A Divide and Conquer Algorithm for Predict+Optimize with Non-Convex Problems

Ali Ugur Guler[1], Emir Demirović [2], Jeffrey Chan [3], James Bailey [1], Christopher Leckie [1], Peter J. Stuckey [4]

[1] University of Melbourne, [2] TU Delft, [3] RMIT University, [4] Monash University
aguler@student.unimelb.edu.au, E.Demirovic@tudelft.nl, jeffrey.chan@rmit.edu.au, baileyj@unimelb.edu.au, caleckie@unimelb.edu.au, peter.stuckey@monash.edu

Abstract

The predict+optimize problem combines machine learning and combinatorial optimization by predicting the problem coefficients first and then using these coefficients to solve the optimization problem. While this problem can be solved in two separate stages, recent research shows end to end models can achieve better results. This requires differentiating through a discrete combinatorial function. Models that use differentiable surrogates are prone to approximation errors, while existing exact models are limited to dynamic programming, or they do not generalize well with scarce data. In this work we propose a novel divide and conquer algorithm based on transition points to reason over exact optimization problems and predict the coefficients using the optimization loss. Moreover, our model is not limited to dynamic programming problems. We also introduce a greedy version, which achieves similar results with less computation. In comparison with other predict+optimize frameworks, we show our method outperforms existing exact frameworks and can reason over hard combinatorial problems better than surrogate methods.

Introduction

Machine Learning (**ML**) models are usually trained to make accurate predictions by minimizing errors, such as mean squared error (**MSE**). These predictions can then be used as coefficients in other decision making processes, such as a combinatorial optimization problem. The performance of these predictions is evaluated by their ability to lead to the correct decisions. Such evidence based decision making arises in many fields like transportation, healthcare, security and education (Horvitz and Mitchell 2010). Consequently, there has been growing interest in ML models for use in optimization problems. These models try to predict coefficients of the optimization problem in such a way that even if the predictions are less accurate, they lead to better decisions. This paradigm is called **predict then optimize** (Elmachtoub and Grigas 2017) or **predict+optimize (PPO)** (Demirovic et al. 2020). In this paper we propose a new framework for PPO to learn coefficients by directly reasoning over discrete combinatorial problems.

Motivation: Traditionally, predictions are the end goal of ML models. A regression model aims to minimize prediction errors. If the model makes perfect predictions, it also leads to the optimal decision. However all models are prone to errors. When there are errors in predictions, MSE does not necessarily represent the performance of the decisions (Ifrim, O'Sullivan, and Simonis 2012). A PPO framework trains parameters with the optimization objective, and can model the underlying problem better. Although PPO can improve decision making, it requires learning through hard, usually non differentiable and discrete functions.

Specialized frameworks called *semi-direct* methods can be used to approximate and minimize the optimization loss for specific problems. These frameworks depend on assumptions of the constraint configurations, and are hard to modify for different constraint sets and problems. We aim to build a general framework, a *direct method*, to understand and reason over an arbitrary optimization problem. Direct methods are not trivial for combinatorial problems. One way to differentiate through combinatorial problems is to use surrogates or relaxations. However, current surrogate models implicitly assume that the relaxed underlying problem will show convex behaviour. For a combinatorial problem this is not guaranteed and in cases where the underlying problem is not convex, gradient descent methods with the relaxed function will induce an approximation error to the optimization objective (Thapper and Živnỳ 2018).

Reasoning over a non convex function is challenging to do in polynomial time and requires more resources than a typical optimization problem. Demirovic et al. (2020) proposed a linear model to directly reason over the *exact* optimization loss of combinatorial problems defined by dynamic programming (DP). It makes use of the DP solution to find the *transition points* where small changes in predictions result in different optimal solutions and thus maps the piecewise-linear function (PWLF) of the combinatorial space. Linear models are widely used in machine learning and their simple structure is suitable for explainable AI applications and helps training the models faster. However state of the art combinatorial solvers are usually faster than the DP solution and in many cases where it is not feasible to scale a DP solution, a dedicated solver scales better. Moreover for any new problem and constraint set a new DP solution has to be formulated and this slows down the application of the framework to new problems.

We propose a novel framework, DNL, to directly reason over the exact optimization loss without solver restrictions. This model can be used with state of the art solvers. Our framework also builds upon the idea of extracting the transition points. However, unlike the DP solution, we use a numerical approach to extract transition points. We show that the predicted objective is a convex PWLF and use this knowledge to apply a divide and conquer algorithm to identify transition points. Our contributions are as follows:

- For problems with linearly parameterized coefficients and linear objectives, we show the predicted objective value is convex.
- We propose a novel direct PPO framework to reason over the exact non-convex optimization problem based on PWLF mapping without solver restrictions.
- We introduce a greedy method to map the PWLF, with similar performance to the full method and with lower complexity.
- We evaluate on 0-1 knapsack problems and a scheduling problem, and compare with the previous state of the art competing approaches. We show our divide and conquer approach achieves identical results to the DP model at knapsack benchmarks and scales better for larger problems. We also demonstrate that for hard non-convex problems DNL performs better than a non-linear exact model SPO-Forest (Elmachtoub, Liang, and McNellis 2020) and is more robust compared to state of the art surrogate models SPO-Relax (Mandi et al. 2020), QPTL (Wilder, Dilkina, and Tambe 2019), IntOpt (Mandi and Guns 2020), DP (Demirovic et al. 2020)).

Related Work

The standard approach to PPO is to separately solve the prediction problem and then the optimization problem. Combined approaches are a relatively new focus. Bengio (1997) showed that hand crafted models for financial portfolio optimization can perform better than a standard loss function. Kao, Roy, and Yan (2009) proposed using a combination of Empirical Optimization and ordinary least squares loss to improve performance for decision driven machine learning. Lim, Shanthikumar, and Vahn (2012) define relative regret in the context of portfolio optimization. Elmachtoub and Grigas (2017) define the general Smart Predict and Optimize (SPO) loss, which we call the regret in our paper. They propose a linear relaxation, SPO+ loss, to train machine learning models. Their work shows SPO+ loss can be used to achieve improved performance for constrained linear programming problems. Amos and Kolter (2017) propose to transform the optimization loss into a quadratic problem using KKT equations. Donti, Amos, and Kolter (2017) show that performance can be improved by using sequential quadratic programming (QP) to compute the new loss, and train the model with respect to it. Wilder, Dilkina, and Tambe (2019) extend the QP approach to linear programming problems. Ferber et al. (2020) extend the approach of Wilder, Dilkina, and Tambe (2019) to directly apply to mixed integer programming by using pure cutting plane methods to solve the MIP, resulting in an LP sufficient to define the MIP optimally. Mandi and Guns (2020) show logarithmic regularization terms can also be used instead of quadratic terms. Mandi et al. (2020) also show that SPO+ loss can be used as a surrogate loss for relaxations of combinatorial problems. Elmachtoub, Liang, and McNellis (2020) train decision trees with exact regret. This approach predicts problem sets as a whole and is not flexible with outlying problem sets. It requires prior knowledge and preprocessing to have reliable results. Luo et al. (2020) propose a specialised framework to optimize virtual machine provisioning. Black-box end to end frameworks are also used to differentiate and learn combinatorial problems (Bello et al. 2016), (Li, Chen, and Koltun 2018), (Niculae et al. 2018). Pogančić et al. (2020) use a black-box approach to predict optimal solutions from coefficient features. Demirovic et al. (2019a) investigate the knapsack problem from a PPO perspective and show how ranking methods can be applied to it. Similarly Demirovic et al. (2019b) introduce transition points for ranking problems. The direct inspiration of our work is that of Demirovic et al. (2020), which shows how to optimize parameters in a learning model directly using regret, as long as the optimization problem has a DP formulation. They build a PWLF using the DP formulation that identifies transition points, where the regret changes. In this work we extend this approach to arbitrary optimization problems by using numerical methods to find the transition points of the regret.

Divide and Learn

Transition points are model parameter values where small perturbations cause a change in the result of the prediction performance. Our Divide and Learn (DNL) framework first extracts these transition points within the search region for all problem sets in a batch and then compares the transition points within themselves to find the optimal model parameters. The algorithm uses coordinate descent to iteratively update all the model parameters (Alg. 1). In this section we formally define the preliminaries and transition points, and detail the technical properties of the DNL framework.

Algorithm 1: DNL algorithm

```
 1: procedure COORDINATE DESCENT($\boldsymbol{\theta}$)
 2:     For Model Parameters $\boldsymbol{\beta}$ and features $\boldsymbol{\theta}$
 3:     $\boldsymbol{\beta} \leftarrow$ initialize with regression
 4:     for all $\boldsymbol{\theta}_{batch} \in \boldsymbol{\theta}$ do
 5:         for all $\beta_i \in \boldsymbol{\beta}$ do
 6:             Define constants, $C_i$, for coordinate descent
 7:             $C_i \leftarrow \sum_{j \neq i} \theta_j * \beta_j$
 8:             $T \leftarrow$ Extract Transition Points$(\theta_i, \beta_i, C_i)$
 9:             $\beta_{opt} \leftarrow$ Compare Transition Points$(T, \theta_i, C_i)$
10:             $\beta_i \leftarrow$ step towards $\beta_{opt}$
```

Preliminaries

We now formally define the PPO problem. Given a set of objective coefficients $\boldsymbol{v}$, we define a linear objective combinatorial problem as:

$$s(\boldsymbol{v}) \equiv \boldsymbol{x} = \underset{\boldsymbol{x} \in C}{\arg\max}\, \boldsymbol{x}^T \boldsymbol{v} \tag{1}$$

where C is a finite combinatorial solution space (usually described implicitly), i.e., the constraints are known and fixed, and the variables have finite domains. The oracle $s(\boldsymbol{v})$ finds a solution that maximizes the objective value of the optimization problem given objective coefficients $\boldsymbol{v}$. In PPO settings objective coefficients are not known beforehand and forecasts, $\boldsymbol{v_p}(\boldsymbol{\theta}, \boldsymbol{\beta})$, are predicted using features $\boldsymbol{\theta}$ and parameters $\boldsymbol{\beta}$. We show the solution of the new parameterized optimization problem as

$$s(\boldsymbol{v_p}) \equiv \boldsymbol{x_p} = \underset{\boldsymbol{x} \in C}{\arg\max}\, \boldsymbol{x}^T \boldsymbol{v_p} \tag{2}$$

Regret: We measure the performance of PPO frameworks using regret. Regret is defined as the cost of making suboptimal decisions with incorrect coefficient predictions. If we define $\boldsymbol{x}^* = s(\boldsymbol{v})$ as the optimal solution of an optimization problem with *true* objective coefficients $\boldsymbol{v}$, and $\boldsymbol{x_p} = s(\boldsymbol{v_p})$ as the optimal solution of an optimization problem with predicted objective coefficients $\boldsymbol{v_p}$, then:

$$Regret : R(\boldsymbol{v_p}, \boldsymbol{v}) = \boldsymbol{x}^{*T} \boldsymbol{v} - \boldsymbol{x_p}^T \boldsymbol{v} \tag{3}$$

The true optimal value $\boldsymbol{x}^{*T}\boldsymbol{v}$ is the boundary for the predicted decisions. The optimal objective value with predicted coefficients $\boldsymbol{x_p}^T\boldsymbol{v}$ can never exceed the true optimal. Therefore the minimum value of regret is zero, and it is achieved when the predicted optimal solution, $\boldsymbol{x_p}$, is equal to a true optimal solution, $\boldsymbol{x}^*$. The **PPO problem** is to find $\boldsymbol{\beta}$ that minimizes $R(\boldsymbol{v_p}, \boldsymbol{v})$.

Transition points: Note that parameterised regret is a piecewise function. The predicted coefficients $\boldsymbol{v_p}$ can only affect regret by changing the solution of the optimization problem. These changes are not continuous and only happen at specific boundaries of the $\boldsymbol{\beta}$ values. Assume for the moment a single (unfixed) parameter β. We call the parameter values β_t where small perturbations change the optimal solution as the **transition points** of the piecewise regret function. Note that for any two points between consecutive transition points $\beta_{t_i} < \beta_1 < \beta_2 < \beta_{t_{i+1}}$, $\boldsymbol{x_p}(\beta_1) = \boldsymbol{x_p}(\beta_2)$, therefore $R(\beta_1) = R(\beta_2)$. This suggests mapping the optimization problem by identifying intervals defined by the transition points. Then we can choose any value in those intervals to train model parameters.

Example 1. *Consider a knapsack problem with three items valued at* $v_1 = 2$, $v_2 = 1$, $v_3 = 3$ *. The capacity is enough for two items. The objective coefficients (e.g., selling price of each item) are unknown but the features:* $\boldsymbol{\theta}^1 = [-1, 3]$, $\boldsymbol{\theta}^2 = [0, 1]$, $\boldsymbol{\theta}^3 = [1, 1]$ *are known. We have a linear model to predict prices from the features. Its parameters are* $\boldsymbol{\beta} = [\beta_1, \beta_2]$ *and* ${v_p}^i = \beta_1 * \theta_1^i + \beta_2 * \theta_2^i + c$ *where* $i \in \mathbb{Z}, 0 < i < 4$*. Assume* β_2 *is fixed and equals 1 and for simplicity the constant c equals 0. We can express the predicted objective coefficients with linear functions (see Fig 1a),* $v_{p1} = -\beta_1 + 3$, $v_{p2} = 1$, $v_{p3} = \beta_1 + 1$*. From Fig 1a and 1b we see that although* v_p *changes with* β_1*, there are only three different solutions (*$\boldsymbol{x} \in \{(1,1,0), (1,0,1), (0,1,1)\}$*) provided by the solver. Each solution represents a sum of linear functions of the chosen items. By combining all separate linear functions, we represent the solution space as a piecewise linear function (PWLF) seen in Fig. 1b.*

In Example 1 there are two transition points: $\beta_1 = 0$, $\beta_1 = 2$. We can express the solution space as:

$$\begin{cases} v_{p1}, v_{p2} \geq v_{p3}, \boldsymbol{x_p} = [1,1,0] & \beta_1 \leq 0 \\ v_{p1}, v_{p3} \geq v_{p2}, \boldsymbol{x_p} = [1,0,1] & 0 \leq \beta_1 \leq 2 \\ v_{p2}, v_{p3} \geq v_{p1}, \boldsymbol{x_p} \equiv [0,1,1] & 2 \leq \beta_1 \end{cases}$$

We define the *predicted optimal value* (POV) and *true optimal value* (TOV), for fixed feature value $\boldsymbol{\theta}$:

$$POV(\boldsymbol{\beta}) = \boldsymbol{x_p}^T \boldsymbol{v_p}$$
$$TOV(\boldsymbol{\beta}) = \boldsymbol{x_p}^T \boldsymbol{v}$$

Note that the predicted coefficients do not directly affect the true objective value shown in Figure 1c. However the transition points of the predicted PWLF are exactly the same as the transition points of the discrete, true objective. Therefore identifying the transition points of the predicted function can dramatically reduce the effort to map the real objective function.

Convex Search Space

Our framework *Divide and Learn (DNL)* predicts coefficients with a linear prediction model. Here we show that for linear models and an arbitrary optimization problem with a linear objective function, the predicted objective value is a convex piecewise function. An optimization problem has a linear objective function when the relationship between the solution vector and the coefficients is linear:

$$Obj(\boldsymbol{x}, \boldsymbol{v}) = \boldsymbol{x}^T \cdot \boldsymbol{v} \tag{4}$$

Here the predicted coefficients are parameterised linear functions

$$v_p = \boldsymbol{\beta}^T \cdot \boldsymbol{\theta} + c \tag{5}$$

where c is a constant. This is the same restriction as Demirovic et al. (2020). Equations 4 and 5 mean that the POV of the optimization problem can be expressed as a sum of linear functions.

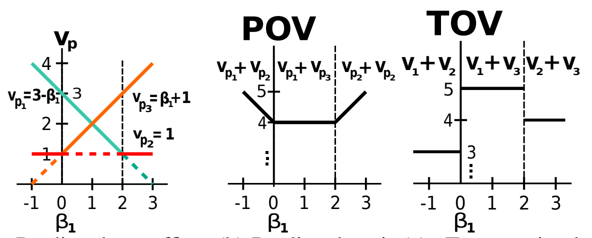
We discuss the properties of POV assuming $\boldsymbol{\beta}$ is a singleton β. Since we use coordinate descent to reason about POV and update one parameter at a time, this is the only case we are interested in.

Lemma 1. *POV is a piecewise linear function (Appx. A.1).*

Lemma 2. *POV is a convex function (Appx. A.2).*

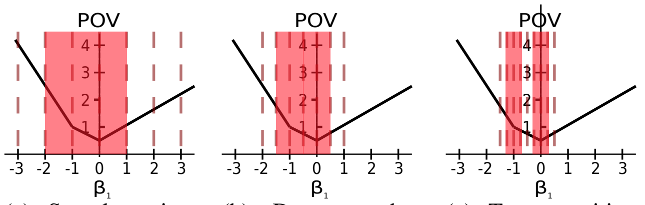
Corollary 1. *For any three values* $\beta_1 < \beta_2 < \beta_3$*, the points* $(\beta_1, POV(\beta_1)), (\beta_2, POV(\beta_2)), (\beta_3, POV(\beta_3))$ *are not collinear iff there is a transition point* β_t *in the range* $\beta_1 < \beta_t < \beta_3$ *(Appx. A.3).*

Figure 1: Piecewise function construction



(a) Predicted coefficient ($\boldsymbol{v_p}$) (b) Predicted optimal value (POV) (c) True optimal value (TOV)

Figure 2: Divide and conquer algorithm



(a) Sample points with large and uniform steps. Transition point in [-2,1] (b) Decrease the step size and narrow the interval to [-1.5,0.5] (c) Two transition points in the intervals [-1.25,-0.75], [-0.25,0.25]

Transition point extraction

We propose a numerical approach to extract transition points without any solver restrictions. Our approach works for any optimization problem with a linear objective and fixed constraints.

Divide and conquer: From Corollary 1, we know that if three arbitrary points of the POV are not collinear, there is at least one transition point between those points. When we decrease the distance between these three values, we can accurately identify the location of transition points. The simplest way to extract transition points is to sample the POV with a fixed step size, and compare collinearity of three consecutive points. This brute force method is infeasible for many problems. For long intervals without transition points, a small step size is redundant. With these insights we apply a divide and conquer algorithm to sample the POV. First we split the search region with ten uniformly spread points, then we test collinearity of these points. If there are points that are not collinear, the intervals these points define are marked as *transition intervals*. A transition interval is an interval with at least one transition point. Then we proceed to decrease the step size as: $step_{n+1} = \frac{step_n}{10}$ and sample the transition intervals. Finally we repeat finding transition intervals, reducing the step size until the step size reaches a desired minimum. By starting with a large step size and iteratively reducing it, we identify long intervals without transition points with minimal processing (Fig 2).

Multivariate model: We described an extraction method for a singleton $\boldsymbol{\beta}$. In reality multivariate models are widely used to predict these coefficients. It is possible to find the transition points of multivariate PWLFs, but the number of transition points can increase exponentially. We use *coordinate descent* to transform a high dimensional linear model into a one dimensional model. We find transition points for each singleton $\boldsymbol{\beta}$ separately (Wright 2015). For a parameter vector $\boldsymbol{\beta} = [\beta_1, ..., \beta_m]$, coordinate descent iterates over $\boldsymbol{\beta}$. In each iteration one parameter, β_k, is considered as a variable, and the rest are fixed as a constant, $v_p = \beta_k \cdot \theta_k + \sum_{n \neq k} \beta_n \cdot \theta_n$. Then for each parameter we consecutively perform transition point extraction and parameter updates. After all transition points are identified we update the parameters as explained in the next section.

Parameter Update: In a PPO setting a dataset is a collection of multiple problem sets. Each problem set has the same constraints, but with different coefficients. We predict unknown coefficients for each problem set with the same model parameters $\boldsymbol{\beta}$, and the goal is to train model parameters to minimize the average regret across all problem sets. For a dataset of size N, and coefficient vector $\boldsymbol{v}$, the dataset is denoted $D = \{\boldsymbol{v}^{(1)}, ..., \boldsymbol{v}^{(N)}\}$. We choose the model parameters $\boldsymbol{\beta}$ to minimize the average regret R.

$$\boldsymbol{\beta} \equiv \arg\min \frac{1}{N} \sum_{n=0}^{N} R(\boldsymbol{v}^{(n)}, \boldsymbol{v_p}^{(n)})$$

Transition point comparison: We express the TOV of each problem set as a piecewise function. The extraction method provides the transition points of each problem set. Let $T^{(i)} = \{\beta_{t_1}^{(i)}, ...\beta_{t_L}^{(i)}\}$ be the set of transition points of size L for problem set i. We construct the intervals of the piecewise function as $I^{(i)} = \{[\beta_{t_l}^{(i)}, \beta_{t_{(l+1)}}^{(i)}], 0 < l < L, l \in \mathbb{Z}\}$. To find the optimal model parameters for problem set i, we calculate the TOV for each interval and pick the best interval. A single sample point for each interval is enough to calculate the TOV and we choose the mid points of the intervals. Let I_{mid} represent the set of mid points then $\beta_{opt}^{(i)} = \arg\min_{\beta \in I_{mid}^{(i)}} R(\boldsymbol{v}^{(i)}, \boldsymbol{v_p}^{(i)})$. The optimal parameter for each problem set can be different. To find the optimal parameters over all problem sets, we compare every interval from each problem set.

$$\beta_{opt} \equiv \underset{\beta \in \bigcup_{i=1}^{N} I_{mid}^{(i)}}{\arg\min} \frac{1}{N} \sum_{n=0}^{N} R(\boldsymbol{v}^{(n)}, \boldsymbol{v_p}^{(n)})$$

With coordinate descent we perform these comparisons for each parameter and update the parameters individually. We use mini-batches to train the model parameters. A mini-batch represents a subset of problem sets. When using mini-batches we construct a quasi-gradient in the direction of the global minimum of that particular mini-batch. Then we update the model parameters with the quasi gradient. For a mini-batch i: $\beta_{new} = \beta_{old} + learning\_rate \cdot \frac{\beta_{opt}^{(i)} - \beta_{old}}{|\beta_{opt}^{(i)} - \beta_{old}|}$

Greedy Methods: To extract transition points and compare them, we repeatedly solve an optimization problem. Many optimization problems are NP-hard, and large problems can be expensive to work with. Therefore we propose a greedy method to partially extract transition points.

Divide and Learn Greedy (DNL-Greedy): Our divide and conquer algorithm repeatedly compares the collinearity of POV samples and there can be many redundant transition points. We propose a greedy extraction method to stop the extraction at the first transition point β_t that improves TOV over the old parameter β_{old}. We prioritize searching regions around the old model parameter. We observed that although TOV is not a convex function, the optimal model parameters can be clustered in similar regions. Our motivation with this greedy method is to quickly iterate over parameters and bypass redundant sampling for the first iterations. Note, even for non convex problems the greedy method can find the global minimum. DNL-Greedy uses only one transition point for each problem set, and reduces comparison complexity by solving N^2 optimization problems, and the number of transition points has minimal effect on the comparison complexity. We show empirically DNL-Greedy achieves similar performance to the full method, and reduces the run time.

Coordinate descent: DNL uses coordinate descent to train model parameters to maximize the TOV. TOV is not a convex function and in order to find the true optimal point we need to test each transition point of the function. Therefore coordinate descent may result in a loss of optimality. This can be overcome by using a multi-dimensional divide and conquer algorithm for multivariate models, however we believe the trade off of an exponential increase in transition points is not worth the more accurate representation of the problem space.

Evaluation and Discussion

We experiment on two optimization problems: 0-1 knapsack and scheduling. We compare four exact models: *DNL*, *DNL-Greedy*, *SPO-Forest* (Elmachtoub, Liang, and McNellis 2020), *dynamic programming (DP)* (Demirovic et al. 2020), three surrogate models: *SPO*-Relax (Mandi et al. 2020), *QPTL* (Ferber et al. 2020), *IntOpt* (Mandi and Guns 2020) and an ML model: *ridge regression*.

Dataset: We use the dataset from the ICON energy challenge (Simonis et al. 2014) for both knapsack and scheduling problems. The same dataset was used in previous work on PPO (Mandi et al. 2020; Demirovic et al. 2020). Data samples are collected from real electricity prices every 30 minutes, from November 2011 to January 2013. In total there are 37877 data samples, each 48 data samples, representing a day, form a problem set. Therefore we use 789 optimization problems to train PPO models.

Methodology: We split the dataset into 70% training set, 10% validation set and 20% test set, resulting in 552, 79 and 157 optimization problems for training, validation and testing. As the dataset is small, we further split the data set into 5 folds to capture the whole distribution. For each fold we train the models 10 times and use the iteration with the least validation regret. Due to the sparsity of data and the combinatorial problem, we observed high variance for all models, including regression. To give a clear representation of the performance, we chose to normalize the regrets with respect to a baseline: regression. We report the normalized mean regret and one standard deviation across folds. Even with normalized regrets, there can be high variance between folds and for scheduling problems we also report the number of folds where a PPO model outperforms the baseline model.

Knapsack problem: We consider a 0-1 knapsack problem of n items: given a capacity limit, W, item weights $\mathbf{w} = [w_1, w_2 ..., w_n]$, we have to predict item values $\boldsymbol{v} = [v_1, v_2 ..., v_n]$. A 0-1 solution vector $\boldsymbol{x} = \arg\max_{(\boldsymbol{x}^T \cdot \mathbf{w} \leq W)} \boldsymbol{x}^T \cdot \boldsymbol{v}$ decides the chosen items.

We run knapsack experiments for both unit weights and varying weights. Knapsack problems with high correlation between item values and weights are harder to solve than those with weak correlations (Pisinger 2005). We generate exactly correlated knapsack problems by choosing weight values in $\{3, 5, 7\}$, and multiplying by the true energy price to generate their true value. We experiment on varying capacity limits from 5% to 90%. For unit knapsack the capacity limits range from 5-45 (10%-90%) and for weighted knapsack the capacity limits range from 12-220 (5%-90%).

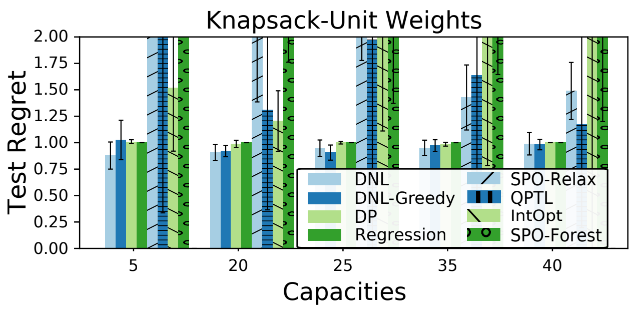
Scheduling: The scheduling problems are modified versions of the ICON challenge (Mandi et al. 2020), (Simonis et al. 2014). There are M machines and N jobs. Each machine can run at most 2 jobs at a given time slot. Each job has a power consumption P_n and a duration D_n. Jobs are not restricted by an earliest starting time and a latest finishing time and can be allocated to any of the 48 time slots. A job can only be run on one machine, and once a job is being processed it cannot be split. All jobs have to be finished in the 24 hour period. The goal of the scheduling is to maximize the energy cost. Energy prices are not known beforehand and have to be predicted. For the experiments we constructed benchmarks with identical jobs with duration $D_n = 4$ time slots. We vary the machine numbers from 1 to 3 and jobs from 1 to 10.

Experiments

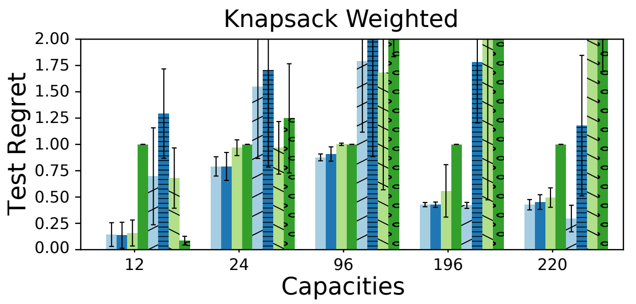
The models are trained with Intel(R) Xeon(R) Gold 6254 CPU @ 3.10GHz processors using 8 cores with 3.10 Ghz clock speed. We use (Gurobi Optimization 2020) to solve knapsack and scheduling problems. For the knapsack problems max training time is set to 4000 seconds ($\approx$ 1 hour). For scheduling problems max training time is set to 12000 seconds ($\approx$ 3.3 hours). Refer to *Appx. B* for detailed hyper parameter configurations. We use early stopping for all PPO models (Bishop 2006), and use the iteration with the least validation regret. We warmstart model parameters with regression (Pratt and Jennings 1996).

Unit knapsack: The unit knapsack problem (Figure 3a) is a simple optimization problem and there is no significant difference between regression and transition point based PPO methods, DNL, DNL-Greedy and DP. Note that identical performance between DNL and DP is expected as they both use the same transition points. We see that numerical approximations of the DNL frameworks are enough to train parameters as good as the exact locations. SPO-Forest has sub-optimal performance, but its performance increases as the capacity of the knapsack problem increases. Similarly, surrogate models also fail to capture the knapsack problem and perform worse than regression. Surrogate models and SPO-Forest also have significantly higher variance when compared to DNL, DNL-Greedy and DP.

Figure 3: Knapsack Benchmarks



(a) Unit Knapsack: normalized mean and one standard deviation. Graph is truncated at 2 for readability. See Appx. E.2 for all capacities



(b) Weighted Knapsack: normalized mean and one standard deviation. The legend is the same as (Fig 3a). Graph is truncated at 2 for some instances for readability. See Appx. E.2 for all capacities

Weighted knapsack: Weighted knapsack is harder than the unit knapsack, and we see benefits of PPO frameworks better. Transition point methods DNL, DNL-Greedy and DP outperform regression for all capacities, and the improvement is significant at capacities 12, 196 and 220. As for the unit knapsack problem the DNL and DP frameworks have similar performance. On capacity 96, DP cannot train under the time limit, and DNL frameworks outperform DP. SPO-Forest outperforms regression only for capacities 12, 196 and 220. QPTL and Intopt performs well for small capacities and SPO is worse than regression for all capacities. Other frameworks have higher variance than DNL and DP.

Scheduling: Scheduling problems are more complex than both the unit and the weighted knapsack problems. Scheduling problems also do not have a feasible DP solution and the DP framework is not applicable. DNL-Greedy outperforms regression for most of the benchmarks , and the only time its performance is significantly worse is with the 2 Machines and 2 Jobs benchmark(Table 1). To the best of our knowledge the only other exact method applicable for these scheduling problems is SPO-Forest, and its performance is sub-optimal compared to all other methods. QPTL also falls behind compared to SPO-Relax and DNL-Greedy, and in many benchmarks is worse than regression. IntOpt performs similar to normal regression and has less variance among surrogates for complex scheduling tasks. DNL-Greedy outperforms SPO-Relax for all benchmarks with 3 machines. Moreover SPO-Relax shows higher variance than DNL-Greedy for a majority of the benchmarks, especially for problems with more than one machine. DNL-Greedy also outperforms regression more consistently compared to SPO-Relax: DNL-Greedy improves 4.1, 3.8, and 4.2 folds on average for benchmarks with 1, 2 and 3 machines, and SPO-Relax improves 3.9, 3, and 3 folds.

Surrogates and Combinatorial Reasoning: Differentiable surrogates reason over non-differentiable combinatorial problems with an approximation error. The relaxations can misrepresent problem spaces with large discrete jumps. We experimented on both problem spaces with small jumps such as unit knapsack, high capacity weighted knapsack and 1 machine scheduling problems, and with large jumps such as low capacity weighted knapsack problems and 3 machines scheduling problems.

There are 48 time slots in a problem set. For scheduling problems each job has identical duration: 4 time slots. Each machine is also identical and can run two jobs simultaneously. When there is one machine and two jobs, 4 to 8 time slots can be allocated. If the problem set has few outlying high values, the difference between the optimal decision and other feasible solutions will be large and the regret function will have large discrete jumps. Whereas for the benchmark with 1 machine and 10 jobs, 20 to 40 time slots are going to be allocated. As the job number is increased the difference between the optimal solution and other feasible solutions will decrease gradually and the problem space will have smaller discrete jumps. Similarly the problem space will be more jagged when the number of machines are increased. We see a similar narrative for the knapsack problem: when capacities are increased the regret will have smaller discrete jumps. SPO-Relax performs relatively better for unit knapsack problems than weighted knapsack and its performance improves as the capacity increases for weighted knapsack problems. Similarly SPO-Relax has good performance and small variance for scheduling problems with 1 machine and high job counts (appx. E.1). However its performance drops sharply when the machine numbers increase or the job numbers decrease. In contrast, DNL-Greedy can reason over both smooth and jagged problem spaces, and improves regret even for hard combinatorial problems like the weighted knapsack benchmarks and scheduling problems with high machine counts.

PPO problems can have small datasets, and this is further magnified by grouping data samples into problem sets. The ICON data set has 37,877 data samples, which is reduced to 789 problem sets with a problem set size of 48 samples. Therefore PPO models should be robust against scarce data. Surrogate models show significantly larger variance across folds than DNL-Greedy and they are more sensitive to the data distribution. This large variance is observed with unit and weighted knapsack benchmarks and is especially pronounced for scheduling problems with higher machines. For benchmarks with three machines, surrogate models other than IntOpt have high variance, more than triple that of DNL. For scheduling problems with 3 machines, QPTL is comparable to SPO-Relax, for others QPTL is outperformed by both SPO-Relax and DNL.

Reliability: PPO problems can be volatile and users may

	2 Machines					3 Machines				
Jobs	DNL-Greedy	SPO-Relax	QPTL	SPO-Forest	IntOpt	DNL-Greedy	SPO-Relax	QPTL	SPO-Forest	IntOpt
1	**0.73±0.33 (5)**	*1.41±1.36* (3)	*1.27±0.29* (1)	*19.39±9.86*, (0)	*1.1±0.43* (3)	**0.73 ± 0.33 (4)**	*1.13±0.96* (3)	*1.28±0.7* (2)	*19.39±9.86* (0)	*1.25±0.34* (2)
2	*1.25±0.96* **(4)**	*1.1±0.96* **(4)**	*1.02±0.57* (3)	*19.39±9.86*, (0)	*1.15±0.55* (2)	**0.75 ± 0.33 (4)**	0.87 ± 0.5, (3)	0.99 ± 0.55 (3)	*19.39±9.86* (0)	*1.14±0.32* (3)
3	**0.82±0.2 (4)**	0.88±0.47 **(4)**	*1.21±0.4* (2)	*19.39±9.86* (0)	0.99±0.14 **(4)**	**0.87±0.14 (4)**	*1.26±1.14***(4)**	*1.02±0.44* (0)	*19.39±9.86* (0)	*1.02±0.19* (3)
4	**0.63±0.3 (5)**	0.8±0.39 (3)	0.98±0.34 (3)	*19.39±9.86* (0)	1.02±0.07 (2)	**0.69±0.3 (5)**	*1.32±1.07* (3)	0.97±0.31 (4)	*19.39±9.86* (0)	0.91±0.07 (5)
5	**0.84±0.14 (4)**	1.0±0.43 (3)	*1.08±0.24* (3)	*4.59±1.4* (0)	*1.02±0.15*, (3)	**0.81±0.1 (5)**	*1.5±1.6* (3)	*1.44±0.67* (2)	*19.39±9.86* (0)	0.92±0.14 (3)
6	**0.83±0.13 (5)**	*1.02±0.35* (2)	*1.16±0.21* (2)	*4.73±0.99* (0)	*1.05±0.2* (3)	**0.74±0.31** (4)	*1.21±0.94* (3)	0.84±0.11 **(5)**	*19.39±9.86* (0)	*1.03±0.14* (3)
7	1.0±0.19 (2)	**0.97±0.36** (2)	*1.08±0.31* **(3)**	*4.44±0.93* (0)	1.0±0.15 **(3)**	**0.9±0.17 (3)**	*1.38±0.78* **(3)**	*1.32±0.43* (2)	*5.58±1.84* (0)	*1.07+0.19* (2)
8	*1.02±0.19* **(3)**	**0.77±0.43 (3)**	*1.26±0.69* (2)	*6.83±1.92* (0)	*1.11+0.51* (3)	**0.83±0.12 (5)**	*1.01±0.5* (3)	*1.11±0.25* (2)	*4.35±1.2* (0)	0.99+0.13 (3)
9	0.91±0.17 **(4)**	**0.82±0.17** (3)	*1.21±0.23* (0)	*4.46±0.63* (0)	0.94+0.02 **(5)**	**0.8±0.21 (4)**	*1.14±0.5* (3)	*1.25±0.28* (1)	*4.44±0.93* (0)	*1.02+0.04* (2)
10	0.99±0.14 (2)	**0.87±0.14 (3)**	*1.07±0.36* **(3)**	*4.27±0.69* (0)	0.93+0.08 **(3)**	**0.84±0.12 (5)**	0.89±0.3 (3)	0.96±0.11 (3)	*4.32±0.7* (0)	*1.03+0.17* (2)
Folds	**38**	30	22	0	31	**42**	31	24	0	28

Table 1: Normalized mean regret and one stds and number of improved folds for scheduling benchmarks. *Mean regret ± std (number folds improved over regression)* is the metric used to demonstrate the performance. At the bottom of the table we also report the total number of improved folds for each number of machines. The regret values are normalized with respect to the baseline ridge regression. Values under 1 represent the cases where the model performs better than the baseline regression. Values over 1 represent the cases when the model performs worse than the regression. See Appx. E.1 for all machines

be concerned about a reliable framework for sensitive decision problems. We have highlighted two reasons for the unpredictability of PPO problem spaces: the combinatorial nature of the problem and data scarcity. Usually it is only possible to see if a model fits a problem *after* training and testing a model. The surrogates' reliability drops and their performance variance increases when the combinatorial nature of the problem increases such as the harder scheduling problems. In contrast DNL-Greedy preserves its performance even with increasing machines and improves 4.1, 3.8 and 4.2 folds on average for benchmarks with 1, 2 and 3 machines.

Our benchmarks are systematically constructed to observe the response of the PPO frameworks to different types of decision problems. In a realistic scenario, it is extremely hard to predict when the problem space is near convex. Unlike the surrogate methods, DNL consistently outperforms regression for both knapsack and scheduling benchmarks, and in the worst case is at least as good as the regression baseline. The only benchmark it significantly underperforms on is scheduling benchmark with 2 jobs and 2 machines, whereas surrogate models are outperformed by regression for both knapsack problems and for all scheduling benchmarks with 3 machines. Therefore if a PPO framework is to be chosen for an unknown decision problem and an unknown dataset, DNL can be chosen for its likelihood to be at least as good as a traditional method.

SPO-Forest and DNL: SPO-Forest is the state of the art direct PPO method based on decision trees. It is limited to problems with linear objectives. In our benchmarks its performance is sub-optimal compared to other models. Unlike other models SPO-Forest uses a multi-output regression model to predict all the items in a problem set. Therefore the item predictions are not independent, and features of one item can affect forecasts of multiple items. On one hand this general structure might capture the complex interdependent relationships, on the other hand fully understanding combinatorial problems requires larger data-sets (Yehuda, Gabel, and Schuster 2020). Many PPO applications have

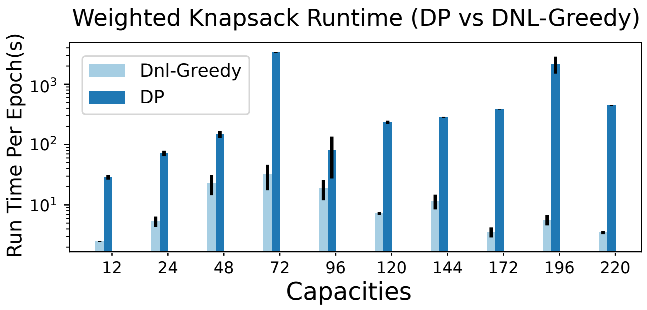


Figure 4: Training time comparison of DNL and DP to train an epoch in log scale(s). DNL is magnitudes order faster than DP and is more robust against high capacity problems.

small numbers of problem sets to train with. SPO-Forest over-fits the average problem set distribution and has difficulties understanding the direct relationship between features and individual items (Appx. D). Therefore it is sensitive to the outliers in problem sets and potentially requires a larger data-set to generalize well.

Run Time

Exact methods deliver better combinatorial reasoning by solving the combinatorial problem multiple times. However this results in longer training times. Our frameworks, DNL and DNL-Greedy, trains faster than the previous transition points model DP. On the other hand exact models are are slower than surrogate models. The regression model has negligible training time compared to predict+optimize models. In this section we compare the run times between different predict+optimize models DP, SPO-Relax, QPTL, SPO-Forest and regression.

DNL and DP: The main advantage of DNL over DP is that DNL can be used with state of the art solvers and significantly improves training time per epoch. Therefore it can be applied with optimization problems that may not be feasible with the DP method. DNL trains faster than DP for all capacities of knapsack problems and the difference is more striking for larger capacities (Figure 4). If we look at the time of early stopping (Figure 5b,5a), we see that for very low capacities DP stops earlier than DNL. However for larger problems DP doesn't scale as well as DNL. At weighted knapsack benchmark with capacity 96 and unit knapsack benchmark with capacity 40 DP returns 0s training time. Counter intuitively this is due to the long training times. DP cannot finish an epoch under 4000 seconds and therefore training terminates without any iteration.

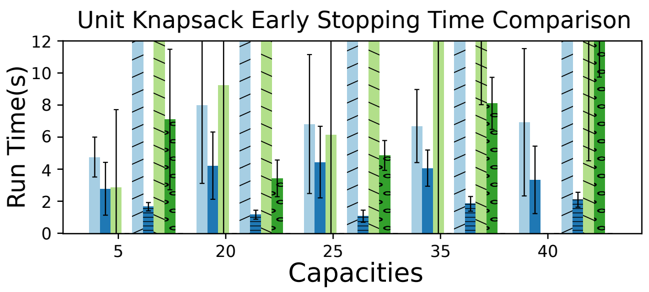
DNL and SPO-Forest: DNL runs faster than SPO-Forest for all knapsack problems (Figure 5b, 5a). We use a larger search space for scheduling benchmarks and SPO-Forest is faster than DNL (Appx. E.1.2). Although SPO-Forest runs faster at scheduling benchmarks, it's regret performance is significantly worse than a simple regression whereas DNL improves regret over regression.

DNL and Surrogate Models: Surrogate models reason over a differentiable surrogate of the combinatorial problem and do not have to solve the decision problem as many times as the exact methods. Therefore they are significantly faster than exact models. However this comes with the cost of not being able to reason over the real combinatorial space and for hard problems they may have larger regret than regression. SPO-Relax is the fastest model for all scheduling benchmarks (Appx. E.1.2). For unit knapsack problems DNL is faster than QPTL and for weighted knapsack problems there isn't a clear difference between run times of QPTL and DNL (Figures 5a, 5b). QPTL and IntOpt are faster than DNL for scheduling (Appx. E.1.2), however like SPO-Forest it has no regret improvement over DNL.

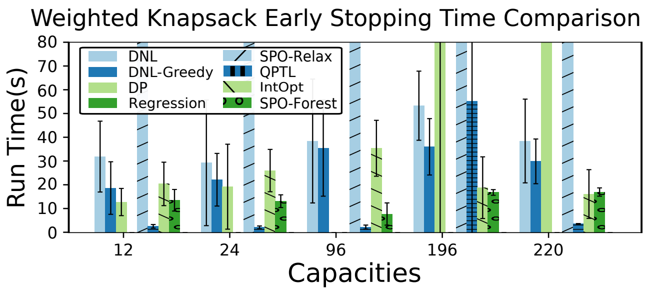
Summary: The DNL framework significantly improves the training time compared to the previous DP approach by using a novel divide and conquer method that can be used with any state of the art solver (Fig 4). However DNL framework requires more time to train compared to surrogate models. Surrogate models build a gradient using a convex surrogate. With this convexity assumption they can update the model parameters with a single solver call and they can be trained faster than the exact models that require multiple calls to the solver. Our weighted knapsack and more complex scheduling experiments suggest that when there is a discrepancy between the non convex real solution space and the surrogates, the surrogate models will have sub-optimal performance. We believe reasoning over non convex underlying problems are likely to require non polynomial time and multiple solver calls are necessary for good performance.

Problems that prioritize performance over training cost benefit from using an exact method, as they are more reliable in the case of a non-convex underlying problem. For problems that are known to behave like a convex problem surrogates may be preferable. However in such cases we observed that a simple regression may also result in a similar or better performance (Table 2). Even the fastest surrogate model needs to solve a non polynomial combinatorial problem and is significantly slower than a standard regression model. Therefore it is likely that if a user is choosing a PPO method over a standard ML model, they are concerned with performance over cost. The DNL method provides an alternative to surrogate models for reliable improvement. Table 2 summarises the trade-offs between performance and run time of the various methods on different problem classes.

Figure 5: Knapsack Run Times



(a) Run time comparison at point of early stopping for unit knapsack benchmarks. Regression training time is negligible compared to PPO frameworks. See Appx. E.2.2 for all capacities



(b) Run time comparison at point of early stopping for weighted knapsack benchmarks. Regression training time is negligible compared to PPO frameworks. See Appx. E.2.2 for all capacities

Models	Performance				Time
	Near-Convex		Non-Convex		
	Unit Knapsack	Easy Scheduling	Knapsack	Hard Scheduling	
Regression	**Very High**	High	High	Average	Very Short
DP	**Very High**	High	**Very High**	N/A	Very Long
DNL-Greedy	**Very High**	High	**Very High**	**Very High**	Long
SPO-Forest	Low	Low	Moderate	Very Low	Average
SPO	Moderate	**Very High**	Low	Average	Short
QPTL	Moderate	Average	Low	Low	Short
INTOPT	Low	Average	Very Low	Average	Short

Table 2: Summary of effectiveness of methods. The comparisons are made with relative performance. For accurate representations see to Figures 3b, 3a and Table 1.

Conclusion

PPO problems are challenging due to the combinatorial nature of the optimization problem. We propose a new divide and conquer method to extract transition points and train parameters using regret, rather than a surrogate or a relaxation. In contrast to the previous DP method, our methods can be applied with any state of the art solvers. Our framework outperforms regression and surrogates for weighted knapsack problems and harder scheduling problems. We also show that DNL is more robust to changes in the scheduling formulation. We next plan to experiment with nonlinear frameworks and transition points for stronger models.

Acknowledgements: This research was supported by The University of Melbourne's Research Computing Services and the Petascale Campus Initiative.

References

Amos, B.; and Kolter, J. Z. 2017. OptNet: differentiable optimization as a layer in neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 136–145.

Bello, I.; Pham, H.; Le, Q. V.; Norouzi, M.; and Bengio, S. 2016. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*.

Bengio, Y. 1997. Using a financial training criterion rather than a prediction criterion. *International Journal of Neural Systems*, 8(04): 433–443.

Bishop, C. M. 2006. *Pattern recognition and machine learning*. springer.

Demirovic, E.; Bailey, J.; Chan, J.; Guns, T.; Kotagiri, R.; Leckie, C.; and Stuckey, P. J. 2019a. A Framework for Predict+Optimise with Ranking Objectives: Exhaustive Search for Learning Linear Functions for Optimisation Parameters. In Kraus, S., ed., *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 1078–1085. IJCAI Press.

Demirovic, E.; Stuckey, P. J.; Bailey, J.; Chan, J.; Leckie, C.; Ramamohanarao, K.; and Guns, T. 2019b. Predict+ Optimise with Ranking Objectives: Exhaustively Learning Linear Functions. In *IJCAI*, 1078–1085.

Demirovic, E.; Stuckey, P. J.; Guns, T.; Bailey, J.; Leckie, C.; Kotagiri, R.; and Chan, J. 2020. Dynamic Programming for Predict+Optimise. In Conitzer, V.; and Sha, F., eds., *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-20)*, 1441–1451. AAAI Press.

Donti, P.; Amos, B.; and Kolter, J. Z. 2017. Task-based end-to-end model learning in stochastic optimization. In *Advances in Neural Information Processing Systems*, 5484–5494.

Elmachtoub, A.; Liang, J. C. N.; and McNellis, R. 2020. Decision trees for decision-making under the predict-then-optimize framework. In *International Conference on Machine Learning*, 2858–2867. PMLR.

Elmachtoub, A. N.; and Grigas, P. 2017. Smart" predict, then optimize". *arXiv preprint arXiv:1710.08005*.

Ferber, A.; Wilder, B.; Dilkina, B.; and Tambe, M. 2020. MIPaaL: Mixed Integer Program as a Layer. In *AAAI*, 1504–1511.

Gurobi Optimization, L. 2020. Gurobi Optimizer Reference Manual.

Horvitz, E.; and Mitchell, T. 2010. From data to knowledge to action: A global enabler for the 21st century. *Computing Community Consortium*, 1.

Ifrim, G.; O'Sullivan, B.; and Simonis, H. 2012. Properties of energy-price forecasts for scheduling. In *International Conference on Principles and Practice of Constraint Programming*, 957–972. Springer.

Kao, Y.-h.; Roy, B. V.; and Yan, X. 2009. Directed regression. In *Advances in Neural Information Processing Systems*, 889–897.

Li, Z.; Chen, Q.; and Koltun, V. 2018. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems*, 539–548.

Lim, A. E.; Shanthikumar, J. G.; and Vahn, G.-Y. 2012. Robust portfolio choice with learning in the framework of regret: Single-period case. *Management Science*, 58(9): 1732–1746.

Luo, C.; Qiao, B.; Chen, X.; Zhao, P.; Yao, R.; Zhang, H.; Wu, W.; Zhou, A.; and Lin, Q. 2020. Intelligent Virtual Machine Provisioning in Cloud Computing. In Bessiere, C., ed., *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, 1495–1502. International Joint Conferences on Artificial Intelligence Organization. Main track.

Mandi, J.; and Guns, T. 2020. Interior Point Solving for LP-based prediction+ optimisation. *arXiv preprint arXiv:2010.13943*.

Mandi, J.; Guns, T.; Demirovic, E.; and Stuckey, P. J. 2020. Smart Predict-and-Optimize for hard combinatorial optimization problems. In Conitzer, V.; and Sha, F., eds., *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-20)*, 1603–1610. AAAI Press.

Niculae, V.; Martins, A.; Blondel, M.; and Cardie, C. 2018. SparseMAP: Differentiable Sparse Structured Inference. In *International Conference on Machine Learning*, 3799–3808.

Pisinger, D. 2005. Where are the hard knapsack problems? *Computers & Operations Research*, 32(9): 2271–2284.

Pogančić, M. V.; Paulus, A.; Musil, V.; Martius, G.; and Rolinek, M. 2020. Differentiation of Blackbox Combinatorial Solvers. In *International Conference on Learning Representations*.

Pratt, L.; and Jennings, B. 1996. A survey of connectionist network reuse through transfer. In *Learning to learn*, 19–43. Springer.

Simonis, H.; O'Sullivan, B.; Mehta, D.; Hurley, B.; and De Cauwer, M. 2014. Energy-Cost Aware Scheduling/Forecasting Competition.

Thapper, J.; and Živnỳ, S. 2018. The limits of SDP relaxations for general-valued CSPs. *ACM Transactions on Computation Theory (TOCT)*, 10(3): 1–22.

Wilder, B.; Dilkina, B.; and Tambe, M. 2019. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 1658–1665.

Wright, S. J. 2015. Coordinate descent algorithms. *Mathematical Programming*, 151(1): 3–34.

Yehuda, G.; Gabel, M.; and Schuster, A. 2020. It's not what machines can learn, it's what we cannot teach. In *International Conference on Machine Learning*, 10831–10841. PMLR.