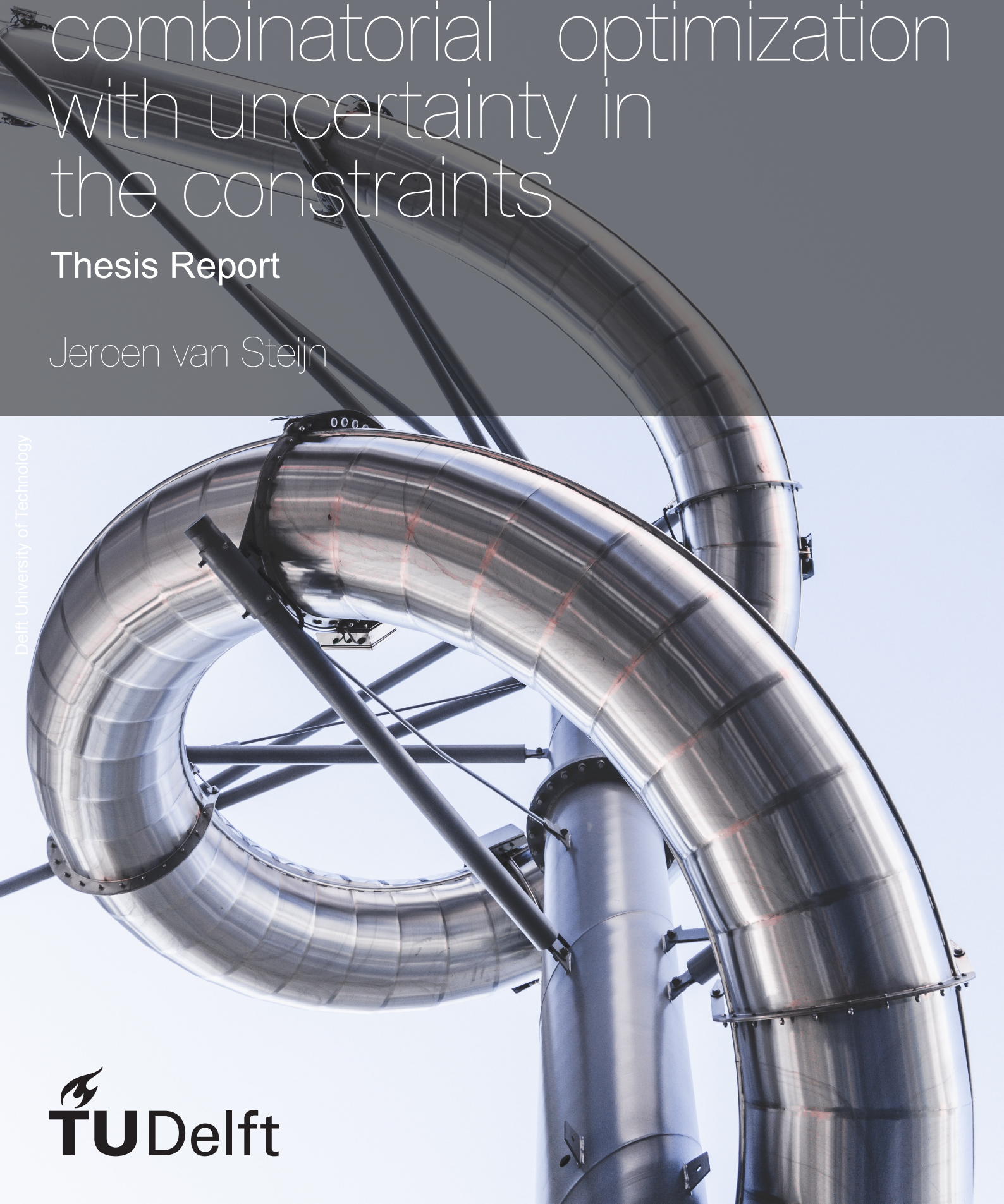
Predict+optimize for combinatorial optimization with uncertainty in the constraints

Thesis Report

Jeroen van Steijn



Delft University of Technology

TU Delft

Predict+optimize for combinatorial optimization with uncertainty in the constraints

Thesis Report

by

Jeroen van Steijn

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday November 18th, 2022 at 2:00 PM.

Thesis Supervisor:	Prof. Dr. M.M. de Weerdt
Daily Supervisor:	K.C. van den Houten
Thesis Committee Member:	Dr. D.M.J. Tax
Project Duration:	March, 2022 - November, 2022
Faculty:	Faculty of Electrical Engineering, Mathematics and Computer Science, Delft

TU Delft

Preface

This thesis is submitted as the final step in obtaining my Master of Science degree in Computer Science at the Delft University of technology. The project was started in light of the AI4B.io Lab, a collaboration between the Royal DSM and TU Delft.

I would first like to thank Mathijs de Weerdt, my thesis supervisor, for his guidance and supervision during the project. You always challenged me to look at problems from a different perspective, and your feedback on my approaches and analysis was invaluable.

Many thanks go out to Kim van den Houten, whom I had the pleasure to have as my daily supervisor. Our discussions always provided me with great insights into the relevance of the problem domain, and I hope our collaboration can contribute to further research in your academic career. Your enthusiasm, great support, and honest feedback have helped make this thesis what it is.

Finally, I would like to thank everyone in the Algorithmics group who participated in a series of discussions on the topic of predict+optimize, as this helped me think about the problem differently and motivated me to find ways to improve the state-of-the-art solutions.

Jeroen van Steijn
Delft, November 2022

Abstract

In this work, it is investigated whether the predict+optimize framework could be utilized for combinatorial optimization problems with a linear objective that have uncertainty in the constraint parameters, such that it outperforms prediction-error-based training. To this end, a predict+optimize formulation of the 0-1 knapsack problem is used, which is an NP-hard combinatorial optimization problem. This formulation was chosen because it has many real-world applications where the constraint parameters are uncertain. In this instance, the challenge is to predict the weights of knapsack items based on feature data and then use these predictions to solve the knapsack problem. The key is that the item weights are uncertain and thus must be estimated, but the quality of the solution is evaluated with respect to the true weights. This uncertainty in the weights can lead to infeasible solutions when evaluated on the true weights. Standard predict+optimize techniques do not account for such infeasibility of candidate solutions, which makes them mostly inapplicable for solving problems with uncertainty in the constraint parameters. To overcome this shortcoming of standard predict+optimize techniques, several correction methods for the evaluation of infeasible solutions during training are compared. Crucially these correction methods can be varied between training and evaluation. By penalizing infeasible solutions linear in the weight of the overweight items during training, predict+optimize is able to outperform standard prediction-error-based techniques. This performance increases to an extent with increased penalization factors, leading to more stable, more optimal results on the validation set.

Contents

1 Introduction

In classical combinatorial optimization, it is assumed that all parameters, or their distribution, are perfectly known prior to optimization. In reality, these are frequently only estimated based on feature data such as expert judgments or historical data. In cases where this feature data is only weakly correlated, the true parameter values can not be perfectly predicted. These predicted parameter values are then used for the optimization algorithm, which can result in a sub-optimal solution when evaluated using the true parameter values. An example of such a problem is the problem of machine scheduling, in which the aim is to optimize a schedule based on the predicted (uncertain) processing time for each task. In many instances, these uncertain processing times are correlated with the features of the environment at the time, such as humidity or temperature for biological processes, as well as features specific to the task, such as the amount and type of product to be produced. The objective is then to use such feature data to make predictions that, when used in the optimization algorithm, lead to the best-performing solution on the true parameter values.

Other optimization problems with uncertainty specifically in the constraint parameters are encountered in areas such as cargo loading. In cargo loading, allocations for items have to be made under a maximum weight constraint. These weights may in some instances only be approximations based on feature data. Traditional approaches to this problem consist of two phases. First, machine learning is used to estimate the uncertain parameter values. This machine learner is trained to minimize the prediction error. Then, in the second phase, optimization is done using the point estimates of the parameters to find an appropriate allocation. However, learning accurate parameter values by minimizing the prediction error does not always lead to solutions of better quality when these predictions are used in the optimization problem. This is because not all errors in the prediction of the parameters have an equal effect on the optimization problem, but the prediction error measures do not take this into account (Elmachtoub & Grigas, 2017). For instance, a small under-prediction in the weight of an item can make the transport vehicle overloaded and may break the transport vehicle. Reversely, a large over-prediction of the weight of an already large item might not influence the cargo allocation at all.

The goal is thus to incorporate the effect of misprediction on the optimization problem into the prediction step. This could be done by using it as a loss function for the machine learner. Due to the discrete nature of the combinatorial optimization problem, this loss function is not differentiable. This means that machine learners can not learn through gradient descent methods that are used with loss functions such as the mean squared error. Various predict+optimize methods have been studied to overcome this problem by approximating the gradient (Elmachtoub & Grigas, 2017; Mandi et al., 2019). These methods have been evaluated exclusively on combinatorial problems with uncertainty in the objective (Mandi et al., 2019). This thesis aims to extend this method to combinatorial optimization problems with uncertainty in the constraint parameters. This is investigated on the 0-1 knapsack problem, a fundamental combinatorial optimization problem. The 0-1 knapsack problem was chosen for its simplistic nature, whilst still being able to capture complex problems suitable for exploring the use of predict+optimize techniques in combinatorial optimization under uncertainty (Demirović et al., 2019).

This work falls into the broader research area of integrating machine learning and combinatorial optimization. Research in this area has focused exclusively on problem settings with uncertainty in the objective, using methods such as branch and bound approaches that integrate machine learners into the search (He et al., 2014), graph neural networks that are used to solve combinatorial optimization problems (Schuetz et al., 2021), or the earlier mentioned predict+optimize methods (Elmachtoub & Grigas, 2017; Mandi et al., 2019). This thesis aims to develop machine learning algorithms specifically designed for combinatorial optimization problems where the constraint parameters are uncertain and can be estimated with machine learning using feature data. This is done by extending the usage of predict+optimize methods to allow them to be used for combinatorial optimization problems with uncertainty in the constraints.

The research question for this project is: 'Can the machine learner in the predict+optimize framework be trained on an accurate reflection of the decision error for combinatorial optimization problems that involve uncertainty in the constraint parameters, such that it outperforms two-stage prediction-error-based methods?

Sub-questions, in order to answer this research question, are:

1. How should candidate solutions produced by predict+optimize which are infeasible given the realized constraint parameters be evaluated, such that predict+optimize can outperform existing prediction-error-based methods for problems with uncertainty in the constraints?
2. Is there an advantage to using a different evaluation of infeasible candidate solutions during training than the evaluation method that is given by the problem context?
3. How does the strength of the correlation between the feature data and the true parameters influence the performance of predict+optimize methods and prediction-error-based methods for problems with uncertainty in the constraint parameters?

This document first provides a theoretical background in Chapter 2. Then, in Chapter 3 the problem description is formally defined. In Chapter 4, the novel method for applying predict+optimize to problems with uncertainty in the constraints is described. Chapter 5 describes the experiments and their results. Finally, a conclusion and discussion of future work are provided in Chapter 6.

2

Background

This chapter discusses the relevant topics from the literature in order to provide sufficient background knowledge on combining machine learning and combinatorial optimization in combinatorial optimization problems with uncertainty in the constraint parameters. Information on current approaches to integrating machine learning and combinatorial optimization is provided, as well as some knowledge required for applying predict+optimize specifically to problems with uncertainty in the constraint parameters.

Because this work investigates how to optimize combinatorial optimization problems with uncertainty in the constraint parameters, Section 2.1 formally defines combinatorial optimization problems. Then, the 0-1 knapsack problem is defined as an instance of such a combinatorial optimization problem. As said in the introduction, the 0-1 knapsack problem was chosen for the experiments in this work.

The classic two-stage approach to solving such combinatorial optimization problems with uncertain parameters is discussed in Section 2.2. This approach is currently often used in practice to solve combinatorial optimization problems with uncertainty in the constraint parameters, and can therefore be used as a baseline. In this approach, a machine learner is trained to predict the uncertain parameter values, after which this machine learner's predictions are used in the optimization algorithm.

Section 2.3 introduces SPO (Smart Predict, Then Optimize), which is a technique that integrates the optimization step into the training of the machine learner. Such techniques are often described as "predict+optimize". As the novel method of this work is to apply predict+optimize techniques to combinatorial optimization with uncertainty in the constraints, understanding how predict+optimize is currently applied to problems with uncertainty in the objective function parameters is a crucial first step.

Next, the concept of gradient descent is introduced in Section 2.4. Because gradient descent is utilized in the two-stage approach and in SPO, understanding the concept of gradient descent and its limitations is relevant to this research. Further details on the predict+optimize framework are provided in Section 2.5 and Section 2.6. Section 2.5 contains a comparison of SPO and other predict+optimize methods, in order to understand some crucial differences between SPO and other methods which can help in the analysis of the results of the experiments. Section 2.6 then describes methods that aim to improve the performance and scalability of SPO for complex NP-Hard problems, in order to show that these methods are applicable to many complex real-world problems.

After that, Section 2.7 provides some examples of the evaluation of infeasible solutions to combinatorial optimization problems. This is relevant specifically because this work investigates problems with uncertainty in the constraint parameters. When the constraint parameters are under-predicted, this means that the solutions based on the predicted parameters can be infeasible when applied to the realized parameter values. Thus, methods of evaluating the objective value of such infeasible solutions are relevant.

Finally, Section 2.8 shows a distinction between epistemic and aleatory uncertainty. It further describes how the type of uncertainty of an uncertain constraint parameter influences typical predict+optimize implementations so that it can be understood for which types of uncertain parameters predict+optimize methods are most suited. This knowledge is especially relevant with regard to the future work proposal presented in Section 6.3.

2.1. Combinatorial optimization and the 0-1 Knapsack problem

As this work aims to optimize combinatorial optimization problems with uncertainty in the constraint parameters, it is useful to first formally define combinatorial optimization problems.

Combinatorial optimization consists of finding an optimal solution from a finite set of candidate solutions, where the set of feasible solutions is discrete and subject to a set of constraints. In many combinatorial optimization problems, exhaustive search is not tractable, since many combinatorial optimization problems are NP-Complete. Combinatorial optimization with a linear objective may be formulated as follows:

$$\begin{gathered} \text{argmax}_X \, W \cdot X \; s.t. \; X \in \mathcal{C} \\ X \in \{0,1\}^n, W \in \mathbf{R}^n \end{gathered} \tag{2.1}$$

The set $\mathcal{C}$ represents the set of feasible solutions given the constraints of the optimization problem. The input is given as W, $\mathcal{C}$, and the goal is then to compute the assignment of n binary variables (X) to maximize (or minimize) the weighted linear sum $W \cdot X$.

One specific example of a combinatorial optimization problem is the 0-1 knapsack problem, an NP-Hard combinatorial optimization problem, which is used in this thesis for its simplicity in implementation and ability to generalize concepts to more complex problems. The 0-1 knapsack problem is defined as follows: given the assignment of items X, each item with a weight w and a value v, determine which items to include so that the total weight is less than or equal to the capacity constraint and the total value is maximized.

To make the formulations in Formula 2.1 specific to the knapsack problem, the following problem is formulated:

$$\begin{aligned} K(v, w, n, c) = \underset{X}{\text{argmax}} & \sum_{i=1}^{n} X_i \cdot v_i \\ s.t. & \sum_{i=1}^{n} X_i \cdot w_i \leq c \end{aligned} \tag{2.2}$$

where:

v_i = the value for item i
w_i = the weight for item i
n = the number of items
c = the capacity of the knapsack

2.2. The two-stage prediction-error-based approach

In order to deal with uncertain constraint parameters for combinatorial optimization problems such as described in Section 2.1, two challenges arise: prediction of the uncertain parameters and optimization of the problem, given these parameters. These two challenges are currently approached using what is called the 'two-stage' approach. First, a machine learning model is trained to predict uncertain parameter values using correlated feature data. This training is done using the prediction error, which is evaluated with a loss function such as the mean squared error (MSE) loss. In the second stage, this machine learner is used to predict the parameter values, which are used for solving a combinatorial optimization problem. The outcome of this optimization is used as the planning, which is executed after the optimization on the realized value of the uncertain parameters.

Machine learning (ML) involves the learning of patterns underlying given data, such that good predictions can be made on novel data using an ML model M_γ, with γ the internal weights of the ML model. If these predictions are numeric, this prediction is a regression; Here, the model M_γ is allowed to alter γ during training such that a loss function $L(M_\gamma(f), \theta)$ is minimized, with $\hat{\theta} = M_\gamma(f)$ the predicted values for the samples with feature data f and θ the realized values. Typically in regression problems, the MSE loss is used as the loss function and is defined in Equation 2.3.

$$MSE = \sum_{i=1}^{n} (\theta_i - \hat{\theta}_i)^2 \tag{2.3}$$

2.3. Smart Predict, Then Optimize

As previously stated in Section 2.2, combinatorial optimization under uncertainty consists of two challenges: prediction of the uncertain parameters and optimization of the problem, given these parameters. Whilst machine learning tools often handle prediction by minimizing the prediction error; this does not consider the effects of the prediction on the outcome of the optimization problem (Elmachtoub & Grigas, 2017). This is where the 'Smart Predict, Then Optimize (SPO)' framework comes in. It defines a new loss function for training machine learners, SPO loss, which considers the decision error of using the predicted values in the optimization stage.

The SPO loss, or regret, is calculated by solving the combinatorial optimization with the predicted values and then evaluating the resulting solution on the true values to calculate the objective value of the 'plan' made by optimizing on the predicted values. The resulting objective value is then compared to the objective value of solving the combinatorial optimization without uncertainty (i.e. with the true values). The difference between these two is the regret. By training on this SPO loss, the machine learner should learn to optimize its predictions such that it minimizes the decision error of the end-to-end predict+optimize problem (Elmachtoub & Grigas, 2017). The SPO loss function is defined as:

$$\ell_{spo}(\theta, \hat{\theta}) = \theta^T w^*(\hat{\theta}) - z^*(\theta) \tag{2.4}$$

In which θ are the actual parameter values, $\hat{\theta}$ are the predicted parameter values by the machine learner, w^* is the optimization algorithm (for example a MIP solver) and z^* is the value of the optimization problem. Note that w^* is often expected to provide a unique optimal solution for any $\hat{\theta}$ (Elmachtoub & Grigas, 2017).

For each instance, the model should predict parameters $\hat{\theta}$. These parameters are then used for solving the optimization problem. The optimal solution to this optimization problem for the predicted values is then used on the real values. The SPO loss is aimed to be minimized over all instances. The problem thus becomes to find a model M that can, for all features f, find the output that minimizes the expected SPO loss over the instances. This model should generalize to a validation set on which no training of the model has been done.

2.4. Gradient descent

The two-stage prediction-error-based approach of Section 2.2 and the SPO method described in 2.3 optimize combinatorial optimization problems with uncertain constraint parameters using machine learning models that are trained using gradient descent. Gradient descent is an iterative optimization algorithm for finding a minimum of a differentiable function. The idea is to take repeated steps in the opposite direction of the gradient of the function at the current point, in order to find the lowest point of the function. In a minimization problem with a differentiable objective function, this procedure thus aims to minimize the cost (Zhang, 2019).

In machine learning, standard gradient descent computes the gradient of the loss function (L) with regard to the entire training set (θ) using differentiation of the loss function. Then the model weights (γ) are updated by taking the old weights (γ_{n-1}) and subtracting the gradient in steps. The size of these steps is decided using a learning rate hyperparameter (η).

$$\gamma_n = \gamma_{n-1} - \eta \cdot \nabla_\gamma L(M_{\gamma_{n-1}}, \theta) \tag{2.5}$$

As it requires a calculation of the gradients for the whole data set to perform just one update, standard gradient descent can be very slow. Standard gradient descent is guaranteed to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces (Ruder, 2016).

2.4.1. Stochastic gradient descent

Gradient descent can converge to a local minimum when optimizing non-convex functions and develop in zig-zag patterns, resulting in slow convergence. An often-used technique to combat these problems is stochastic gradient descent. Stochastic gradient descent is a stochastic approximation of gradient descent. This approximation is made by calculating the gradient on only a part of the data set, which in addition to increasing convergence speed also reduces computational complexity (Ruder, 2016). The Adam optimizer extends the work in stochastic gradient descent with an adaptive learning rate, which is individual per parameter. Since the learning rate parameter is adaptive, Adam requires less hyperparameter tuning than methods such as stochastic gradient descent. The adaptive tuning of the learning rate per parameter makes Adam perform better for non-stationary objectives and problems with very noisy or sparse gradients (Kingma & Ba, 2014).

2.4.2. Gradient descent in predict+optimize

Training a machine learner to minimize SPO loss comes with some challenges. One challenge is that the loss function is in most cases not differentiable, thus making it more difficult to train a machine learner. To overcome this the work from Elmachtoub and Grigas (2017) defines a more tractable upper bound of the SPO loss called SPO+ loss.

Remember that the SPO loss is defined as $\theta^T w^*(\hat{\theta}) - z^*(\theta)$, where $\hat{\theta}$ are the predicted parameters, w is an optimizer which provides a solution based on the predicted parameters, and $z^*(\theta)$ is the optimal solution given the true parameter values. $\theta^T w^*(\hat{\theta})$ is then the objective value when the optimization based on the predicted values is evaluated on the true values. This loss function is not differentiable and not convex due to the discrete nature of the underlying combinatorial optimization problem. Through duality theory, Elmachtoub and Grigas (2017) arrive at the SPO+ loss, which is a more tractable and convex upper bound of the SPO loss. Whilst the SPO+ loss is still not differentiable, Elmachtoub and Grigas (2017) show that there is a differentiable sub-gradient for the SPO+ loss: $w^*(\theta) - w^*(2\hat{\theta} - \theta)$. In essence, this evaluates the cost of the solution when optimizing on the true parameters and the cost of the solution when a convex combination of the true parameters and the predicted parameters ($2\hat{\theta} - \theta$) is used to find a solution. This differentiable sub-gradient allows for the optimization of the machine-learning model through gradient descent. Potential downsides of this are that $w^*(\theta) - w^*(2\hat{\theta} - \theta)$ can get stuck in local optima, and that it depends on the smoothness of the objective function in order to be able to find more optimal predictions that minimize the regret. Whilst the Adam optimizer could overcome some of these local optima due to its use of stochasticity; it is still not guaranteed to find optimal results for such objectives (Kingma & Ba, 2014).

2.5. Approximate and exact methods in predict+optimize

Besides the SPO+loss-based training described in Section 2.4.2, multiple methods that can train a machine learner to minimize regret have been proposed. These can be divided into two categories: approximate methods and exact methods. Approximate methods use approximations of the gradient (like the sub-gradient SPO+ loss) or continuous relaxations of the discrete combinatorial optimization problem in order to train a machine-learning model. Two notable examples are QPTL and IntOp. QPTL relaxes the objective in constrained optimization to be able to differentiate it (Wilder et al., 2018), whereas IntOp utilizes an interior point based approach (Mandi & Guns, 2020). The largest difference between SPO and these other approaches is that SPO can utilize any black-box solver as an optimization oracle, instead of relying on specific structures in the solver (Elmachtoub & Grigas, 2017; Mandi & Guns, 2020).

In contrast, exact approaches exploit the problem structure of the combinatorial optimization problem and try to find a model through search methods such as divide and conquer. These methods directly lower the regret instead of using an approximation. The downside of the exact approaches is that they are limited in applicability (only problems of a specific structure can be solved), require implementation specific to each problem, and are more computationally expensive (Guler et al., 2022; Hu et al., 2022a). In a comparison by Guler et al. (2022), exact methods are shown to be performing better on non-convex optimization problems, whereas approximate methods perform better on near-convex optimization problems. Table 2.1 provides an overview of predict+optimize methods.

Method	Technique	Approximate / Exact	Applied to problems with uncertain parameters in
SPO (Elmachtoub & Grigas, 2017)	Approximate the gradient through a differentiable surrogate for the regret function, formulated using duality theory.	Approximate	Objective
MiPaal (Ferber et al., 2019)	Use a continuous relaxation of the combinatorial optimization problem and KKT conditions for gradient approximation.	Approximate	Objective
QPTL (Wilder et al., 2018)	Use a continuous relaxation of the combinatorial optimization problem and KKT conditions for gradient approximation.	Approximate	Objective
IntOpt (Mandi & Guns, 2020)	Use an interior-point method instead of KKT conditions for the gradient approximation.	Approximate	Objective
Ranking Objectives (Demirovic et al., 2019)	Exploit problem structure of ranking problems to find a search method that can directly lower regret.	Exact	Objective
Dynamic Programming (Demirovic et al., 2020)	Exploit problem structure of problems with dynamic programming solutions to find a search method that can directly lower regret.	Exact	Objective
Branch & Learn (Hu et al., 2022a)	Exploit problem structure of problems with recursive solutions to find a search method that can directly lower regret.	Exact	Objective
Divide and Learn (Guler et al., 2022)	Use a numerical approach for finding the transition points of a combinatorial optimization problem, after which a divide and conquer algorithm is applied to find a model that minimizes the regret.	Exact	Objective

Table 2.1: Comparison of predict+optimize methods

As can be seen in Table 2.1, none of the aforementioned approximate and exact methods for training machine learning models to minimize regret have previously been applied to problems with uncertainty in the constraint parameters.

2.6. Performance optimization for predict+optimize for NP-Hard problems

All of the predict+optimize methods described in Section 2.5 require repeatedly finding optimal solutions to the combinatorial optimization problem during the training of the model. Real-world applications of predict+optimize methods are often rather complex, NP-hard optimization problems. Finding optimal solutions for NP-Hard problems is computationally challenging. The computational complexity of predict+optimize methods can thus be a big limitation of predict+optimize methods. In many optimization contexts, there is often limited time for optimization and a limited computation budget. It is thus investigated if the solver can be replaced with a less computationally challenging relaxation of the problem during training. This approach is called SPO-Relax (Mandi et al., 2019), and it can indeed provide significant increases in performance with equal results to SPO+ Loss training. Furthermore, warm-starting the learning by training on the prediction error, and re-using solutions for the solver are techniques that can help lessen the computational load (Mandi et al., 2019). These results show the feasibility of utilizing predict+optimize techniques for complex NP-hard problems at large scales.

2.7. Objective value calculation for infeasible solutions

As already explained in Chapter 1, applying predict+optimize methods to problems with uncertainty in the constraint parameters can lead to infeasible solutions. The reason for this is that solutions created on uncertain parameters may be infeasible when they are applied using the true parameters. As predict+optimize methods require that the objective value of such solutions is evaluated, finding methods of evaluating infeasible solutions is relevant.

An example of a search method where this use case arises is evolutionary algorithms for combinatorial optimization problems (Ray et al., 2009). In contrast to predict+optimize, these specific evolutionary algorithms do not learn to make predictions for uncertain problem parameters. Instead, they can be used as solvers for the knapsack problem with known parameters. Because evolutionary algorithms can lead to infeasible candidate solutions in their solving process, an appropriate objective value evaluation for infeasible solutions can help the solving process. Several functions for evaluating the objective value of infeasible solutions are possible. For example in Syarif et al. (2016), reparation and penalization strategies are used for evaluating infeasible solutions for the knapsack problem. From this, it can be seen that altering the objective value calculation for infeasible solutions can help search methods in approaching optimal solutions (Ray et al., 2009).

2.8. Epistemic and aleatory uncertainty in uncertain parameters

Both the two-stage prediction-error-based approach of Section 2.2 and the predict+optimize approach of Section 2.3 optimize combinatorial optimization problems under uncertain constraint parameters. In optimizing using uncertain constraint parameters, it is important to know that two types of uncertainty can be distinguished: epistemic uncertainty and aleatory uncertainty. This distinction is especially valuable as the amount of each type of uncertainty influences how well a single-point predictor can provide information about an uncertain parameter to the optimization algorithm.

Epistemic uncertainty derives from the lack of knowledge of a parameter. An example of completely epistemic uncertainty is interpreting a rare word (one that has not previously been observed) in a language model. There is no knowledge of the meaning of this word, even though there could have been if there had been enough data. In contrast, aleatory uncertainty refers to uncertainty caused by probabilistic variants in a random event. A good example of an event with only aleatory uncertainty is rolling a die. Even if all parameters are known, there is still uncertainty about the process that can not be predicted. Both types of uncertainty often co-exist for an uncertain parameter (Hüllermeier & Waegeman, 2021). In both prediction-error and predict+optimize implementations, typically a single value is predicted for an uncertain parameter. Thus only this single prediction is provided as information about the uncertain parameter to the solver. This will favour parameters containing mostly epistemic uncertainty because epistemic uncertainty implies predictability to a single value if only enough information is known. In contrast, for uncertain parameters that also contain significant aleatory uncertainty, learning the underlying distribution of the uncertain parameter may provide more information. Because predict+optimize methods make point predictions instead of predicting distributions of uncertain parameters, the applicability of predict+optimize methods to problems where the uncertain parameters contain significant aleatory uncertainty may be lacking. A research direction that investigates the applicability of predict+optimize to such problem settings is discussed in Section 6.3.

3

Problem Description

The 0-1 Knapsack problem is a combinatorial optimization problem that is characterized by a capacity constraint. If the exact item weights are unknown, this is a typical example of a combinatorial optimization problem with uncertainty in the constraints. Such optimization problems with uncertainty in the constraints are crucial problems in today's world. For example, when budgeting it can be uncertain exactly how much each item will cost, yet it is important to make optimal use of the budget in order to reach a given objective. Or when creating a schedule, the duration of each task may be uncertain. Even though these values that are relevant to the constraints are uncertain, with each of these examples, it is possible to estimate the value of these uncertain parameters based on similar items in the past and their features.

This estimation is often done using machine learning models (Meilanitasari & Shin, 2021). Optimizing a model M_γ is done by searching for the internal weights γ of the model that can select any translation function from input features f to predicted parameter values $\hat{\theta}$. The training of the model's weights γ is often done using the prediction error (the difference between θ, the true parameter values and $\hat{\theta}$) using some distance function such as the mean squared error (MSE, see equation 2.3). However, not all mispredictions have an equal effect on regret, and since training to lower the prediction error does not take the regret into account, this may lead to sub-optimal solutions.

Learning the optimal model weights γ directly using gradient descent on the regret calculation is not possible, because inside the regret calculation there is a translation from an $\hat{\alpha}$ to its objective value, which is a discrete function which can take any arbitrary shape, and is thus not differentiable. In this work, the SPO+ loss sub-gradient is used as described in Section 2.4.2, because of its applicability to the 0-1 Knapsack problem and prior results in works such as Demirović et al. (2019) and Mandi et al. (2019).

To study how to best use this available information to make predictions such that the regret of the optimization is minimized, the 0-1 knapsack problem with uncertainty in the weight parameters is used. Specifically, it is investigated if the SPO+loss learning method (Elmachtoub & Grigas, 2017) can be utilized for such problems with uncertainty in the constraints.

A formal definition of the 0-1 knapsack problem with uncertain item weights and predictive features is presented below:

$$\begin{aligned} \underset{\gamma}{\text{argmin}} \sum_{p=1}^{P} obj(\alpha, \theta_p, v_p, c) - obj_{corrected}(\hat{\alpha}, \theta_p, v_p, c) \\ s.t.\ \alpha = K(v_p, \theta, n, c), \\ \hat{\alpha} = K(v_p, \hat{\theta}, n, c), \\ \hat{\theta} = M_\gamma(f_p) \end{aligned} \tag{3.1}$$

where:

$p \in \mathbb{Z}^+$	= the problem instance index
$obj(a, w, v, c) \in \mathbb{R}$	= provides the objective value of an assignment solution a for the provided **true** weights w and values v under capacity constraint c
$obj_{corrected}(a, w, v, c) \in \mathbb{R}$	= provides the objective value of an assignment solution a for the provided **true** weights w and values v under capacity constraint c, with some correction method for infeasible assignments a.
$K(v, w, n, c) \in \{0, 1\}^n$	= provides the assignment solution X when solving the knapsack problem $K(v, w, n, c)$
$f_p \in (\mathbb{R}^9)^n$	= the list of 9 features for all items in problem instance p
$v_p \in \mathbb{Z}^{+n}$	= the list of values for all items in problem instance p
$\theta_p \in \mathbb{Z}^{+n}$	= the true weight for all items in problem instance p
$M_\gamma(f_p) \in \mathbb{Z}^n$	= the model with internal weights γ that can output the predicted weights $\hat{\theta}$ based on a provided feature set f
$n \in \mathbb{Z}^+$	= the number of items per problem instance
$c \in \mathbb{Z}^+$	= the capacity of the knapsack

Note that $\hat{\alpha}$ is the assignment when optimizing the knapsack problem on the predicted weights $\hat{\theta}$. A new problem arises when predict+optimize is used for problems with uncertainty in the constraint parameters. Because $\hat{\alpha}$ is based on estimated item weights $\hat{\theta}$, if any items that are selected in $\hat{\alpha}$ are underpredicted, $\hat{\alpha}$ can be an infeasible assignment given the true weights θ. Current predict+optimize methods do not take the feasibility of $\hat{\alpha}$ into account, making them less applicable to problems with uncertainty in the constraint parameters. To overcome this shortcoming of predict+optimize, $obj_{corrected}(\hat{\alpha}, \theta_p, v_p, c)$ should take the feasibility of the assignment $\hat{\alpha}$ on the true weights into account. This should be done both during the training of model M and when evaluating the predictions made by M on the validation set.

4

Method

In this chapter, a novel approach to dealing with problems with uncertainty is described. First, in Section 4.1, several different correction methods are proposed in order to find how candidate solutions produced by predict+optimize that are infeasible when evaluated on the realized constraint parameters should be evaluated. Then, it is proposed that these correction methods can be varied between the training and evaluation stages of the machine learner in Section 4.2. Finally, Section 4.3 describes how each research question will be answered in the experiments.

4.1. Correction methods

Given that the knapsack item weights are uncertain, the machine learner may under-predict the item weights. When optimization is done on these under-predicted item weights, this leads to an assignment where more items are included, than if the optimization is done on the true weights. The knapsack assignments made using predicted weights can thus be infeasible assignments when evaluated using the true weights. In order to deal with infeasible assignments given the true weights, a regret has to be calculated based on such assignments. Thus, the objective value calculation of the assignment made on the predicted weight must be corrected for infeasible solutions by using a correction method. This correction can be done using various methods, which are described below.

Finding the items that should be removed in order to make a candidate solution feasible is itself another NP-hard 0-1 knapsack problem. In order to keep run-time complexity low, these correction methods are greedy methods that sort the items by predicted weight-to-value ratio and remove or penalize the items with the lowest weight-to-value ratio first.

Option 1: Repairing through the elimination of over-weight items
The first strategy is repairing infeasible solutions to become feasible, and then evaluating the feasible version of the solution. In order to do this greedily, all items are put in the knapsack as long as they fit. This is done in order of the weight-to-value ratio of the items so that the most efficient items are added first. When an item is encountered that does not fit in the knapsack anymore, its value is not added to the objective value. The strategy is described in Algorithm 1 below.

Option 2: Rejecting infeasible solutions
Another strategy is to reject infeasible solutions and thus evaluate those solutions as having 0 objective value. This is essentially an evaluation with hard constraints, which makes this strategy interesting when comparing the results on the validation set in order to see only feasible solutions. The strategy is described in Algorithm 2 below.

Option 3 & 4: Penalization of infeasible solutions through linear penalties in weight or value Finally, a penalization of infeasible assignments can be a correction. Here, items are again sorted by the weight-to-value ratio. Each item that is assigned to the knapsack is added to the knapsack. However, when an item is added, it is checked if adding this item violates the capacity constraint. If this is the case, the objective value is penalized. This penalization can be linear in the item weight (option 3) or the item value (option 4). The penalization can be scaled using a constant integer P, which is multiplied by the value or weight of the overweight item. This amount is then subtracted from the objective value.

It is expected that options 1 and 2 limit the amount of learning because the information on the distance of a candidate solution to a feasible solution is lost during training. This will make the gradient function less smooth, which makes it less likely that gradient descent will find better solutions. With penalization options (3 and 4) the smoothness of the gradient (and thus the amount of learning possible for gradient-descent) is expected to depend on their respective P values. Low P values are similar to a repair method, in which there is no or only a small downside to selecting overweight solutions. Higher P values should emphasize the importance of finding feasible solutions over finding optimal solutions that are infeasible. The penalization correction methods are described in Algorithm 3 and Algorithm 4 below.

Algorithm 1 Repairing through eliminating over-weight items from objective value calculation.

procedure objectiveRepair(α, θ, v, c)
    $used \leftarrow 0$
    $objectiveValue \leftarrow 0$
    $\alpha \leftarrow$sort(α, θ, v) ▷ Sorting in descending order of predicted-weight to value ratio
    for $\alpha_i \in \alpha$ **do**
        $newUsed \leftarrow used + \alpha_i \times w_i$
        if $newUsed < capacity$ **then**
            $used \leftarrow used + \alpha_i \times w_i$
            $objectiveValue \leftarrow objectiveValue + \alpha_i \times v_i$
        end if
    end for
    return $objectiveValue$
end procedure

Algorithm 2 Infeasible solutions are evaluated as having 0 objective value.

procedure objectiveRejection(α, θ, v, c)
    $usedWeight \leftarrow 0$
    $objectiveValue \leftarrow 0$
    for $\alpha_i \in \alpha$ **do**
        $used \leftarrow used + \alpha_i \times w_i$
        $objectiveValue \leftarrow objectiveValue + \alpha_i \times v_i$
    end for
    if $used > capacity$ **then**
        return 0
    end if
    return $objectiveValue$
end procedure

Algorithm 3 Penalizing linear in weight of the over-weight items, with a varying factor P for the strength of the penalization.

procedure objectivePenalizeWeight(α, θ, v, c, P)
    $used \leftarrow 0$
    $objectiveValue \leftarrow 0$
    $\alpha \leftarrow$sort(α, θ, v) ▷ Sorting in descending order of weight to value ratio
    for $\alpha_i \in \alpha$ **do**
        $newUsed \leftarrow used + \alpha_i \times w_i$
        $objectiveValue \leftarrow objectiveValue + \alpha_i \times v_i$
        if $newUsed < capacity$ **then**
            $used \leftarrow newUsed$
        else
            $objectiveValue \leftarrow objectiveValue - \alpha_i \times w_i \times P$
        end if
    end for
    return $objectiveValue$
end procedure

Algorithm 4 Penalizing linear in value of the over-weight items, with a varying factor P for the strength of the penalization

procedure objectivePenalizeValue(α, θ, v, c, P)
    $used \leftarrow 0$
    $objectiveValue \leftarrow 0$
    $\alpha \leftarrow$sort(α, θ, v) ▷ Sorting in descending order of weight to value ratio
    for $\alpha_i \in \alpha$ **do**
        $newUsed \leftarrow used + \alpha_i \times w_i$
        $objectiveValue \leftarrow objectiveValue + \alpha_i \times v_i$
        if $newUsed < capacity$ **then**
            $used \leftarrow newUsed$
        else
            $objectiveValue \leftarrow objectiveValue - \alpha_i \times v_i \times P$
        end if
    end for
    return $objectiveValue$
end procedure

4.2. Varying training and evaluation correction methods

The correction methods should be applied both to infeasible solutions during training, as well as in the evaluation on the validation set. However, in these two scenarios, the correction method has a different purpose. During training, the correction method should focus on making the gradient smoother so that more optimal learning of the underlying decision problem can take place. In contrast, during the evaluation, the correction method is given by the problem application's context and how hard its constraints are. It should thus be considered to evaluate infeasible solutions differently during training than during evaluation. For example, the learner could be trained using a penalization on infeasible solutions to make the gradient more smooth and thus improve learning. After training, the results could be evaluated by rejecting infeasible solutions, if this is what the context of the specific combinatorial optimization problem aims to optimize.

Given that no particular hardness of constraints is investigated in this work, both rejection and penalization correction methods are used as evaluation methods to gather more insights into the feasibility of the provided solutions. Reparation was not compared as a correction method in evaluation in these experiments, as it disregards the feasibility of solutions, and thus existing predict+optimize methods are already applicable. To conclude, all proposed correction methods are used in training and compared on both evaluation methods in order to make a complete comparison.

4.3. Answering the research questions using experiments

The primary goal of this research is to investigate if the machine learner in the predict+optimize framework can be trained on an accurate reflection of the decision error for combinatorial optimization problems that involve uncertainty in the constraints, such that it outperforms two-stage prediction-error-based methods. To answer this question, the NP-hard 0-1 knapsack problem will be used as an instance of such a combinatorial optimization problem.

The first sub-question of this research is: *'How should candidate solutions produced by predict+optimize which are infeasible when evaluated on the realized constraint parameters be evaluated?'*. To answer this concretely, all correction methods from Section 4.1 will be evaluated against each other on the 0-1 knapsack problem. To this end, an experiment can be set up such that predict+optimize learners that use these different correction methods in training can be compared. For the penalization method, different penalization factors will be experimented with for a full comparison. Sufficient training samples will be provided for convergence to the optimum result for the learners. Therefore, an experiment that finds the number of training samples required for convergence will first be conducted.

To compare the different correction methods used in training, the same correction method will be used in the evaluation between them. For example, when comparing training predict+optimize with a rejection of infeasible solutions and training predict+optimize with a penalization of infeasible solutions at $P = 10$, both will be evaluated using the same objective value calculation and thus correction method, in order to make it a fair comparison. This experiment will help answer the second sub-question of this research: *'is there an advantage to using a different evaluation of infeasible candidate solutions during training than the evaluation method that is given by the problem context?'*.

To make the comparison between these correction methods in training complete, the noise in the feature data will be varied. With little to no noise, it is expected that the two-stage prediction-error-based technique works perfectly. In practice, perfect predictions are often not possible due to noise in the feature data, and this is where the need for predict+optimize arises. This experiment will thus answer the third sub-question of this research: *'How does the strength of the correlation between the feature data and the true parameters influence the performance of predict+optimize methods and prediction-error-based methods for problems with uncertainty in the constraint parameters?'*.

Finally, to show the generalization of the results to realistic scenarios, an experiment on a benchmark data set will be conducted to see if the findings still hold when there is a more realistic variance of the weight-to-value ratio between the knapsack items, and the weights and values are distributed on larger scales.

5

Experiments and results

In this chapter, the setup of the experiments conducted to answer the research questions is stated. In addition, the results of these experiments are discussed. First, Section 5.1 describes the experimental setup. Then, the number of training samples required for convergence of the machine learners is investigated in Section 5.3. In Section 5.4 a comparison is made between the different correction methods and the two-stage prediction-error-based method to answer the first and second sub-research questions. In order to investigate how both predict+optimize methods and two-stage prediction-error-based methods perform under lower and higher correlations between the feature data and true weights, varying amounts of noise are added to the feature data. These experiments are described in Section 5.5. Finally, an evaluation of the performance of predict+optimize with the best-performing correction method and the two-stage prediction-error-based approach on a benchmark data set is provided in Section 5.6.

5.1. Experimental Setup

The experiments described in the following sections are based on the problem setting of the 0-1 knapsack problem with uncertain weights. Two approaches are compared. The baseline approach, which is currently often used in practice, is solving the problem by obtaining a point prediction of the item weights first, and then solving the problem given this point prediction. The machine learner is then trained on the prediction loss using the MSE as the loss function. This baseline approach is referred to later as ”MSE”. The novel approach is to train the machine learner on the decision error using the SPO+ loss gradient in combination with some correction method in order to be able to evaluate infeasible solutions. The hypothesis is that this novel approach will yield a lower regret by integrating the prediction and optimization phase in the machine learner. First, in Section 5.1.1, the details are provided about how multiple 0-1 knapsack instances with uncertain weights and predictive features are generated to compare the correction methods. Then, in Section 5.1.2 and 5.1.3 the training and evaluation process for the machine learning models are described.

5.1.1. Generated knapsack instances

In order to compare the different correction methods to the two-stage approach and to each other, several instances of the 0-1 knapsack with uncertain weights problem should be available. More detailed information on the structure of the problem instances is provided below:

- Each item has a true weight which is unknown at the time of the optimization and 9 features that can (partially) predict its true weight. The features consist of 9 floats, which are normalized before they are used in the model.
- 25% of the items available are used as the validation set.
- Instances of 48 items are created from this data by using the ordering in which the items are generated (i.e. the first 48 items are the first instance, the next 48 items are the second instance, etc.).
- The capacity of each knapsack instance is 60.
- The weight-to-value ratio of all of the items is 5.

The value for the number of items per knapsack, capacity and the weights and values used roughly match those in the experiments in Mandi et al. (2019). The capacity of each knapsack was taken so that each item should fit within the capacity of the knapsack, and up to six items can fit in a knapsack. This makes for a complex decision between 48 different items in each instance. In the instance generation procedure, nine linear coefficients ($linear\_c$) are first selected from a uniform range between 0.1 and 1. These coefficients, multiplied by the nine features that are generated for each item, result in the true weight in the scenario without noise. The weights are picked uniformly, after which 8 random features are generated. The ninth feature is picked in such a way that the multiplication with linear coefficients results in the selected true weight, creating a linear relationship between the features and true weights. The value is picked using the selected weight and the weight-to-value ratio ($weightToValueRatio$).

The correlation between the predictive features and the true weight can be tuned by adding uniformly distributed noise over each item's features. The range of this noise is dependent on the "noise" parameter. This noise parameter should be varied in order to test the methods under increasing amounts of uncertainty. In order to keep the noise levels stable, and not add any noise in the generation procedure other than the noise variable, the weight-to-value ratio is not varied between the items. Such instances, which are described as *strongly correlated* instances in Pisinger (2005), result in a relatively harder 0-1 knapsack problem to solve. The full instance generation procedure can be found in Algorithm 5. An example of a single knapsack problem instance generated from this procedure is provided in Appendix A.

Algorithm 5 Instance generation procedure

procedure generateInstance($nrItems$, $weightToValueRatio$, $noiseSize$)
    $minWeight \leftarrow 10$
    $maxWeight \leftarrow 50$
    $linear\_c \leftarrow [\,]$
    for $c\_index \in 0...9$ **do**
        $linear_c[c_index] \leftarrow$randUniform$(0.1, 1)$
    end for
    $result \leftarrow [\,]$
    for $i \in 0....nrItems$ **do** $weight \leftarrow$randUniform$(minWeight, maxWeight)$
        $features \leftarrow [\,]$
        for $j \in 1...8$ **do** $feature \leftarrow$randUniform$(1, 10)$
            $features \leftarrow features + feature$
        end for
        $weightBeforeCorrection \leftarrow$sum$(linear_c * features)$
        $weightDifference \leftarrow weight - weightBeforeCorrection$
        $features[9] \leftarrow weightDifference/linear\_c[9]$
        $value \leftarrow$round$(weight/weightToValueRatio)$
        for $feature \in features$ **do** $feature \leftarrow feature +$ randUniform$(0, noiseSize)$
        end for
        $result \leftarrow result + [value, weight, features]$
    end for
    return result
end procedure

5.1.2. Training process

The machine learning model used is Linear Regression. The gradient optimization is done using the Adam optimizer (Kingma & Ba, 2014). Training on the training samples is executed over multiple 'epochs'. Each epoch goes over all training instances in a randomly shuffled order. For prediction error-based training (MSE) the model is trained on each item in the training set during each epoch. This requires no evaluation of the combinatorial optimization problem during training. Thus, the combinatorial optimization problem is only solved on the validation set, in order for it to be used as an evaluation mechanism.

The training process when training on SPO+ loss is described in Algorithm 6 and implements a gradient descent over the problem description in Equation 3.1. During each epoch, all instances are visited in a shuffled order. For each instance, the model is trained by first predicting the weights of the items based on their features. This prediction is then used for the SPO+ loss gradient calculation, which evaluates the regret by comparing the objective value of solving the knapsack using a convex combination of the prediction and the true weights ($2 * \hat{\theta} - \theta$) as the weights and the objective value when solving on the true weights (θ). This difference is referred to as surrogate regret. The gradient calculation is then done using the model's built-in back-propagation ($Model.backward$ in PyTorch (Paszke et al., 2019)), which captures the influence that each of the internal model weights γ had on the prediction. By multiplying that with the surrogate regret, the SPO+ loss gradient is calculated. This gradient is then forwarded to the Adam optimizer which decides how to update the model weights over time, after which the model weights are updated.

Algorithm 6 SPO+ Loss training loop

procedure train
  for $epochNr \in 0....numEpochs$ **do**
    $instances =$ Shuffle($trainingInstances$)
    for $instance \in instances$ **do**
      $\hat{\theta} =$ Model.Predict(f)
      $\hat{\alpha} = x(2 * \hat{\theta} - \theta$, instance)
      $spoEvaluated = obj_{corrected}(\hat{\alpha}, \theta)$
      $spoTrue = obj(x(\theta, instance), \theta)$
      $grad =$ Model.backward() * ($spoEvaluated$ - $spoTrue$)
      $Model =$ Model.updateGradient(adam.Optimize($grad$))
    end for
  end for
end procedure

Solving the 0-1 knapsack problems in order to find the outcome of $spoTrue$ and $\hat{\alpha}$ is done by formulating a MIP and using the Gurobi solver (Gurobi Optimization, LLC, 2022). This solving is referred to as x in Algorithm 6.

5.1.3. Evaluation process

Evaluation is done at the end of each epoch training epoch, by letting the trained model predict the parameters ($\hat{\theta}$) of all validation instances. The predicted parameters are then evaluated by solving the 0-1 knapsack problem using these predictions on all validation instances, which outputs an allocation $\hat{\alpha}$ for each instance. The objective value is then calculated using the allocation made and evaluating it on the true, realized values, and compared to the objective value achieved when optimizing on the true values. Suppose the allocation $\hat{\alpha}$ is infeasible on the true values, which can only be the case when there is uncertainty in the constraints. In that case, the evaluation mechanism ($obj_{corrected}$) should be the same across different learners, in order to be able to compare them fairly. This evaluation mechanism could be a penalization with the same P value in either weight of values, reparation to a feasible solution or rejection of all infeasible solutions as described in Section 4.1.

Algorithm 7 Evaluation loop

```
1:  procedure evaluate
2:      for epochNr ∈ 0....numEpochs do
3:          trainEpoch()
4:          for instance ∈ validationInstances do
5:              θ̂ = Model.Predict(f)
6:              α̂ = x(θ̂, instance)
7:              achieved = obj_corrected(α̂, θ)
8:              spoTrue = obj(x(θ, instance), θ)
9:              regret = true − achieved
10:         end for
11:     end for
12: end procedure
```

Solving the 0-1 knapsack problems in order to find $spoTrue$ and $\hat{\alpha}$ is done by formulating a MIP and using the Gurobi solver (Gurobi Optimization, LLC, 2022). This solving is referred to as x in Algorithm 7.

5.2. Overview of the experiments

In Table 5.1, an overview of the experiments is provided. First, experiment 1 is used to select an appropriate size for the training set. Then, experiments 2 through 4 show the performance of each correction method, compared to the baseline two-stage approach, on instances with different correlations between the features and true weights. Experiment 5 combines these results and shows the influence of the different correlations on the performance of the different learners. Finally, experiment 6 applies the best-performing learning method and the two-stage prediction-error-based approach to a benchmark data set.

Experiment	Instance properties	Learner properties in training	Evaluation properties
1	Self-generated with varying amounts of training data compared.	MSE, varying P levels and penalization linear in weights.	Reject and penalization linear in values, $P = 2$.
2	Self-generated with varying noise.	MSE, repair method.	Reject and penalization linear in values, $P = 2$.
3	Self-generated with varying noise.	MSE, rejection method.	Reject and penalization linear in values, $P = 2$.
4	Self-generated with varying noise.	MSE, varying P levels and penalization linear in weights.	Reject and penalization linear in values, $P = 2$.
5	Self-generated with varying noise levels compared.	MSE, varying P levels and penalization linear in weights.	Reject and penalization linear in values, $P = 2$.
6	Altered benchmark data set.	MSE, best performing correction method in experiment 5.	Reject and penalization linear in values, $P = 2$.

Table 5.1: Overview of the varying experiments based on the described methodologies

5.3. Influence of the number of training samples

The first experiment investigates the number of training samples required for the convergence of the learner to its most optimal performance on the validation set.

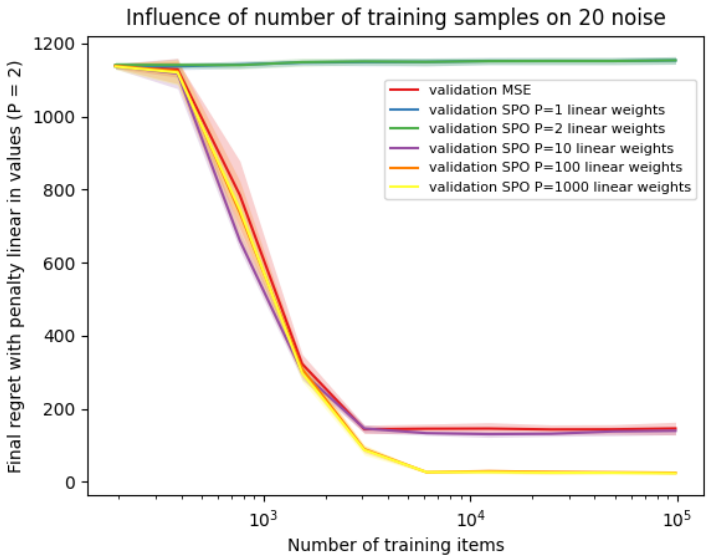


Figure 5.1: Experiment 1: the number of training items required for convergence is around 10^4 for both MSE and SPO based learning.

The correction method used for this evaluation is a penalty linear in the value of the overweight items with $P = 2$ (referred to as "penalty linear in values $P = 2$"). The feature data contains 20 uniformly distributed noise, which is the highest amount used in further experiments. This is thus the weakest correlation between the feature data and the true item weights. Similar results were found for lower noise levels.

From figure 5.1 it can be observed that the number of training samples required for convergence on the validation set is roughly 10^4 training items. This number is in the same order of magnitude between MSE and SPO. The total number of instances used in further experiments is 26.496, which is 552 knapsack instances of 48 items. This number was selected so that the threshold for convergence was easily met, and so that enough validation instances would be available (25% of the instances is used as validation instances). The performance difference between the different correction methods is researched in further experiments.

5.4. Comparison of correction methods

To answer the first research question: *'How should candidate solutions produced by predict+optimize which are infeasible given the realized constraint parameters be evaluated, such that predict+optimize can outperform existing prediction-error-based methods for problems with uncertainty in the constraints?'*, a comparison between the different correction methods during training is made. All methods are evaluated using two correction methods on the validation set: the first correction method used during evaluation rejects all infeasible solutions (referred to as "rejection"). This represents the performance of the learners on problems with hard constraints. The second correction method is an evaluation using a linear penalization, where the penalization is linear in the value of the overweight item, which means adding an overweight item is never beneficial. The P value of this evaluation was selected to be 2, so that any items that are overweight subtract their value from the total objective value once, instead of adding it (referred to as "penalty linear in values $P = 2$").

Repair during training

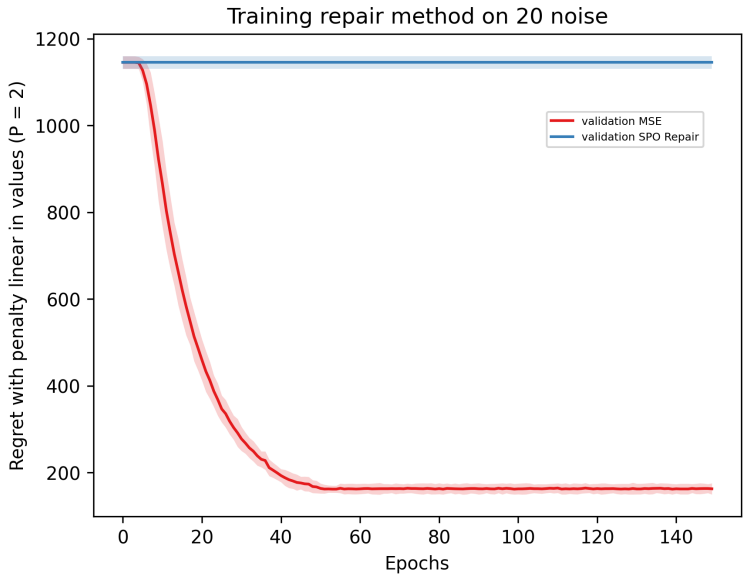


Figure 5.2: Experiment 2: Prediction-error-based training of the model outperforms SPO training with a repair of infeasible solutions. SPO training with repair is stuck in a local optimum and is not provided additional information for lowering regret. The experiment was repeated on 5 seeds, shaded areas depict the standard deviation.

As seen in Figure 5.2, repairing infeasible solutions during training does not result in improved solutions by the SPO learner when compared to the two-stage prediction-error-based approach. The regret for SPO stays constant over each epoch and does not decrease. The performance of the two-stage approach MSE-based learner improves from 1200 regret to 200 regret. This means that the predicted weights produced by the learner lead to solutions that are getting closer in objective value to the solution given the true weights for the validation set.

An intuition that explains the results of the SPO learner is that repairing infeasible solutions during training loses information about why the predicted weights resulted in an infeasible solution. This could cause the gradient to get stuck in a local optimum, as described in section 2.4.2. At different noise levels and with both correction methods in evaluation, the same behaviour is observed. To see these results, please refer to Appendix B.

Reject during training

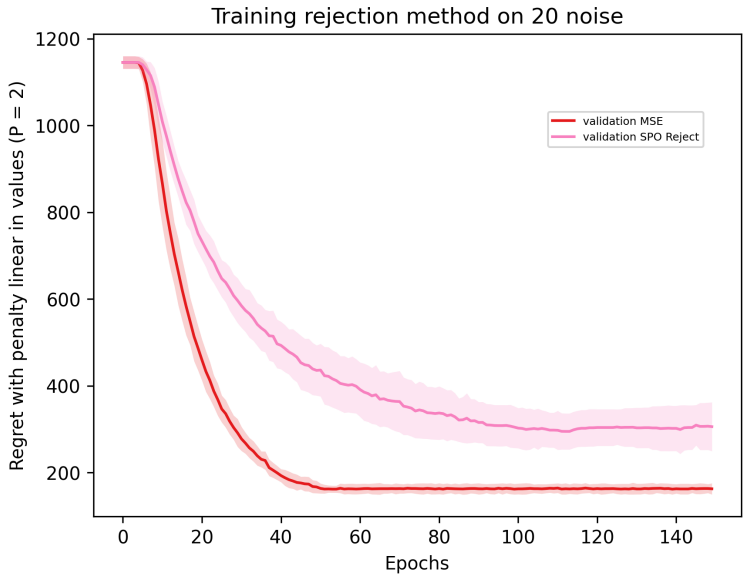
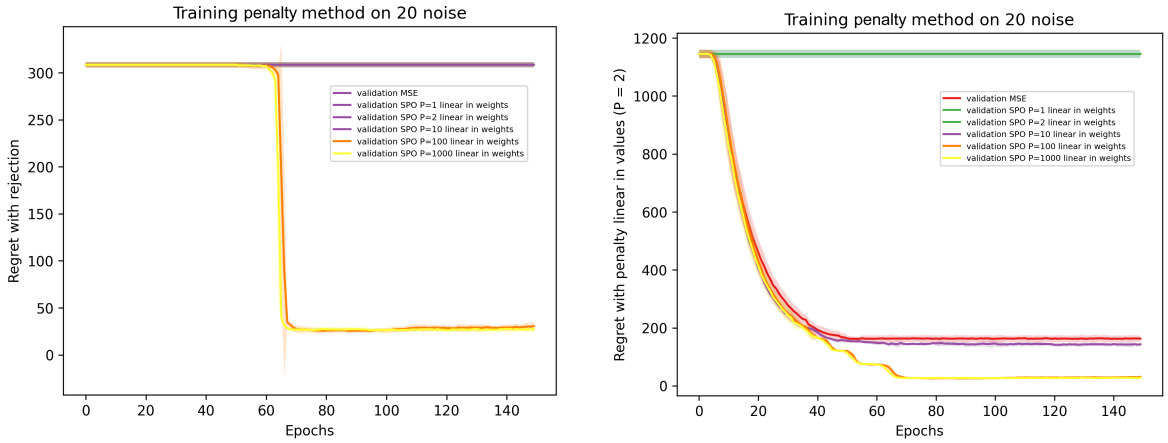


Figure 5.3: Experiment 3: Prediction-error-based training of the model outperforms SPO training with a rejection of infeasible solutions. The experiment was repeated on 5 seeds, shaded areas depict the standard deviation.

As seen in Figure 5.3, rejecting infeasible solutions during training results in some learning for SPO, but a slower convergence and higher regret when compared to the classic two-stage prediction-error-based approach. This can again be caused an information loss during training because rejecting infeasible solutions will not provide information to the gradient about the influence of the prediction of each item on the feasibility of the solution. These results were compared with both rejection and penalization evaluation methods and at lower noise levels. In these experiments, the same behaviour was observed. To see these plots, please refer to Appendix C.

Penalty linear in weight



(a) Reject infeasible solutions in evaluation.

(b) Linear penalty in value with $P = 2$ in evaluation for infeasible solutions.

Figure 5.4: Experiment 4: SPO training with a penalization of infeasible solutions outperforms prediction-error-based training. The experiment was repeated on 5 seeds, shaded areas depict the standard deviation.

In Figure 5.4 it can be seen that training based on SPO+ loss-based training with penalization is superior to training based on prediction loss when there is a weak correlation between the item features and the true weights. This can be observed from the final regret achieved. In Figure 5.4(a) the validation regret is evaluated with rejection. With this evaluation method, prediction-error-based learning does not find completely feasible solutions and thus has a continious regret of 300. In contrast, SPO+ loss based learning with penalization of infeasible solutions and $P \geq 100$ achieves an average regret of around 30 after convergence at epoch 70. Figure 5.4(b) shows the same learning methods, but evaluated using penalization of infeasible solutions. With this evaluation method, prediction-error-based learning achieves an average regret of around 200 after convergence at epoch 50, whereas SPO+ loss-based learning with penalization of infeasible solutions and $P \geq 100$ achieves an average regret of around 30 after convergence at epoch 70.

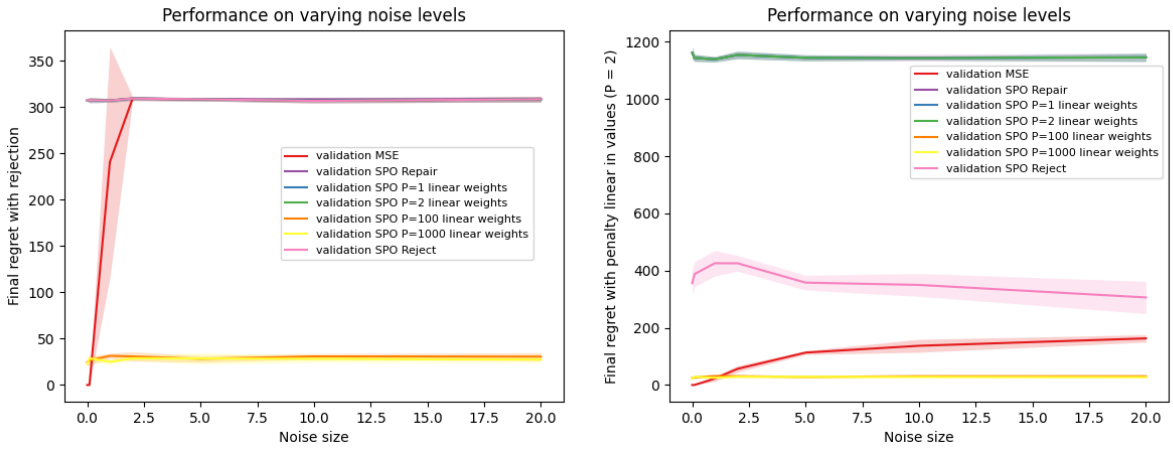
These results show the added value of incorporating the optimization problem in the learner when there is uncertainty in the constraint parameters. By utilizing a linear penalty function during training, the learner can lower regret in the evaluation on the validation set. With higher P values, the SPO+ loss-based training was able to find more feasible solutions, as can be seen in Figure 5.4(a), in which the evaluation rejects infeasible solutions.

With a penalty function linear in weight, the P parameter for scaling that penalty should behave so that most (or preferably all) allocations that are infeasible have an objective value lower than the optimal feasible allocation given the true weights. Since the penalty is linear in the weights, and the objective function depends only on the value of the items and the allocation, the P for which this property holds depends on the weight-to-value ratio. In these generated instances, a P value of at least 5 is necessary not to have any items for which it is better to include overweight items. This explains why $P = 1$ and $P = 2$ do not yield any improved results. Note that penalization linear in values is not compared as a training method, as this is essentially the same as penalizing linear in weight when the weight-to-value ratios are not varied between items.

Interestingly, the higher penalties $P = 100$ and $P = 1000$, were able to outperform $P = 10$ significantly. This indicates that not only exceeding the weight-to-value ratio but also how much it is exceeded, can influence the results of the learning using the SPO+ loss gradient. By increasing the penalization during training, the gradient descent can find better results on the validation set, when evaluated both using rejection and penalization. This advantage to higher penalization factors might be explained by its influence on the gradient descent on the approximate SPO+ loss gradient. To see the plots for different noise levels, please refer to Appendix D.

5.5. Influence of noise in feature data

In order to answer the third research question: ”How does the noise in the available feature data influence the performance of predict+optimize with uncertainty in the constraints methods compared to existing prediction-error-based methods?”, the results of all of the experiments that vary the noise between the features and the true weights are combined. For clarity, only the final regret of each training method on each amount of noise is plotted.



(a) Reject infeasible solutions in evaluation.

(b) Linear penalty in value with $P = 2$ in evaluation for infeasible solutions.

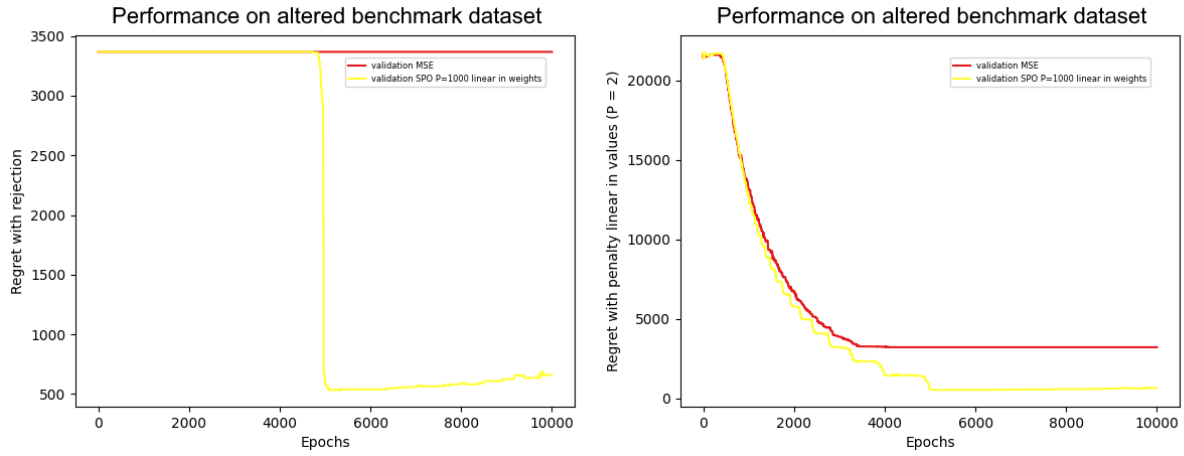
Figure 5.5: Experiment 5: SPO-based learning performs better than prediction-error-based training when there is a weak correlation between the feature data and true weights. The experiment was repeated on 5 seeds, shaded areas depict the standard deviation.

As seen in Figure 5.5, MSE can predict true values perfectly, thus realising a regret of 0, when there is no noise in the features. This is expected because without noise the linear relation between the features and the true weight can be perfectly learned by linear regression, and perfect predictions lead to an optimal optimization and thus 0 regret. Increasing the noise makes training with MSE inferior to SPO-based learning with $P \geq 10$. This is due to SPO's ability to learn from the impact of under-prediction of the constraints, thus opting for safer predictions so that constraints are not violated during the optimization. This can be seen in Figure 5.5(a) where the solutions are evaluated using rejection. At high noise levels, MSE provides exclusively infeasible solutions, whereas SPO with $P \geq 100$ provides more feasible solutions as can be seen from the lower average regret on the validation set.

5.6. Performance on altered benchmark data

To see if the methodology of the correction method with penalization linear in weights generalizes to more realistic problem instances, the same methodology should be applied to a benchmark data set with more realistic item weights and values. However, none of the publicly available 0-1 knapsack benchmark data sets have the properties that are required to test predict+optimize methods on the 0-1 knapsack with uncertainty in the constraints problem. What is missing is: 1) correlated feature data to predict uncertain weights, and 2) a selection of multiple sub-problems to solve instead of having to solve one large problem.

Pisinger (2005) provides several large-scale instances of the 0-1 knapsack problem, with varying correlations between the weight and value of items. The data set selected is a large, weakly correlated instance: this is described as perhaps the most realistic in management problems, as it is well known that the return of an investment is generally proportional to the sum invested within some small variations. To use this data set, feature data has to be generated, similarly to Algorithm 5. Furthermore, a capacity and number of items has to be selected so that the subsets of the instance could be used as a single knapsack problem. This capacity is selected at 3000 so that all items fit, and the number of training items used per instance is again 48.



(a) Reject infeasible solutions in evaluation.

(b) Linear penalty in value with $P = 2$ in evaluation for infeasible solutions.

Figure 5.6: Experiment 6: SPO-based learning with penalization outperforms prediction-error-based learning on the benchmark data set from Pisinger (2005).

The benchmark data set contains items with larger weights and values and has more variance in the weight-to-value ratio. This means that more epochs are required to converge to an optimal result, as shown in Figure 5.6. Furthermore, this figure shows that SPO with a high penalization linear in the weights of overweight items achieves a lower regret on the validation set than MSE, both for rejection of infeasible solutions and penalization of infeasible solutions. Interestingly, in Figure 5.6(a) it is observed that after converging to the best performance around 5000 epochs, SPO finds slightly worse solutions starting around epoch 7000. This could indicate that the SPO+ loss gradient is overfitting slightly to the training set, which can happen when training for too long after final convergence. This is especially known to be a problem for approximate gradients and can be largely resolved using an early stopping criterion or treating the number of epochs as a hyperparameter (Yao et al., 2007).

6

Conclusion

This work investigates the applicability of the predict+optimize framework for combinatorial optimization problems with uncertainty in the constraints. This uncertainty in the constraints brings a new challenge in comparison to optimization problems with uncertainty in the objective. Assignments created on predicted parameter values can be infeasible when evaluated on the realized parameter values. Thus, a novel approach to correcting regret for infeasible solutions in training and evaluation had to be found. For the comparison, the NP-hard 0-1 knapsack problem with uncertain weights was considered. The performance was compared against a baseline two-stage approach, in which a machine learner is first trained on the MSE loss, and then optimization is done based on the predicted parameters.

In this chapter, the main contributions and answers to the research questions are discussed in Section 6.1. Then, a discussion on potential improvement points of this research is given in Section 6.2. Finally, some suggestions for future work are presented in Section 6.3.

6.1. Contributions

The contributions of this research are three-fold:

First, the addition of correction methods for infeasible solutions in the regret calculation was proposed. This, in order to overcome the challenge of evaluating infeasible candidate solutions in predict+optimize, which arises from utilizing predict+optimize for problems with uncertainty in the constraints. Instances were synthetically generated, and the correlation between the features and the true weights was tweaked using a noise parameter. For instances where the noise level, and thus correlation, was chosen such that perfect prediction is impossible to achieve, it was observed that predict+optimize with a correction method achieves a lower regret on the validation set than the two-stage prediction-error-based learning. By considering the error of the decision problem during learning with predict+optimize and utilizing a penalization of infeasible solutions, predictions of uncertain parameters can be made such that they result in an overall lower regret on the validation set.

Secondly, a comparison of several correction methods was performed. The advantage to predict+optimize over the two-stage prediction-error-based approach was observed exclusively using a linear penalization of infeasible solutions during training, which helped create a smoother gradient and resulted in faster convergence to a lower final regret. Other correction methods, such as evaluating a repaired version of the candidate solution or rejecting all infeasible solutions, were not able to outperform the two-stage prediction-error-based methods due to their inherent information loss in combination with the approximation of the regret using the SPO+ loss gradient. Notably, a higher penalization factor showed better results, even when compared to penalization factors already sufficient for penalizing all overweight items. This indicates that for problems with uncertainty in the constraints, larger penalization factors during training can help the SPO+ loss gradient find better-performing model weights.

Finally, the method of utilizing higher penalization factors during training achieves better results when evaluated using both rejection of infeasible solutions and penalization of infeasible solutions on the validation set. There is thus an advantage to using a different correction method for infeasibilities in training to help smoothen the SPO+ loss gradient, compared to the correction method that may be required for the problem context in evaluation.

6.2. Discussion

Whilst the results show the potential of utilizing predict+optimize for combinatorial optimization with uncertainty in the constraints; some limitations apply to the generalization of these results that should be noted.

1. A shortcoming of the experiment on benchmark data in Section 5.6 is that the features and division in multiple sub-problems had to be generated for this specific research. The creation of multiple benchmark test data sets for predict+optimize with uncertainty in the constraints allows for better comparison between different gradient approximation approaches and different correction methods. Two key properties for these data sets are 1) features are provided that have some (varying by set) level of correlation to the uncertain constraint. 2) a large number of instances are provided for each of these sets so that the end-to-end (features to prediction) problem can be trained on.
2. The results of this work have shown that predict+optimize can outperform the classic two-stage prediction-error-based approach for the NP-Hard 0-1 knapsack problem with uncertainty in the weights. Whilst this problem already has great applicability in the real world, more complex combinations of constraints are often encountered, for which a new correction method would have to be constructed. Although the same method of penalization and correction in the regret calculation can be applied generically, it would be very relevant to find a generic way to create correction methods that work well in combination with predict+optimize and the SPO+ loss gradient.
3. Using other models such as neural networks, instead of the more rudimentary linear regression model, could increase performance by allowing it to learn non-linear functions. Combinations of constraints in combinatorial optimization may be highly non-linear, as can be seen in job shop problems, which can have a combination of constraints such as sequence dependencies and uncertain processing times. Using neural networks to learn non-linear functions may increase performance on problems with non-linear dependencies by realizing a more accurate reflection of the effect of the predictions on the final decision error in the model. Utilizing neural networks comes at the cost of additional hyperparameter tuning.

6.3. Future Work

In this section, some suggestions are made for directions of future research into the topics touched upon in this work.

First and foremost, further research into the effects of approximating the gradient of predict+optimize is warranted. No guarantees are provided about when this gradient approximation is or is not optimal. This applies to both optimization problems with uncertainty in the constraints and the objective. Having more knowledge about this gradient approximation and its limitations would help with applying predict+optimize in more complex real-world scenarios. Moreover, exact approaches that exploit the structure of optimization problems to train models without computing gradients have recently appeared, such as the Branch & Learn framework (Hu et al., 2022a). These approaches are limited in applicability to certain problems and involve additional computation to find an optimal modal. To their advantage, exact approaches directly optimize the regret instead of approximating it. Integrating correction methods with these exact approaches such that they can be applied to optimization problems with uncertainty in the constraints is thus another possible research direction. Preliminary results for such a comparison are discussed in Appendix E.

Besides researching the effects of the gradient approximation, it could be investigated if predict+optimize techniques can be applied to other problem scenarios. One such problem scenario is one where the constraint parameters have multiple possible realizations that follow a certain distribution. This means that the constraint parameters contain more aleatory uncertainty, as described in Section 2.8. Such problems are currently often solved using techniques such as robust optimization. In the problem instances used in this work, each item's weight is only uncertain due to noise in the feature data. The predicted weights are then used to create an assignment by solving a MIP. However, in practice, it may be the case that there is inherent aleatory uncertainty in the constraint parameter such that it can take any value from a certain underlying distribution. When this distribution is unknown at the time of optimization, a machine learner might be able to predict the parameters of the distribution using feature data. A predict+optimize machine learner could learn to create better predictions of this distribution such that the regret in the optimization problem is minimized.

As this is a specific type of combinatorial optimization problem with uncertainty in the constraints, again a correction method would have to be applied in order to be able to learn using the SPO+ loss gradient. Alternatively, an exact approach to lowering regret may be able to find better solutions without the use of robust optimization. An example of the outcome of an experiment on a 0-1 knapsack problem where each item's weight is taken from a uniform distribution and re-used in multiple problems with the same features is provided below. In this instance, a MIP solver is used as the solver.

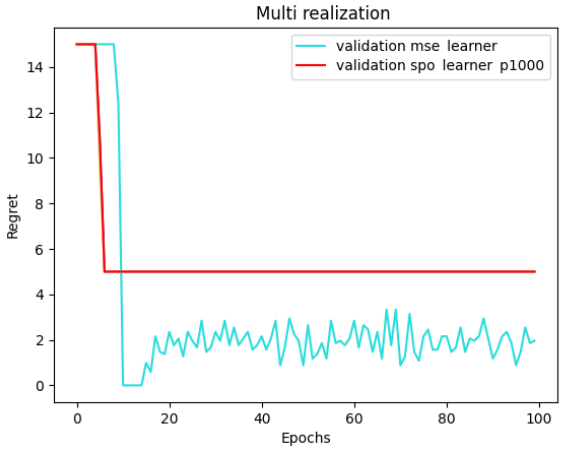


Figure 6.1: Prediction-error-based training outperforms SPO-based training with penalization on a data set with high aleatory uncertainty and no epistemic uncertainty.

As can be observed in Figure 6.1, using SPO+ loss and high penalization, the learner is not able to find predictions such that the regret is lower than when prediction-error-based training is applied on this instance. This highlights the limitations of the MIP solving on items that have an aleatory uncertainty in the parameters. Replacing the MIP solver with a robust optimization, and having the machine learner predict the distribution of the uncertain parameters should provide a better comparison between predict+optimize and prediction-error-based approaches, taking advantage of the fact that the SPO framework can utilize any black-box solver (Elmachtoub & Grigas, 2017).

Finally, applying and evaluating predict+optimize on large-scale real-world problems with uncertainty in both dependency and constraint parameters, such as a flexible job shop problem, remains a promising research direction where many of the earlier mentioned challenges are combined.

References

Demirovic, E., Stuckey, P. J., Bailey, J., Chan, J., Leckie, C., Ramamohanarao, K., & Guns, T. (2019). Predict+optimise with ranking objectives: Exhaustively learning linear functions. *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, 1078–1085. https://doi.org/10.24963/ijcai.2019/151

Demirovic, E., Stuckey, P. J., Guns, T., Bailey, J., Leckie, C., Ramamohanarao, K., & Chan, J. (2020). Dynamic programming for predict+optimise. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(02), 1444–1451. https://doi.org/10.1609/aaai.v34i02.5502

Demirović, E., Stuckey, P. J., Bailey, J., Chan, J., Leckie, C., Ramamohanarao, K., & Guns, T. (2019). An investigation into prediction + optimisation for the knapsack problem. In L.-M. Rousseau & K. Stergiou (Eds.), *Integration of constraint programming, artificial intelligence, and operations research* (pp. 241–257). Springer International Publishing.

Elmachtoub, A. N., & Grigas, P. (2017). Smart "predict, then optimize". https://doi.org/10.48550/ARXIV.1710.08005

Ferber, A. M., Wilder, B., Dilkina, B., & Tambe, M. (2019). Mipaal: Mixed integer program as a layer. *CoRR*, *abs/1907.05912*. http://arxiv.org/abs/1907.05912

Guler, A. U., Demirović, E., Chan, J., Bailey, J., Leckie, C., & Stuckey, P. J. (2022). A divide and conquer algorithm for predict+optimize with non-convex problems. *Proceedings of the AAAI Conference on Artificial Intelligence*, *36*(4), 3749–3757. https://doi.org/10.1609/aaai.v36i4.20289

Gurobi Optimization, LLC. (2022). Gurobi Optimizer Reference Manual. https://www.gurobi.com

He, H., Daume III, H., & Eisner, J. M. (2014). Learning to search in branch and bound algorithms. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, & K. Weinberger (Eds.), *Advances in neural information processing systems*. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2014/file/757f843a169cc678064d9530d12a1881-Paper.pdf

Hu, X., Lee, J. C. H., Lee, J. H. M., & Zhong, A. Z. (2022a). Branch & learn for recursively and iteratively solvable problems in predict+optimize. https://doi.org/10.48550/ARXIV.2205.01672

Hu, X., Lee, J. C. H., Lee, J. H. M., & Zhong, A. Z. (2022b). *Correctional regret for predict+optimize with unknown objectives and constraints* [Work in progress].

Hüllermeier, E., & Waegeman, W. (2021). Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine Learning*, *110*(3), 457–506. https://doi.org/10.1007/s10994-021-05946-3

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. https://doi.org/10.48550/ARXIV.1412.6980

Mandi, J., Demirovic, E., Stuckey, P. J., & Guns, T. (2019). Smart predict-and-optimize for hard combinatorial optimization problems. *CoRR*, *abs/1911.10092*. http://arxiv.org/abs/1911.10092

Mandi, J., & Guns, T. (2020). Interior point solving for lp-based prediction+optimisation. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems* (pp. 7272–7282). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2020/file/51311013e51adebc3c34d2cc591fefee-Paper.pdf

Meilanitasari, P., & Shin, S.-J. (2021). A review of prediction and optimization for sequence-driven scheduling in job shop flexible manufacturing systems. *Processes*, *9*(8). https://doi.org/10.3390/pr9081391

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Pisinger, D. (2005). Where are the hard knapsack problems? *Computers Operations Research*, *32*(9), 2271–2284. https://doi.org/https://doi.org/10.1016/j.cor.2004.03.002

Ray, T., Singh, H. K., Isaacs, A., & Smith, W. (2009). Infeasibility driven evolutionary algorithm for constrained optimization. In E. Mezura-Montes (Ed.), *Constraint-handling in evolutionary optimization* (pp. 145–165). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-00619-7_7

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *CoRR*, *abs/1609.04747*. http://arxiv.org/abs/1609.04747

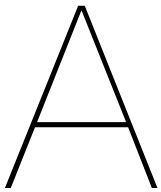
Schuetz, M. J. A., Brubaker, J. K., & Katzgraber, H. G. (2021). Combinatorial optimization with physics-inspired graph neural networks. *CoRR*, *abs/2107.01188*. https://arxiv.org/abs/2107.01188

Syarif, A., Aristoteles, A., Dwiastuti, A., & Malinda, R. (2016). Performance evaluation of various genetic algorithm approaches for knapsack problem. *11*, 4713–4719.

Wilder, B., Dilkina, B., & Tambe, M. (2018). Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. *CoRR*, *abs/1809.05504*. http://arxiv.org/abs/1809.05504

Yao, Y., Rosasco, L., & Caponnetto, A. (2007). On early stopping in gradient descent learning. *Constructive Approximation*, *26*, 289–315. https://doi.org/10.1007/s00365-006-0663-2

Zhang, J. (2019). Gradient descent based optimization algorithms for deep learning models training. *CoRR*, *abs/1903.03614*. http://arxiv.org/abs/1903.03614



Example of a generated knapsack instance

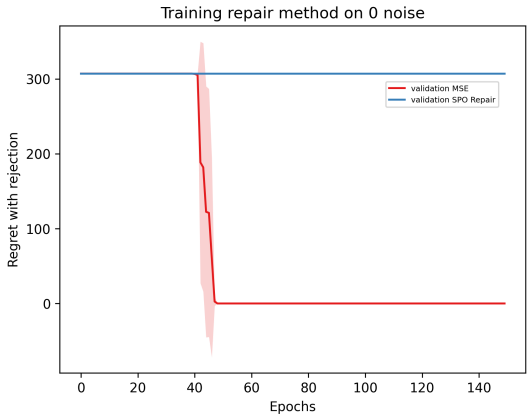
```
feature_0, feature_1, ..., feature_8, value, true_weight
18.1419876318491, 25.07238286196479, ...,-121.22859532901323, 29, 6
1.3295175675373141, 6.811178653764516, ...,-79.91749418761856, 37, 7
13.185756622666858, 11.59506959310166, ...,-34.97225649075373, 29, 6
16.60703839067699, 19.736488460826457, ...,,-129.2471313607014, 47, 9
...
```

1. The instance file contains 48 lines (i.e. items) per knapsack instance
2. In total the file contains 26.496 lines (i.e. items), which is a total of 552 separate knapsack problems

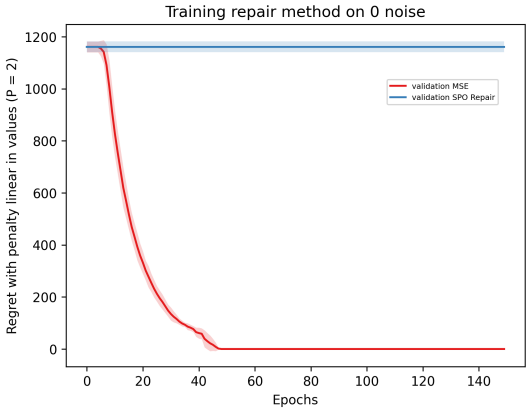
B

Performance for specific noise levels: training with repair

0 noise



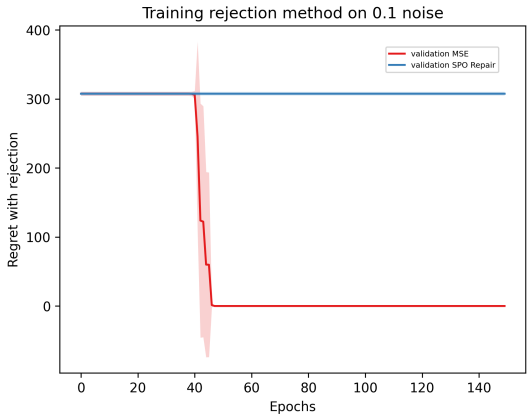
(a) Reject infeasible solutions in evaluation.



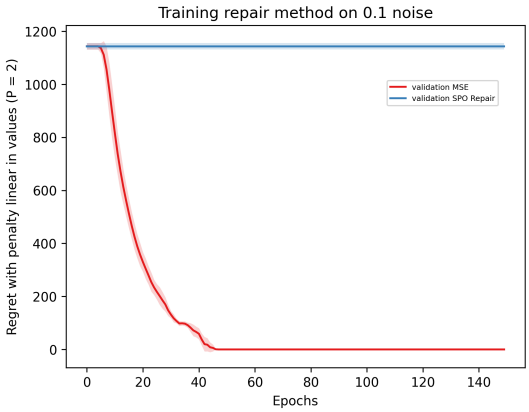
(b) Linear penalty in value with $P = 2$ for infeasible solutions.

Figure B.1: Comparing MSE and SPO+ loss training with repairing infeasible solutions. Instance features contain no noise.

0.1 noise



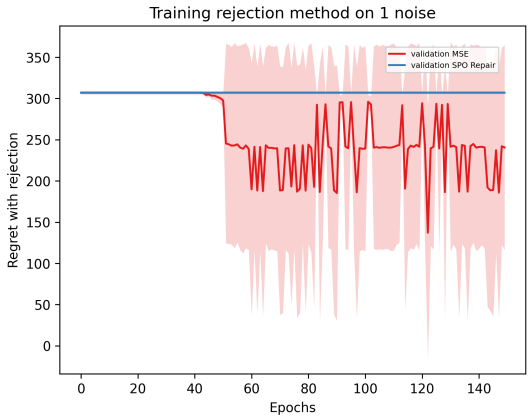
(a) Reject infeasible solutions in evaluation.



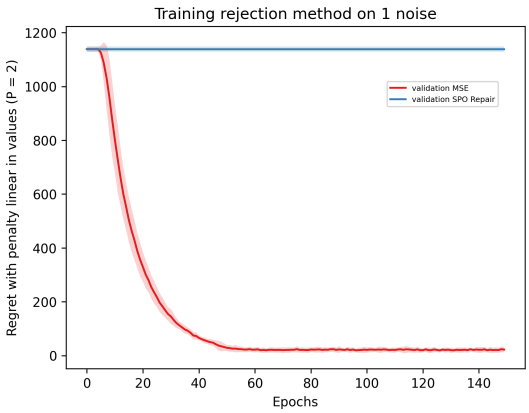
(b) Linear penalty in value with $P = 2$ for infeasible solutions.

Figure B.2: Comparing MSE and SPO+ loss training with repairing infeasible solutions. Instance features contain some noise (0.1 uniform).

1.0 noise



(a) Reject infeasible solutions in evaluation.

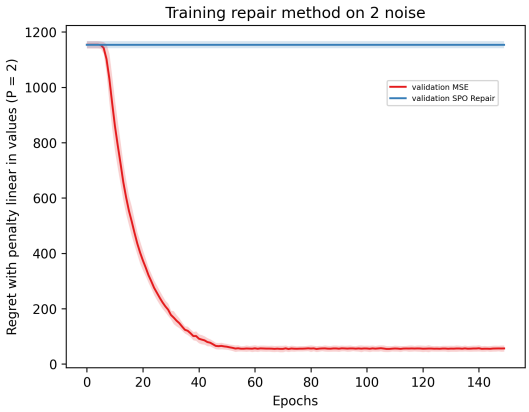


(b) Linear penalty in value with $P = 2$ for infeasible solutions.

Figure B.3: Comparing MSE and SPO+ loss training with repairing infeasible solutions. Instance features contain some noise (1 uniform).

2 noise

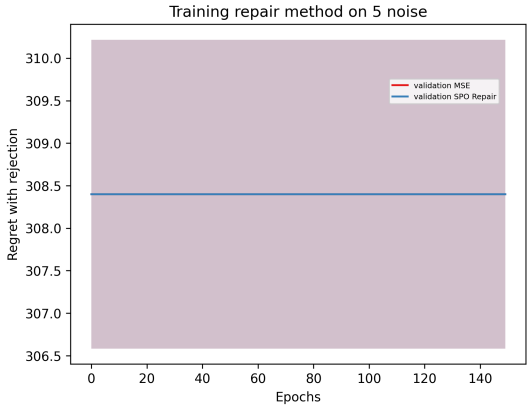
(a) Reject infeasible solutions in evaluation.



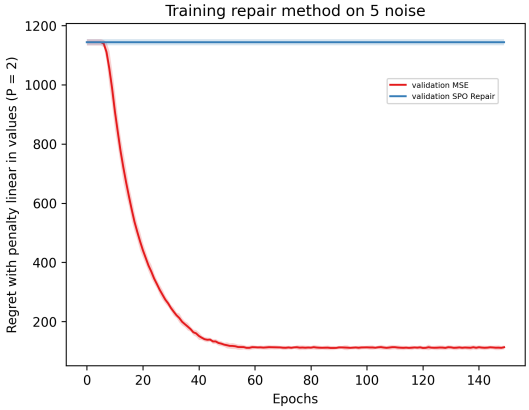
(b) Linear penalty in value with $P = 2$ for infeasible solutions.

Figure B.4: Comparing MSE and SPO+ loss training with repairing infeasible solutions. Instance features contain some noise (2 uniform).

5 noise



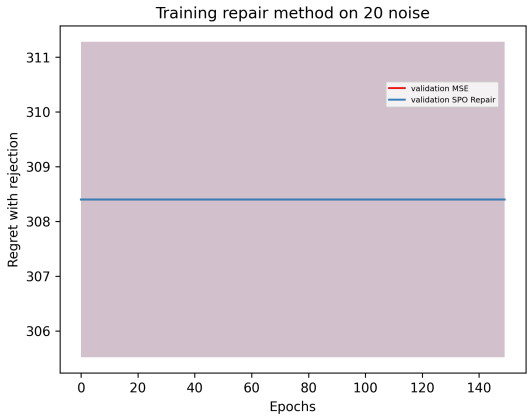
(a) Reject infeasible solutions in evaluation.



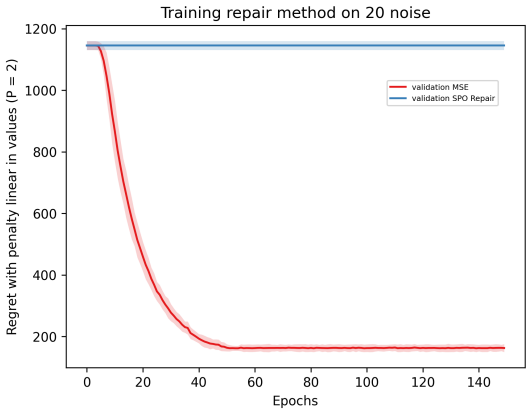
(b) Linear penalty in value with $P = 2$ for infeasible solutions.

Figure B.5: Comparing MSE and SPO+ loss training with repairing infeasible solutions. Instance features contain some noise (5 uniform).

20 noise



(a) Reject infeasible solutions in evaluation.



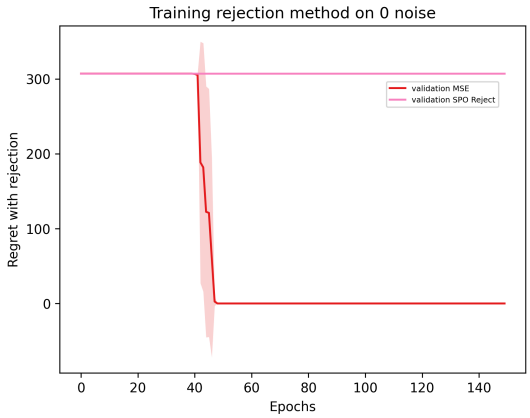
(b) Linear penalty in value with $P = 2$ for infeasible solutions.

Figure B.6: Comparing MSE and SPO+ loss training with repairing infeasible solutions. Instance features contain some noise (20 uniform).

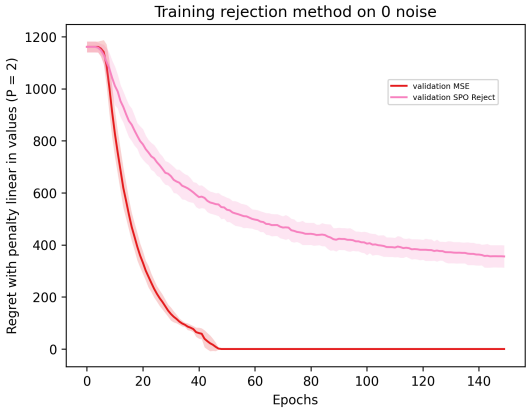
C

Performance for specific noise levels: training with rejection

0 noise



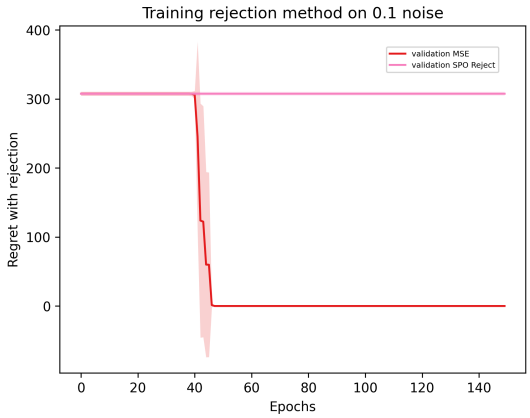
(a) Reject infeasible solutions in evaluation.



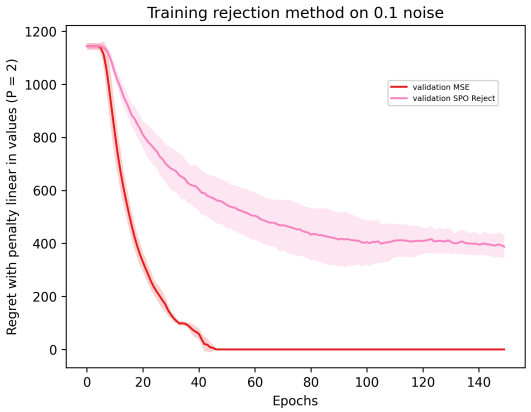
(b) Linear penalty in value with $P = 2$ for infeasible solutions.

Figure C.1: Comparing MSE and SPO+ loss training with rejecting infeasible solutions. Instance features contain no noise.

0.1 noise



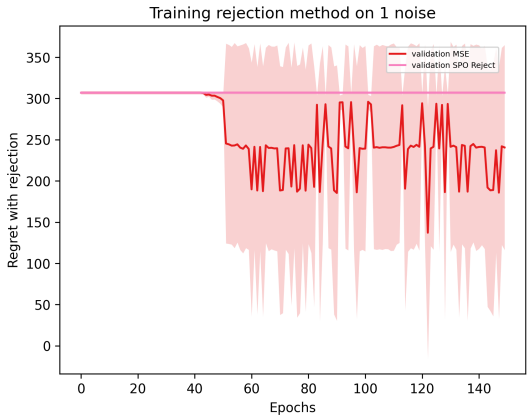
(a) Reject infeasible solutions in evaluation.



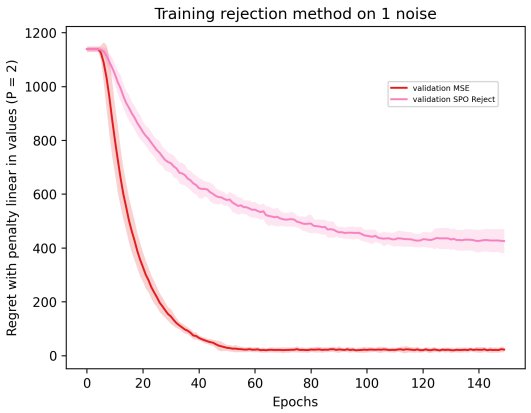
(b) Linear penalty in value with $P = 2$ for infeasible solutions.

Figure C.2: Comparing MSE and SPO+ loss training with rejecting infeasible solutions. Instance features contain some noise (0.1 uniform).

1.0 noise



(a) Reject infeasible solutions in evaluation.

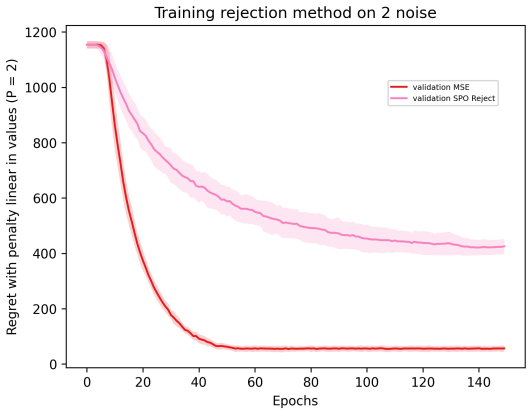


(b) Linear penalty in value with $P = 2$ for infeasible solutions.

Figure C.3: Comparing MSE and SPO+ loss training with rejecting infeasible solutions. Instance features contain some noise (1 uniform).

2 noise

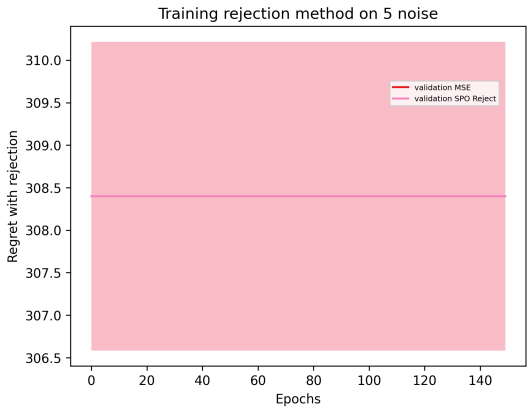
(a) Reject infeasible solutions in evaluation.



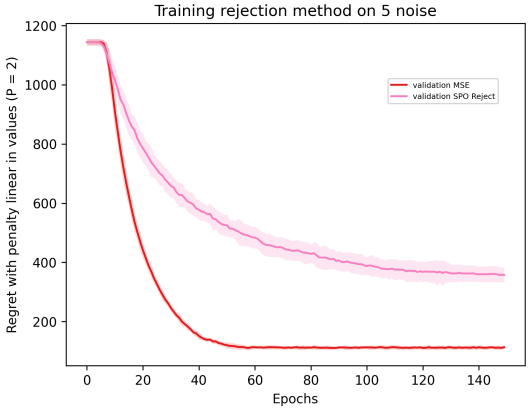
(b) Linear penalty in value with $P = 2$ for infeasible solutions.

Figure C.4: Comparing MSE and SPO+ loss training with rejecting infeasible solutions. Instance features contain some noise (2 uniform).

5 noise



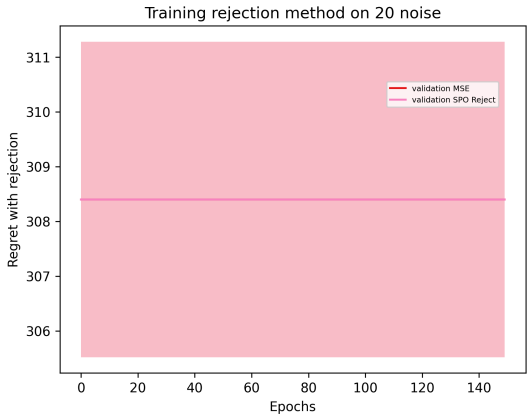
(a) Reject infeasible solutions in evaluation.



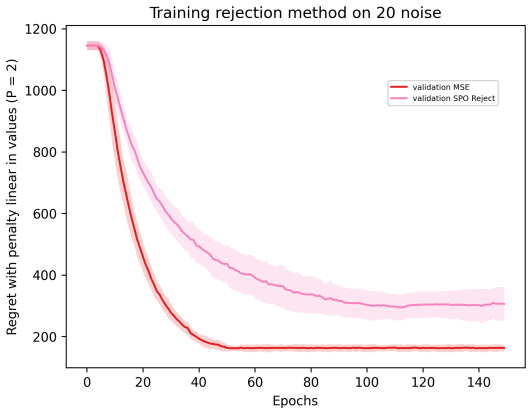
(b) Linear penalty in value with $P = 2$ for infeasible solutions.

Figure C.5: Comparing MSE and SPO+ loss training with rejecting infeasible solutions. Instance features contain some noise (5 uniform).

20 noise



(a) Reject infeasible solutions in evaluation.



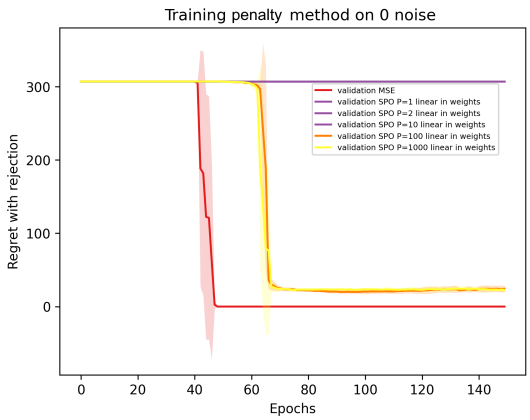
(b) Linear penalty in value with $P = 2$ for infeasible solutions.

Figure C.6: Comparing MSE and SPO+ loss training with rejecting infeasible solutions. Instance features contain some noise (20 uniform).

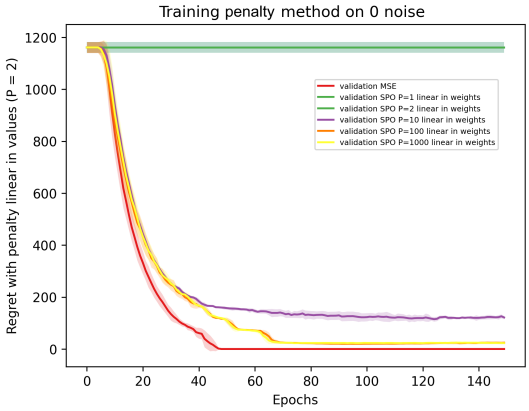
D

Performance for specific noise levels: training with penalty linear in weights

0 noise



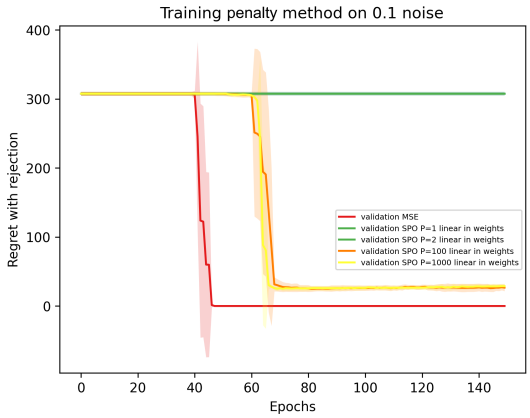
(a) Reject infeasible solutions in evaluation.



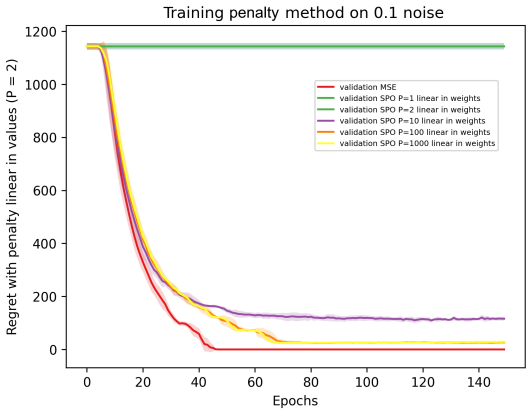
(b) Linear penalty in value with $P = 2$ for infeasible solutions.

Figure D.1: Comparing MSE and SPO+ loss training with penalty linear in weights and different P values. Instance features contain no noise.

0.1 noise



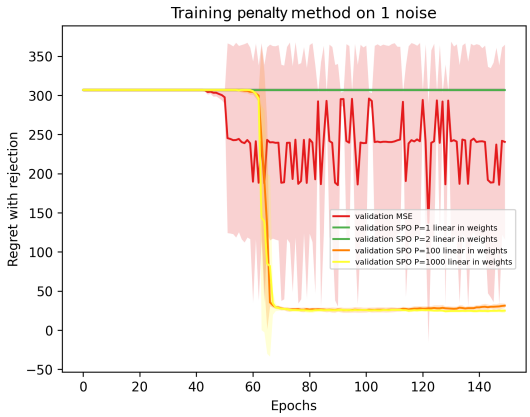
(a) Reject infeasible solutions in evaluation.



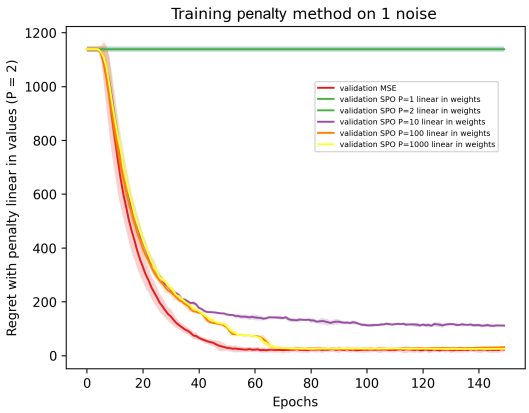
(b) Linear penalty in value with $P = 2$ for infeasible solutions.

Figure D.2: Comparing MSE and SPO+ loss training with penalty linear in weights and different P values. Instance features contain some noise (0.1 uniform).

1.0 noise



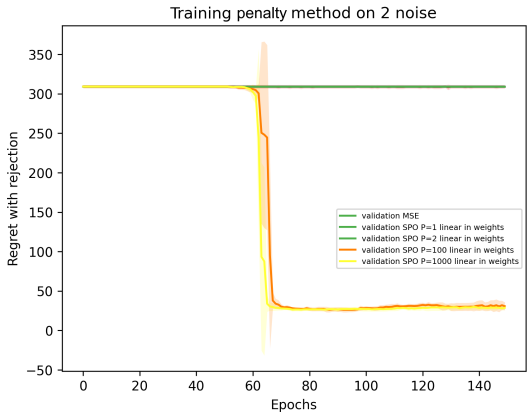
(a) Reject infeasible solutions in evaluation.



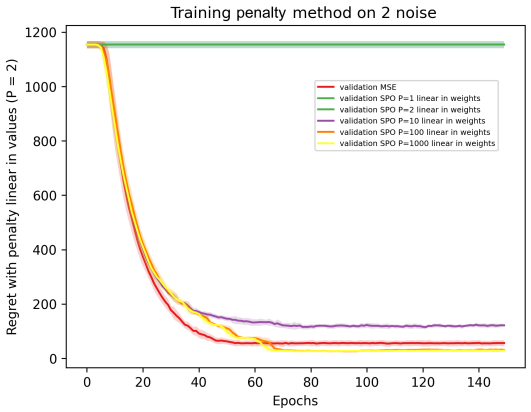
(b) Linear penalty in value with $P = 2$ for infeasible solutions.

Figure D.3: Comparing MSE and SPO+ loss training with penalty linear in weights and different P values. Instance features contain some noise (1 uniform).

2 noise



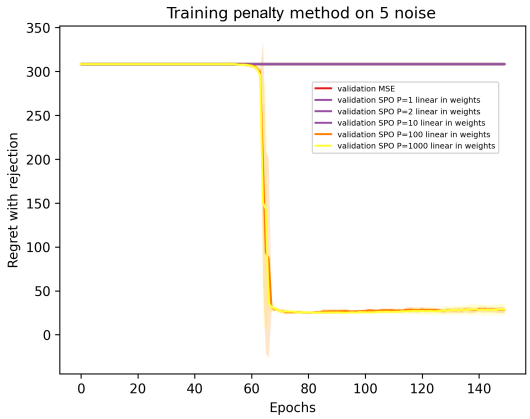
(a) Reject infeasible solutions in evaluation.



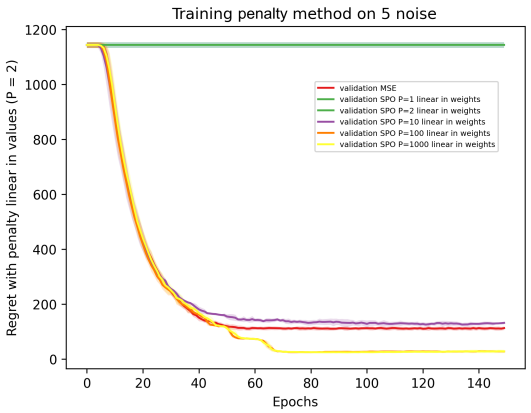
(b) Linear penalty in value with $P = 2$ for infeasible solutions.

Figure D.4: Comparing MSE and SPO+ loss training with penalty linear in weights and different P values. Instance features contain some noise (2 uniform).

5 noise



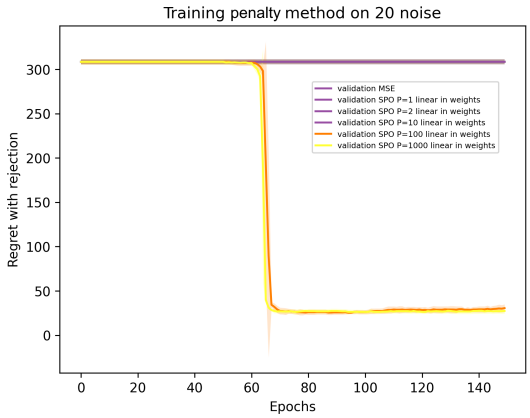
(a) Reject infeasible solutions in evaluation.



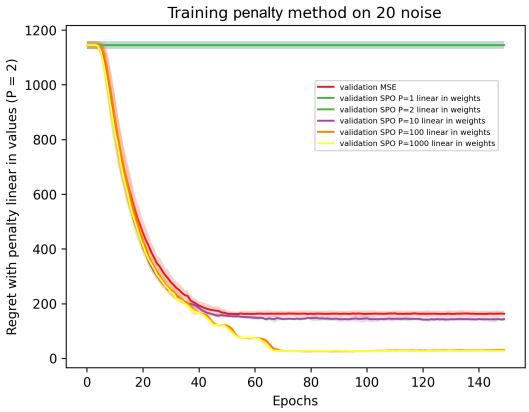
(b) Linear penalty in value with $P = 2$ for infeasible solutions.

Figure D.5: Comparing MSE and SPO+ loss training with penalty linear in weights and different P values. Instance features contain some noise (5 uniform).

20 noise



(a) Reject infeasible solutions in evaluation.



(b) Linear penalty in value with $P = 2$ for infeasible solutions.

Figure D.6: Comparing MSE and SPO+ loss training with penalty linear in weights and different P values. Instance features contain some noise (20 uniform).

E

Preliminary results: comparing exact and approximate predict+optimize methods for problems with uncertainty in the constraint parameters

An overview of several methods for predict+optimize is provided in Section 2.5. In this section, a distinction is made between approximate methods that indirectly aim to lower the regret, and exact methods which can directly learn to lower the regret. Both types have not been applied to problems with uncertainty in the constraint parameters before. In this work, a novel approach is presented to apply the approximate SPO+ loss method to problems with uncertainty in the constraint parameters. Exact methods could provide more optimal models as they directly lower the regret, but this can come at the cost of additional computation time. It is thus useful to compare the performance of both methods and investigate which method can best be applied to (which type of) problems with uncertainty in the constraint parameters.

In the comparison of predict+optimize methods, the Branch & Learn framework (Hu et al., 2022a) is mentioned as an exact search method that can directly lower the regret for all problems with recursive solutions. Parallel to the writing of this thesis, the authors of the Branch & Learn framework are working on applying this exact predict+optimize method to problems with uncertainty in the constraint parameters. Similar to the approach in this work, their method corrects the regret using a correction method for infeasible solutions in the regret calculation (Hu et al., 2022b).

In collaboration with them, an experiment is created to compare their approach (B&L-c) to the best-performing approach of this thesis: SPO+ loss-based learning with a penalty method linear in the weight of the overweight items, and $P = 1000$ (SPO-c). This experiment is executed on the 0-1 knapsack problem, with real-life data from the ICON energy-aware scheduling competition, which also appears in previous work on predict+optimize (Demirović et al., 2019). As this data relates to a setting with uncertain item values, a translation to a problem with uncertain weights is necessary. This is done by utilizing the original item values as the item weights, and generating new item values which are directly correlated to the original item weights, with a multiplication factor and a small, normally distributed, variance. Three capacities are selected (100, 200, 300) based on an analysis of the item weights.

For a full comparison, both approaches are compared on two problem types: one where there is a hard constraint, and one where there is a soft constraint. The hard constraint implies that all solutions that are infeasible on the validation set are evaluated as 0 objective value, as described in Algorithm 2. The soft constraint uses the penalty method linear in the value of the overweight items as described in Algorithm 4, with a penalty factor of 0.1. The performance of 6 different methods is compared:

1. Branch and learn with correctional regret (B&L-c), an exact predict+optimize method, which trains the prediction model with the corrected regret function that evaluates infeasible solutions the same as in evaluation (a hard or soft constraint respectively).
2. SPO+ loss-based learning with a penalty linear in the weight of the overweight items and $P = 1000$ (SPO-c)
3. Four classical regression methods: Ridge regression, k-nearest neighbours regression (KNN), classification and regression tree (CART) and random forest regression (RF), all of which train the prediction models with a prediction-error-based loss function.

Due to time constraints, only preliminary results of this experiment are available, and a thorough analysis of these results is left for further research.

Capacity	B&L-c	SPO-c	Ridge	KNN	CART	RF
100	68.93±6.22	86.88	72.69±3.88	75.04±6.01	78.03±4.91	74.39±5.18
200	109.29±6.75	124.87	120.67±5.90	115.75±6.38	119.57±7.01	115.72±6.00
300	117.03±5.77	132.72	160.50±7.24	152.07±5.96	155.65±11.25	151.01±7.17

Table E.1: The regret of different learning methods on the 0-1 knapsack problem with a soft capacity constraint.

Capacity	B&L-c	SPO-c	Ridge	KNN	CART	RF
100	133.03±12.55	146.135	152.23±11.69	210.02±14.50	231.98±14.10	180.60±12.74
200	337.65±25.48	312.81	401.72±26.33	499.96±33.25	566.34±33.86	462.42±30.25
300	529.60±38.34	459.47	616.32±52.28	721.81±46.79	797.30±45.00	684.47±40.46

Table E.2: The regret of different learning methods on the 0-1 knapsack problem with a hard capacity constraint.

Table E.1 shows the results for the 0-1 knapsack problem with a soft constraint. Here, B&L-c has the best performance in all problem settings. In Table E.2 the results for the 0-1 knapsack problem with a hard constraint are shown. Here SPO-c appears as the best-performing method for capacity 200 and 300, whereas B&L-c has the best performance for capacity 100. No standard deviation is provided for SPO-c due to time constraints. The next steps for the comparison are: gathering the final results, creating a more thorough analysis, applying both methods to several different problem settings and including the computation time in the comparison.