Dynamic Programming for Predict+Optimise

Emir Demirović,[1] **Peter J. Stuckey,**[2,3] **James Bailey,**[1] **Jeffrey Chan,**[4]
Christopher Leckie,[1] **Kotagiri Ramamohanarao**,[1] **Tias Guns**[5]
[1]University of Melbourne, Australia
[2]Monash University, Australia
[3]Data61, Australia
[4]RMIT University, Australia
[5]Vrije Universiteit Brussel, Belgium
{emir.demirovic, baileyj, caleckie, kotagiri}@unimelb.edu.au,
peter.stuckey@monash.edu, jeffrey.chan@rmit.edu.au, tias.guns@vub.be

Abstract

We study the predict+optimise problem, where machine learning and combinatorial optimisation must interact to achieve a common goal. These problems are important when optimisation needs to be performed on input parameters that are not fully observed but must instead be estimated using machine learning. We provide a novel learning technique for predict+optimise to directly reason about the underlying combinatorial optimisation problem, offering a meaningful integration of machine learning and optimisation. This is done by representing the combinatorial problem as a piecewise linear function parameterised by the coefficients of the learning model and then iteratively performing coordinate descent on the learning coefficients. Our approach is applicable to linear learning functions and any optimisation problem solvable by dynamic programming. We illustrate the effectiveness of our approach on benchmarks from the literature.

Introduction

Machine learning and combinatorial optimisation are two essential components of decision-making systems. The former aims to provide predictive models based on historical data, while the latter prescribes optimal actions given input data. While there has been significant progress in these areas individually, an established methodology for solving problems which require both remains an open question.

We study the predict+optimise problem (Demirović et al. 2019a; Wilder, Dilkina, and Tambe 2019; Donti, Amos, and Kolter 2017), where the task is to learn input parameters to a combinatorial optimisation problem, such that the resulting solutions are deemed desirable. Therefore, optimisation needs to be performed with input that is approximated. Traditionally, predict+optimise is viewed as a two-stage process: predict the input parameters and then optimise. This is illustrated in the following example, which will be used as a running example throughout the paper.

Example 1. *Consider a parametrised project-funding problem. The input parameters are integers that represent the benefit of funding each individual project p_i. These estimates are based on the novelty n_i of the project and the experience e_i of the investigator. In addition, each project has a cost c_i and a limited amount of funding is available. The committee decided to use a linear weighting scheme to determine the perceived value v_i of each project. They fixed the coefficient for the experience component and are deciding on the coefficient α for the novelty part, i.e., $v'_i(\alpha) = n_i \cdot \alpha + e_i$. Once the coefficient α is agreed upon, the task is to select the subset of projects to maximise the estimated value. Consider four projects with features $p_i = (n_i, e_i, c_i)$ as $(-1, 10, 2), (1, 2, 1), (-0.5, 5, 1)$, and $(2, -5, 1)$. Note that all but the first project are of unit cost. Assume the funding budget is set to two units. By varying the coefficient α, different optimal outcomes can be observed. Figure 1(a) shows the relationship between α, the funding decisions, and their total estimated value.* □

In the example above, the output is the list of projects that are to be selected for funding. The estimated values serve merely as proxies for deciding the outcome. However, the *true* values of funding the project will only become apparent after the projects are completed. Therefore, the aim is to learn a predictive model that results in a decision that maximises the true value, rather than the estimated values.

The main issue that arises is that conventional learning metrics, such as mean-square error, do not necessarily reflect the outcome of the optimisation procedure.

Example 1 (continued)**.** *The committee believes it would be best to equally weigh the novelty and experience criteria, i.e., set $\alpha = 1$. This results in funding project p_1. However, a machine learning expert recommends using a data-driven approach. When observing projects with similar features that were funded in previous years, the historical data suggests the projects are likely to result in values $V = (v_1, v_2, v_3, v_4) = (14, 11, 12, 10)$. Linear regression, a standard machine learning algorithm that minimises the mean-square error, i.e., $mse(V, V'(\alpha)) = \sum(v_i - v'_i(\alpha))^2$, suggests setting the novelty coefficient to $\alpha = 5$. This would lead to funding projects p_2 and p_4, which has a greater outcome value than project p_1. However, had the machine learning algorithm understood the underlying combinatorial optimisation problem, a possible suggestion would be $\alpha = 3$. This would result in funding projects p_2 and p_3, which is the optimal funding decision according to histori-*

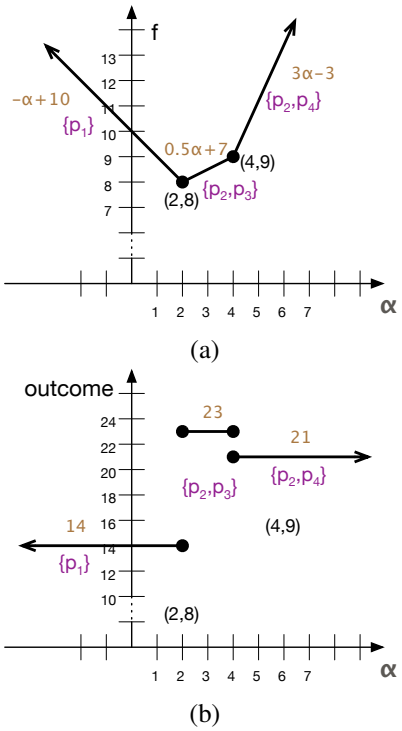

Figure 1: (a) Total estimate and (b) outcome values as a function of the novelty weight α. The purple curly brackets denote the funded projects for values of α along the segment, i.e., $\alpha \in (-\infty, 2) \rightarrow \{p_1\}$; $\alpha \in [2, 4) \rightarrow \{p_2, p_3\}$, and $\alpha \in [4, \infty) \rightarrow \{p_2, p_4\}$.

cal data V and the funding budget. The estimated *values do not accurately represent the* true *values, but serve as proxy values which lead to the best decision. Figure 1(b) shows how α changes outcome value, the total value of the funded projects, based on the historical data.* □

The piecewise function in Figure 1b was key in providing the best predictive model in the previous example. It shows possible outcomes as a function of the learning coefficient α, allowing us to easily select the best value. The construction of this function is precisely the main contribution of our paper. We provide a technique to build such a function for optimisation problems solvable by dynamic programming. When applied to a machine learning setting, such as predict+optimise, the piecewise function representation enables the learning algorithm to reason over the effect on the combinatorial optimisation problem. This is in contrast to previous works, which consider a relaxation of the (**NP**-hard) optimisation problem.

Related work is divided into three groups: 1) *direct* methods aim to interact with the optimisation problem during training, in most cases considering an approximation of the underlying combinatorial problem, 2) *semi-direct* methods take into account features of the optimisation problem but do not directly interact with the problem during training, and 3) *indirect* methods are oblivious to the optimisation problem, i.e., standard machine learning algorithms such as linear regression. Related work, including knowledge-compilation techniques, preference elicitation, and Lagrangian relaxation, are discussed after the section on preliminaries.

Our method falls into the *direct* category: we provide the means to directly reason over the combinatorial problem. This was previously only achieved for combinatorial problems with ranking objective funtions (Demirović et al. 2019a). In our work, we cover a new class of problems, namely those conventionally solvable by dynamic programming, which includes weakly **NP**-hard problems.

The main step is, as illustrate above, to represent the combinatorial problem as a piecewise linear function based on historical data to capture the possible outcomes for any value of the learning coefficient. This is accomplished by using an algebra over (piecewise) linear functions and dynamic programming algorithms. Each segment can be mapped to a solution of the combinatorial problem, which is considered optimal given the objective function defined on the interval. The main purpose of the representation is its application to predict+optimise: once the piecewise representation is built, we can efficiently evaluate the quality of each parameter value with respect to the target solution based on historical data. As a result, we derive a novel learning algorithm that is able to reason directly over combinatorial problems.

We summarise our contributions as follows:

- We provide a technique for accurately representing the behaviour of optimal solutions for combinatorial optimisation problems, where the objective is given as a function over a parameter. We consider the class of problems that are conventionally solvable by dynamic programming.
- We apply our approach to the predict+optimise setting, where machine learning is used to estimate the input parameters of a combinatorial optimisation problem. Our representation enables the machine learning algorithm to estimate the learning coefficients *directly* considering the combinatorial problem, rather than relying on a relaxation of the problem, leading to more accurate predictions.

Preliminaries

We define an **optimisation problem** as:

$$\max_{X \in C} obj(X, P), \tag{1}$$

where P is the vector of optimisation parameters, C is the set of feasible solutions implicitly defined by a set of constraints, and obj is the objective function. *Solving* corresponds to computing an *optimal solution* X^* that maximises the objective while respecting the constraints, i.e. $X^* \in \{X | X \in C \wedge \forall X' \in C : obj(X, P) \geq obj(X', P)\}$.

In the **predict+optimise** setting, the aim is to compute a solution X that maximises the objective in the optimisation problem, but the parameters $P = (p_1, p_2, ..., p_n)$ are hidden. To assist with the optimisation, a set of attribute vectors A are given, where $A_i = (a_{i1}, a_{i2}, ..., a_{im})$ encodes partial information about the optimisation parameter p_i. A common approach is to compute a solution for the original optimisation problem using estimated parameters $p'_i = f(A_i)$ instead

of p_i for a chosen prediction function f. Predicted parameters p'_i are used to construct a solution, but its optimality is evaluated with respect to the hidden parameter P.

The challenge is to select a function f that, when used to provide estimates, leads to an optimal solution with respect to the true parameters. Given historical data (A_i, p_i), the aim of learning is to compute an f that minimises *regret*, i.e., the difference between the optimal solution and the resulting solution. Assume f belongs to a predetermined family of functions $\{f_\alpha\}$ defined by its *m*-dimensional vector α. Predict+optimise problem can be posed in terms of α:

$$\min_{\alpha} Regret(\alpha) = \min_{\alpha}(obj(X^*, P) - obj(X'(\alpha), P)) \quad (2)$$

$$X'(\alpha) = \arg\max_{X \in C}\{obj(X, f_\alpha(A))\} \quad (3)$$

The equations represent the learning of α based on historical data (A_i, p_i) during *training*. In practical scenarios where the training set consists of multiple benchmarks, the regret is computed as the sum of regrets of each benchmark. Once an unknown instance consisting of attribute vectors A'_i is given, the solution is determined using the calculated f_α.

Example 1 (continued)**.** *The project-funding problem can be posed as* $obj(X, P) = \max_X(X \cdot P)$ *and* $C = \{X \mid X \in \{0,1\}^n \wedge X \cdot K \leq b\}$*, where* $P = (p_1, p_2, p_3, p_4)$ *and* $K = (2, 1, 1, 1)$ *represents the value and costs of the projects, respectively. Available units of funding are given by* $b = 2$ *and the final decision is encoded using* X*, where* x_i *denotes if the ith project has been selected for funding.*

The true value of the projects constitute the parameter vector P*. The attribute vector for the ith project is a pair representing the novelty of the project* n_i *and the experience of the investigator* e_i*, i.e.,* $A_i = (n_i, e_i)$*. Based the attributes, an estimated value* $v'_i = f(A_i)$ *is computed.*

The committee could have considered two learned functions for the attribute vectors $A_i = (n_i, e_i)$*:* $f_1(A_i) = 5 \cdot n_i + e_i$ *and* $f_2(A_i) = 3 \cdot n_i + e_i$*. The function* f_2 *results in estimates* $P'_2 = (7, 5, 3.5, 1)$*, leading to a solution that selects* p_2 *and* p_3 *and has zero regret given true parameter values* $P = (14, 11, 12, 10)$*, as opposed to using* f_1*.* □

In the following, we assume linear learning functions, i.e., $f_\alpha(A) = \alpha \cdot A$, which are widely used in machine learning, e.g., regression and linear support vector machines. The ease of interpreting these functions make them suitable for a variety of explainable AI applications (Azizi et al. 2018).

Similar definitions have appeared in literature, e.g., defining predict+optimise in terms of *expected regret* (Demirović et al. 2019b) and considering linear programs (Elmachtoub and Grigas 2017; Wilder, Dilkina, and Tambe 2019).

Related Work

Predict+optimise can be partitioned into three groups. **Indirect** methods use standard learning methods and loss functions that are independent of the optimisation problem. **Semi-direct** methods (Demirović et al. 2019b) design the loss function with respect to the optimisation problem but are otherwise similar in usage to indirect techniques, e.g., learning to classify items in knapsack problems as *desirable* and *not desirable* based on the optimisation result, which can offer advantages over conventional ranking approaches. **Direct** methods (Demirović et al. 2019a; Elmachtoub and Grigas 2017; Wilder, Dilkina, and Tambe 2019; Donti, Amos, and Kolter 2017) interact with the optimisation problem during training. The issue is that most modern machine learning algorithms rely on gradients, but the regret function in predict+optimise is nondifferentiable. As an alternative, convex surrogates of combinatorial problems are employed in training for which gradient descent techniques can be applied. The common theme of previous work is that the problem used in learning is *simplified* and not considered *directly*. The first technique to fully integrate the optimisation problem in its original form, rather than a relaxation, was based on transition points with linear learning functions (Demirović et al. 2019a), but it is restricted to ranking problems. In this work, we extend this framework by including problems amenable to dynamic programming.

Knowledge-compilation approaches based on binary- and multi-decision diagrams (Berman, Cire, and van Hoeve 2016; Hadzic et al. 2008; de Uña et al. 2019) compactly represent *all* solutions. In a sense, these approaches encode the dynamic programming structure. One could draw the connection to our work, which represents *optimal* solutions as a *function of a parameter* using dynamic programming.

Lagrangian relaxation computes the *dual* of a combinatorial problem as a piecewise linear function to compute a *lower/upper bound*, whereas our approach *precisely* represents the *behaviour of optimal solutions as the learning coefficient changes* with the aim of computing the *regret*.

Preference elicitation (Dragone, Teso, and Passerini 2018; Teso, Passerini, and Viappiani 2016) is concerned with interactively learning to synthesize structured objects from data. Our setting bears similarities, e.g., both frameworks learn a linear function whilst minimising a related notion of regret. *Solutions* and *weights* in preference elicitation correspond to *attribute vectors* and α. However, there are notable differences. Incremental preference elicitation considers querying the user, uses a single optimisation problem, and considers pairwise comparisons with additional constraints. In contrast, our setting assumes a fixed dataset, simultaneously optimises multiple problems with the same α, uses a test set in addition to a training set, and our approach deals with optimisation problems solvable by dynamic programming. In both settings, the predictions are used in the objective, but our work is concerned with learning the weights of the objective on a per-instance basis.

Parameterised Optimisation Problems

We introduce *parameterised optimisation problems*, where the coefficients in the objective are functions rather than constants. This notion plays an important role in our method, as predict+optimise can be seen as the problem of selecting the best parameter of a parameterised optimisation problem that leads to a solution that minimises regret.

For a set of linear functions $g_i(\alpha) = s_i \cdot \alpha + c_i$ and their corresponding vector $G(\alpha) = (g_1(\alpha), g_2(\alpha), ..., g_n(\alpha))$, we

consider the family of combinatorial optimisation problems, parameterised by $\alpha \in \mathbf{R}$:

$$COP(\alpha) = \max_{X \in C} obj(X, G(\alpha)). \quad (4)$$

The variables X appearing in the objective are assumed to be linear with respect to $G(\alpha)$. This does not limit the objective to additively separable objective functions, because for instance min/max functions are supported, but quadratic functions are not included. These restrictions are imposed by the limitations of our algebra (see next section).

For a *fixed* $\alpha \in \mathbf{R}$, we obtain a standard optimisation problem as defined in the preliminary section, which can thus be solved using conventional optimisation techniques.

Notice that varying α has an impact on the resulting objective value, but it does not necessarily lead to a change in solution, i.e., the assignment of X.

Our aim is to understand the behaviour of solutions with respect to the parameter α. We would like to compute the set of *transition points* $\alpha_i \in \mathbf{R}$ for which $COP(\alpha_i - \epsilon)$ and $COP(\alpha_i + \epsilon)$ have different solutions for some infinitesimal $\epsilon > 0$. This would allow us to represent the underlying solution structure, as the solutions to the problem do not change for parameter values that lie in between the transition points.

The transition points capture the structure of the problem, since the solution changes in the neighbourhood of these points, e.g., in Figure 1, $\alpha = 2$ is a transition point as $COP(1.99)$ and $COP(2.01)$ have different solutions. These points play a critical role in enabling machine learning to directly reason about the combinatorial problem.

Example 1 (continued)**.** *The project-funding problem, parametrised by the novelty weight α, can be written as:* $COP(\alpha) = \max_X G(\alpha) \cdot X = \max_X g_1(\alpha) \cdot x_1 + g_2(\alpha) \cdot x_2 + g_3(\alpha) \cdot x_3 + g_4(\alpha) \cdot x_4 = \max_X (-\alpha + 10) \cdot x_1 + (\alpha + 2) \cdot x_2 + (-0.5\alpha + 5) \cdot x_3 + (2\alpha - 5) \cdot x_4$*, where* $C = \{(x_1, x_2, x_3, x_4) \mid 2x_1 + x_2 + x_3 + x_4 \leq 2 \wedge x_i \in \{0, 1\}\}$.

For $\alpha = 0$, we have $COP(0) = \max_X 10x_1 + 2x_2 + 5x_3 - 5x4$. Thus, the solution is $(x_1, x_2, x_3, x_4) = (1, 0, 0, 0)$ with the objective of 10. The solution is the same for $COP(1)$, although the objective value is 9.

Figure 1 shows the behaviour of the project-funding example as the parameter α changes. The graph displays the objective value for every α. The transition points are 2 and 4. Each $\alpha \in (-\infty, 2)$ has the solution $(x_1, x_2, x_3) = (1, 0, 0, 0)$. Similarly, for $\alpha \in (2, 4)$, the solution is $(x_1, x_2, x_3, x_4) = (0, 1, 1, 0)$, and $\alpha \in (4, \infty)$ has the solution $(x_1, x_2, x_3, x_4) = (0, 1, 0, 1)$. □

Dynamic Programming

We represent the solution structure of parameterised optimisation problems using piecewise linear functions. Such a representation will be used to determine the best parameter value in predict+optimise. For problems solvable by dynamic programming, the transition points can be computed.

Standard dynamic programming algorithms require the coefficients of the objective to be precisely stated as numbers, based on which a dynamic programming table is built. To use such algorithms for our parameterised combinatorial optimisation problems, the key is to *generalise* the coefficients and the table computation to *linear functions* rather than integers or reals. For example, in our setting, the *i*th coefficient is specified as linear function $f_i(\alpha) = s_i \cdot \alpha + c_i$. The classical optimisation formulation is obtained by setting $s_i = 0$. We generalise standard operations on numbers, such as *addition*, *min*, and *max*, to linear functions by using piecewise linear functions, which are detailed below.

When combining dynamic programming algorithms with piecewise linear functions, we can compute the dependency between the parameter α and the solutions of parametrised optimisation problem $COP(\alpha)$. As a result, we obtain a piecewise linear function that can be queried to provide the solution and objective value of $COP(\alpha)$. More importantly, the resulting function provides the desired transition points. The key observation is that the solution does not change for α values that lie *in between* transition points. We proceed to describe our algebra that generalises standard operations on numbers and follow up with a detailed example.

An Algebra of Piecewise Linear Functions

We represent the solution structure of parameterised optimisation problems using piecewise linear functions. Before detailing our approach, we describe our algebra on piecewise linear functions used throughout this paper. This can be seen as a special case of the general piecewise function algebra (Mohrenschildt 1998) with specialised algorithms to ensure a compact representation.

Definition 1. *A piecewise linear function $f_I^F : \mathbb{R} \to \mathbb{R}$ is represented as a tuple $(\mathbf{I}, \mathbf{F})$, where $\mathbf{I}$ is a set of nonoverlapping contiguous intervals and $\mathbf{F}$ is a set of linear functions $f_i : \mathbb{R} \to \mathbb{R}$ of the form $f_i(\alpha) = s_i \cdot \alpha + c_i$. Each interval $I_i \in \mathbf{I}$ is associated with exactly one $f_i \in \mathbf{F}$, denoted as $\mathbf{F}(I_i) = f_i$. The value of f_I^F at a point $\alpha \in \mathbb{R}$ is given as $f_I^F(\alpha) = f_i(\alpha)$, where $\alpha \in I_i \in \mathbf{I}$ and $\mathbf{F}(I_i) = f_i$.*

Example 2. *Figure 1(a) shows the piecewise linear function with intervals $\mathbf{I} = \{I_1, I_2, I_3\}$, $I_1 = (-\infty, 2)$, $I_2 = [2, 4)$, $I_3 = [4, \infty)$, $\mathbf{F}(I_1)(\alpha) = -\alpha + 10$, $\mathbf{F}(I_2)(\alpha) = \frac{1}{2}\alpha + 7$, and $\mathbf{F}(I_3)(\alpha) = 3\alpha - 3$.* □

The interpretation of the (pointwise) operations of addition and maximum on piecewise linear functions is analogous to real numbers and map to a piecewise linear function. We describe the operation *max* in detail and sketch the *addition* operation. Both operations are linear in their input.

Max: Algorithm 1. The set $\mathbf{I^T}$ is computed by intersecting intervals from the two input piecewise linear functions. For each newly computed interval, the two linear functions corresponding to the linear functions defined on the given interval by the input functions are considered. If the intersection point m of these functions is not in the given interval, then one of the two functions is greater along the interval. The interval is thus assigned to the greater function. Otherwise, the interval is split into two parts according to the intersection point m. Each of the linear function achieves a greater value than the other only on one part. The appropriate functions are identified and mapped to the intervals.

Addition: As in the *max* operation, the set of intersection intervals $\mathbf{I^T}$ is computed. Each newly computed interval is

Algorithm 1: Compute the max of two piecewise linear functions

input: Piecewise linear functions $f^{\mathbf{F_1}}_{\mathbf{I_1}}(\alpha)$ and $f^{\mathbf{F_2}}_{\mathbf{I_2}}(\alpha)$
output: Piecewise linear function $f^{\mathbf{F_3}}_{\mathbf{I_3}}(\alpha) \leftarrow \max(f^{\mathbf{F_1}}_{\mathbf{I_1}}(\alpha), f^{\mathbf{F_2}}_{\mathbf{I_2}}(\alpha))$

begin
  $\mathbf{I^T} \leftarrow \{I_i \cap I_j \mid I_i \in \mathbf{I_1} \wedge I_j \in \mathbf{I_2}\}$
  $\mathbf{F_3} \leftarrow$ empty mapping
  $\mathbf{I_3} \leftarrow \emptyset$
  for $I_t \in \mathbf{I^T}$ **do**
    $f_1(\alpha) \leftarrow \mathbf{F_1}(I_t) = s_1 \cdot \alpha + c_1$
    $f_2(\alpha) \leftarrow \mathbf{F_2}(I_t) = s_2 \cdot \alpha + c_2$
    $m \leftarrow$ the point such that $f_1(m) = f_2(m)$
    if $m \notin I_t$ *or* m *does not exist* **then**
      $f_3 \leftarrow$ the function that has greater value in the interval I_t among f_1 and f_2
      $\mathbf{I_3} \leftarrow \mathbf{I_3} \cup I_t$
      $\mathbf{F}_3(I_t) \leftarrow f_3$
    else
      $I_a \leftarrow \{x \mid x < m \wedge x \in I_t\}$
      $I_b \leftarrow \{x \mid x \geq m \wedge x \in I_t\}$
      $f_a \leftarrow$ the function that has greater value at $x \in I_a$ among f_1 and f_2
      $f_b \leftarrow$ the function that has greater value at $x \in I_b$ among f_1 and f_2
      $\mathbf{I_3} \leftarrow \mathbf{I_3} \cup \{I_a, I_b\}$
      $\mathbf{F_3}(I_a) \leftarrow f_a$
      $\mathbf{F_3}(I_b) \leftarrow f_b$
  return $f^{\mathbf{F_3}}_{\mathbf{I_3}}$

mapped to a linear function representing the sum of the two input functions on the given interval.

Detailed Example

Our approach can be applied to any parametrised optimisation problem that, in its original non-parametrised form, is solvable by dynamic programming. We illustrate our approach on the knapsack problem, which corresponds to the project-funding problem used in the examples.

Consider the parameterised knapsack problem that corresponds to the problem from Example 1. We denote with $B(s, c)$ the piecewise linear function with a single interval $I_1 = (-\infty, \infty)$ and $\mathbf{F}(I_1)(\alpha) = s \cdot \alpha + c$. Assume each value p_i is given by the linear function $f_i(\alpha)$, the dynamic program can be rewritten as a construction of a piecewise linear function algebra expression $k(i, W)$ in a similar fashion as for the standard knapsack problem, but using linear functions rather than constants for the values of items:

$$k(i,W) = \begin{cases} B(0,0), & W = 0 \vee \\ & i = 0 \\ k(i-1,W), & w_i > W \\ \max(k(i-1,W), & w_i \leq W \\ \quad k(i-1,W-w_i) + f_i(\alpha)) & \end{cases} \quad (5)$$

Example 1 (continued)**.** *Consider the project-funding running example with four projects of weights* $2, 1, 1, 1$ *and profits* $-\alpha+10$, $\alpha+2$, $-\frac{1}{2}\alpha+5$ *and* $2\alpha-5$ *with a capacity limit* $W = 2$*. The overall problem is* $k(4, 2)$*. The functions representing the profits are* $f_1 = B(-1, 10)$*,* $f_2 = B(1, 2)$*,* $f_3 = B(-\frac{1}{2}, 5)$ *and* $f_4 = B(2, -5)$*. The equations for the piecewise linear functions are:*

$$\begin{aligned} k(4,2) &= \max(k(3,1) + f_4, k(3,2)) \\ k(3,2) &= \max(k(2,1) + f_3, k(2,2)) \\ k(3,1) &= \max(k(2,0) + f_3, k(2,1)) \\ k(2,2) &= \max(k(1,1) + f_2, k(1,2)) \\ k(2,1) &= \max(k(1,0) + f_2, k(1,1)) \\ k(2,0) &= k(1,0) \\ k(1,2) &= \max(k(0,0) + f_1, k(0,2)) \\ k(1,1) &= k(0,1)) \\ k(1,0) &= B(0,0) \\ k(0,\_) &= B(0,0). \end{aligned}$$

Clearly $k(1,1) = k(2,0) = B(0,0)$*. We can compute* $k(1,2) = \max(B(-1,10), B(0,0))$*. There is an intersection point at* $\alpha = 10$ *so we arrive at* $k(1,2) = \{(-\infty, 10) \mapsto -\alpha + 10, [10, \infty) \mapsto 0\}$*. Now* $k(2,1) = \max(B(1,2), B(0,0))$ *which has an intersection point at* $\alpha = -2$ *so we arrive at* $k(2,1) = \{(-\infty, -2) \mapsto 0, [-2, \infty) \mapsto \alpha + 2\}$*. Similarly* $k(2,2) = \max(B(1,2), k(1,2))$*. The first interval of* $k(1,2)$ *intersects* $\alpha+2$ *at* $\alpha = 4$*, the second interval* $B(1,2)$ *intersects at* $\alpha-2$ *outside* $[10, \infty)$*, in this segment* $B(1,2)$ *dominates. The result is* $k(2,2) = \{(-\infty, 4) \mapsto -\alpha + 10, [4, \infty) \mapsto \alpha + 2\}$*.*

To compute $k(3,1) = \max(B(-\frac{1}{2}, 5), k(2,1))$ *we find the first interval intersects at* $\alpha = \frac{5}{2}$ *outside the range* $(-\infty, -2)$ *and* $B(-\frac{1}{2}, 5)$ *dominates. The second interval intersects at* $\alpha = 2$ *before which* $B(-\frac{1}{2}, 5)$ *is better. We compute* $k(3,1) = \{(-\infty, 2) \mapsto -\frac{1}{2}\alpha+5, [2, \infty) \mapsto \alpha+2\}$*.*

Now $k(3,2) = \max(k(2,1) + B(-\frac{1}{2}, 5), k(2,2)$*. The first argument is* $\{(-\infty, -2) \mapsto -\frac{1}{2}\alpha+5, [-2, \infty] \mapsto \frac{1}{2}\alpha+7\}$*. We compute the intervals* $(-\infty, -2), [-2, 4), [4, \infty)$*. In the first interval* $k(2,2)$ *dominates, in the second there is an intersection at* $\alpha = 2$ *before which* $k(2,2)$ *dominates, in the third interval there is an intersection point at* $\alpha = 10$*. We arrive at* $k(3,2) = \{(-\infty, 2) \mapsto -\alpha + 10, (2, 10] \mapsto \frac{1}{2}\alpha + 7, (10, \infty) \mapsto \alpha + 2\}$

Finally $k(4,2) = \max(k(3,1) + B(2,-5), k(3,2))$*. The first argument is* $\{(-\infty, 2) \mapsto \frac{3}{2}\alpha, [2, \infty) \mapsto 3\alpha - 3\}$*. The intervals are* $(-\infty, 2), (2, 4], (4, 10], (10, \infty)$*. In the first* $-\alpha + 10$ *dominates, in the second* $\frac{1}{2}\alpha + 7$ *dominates and in the third and fourth* $3\alpha - 3$ *dominates. The final result is* $k(4,2) = \{(-\infty, 2) \mapsto -\alpha+10, (2, 4] \mapsto \frac{1}{2}\alpha+7, [4, \infty) \mapsto 3\alpha - 3\}$ *which is* $\mathbf{F}$ *from Example 2 and Figure 1(a).* □

Application to Predict+Optimise

Our piecewise linear function representation of parameterised combinatorial problems can be applied to predict+optimise, i.e., to compute the function over the attributes to minimise regret. The main idea is that such a representation of $COP(\alpha)$ explicitly stores the *transition points*, i.e., the values α_i where $COP(\alpha_i - \epsilon)$ and

$COP(\alpha_i + \epsilon)$ have different solutions for some small $\epsilon > 0$. Therefore, the solution of a $COP(\alpha)$ stays *fixed* for values of the parameter in between consecutive transition points. In predict+optimise, a change of a predicted solution is a sufficient condition for a change of regret. Thus, the regret can be *minimised* by considering a *single point* from *each* interval defined by the transition points. For Ex. 1, the regret on each interval is 23 minus the value illustrated in Figure 1(b). Recall that our piecewise linear function approach is only applicable to problems that admit dynamic programming.

The key is that the piecewise linear function representation enables us to directly reason over the combinatorial optimisation problem, rather than considering a relaxation of the problem as in previous works. The concept of transition points has been introduced in a *ranking* setting (Demirović et al. 2019a). In this work, we extend the framework to support problems amenable to dynamic programming.

Our algorithm for predict+optimise is summarised in Algorithm 2. It applies a steepest coordinate descent technique for each learning coefficient in $\alpha = (\alpha_1, ..., \alpha_m)$ one by one. The input is a predict+optimise problem, with an optimisation problem solvable by dynamic programming, and a linear learning function parametrised by the n-dimensional vector α. The algorithm starts from the initial coefficients α derived from linear regression and iteratively searches for an improving assignment. In each iteration, every component of the vector α, except one, is fixed to its current value. Therefore, the problem is effectively transformed into a parametrised optimisation problem. The *complete* set of transition points, which effectively define the intervals $\mathbf{I}$, is *efficiently* computed by representing the subproblem as a piecewise linear function $f^{\mathbf{F}}_{\mathbf{I}}$. The representative points Γ consist of a point from each interval in $\mathbf{I}$, e.g., transition points $\{5, 13\}$ yield the intervals $\mathbf{I} = \{[-\infty, 5), [5, 13), [13, +\infty)\}$, and $\Gamma = \{4, 9, 14\}$. The unfixed coefficient is assigned the point from Γ that minimises regret. This process is repeated until improvements are no longer possible or a timeout occurs. The algorithm generalises over multiple optimisation problems by expanding the intervals to consider all transition points and the sum of the regret of each instance.

In each iteration, the single-parameter optimisation problem is solved to optimality with respect to regret. Quantifying a global optimality gap is not straight-forward as the regret is nonlinear and noncontinuous, which prevents conventional analysis. Note that, intuitively, the algorithm terminates since in each iteration the algorithm searches for a strictly better solution.

Algorithm 2: Coordinate descent for predict+optimise using piecewise linear functions.

input: A predict+optimise problem with an optimisation problem (obj, C) amenable to dynamic programming and a training set $\{(A_i, p_i)\}$ for $i \in [1, 2, ..., n]$ and $A_i = (a_{i1}, a_{i2}, ..., a_{im})$

output: The coefficients α that lead to minimum regret for the learning function $f_\alpha(A) = \sum \alpha_i \cdot a_i$

begin
- $\alpha \leftarrow \arg\min_\alpha \{\sum (f_\alpha(A_i) - p_i)^2\}$, i.e. initialise using linear regression
- **while** *not coverged* $\wedge$ *resources remain* **do**
  - **for** $k \in [1, 2, ..., m]$ **do**
    - **for** $i \in [1, 2, ..., n]$ **do**
      - $c_i \leftarrow \sum_{j \in [1,2,...,m] \wedge j \neq k} (\alpha_j \cdot a_{ij})$
      - $f_i(\alpha_k) \leftarrow a_{ik} \cdot \alpha_k + c_i$
    - $F(\alpha_k) = (f_1(\alpha_k), f_2(\alpha_k), ..., f_n(\alpha_k))$
    - Let $f^{\mathbf{F}}_{\mathbf{I}}$ be the result of converting of $COP(\alpha_k) = \max_{X \in C} obj(X, F(\alpha_k))$ into a piecewise linear function using our dynamic programming approach
    - $\Gamma \leftarrow \{max(I_1) - 1\} \cup \{\frac{min(I_j) + max(I_j)}{2} \mid 2 \leq j < |\mathbf{I}|\} \cup \{min(I_{|\mathbf{I}|}) + 1\}$
    - $\alpha_k \leftarrow \min_{\gamma \in \Gamma} Regret(\gamma)$
- **return** α

Experimental Evaluation

The aims of this section are: (a) to illustrate the applicability of our approach by comparing with state-of-the-art for the predict+optimise knapsack; (b) to show the scalability of our approach on two dynamic programming problems: knapsack and shortest path for directed-acyclic graphs; and (c) to demonstrate the strength of our approach by constructing benchmarks where the linear relaxation is not an accurate indicator for the quality of the predictions. State-of-the-art approaches, which rely on a relaxation of the problem, cannot infer the correct dynamics. In contrast, our approach performs perfectly since it captures the connection between the predictions and the underlying optimisation problem.

Predict+Optimise Knapsack Problem

Benchmarks and data. The *predict+optimise knapsack* dataset is based on two years of real-life energy price data. The data was used in the ICON energy-aware scheduling competition and a number of publications (Dooren et al. 2017; Grimes et al. 2014). The goal is to predict energy prices based on weather conditions and day-ahead estimates. Benchmarks represent days and each optimisation parameter encodes the price for one half-hour. Item weights take values 3, 5, and 7. There are 37,872 benchmarks, 48 optimisation parameters per benchmark, and eight attributes. We refer to (Demirović et al. 2019b) for more details. Note the knapsack corresponds to the project-funding problem.

Learning methods We compare with the state-of-the-art. **(Indirect)** *kNN*, k-nearest neighbour; *Ridge* regression; *SVMRank*; **(direct)** *SPO*, smart predict then optimise (Elmachtoub and Grigas 2017); *QPTL*, quadratic programming task loss (Wilder, Dilkina, and Tambe 2019) **(semi-direct)** *SVMR-s*, learn-to-partition (Demirović et al. 2019b).

Methodology. Training and test sets are divided at a 70%-30% ratio. Our coordinate descent approach optimises one parameter at a time. For other methods, we performed 5-fold

Table 1: The entry (x, y) denotes the average regret for the training and test set. Our approach is labelled DP+PLF.

	Indirect			Semi-direct	Direct		
Capacity	kNN	ridge	SVMR	SVMR-s	SPO	QPTL	DP+PLF
25 (10%)	(⊥; 120)	(95; 94)	(97; 97)	(88; 89)	(126; 106)	(177; 142)	**(86, 89)**
75 (30%)	(⊥; 176)	(137; 147)	(145; 155)	(143; 155)	(259; 223)	(260; 237)	**(126, 141)**
125 (50%)	(⊥; 128)	(108; **109**)	(117; 121)	(124; 126)	(136; 113)	(236; 191)	(**103**, 112)

Table 2: Average runtime in seconds and number of transition points (#TP) for ten random knapsack and shortest path benchmarks. The parameter n denotes the number of items and nodes for the knapsack and shortest path problems.

	Knapsack		*Shortest path*	
n	**time**	**#TP**	**time**	**#TP**
10	0	6	0	7
50	2	28	0	19
100	25	52	2	36
150	95	83	6	43
200	303	101	18	64
300	1287	153	98	99

hyperparameter tuning with regret as the measure.

Comparison with the state-of-the-art. We set the capacity of the knapsack to 10%, 30% and 50% of the sum of item weights. In Table 1, each entry (x, y) represents the average regret for the training (x) and testing set (y). The best previous approach for this problem was ridge regression, an indirect method not exploiting the optimisation problem. Our approach uniformly outperforms other approaches on the training set, demonstrating the ability to reason directly with the combinatorial problem. The result generalises for the test set of the harder (smaller capacity) instances.

Scalability

We experiment with the predict+optimise knapsack and shortest path for directed-acyclic graphs. The aim is to show scalability. The item-values and edge-weights are the learnt parameters. Note that the knapsack problem is $\mathbf{NP}$-hard. Nevertheless, both problems admit a dynamic programming algorithm and thus our approach can be applied. Synthetic data to test scalability was generated as follows:

- Knapsack: The capacity is set to 15% of the total sum of the weights of the n items. Each weight is generated randomly from the interval $[1, 10]$.
- Directed-acylic graphs: we create an edge (i, j) with $i < j$ and $|i - j| \leq \lfloor \frac{n}{3} \rfloor$ with 50% chance, where i and j are indices of nodes. We compute the cost of the paths between all pairs of n nodes.

The attributes are $A_i = (a_{i1}, a_{i2})$, with $a_{i1}, a_{i2} \in [1, 360]$ and $p_i = 1000 \cdot sin(a_{i1})sin(a_{i2})$. The slopes and constants for the initial linear functions were chosen from the interval $[1, 100]$. Table 2 shows the runtime and number of transition points in the resulting piecewise linear functions. Our approach scales reasonably with the benchmark size. Faster runtimes can be achieved by optimising our implementation.

Combinatorial reasoning vs relaxations

Representing parametrised optimisation problems through piecewise functions reveals the solution structure. Thus, we may select the value of the parameter that directly leads to the desired solutions and minimise regret, rather than relying on accurately predicting the coefficients. This differs from other state-of-the-art approaches for predict+optimise which aim to approximate the underlying combinatorial problem. These relaxation schemes capture the problem structure to an extent, but may not be reliable for $\mathbf{NP}$-hard problems. In contrast, our approach can directly understand the underlying combinatorial problem.

To highlight the strength of our approach, we constructed simple knapsack benchmarks: each instance contains only *three* items. Our approach can perfectly solve the problems, whereas the state-of-the-art does not accurately capture the underlying structure despite the simplicity of the benchmarks. These benchmarks are not meant as realistic examples, but rather to illustrate the strength of reasoning directly over the combinatorial problem and shows the limitations of using relaxations for predict+optimise.

The baseline benchmark consists of three items with profits given as: $(p_1, p_2, p_3) = (4, 5, 7)$, the attributes are 2-D vectors: $(a_1, a_2, a_3) = ((1, 1), (2, 1), (5, 1))$, and the weights as given as $(w_1, w_2, w_3) = (5, 5, 6)$. The capacity is set to ten. The *i-th* benchmark is generated by multiplying the baseline profits and attributes vector by the integer i. We generate ten benchmarks by iterating $i \in [1, 2, ..., 10]$.

We tested the benchmarks using the state-of-the-art in a similar manner as in the previous knapsack section. All approaches, except ours, result in a non-zero regret. Linear regression attempts to minimise the squared error of the predictions. However, as the profits are not linear with respect to its attributes, this results in predicting the value of the third item to be greater than the sum of the other two. SVMRank-p divides the items into two partitions based on the linear relaxation and learns a linear ranking to separate the two classes. However, it incorrectly places the second and third item in the same partition, and thus the resulting predictions are not accurate. SPO computes a convex surrogate of the loss functions, which cannot precisely capture the loss for these benchmarks. Note that our method does not produce accurate predictions when compared to the true profits, but the predictions lead to the correct solutions.

Conclusion

We presented a novel method for predict+optimise. It combines dynamic programming and a piecewise linear function algebra to represent the solution structure of optimisation problems. Such a representation is used in our steepest coordinate algorithm, where a series of single-parameter predict+optimise problems are iteratively solved to optimality. The main advantage is that our approach can optimise the learning coefficients *directly* with respect to regret rather than use a relaxation of the optimisation problem. The experiments illustrate its effectiveness for predict+optimise.

References

Azizi, M. J.; Vayanos, P.; Wilder, B.; Rice, E.; and Tambe, M. 2018. Designing fair, efficient, and interpretable policies for prioritizing homeless youth for housing resources. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26-29, 2018, Proceedings*, 35–51.

Berman, D.; Cire, A.; and van Hoeve, W. 2016. *Decisions diagrams for optimization*. Springer.

de Uña, D.; Gange, G.; Schachte, P.; and Stuckey, P. J. 2019. Compiling CP subproblems to MDDs and d-DNNFs. *Constraints* 24(1):56–93.

Demirović, E.; Bailey, J.; Chan, J.; Gins, 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. https://www.ijcai.org/proceedings/2019/0151.pdf.

Demirović, E.; Guns, T.; Stuckey, P. J.; Bailey, J.; Chan, J.; Leckie, C.; and Kotagiri, R. 2019b. An investigation into prediction + optimization for the knapsack problem. In Rousseau, L.-M., and Stergiou, K., eds., *Proceedings of Sixteenth International Conference on Integration of Artificial Intelligence and Operations Research techniques in Constraint Programming (CPAIOR2019)*, number 11491 in LNCS, 241–257. Springer.

Donti, P. L.; Amos, B.; and Kolter, J. Z. 2017. Task-based end-to-end model learning in stochastic optimization. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017)*, 5484–5494.

Dooren, D. V. D.; Sys, T.; Toffolo, T. A. M.; Wauters, T.; and Berghe, G. V. 2017. Multi-machine energy-aware scheduling. *EURO J. Computational Optimization* 5(1-2):285–307.

Dragone, P.; Teso, S.; and Passerini, A. 2018. Pyconstruct: Constraint programming meets structured prediction. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, 5823–5825. https://www.ijcai.org/proceedings/2018/0850.pdf.

Elmachtoub, A. N., and Grigas, P. 2017. Smart "predict, then optimize". Technical report. https://arxiv.org/pdf/1710.08005.pdf.

Grimes, D.; Ifrim, G.; O'Sullivan, B.; and Simonis, H. 2014. Analyzing the impact of electricity price forecasting on energy cost-aware scheduling. *Sustainable Computing: Informatics and Systems* 4(4):276–291. Special Issue on Energy Aware Resource Management and Scheduling (EARMS).

Hadzic, T.; Hooker, J. N.; O'Sullivan, B.; and Tiedemann, P. 2008. Approximate compilation of constraints into multivalued decision diagrams. In Stuckey, P. J., ed., *Proceedings of the Fourteenth International Conference on Principles and Practice of Constraint Programming, CP2008*, 448–462. Springer.

Mohrenschildt, M. V. 1998. A normal form for function rings of piecewise functions. *Journal of Symbolic Computation* 26(5):607 – 619.

Teso, S.; Passerini, A.; and Viappiani, P. 2016. Constructive preference elicitation by setwise max-margin learning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, 2067–2073.

Wilder, B.; Dilkina, B.; and Tambe, M. 2019. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*, 1658–1666. https://doi.org/10.1609/aaai.v33i01.33011658.