ returned by Convert, the set of the picked vertices, i.e., the estimated solution, remains the same. When the edge values are unknown, the estimated edge values might cause an edge to be wrongly removed. The selected vertices might not cover all the edges that need to be covered. We thus propose one correction function:

– **Correction Function A**: if there is an edge not covered by the selected vertices, add both of the edge endpoints to the selection.

Since the solution set is discrete and finite, the estimated solution $x^*(a^i\gamma + b^i)$ remains the same in each interval $I \in \mathbb{I}(E^i(\gamma))$, where $E^i(\gamma)$ is the piecewise linear function returned from Convert. Using Correction Function A, the corrected solution $x^*_{corr}(a^i\gamma + b^i, \theta^i)$ remains the same in each interval $I \in \mathbb{I}(C^i(\gamma))$, where $C^i(\gamma)$ is the piecewise constant function returned from Convert. Since there is no penalty function in this example, $Pen(x^*(a^i\gamma + b^i) \rightarrow x^*_{corr}(a^i\gamma + b^i, \theta^i)) = 0$ is a constant function. Therefore, the post-hoc regret function $L(\gamma)$ returned from Evaluate is a piecewise constant function, and we can easily sum up two piecewise constant functions and minimizes $L(\gamma)$ in Lines 9 and 10 respectively in Algorithm 1.

5   Experimental Evaluation

In this section, we evaluate the proposed B&L-C framework and the post-hoc regret function on the three optimization problems mentioned in Sect. 4. We compare the proposed framework (B&L-C) with 7 different methods: the B&L framework [10], a state-of-the-art approximation method (IntOpt-C) [9], and 5 classical regression methods including linear regression (LR), k-nearest neighbors (k-NN), classification and regression tree (CART), random forest (RF) and neural network (NN) [6]. Below we briefly discuss the experiment setting of each problem:

MFP with Unknown Edge Capacities. Our aim is to use this problem to compare the proposed B&L-C framework with IntOpt-C [9]. Therefore, we use the same dataset and follow the experiment setting in the work of IntOpt-C. The real-life dataset [17] is used on three real-life graphs: POLSKA [15], with 12 vertices and 18 edges, USANet [13], with 24 vertices and 43 edges, and GÉANT [12], with 40 vertices and 61 edges. In this dataset, each unknown edge capacity is related to 8 features. Following the setting in IntOpt-C, we divide the dataset into two sets: training and test. For experiments on

POLSKA and USANet, 610 instances are used for training and 179 instances for testing the model performance, while for experiments on GÉANT, 490 instances are used for training and 130 instances for testing the model performance.

0-1 Knapsack with Unknown Weights. In this experiment, each instance consists of 10 items. The weights W will be predicted from data, while values V and capacity C are given. Given that we are unable to find datasets specifically for the 0-1 knapsack problem, we follow the experimental approach in the previous works of P+O [10,14] and use real data from a different problem (the ICON scheduling competition) [17] as numerical values required for our experiment instances. In this dataset, each unknown weight is related to 8 features. We use a 70%/30% training/testing data split: 210 instances are used for training and 90 instances for testing the model performance.

We generate the values following the generation method proposed by Pisinger [16], which is widely used to generate knapsack data [1,11]. Three groups of values, which are uncorrelated, weakly correlated, and almost strongly correlated with the weights, are considered. Suppose the value of the ith item is v_i, the weight of the ith item is w_i. These 3 groups of values are generated as: 1) uncorrelated: v_i is randomly chosen in $[1, R]$, 2) weakly correlated: v_i is randomly chosen in $[\max\{1, w_i - R/10\}, w_i + R/10]$, 3) almost strongly correlated: v_i is randomly chosen in $[w_i + R/10 - R/500, w_i + R/10 + R/500]$, where R is set to be 500 since the weights in the dataset are around 40 to 60. Since the average total weight of each instance is around 400, we conduct experiments on the 0-1 knapsack problem with 100, 200, and 300 capacities. The σ in Penalty Function I is set to be 0.1 and K in Penalty Function II is set to be 500.

MCVC with Unknown Costs and Edge Values. Since the MCVC is an NP-hard problem, we conduct experiments on two small graphs from the Survivable Network Design Library [15]: ABILENE, with 12 vertices and 15 edges, and PDH, with 11 vertices and 34 edges. Given that we are unable to find datasets specifically for the MCVC, we use the same real data from the ICON scheduling competition [17] as numerical values required for our experiment instances. We use a 70%/30% training/testing data split: 210 instances are used for training and 90 instances for testing the model performance.

5.1   B&L-C Versus IntOpt-C

In the first experiment, we compare our exact method (B&L-C) against an approximation method (IntOpt-C) in terms of solution quality and runtime. We conduct experiments on the MFP with unknown edge capacities, which can be solved by both the approximation method IntOpt-C [9] and the proposed exact method B&L-C. Following the experiment setting in IntOpt-C [9], Correction Function A is used and there is no penalty function here. We run 10 simulations on each graph and compare the solution quality and the runtime of each method.

Table 1 reports the mean post-hoc regrets and standard deviations for each approach. At the bottom of the table, we also report the *average True Optimal Values* (TOV) for reference. Note that B&L performs training with the regret but the testing is

Table 1. Mean post-hoc regrets and standard deviations for the MFP with unknown edge capacities using Correction Function A and no penalty function.

PReg	POLSKA	USANet	GÉANT
B&L-C	**9.07 ± 0.67**	**14.44 ± 1.12**	**10.18 ± 1.02**
B&L	17.01±2.00	21.79±1.53	17.04±2.11
IntOpt-C	10.00±0.67	16.64±1.34	10.84±1.10
Ridge	11.20±0.73	19.52±1.16	12.47±1.14
k-NN	14.39±0.83	22.89±1.58	15.13±1.08
CART	16.65±1.06	24.15±1.51	17.01±1.59
RF	12.30±0.90	22.27±1.34	12.52±1.19
NN	12.18±1.08	18.62±1.23	12.05±1.13
TOV	88.66±1.10	96.22±1.38	98.71±1.98

Table 2. Average runtimes (in seconds) for the MFP with unknown edge capacities.

Runtime(s)	POLSKA	USANet	GÉANT
B&L-C	66.54	411.67	48.32
B&L	40.30	288.43	29.90
IntOpt-C	18.65	132.22	15.48

with the post-hoc regret, while B&L-C and IntOpt-C use post-hoc regret in both training and testing. The results show that B&L-C always achieves the best performance, while IntOpt-C achieves the second-best performance in all cases. Compared with IntOpt-C, B&L-C obtains 9.29% smaller regret on POLSKA, 13.20% smaller regret on USANet, and 6.08% smaller regret on GÉANT. Considering the relative error, B&L-C achieves 10.23%, 15.01%, and 10.31% relative error on POLSKA, USANet, and GÉANT respectively. We also observe that using regret as the loss function, B&L does not have a better performance than the classical regression methods when the unknown parameters appear in constraints. Table 2 shows the average runtimes of B&L-C using Correction Function A, B&L, and IntOpt-C. Here, the runtime refers to the training time of the prediction model. The results show that the runtime of B&L-C using Correction Function A is larger than that of IntOpt-C, while the runtime of B&L is not that much larger than that of IntOpt-C. The reason is that, in this problem, the post-hoc regret function using Correction Function A is a piecewise rational linear function, while the regret function is a piecewise linear function. To compute the minimum of a piecewise rational linear function, grid search is used and is quite time-consuming. But the minimum of a piecewise linear function can be computed easily and grid search is not needed.

In conclusion, we observe that B&L-C can achieve much better solution quality but need longer runtime than IntOpt-C.

5.2 Post-hoc Regret on More General Problems

IntOpt-C is only applicable for packing and covering linear programming problems. We investigate the performance of post-hoc regret on two integer programming problems: 0–1 knapsack with unknown weights, and a variant of MCVC with unknown costs and edge values, both of which cannot be solved by IntOpt-C.

0-1 Knapsack with Unknown Weights. In this experiment, we use Correction Function A and Penalty Function I as an example, to show that the B&L-C framework can deal with 0-1 knapsack with unknown weights. Table 3 reports the solution qualities for

Table 3. Mean post-hoc regrets and standard deviations for the 0-1 knapsack problem with unknown weights and 3 groups of values using Correction Function A with Penalty Function I.

Preg		Cap=100	Cap=200	Cap=300
Uncorrelated	B&L-C	**112.66 ± 19.70**	**91.45 ± 9.43**	**53.90 ± 8.33**
	B&L	165.57±18.35	199.02±48.07	123.06±25.26
	Ridge	175.05±24.22	201.00±24.52	145.73±23.04
	k-NN	188.73±22.95	239.83±30.24	189.11±37.25
	CART	185.33±22.53	215.83±24.14	174.44±22.06
	RF	179.34±21.99	213.53±23.63	159.27±29.35
	NN	159.75±40.05	172.87±74.31	120.21±70.43
	TOV	942.76±37.06	1712.67±41.63	2174.50±42.63
Weakly Correlated	B&L-C	**20.81 ± 2.61**	**18.88 ± 1.56**	**12.91 ± 1.16**
	B&L	31.73±6.56	36.45±6.36	25.31±5.65
	Ridge	28.98±3.39	36.00±3.81	26.81±2.40
	k-NN	31.22±3.93	42.00±4.95	34.32±5.08
	CART	31.61±4.68	38.80±3.72	31.83±3.49
	RF	30.04±4.08	38.05±4.94	29.34±3.51
	NN	27.08±5.11	31.18±11.57	22.12±9.80
	TOV	165.96±4.62	309.28±4.69	397.94±5.77
Almost Strongly Correlated	B&L-C	**44.62 ± 4.01**	**59.77 ± 3.91**	**43.93 ± 3.41**
	B&L	51.95±5.17	84.11±13.64	97.72±22.87
	Ridge	49.98±3.47	82.92±8.01	96.53±10.61
	k-NN	51.73±3.38	87.15±7.68	110.98±12.66
	CART	52.40±4.48	81.02±6.73	101.58±8.95
	RF	50.75±4.29	80.72±9.87	98.31±12.44
	NN	50.11±3.63	75.34±18.42	85.56±32.39
	TOV	209.40±5.92	441.17±9.05	654.16±10.99

each approach across 10 runs on 0-1 knapsack problem with unknown weights and 3 different groups of values (uncorrelated, weakly correlated, and almost strongly correlated).

As shown in Table 3, B&L-C has the smallest mean post-hoc regrets in all cases. In the experiments on uncorrelated values, B&L-C obtains at least 29.48%, 47.10%, and 55.16% when the capacity is 100, 200, and 300 respectively. In the experiments on weakly correlated values, B&L-C obtains at least 23.13%, 39.45%, and 41.61% when the capacity is 100, 200, and 300 respectively. In the experiments on almost strongly correlated values, B&L-C obtains at least 10.72%, 20.67%, and 48.65% when the capacity is 100, 200, and 300 respectively. These results indicate that using Correction Function A, when the capacity grows larger, the advantage of B&L-C is more evident.

We also report the relative errors in these experiments. In the experiments on uncorrelated values, B&L-C achieves 11.95%, 5.34%, and 2.48% relative error when the capacity is 100, 200, and 300 respectively. In the experiments on weakly correlated values, B&L-C achieves 12.54%, 6.10%, and 3.25% relative error when the capacity is 100, 200, and 300 respectively. In the experiments on almost strongly correlated values, B&L-C achieves 21.31%, 13.55%, and 6.72% relative error when the capacity is 100, 200, and 300 respectively. These results indicate that when the values are more correlative with the weights, the relative error is larger. Besides, using Correction Function A, the relative error becomes smaller when the capacity grows larger.

Table 4. Mean post-hoc regrets and standard deviations for the MCVC with unknown costs and edge values.

PReg	ABILENE	PDH
B&L-C	**11.83 ± 2.79**	**55.94 ± 8.46**
B&L	15.26±3.56	73.6±8.55
Ridge	19.3±3.05	65.23±6.76
k-NN	33.08±4.55	70.52±6.72
CART	28.6±5.67	66.03±7.39
RF	27.91±4.25	65.29±8.01
NN	14.14±2.42	70.65±5.69
TOV	275.33±5.43	491.18±12.75

Table 5. Mean post-hoc regrets and standard deviations for the MFP with unknown edge capacities using Correction Function B without/with Penalty Function I.

PReg	Correction Function B			Correction Function B & Penalty Function I		
	POLSKA	USANet	GÉANT	POLSKA	USANet	GÉANT
B&L-C	**1.41 ± 0.26**	**4.49 ± 0.67**	**1.03 ± 0.24**	**6.14 ± 1.08**	**16.89 ± 1.10**	**2.04 ± 0.27**
B&L	1.51±0.30	10.09±1.37	5.82±1.68	8.59±0.45	20.89±1.55	7.00±2.01
Ridge	1.52±0.30	7.11±0.88	1.20±0.34	8.54±0.47	18.72±1.10	2.47±0.27
k-NN	2.42±0.36	8.03±0.86	1.47±0.52	8.22±0.53	23.57±1.04	3.10±0.46
CART	3.29±0.69	10.84±1.28	1.75±0.52	8.13±0.88	28.79±1.96	3.71±0.82
RF	1.81±0.33	8.72±1.19	1.27±0.41	7.27±0.52	21.30±1.36	2.54±0.44
NN	1.83±0.29	5.52±0.73	1.22±0.37	8.95±0.44	18.98±1.02	2.45±0.54
TOV	88.66±1.10	96.22±1.38	98.71±1.98	88.66±1.10	96.22±1.38	98.71±1.98

MCVC with Unknown Costs and Edge Values. Table 4 shows the solution qualities for each approach across 10 runs on the MCVC experiment. B&L-C achieves the best performance in both of the two graphs. B&L-C obtains at 16.13%-64.24% smaller post-hoc regret in ABILENE, and 14.24%-23.99% in PDH. Considering the relative error, B&L-C achieves 4.30% relative error in ABILENE, and 11.39% relative error in PDH.

5.3   Different Combinations of Correction Functions and Penalty Functions

The correction function and the penalty function are problem and application-specific. Even in the same problem but different scenarios, the correction function and the

penalty function could be different. Here, we try out different combinations of correction functions and penalty functions in the same problem to provide insights for defining the appropriate post-hoc regret.

MFP with Unknown Edge Capacities. We conduct experiments on the MFP using Correction Function B with/without Penalty Function I. The experiment results are shown in Table 5. Since the correction function is changed, the gradient of the post-hoc regret with respect to edge capacities is also changed and thus IntOpt-C cannot be used.

First, to compare the performance of post-hoc regret with different correction functions, we compare the results of using Correction Function A and B in Tables 1 and 5 respectively. Table 5 shows that B&L-C always achieves the best performance. Compared with other methods, B&L-C obtains at least 7.03% smaller regret on POLSKA, 18.50% smaller regret on USANet, and 14.19% smaller regret on GÉANT. Considering the relative error, B&L-C achieves 1.59%, 4.67%, and 1.04% relative error on POLSKA, USANet, and GÉANT respectively. The results show that B&L-C using Correction Function B can achieve smaller mean post-hoc regret than B&L-C using Correction Function A, which indicates that using Correction Function B is more suitable.

Second, we investigate the performance of post-hoc regret when using penalty functions on the MFP. Due to the page limit, we opted to present only one value of K, which is the units of flow to be deducted. Here, K is set to be 10. For experiment results and analysis on $K = \{30, 50\}$, please refer to the extended version on arXiv with the same title. Results in Table 5 show that when using Correction Function B & Penalty Function I, B&L-C also achieves the best performance on all of the three graphs. Since the penalty term exists and is always non-negative, the mean post-hoc regrets here are larger than the mean post-hoc regrets of using Correction Function B without a penalty function. Compared with other methods, B&L-C obtains at least 15.46% smaller regret on POLSKA, 9.79% smaller regret on USANet, and 16.47% smaller regret on GÉANT. Considering the relative error, B&L-C achieves 6.93%, 17.53%, and 2.07% relative error on POLSKA, USANet, and GÉANT respectively.

Table 6. Mean post-hoc regrets and standard deviations for the 0-1 knapsack problem with weakly correlated values using different correction functions with Penalty Function I.

PReg	Corection Function B			Correction Function C		
	Cap=100	Cap=200	Cap=300	Cap=100	Cap=200	Cap=300
B&L-C	**24.25 ± 2.59**	**32.87 ± 3.55**	**31.70 ± 1.89**	**53.48 ± 5.27**	**116.56 ± 14.06**	**130.79 ± 9.80**
B&L	32.59±6.32	40.24±5.31	36.35±5.72	58.67±7.41	124.67±23.56	174.70±26.32
Ridge	29.88±3.22	39.62±3.60	36.27±2.33	55.31±7.29	124.23±14.73	160.04±20.54
k-NN	32.96±3.72	45.98±4.16	41.73±4.11	62.72±6.63	123.12±14.03	153.48±21.53
CART	35.14±4.71	45.79±4.37	42.28±2.79	70.97±11.57	141.80±19.47	171.11±23.31
RF	32.83±3.97	43.51±4.91	37.30±2.37	64.01±11.51	126.48±22.04	158.52±26.99
NN	28.60±4.58	39.05±8.02	38.52±11.05	74.05±28.73	163.71±64.50	213.17±80.20
TOV	165.96±4.62	309.28±4.69	397.94±5.77	165.96±4.62	309.28±4.69	397.94±5.77

0-1 Knapsack with Unknown Weights. Experiments on weakly correlated values are conducted as examples to show the performance difference of post-hoc regret when using different correction functions and penalty functions in the 0-1 knapsack problem.

First, to compare the performance when using different correction functions, we fix the penalty function as Penalty Function I. Experiments on 0-1 knapsack using Correction Function A & Penalty Function I are conducted in Sect. 5.1, and we only conduct experiments using Correction Function B and C here. The results are shown in Table 6. B&L-C outperforms other approaches. Besides, the mean post-hoc regrets achieved by B&L-C using Correction Functions B and C are both larger than that achieved by B&L-C using Correction Function A, which indicates that when using Penalty Function I, Correction Function A is more suitable than Correction Functions B and C.

Second, to compare the performance when using different penalty functions, we conduct experiments using Penalty Function II and Correction Function A, B, and C respectively. Experiment results are shown in Table 7. As Table 7 shows, B&L-C outperforms other approaches. The results show that when using Penalty Function II, B&L-C

Table 7. Mean post-hoc regrets and standard deviations for the 0-1 knapsack problem with unknown weights and weakly correlated values using Penalty Function II.

PReg		Cap=100	Cap=200	Cap=300
Correction Function A	B&L-C	**127.13 ± 27.52**	**202.99 ± 48.62**	**231.32 ± 58.74**
	B&L	280.65±73.44	482.37±134.13	668.78±167.01
	Ridge	263.55±68.79	439.81±101.30	608.07±121.73
	k-NN	323.26±67.78	445.81±119.63	566.84±159.66
	CART	566.15±188.48	687.51±184.25	697.52±176.48
	RF	378.68±131.24	536.21±170.43	613.09±153.58
	NN	357.23±121.00	614.05±191.50	608.49±200.71
Correction Function B	B&L-C	**127.24 ± 27.60**	**186.84 ± 37.45**	**193.92 ± 39.52**
	B&L	281.43±73.56	491.13±158.54	726.18±175.94
	Ridge	264.37±68.94	435.33±97.59	596.34±121.07
	k-NN	323.73±66.71	437.21±115.41	556.91±159.57
	CART	563.81±187.58	676.10±180.67	685.91±176.05
	RF	380.11±±132.23	527.84±163.75	595.89±151.56
	NN	356.05±119.96	552.03±217.25	592.49±190.67
Correction Function C	B&L-C	**173.12 ± 43.50**	**255.03 ± 73.81**	**335.59 ± 82.28**
	B&L	566.25±155.93	1546.77±518.46	2736.39±628.02
	Ridge	543.03±149.36	1472.25±283.24	2328.08±430.10
	k-NN	658.57±156.24	1376.22±322.83	2069.61±493.77
	CART	1030.83±320.56	1991.15±453.46	2581.91±509.94
	RF	775.12±270.11	1623.26±483.81	2322.75±582.54
	NN	697.35±269.30	1890.08±712.55	2485.80±892.67
TOV		165.96±4.62	309.28±4.69	397.94±5.77

using Correction Function B achieves smaller mean post-hoc regret than B&L-C using Correction Function A or C. This indicates that when using Penalty Function II, Correction Function B is more suitable to be used. The reason for this phenomenon lies in the definitions of Penalty Function II and Correction Function B. In Penalty Function II, removing more items leads to a larger penalty, while Correction Function B removes the selected items in the estimated solution one by one in decreasing order of the weights, thus will remove fewer items than Correction Functions A and C. We also notice that when the capacity is 100, B&L-C using Correction Function A outperforms B&L-C using Correction Function B. We give the explanation below. Since the average weight of the items is around 40 to 60, when the capacity is 100, the number of the selected items in the estimated solution and the number of the removal items from the post-hoc correction are both very small and the latter ones in Correction Function A and B may be almost the same. Under this situation, the penalty terms $(Pen(x^*(\hat{\theta}) \rightarrow x^*_{corr}(\hat{\theta}, \theta)))$ of using Correction Functions A and B in the post-hoc regret function are almost the same, then selecting items with higher value and has larger corrected optimal value $(obj(x^*_{corr}(\hat{\theta}, \theta), \theta))$ can achieve smaller post-hoc regret. Therefore, B&L-C using Correction Function A performs better.

6   Conclusion

We propose the first exact method for Predict+Optimize with unknown parameters in both the objective and constraints. The proposed framework is an extension of Branch & Learn, a framework for problems with only unknown objectives, and can handle recursively and iteratively solvable problems. Extensive experiments are conducted to compare the proposed method with the state-of-the-art Predict+Optimize approach and investigate the performance of the post-hoc regret on more general problems. Furthermore, we empirically study different combinations of correction functions and penalty functions to gain insights for defining post-hoc regret in different scenarios.

References

1. Cappart, Q., Moisan, T., Rousseau, L.M., Prémont-Schwarz, I., Cire, A.A.: Combining reinforcement learning and constraint programming for combinatorial optimization. In: Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence, vol. 35, pp. 3677–3687 (2021)
2. Collet, M., Gotlieb, A., Lazaar, N., Carlsson, M., Marijan, D., Mossige, M.: RobTest: a CP approach to generate maximal test trajectories for industrial robots. In: Simonis, H. (ed.) CP 2020. LNCS, vol. 12333, pp. 707–723. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58475-7_41

3. Demirović, E., et al.: Predict+Optimise with ranking objectives: exhaustively learning linear functions. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, pp. 1078–1085 (2019)

4. Demirović, E., et al.: Dynamic programming for Predict+Optimise. In: Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence, pp. 1444–1451 (2020)

5. Elmachtoub, A.N., Grigas, P.: Smart predict, then optimize. Manag. Sci. **68**(1), 9–26 (2022)

6. Friedman, J., Hastie, T., Tibshirani, R.: The Elements of Statistical Learning, vol. 1, Number 10. Springer series in statistics, New York (2001). https://doi.org/10.1007/978-0-387-21606-5

7. Genc, B., O'Sullivan, B.: A two-phase constraint programming model for examination timetabling at university college cork. In: Simonis, H. (ed.) CP 2020. LNCS, vol. 12333, pp. 724–742. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58475-7_42

8. de Givry, S., Lee, J.H.M., Leung, K.L., Shum, Y.W.: Solving a judge assignment problem using conjunctions of global cost functions. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 797–812. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10428-7_57

9. Hu, X., Lee, J.C., Lee, J.H.: Predict+Optimize for packing and covering LPs with unknown parameters in constraints. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (2022)

10. Hu, X., Lee, J.C., Lee, J.H., Zhong, A.Z.: Branch & learn for recursively and iteratively solvable problems in Predict+Optimize. Adv. Neural Inf. Process. Syst. **35** (2022)

11. Li, D., et al.: A novel method to solve neural knapsack problems. In: Proceedings of the Thirty-Eighth International Conference on Machine Learning, pp. 6414–6424. PMLR (2021)

12. LLC, M.: Geant topology map dec2018 copy (2018). https://www.geant.org/Resources/Documents/GEANT_Topology_Map_December_2018.pdf. Accessed 10 Sep 2020

13. Lucerna, D., Gatti, N., Maier, G., Pattavina, A.: On the efficiency of a game theoretic approach to sparse regenerator placement in WDM networks. In: GLOBECOM 2009–2009 IEEE Global Telecommunications Conference, pp. 1–6. IEEE (2009)

14. Mandi, J., Stuckey, P.J., Guns, T., et al.: Smart predict-and-optimize for hard combinatorial optimization problems. In: Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence, vol. 34, pp. 1603–1610 (2020)

15. Orlowski, S., Pióro, M., Tomaszewski, A., Wessäly, R.: SNDlib 1.0-Survivable network design library. In: Proceedings of the Third International Network Optimization Conference, April 2007. http://www.zib.de/orlowski/Paper/OrlowskiPioroTomaszewskiWessaely2007-SNDlib-INOC.pdf.gz, http://sndlib.zib.de, extended version accepted in Networks, 2009

16. Pisinger, D.: Where are the hard knapsack problems? Comput. Oper. Res. **32**(9), 2271–2284 (2005)

17. Simonis, H., O'Sullivan, B., Mehta, D., Hurley, B., Cauwer, M.D.: Energy-Cost Aware Scheduling/Forecasting Competition (2014). http://challenge.icon-fet.eu/sites/default/files/iconchallenge.pdf

18. Von Mohrenschildt, M.: A normal form for function rings of piecewise functions. J. Symb. Comput. **26**(5), 607–619 (1998)

19. Waissi, G.R.: Network flows: theory, algorithms, and applications (1994)

20. Wilder, B., Dilkina, B., Tambe, M.: Melding the data-decisions pipeline: decision-focused learning for combinatorial optimization. In: Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, pp. 1658–1665 (2019)