An investigation into prediction + optimisation for the knapsack problem

Emir Demirović[1], Peter J. Stuckey[2], James Bailey[1], Jeffrey Chan[3], Chris Leckie[1], Kotagiri Ramamohanarao[1], and Tias Guns[4]

[1] University of Melbourne, Australia
[2] Monash University and Data61, Australia
[3] RMIT University, Australia
[4] Vrije Universiteit Brussel, Belgium

Abstract. We study a *prediction+optimisation* formulation of the knapsack problem. The goal is to predict the profits of knapsack items based on historical data, and afterwards use these predictions to solve the knapsack. The key is that the item profits are not known beforehand and thus must be estimated, but the quality of the solution is evaluated with respect to the true profits. We formalise the problem, the goal of minimising expected regret and the learning problem, and investigate different machine learning approaches that are suitable for the optimisation problem. Recent methods for linear programs have incorporated the linear relaxation directly into the loss function. In contrast, we consider less intrusive techniques of changing the loss function, such as standard and multi-output regression, and learning-to-rank methods. We empirically compare the approaches on real-life energy price data and synthetic benchmarks, and investigate the merits of the different approaches.

Combinatorial optimisation is crucial in today's society and used throughout many industries. In this paper, we work with the fundamental knapsack problem, which has been studied for over a century and is well understood [17, 9]. It is studied in fields such as combinatorics, computer science, complexity theory, cryptography, and applied mathematics. It has numerous applications, including resource allocation problems where the aim is to select as many resources as possible under given financial constraints. The knapsack problem is NP-hard, though highly efficient solution methods exist for reasonably sized instances [9].

In traditional optimisation, it is assumed that all parameters, e.g. the profits and weights in a knapsack, are precisely known beforehand. In practice, these are often crude estimates based on domain expertise or historic data. As we enter the age of big data, large amounts of data is available and thus parameters can be estimated with greater precision. For example, ongoing promotion and current weather might influence the demand. The question that arises is whether such contextual data, together with historical data, can be used to improve decision making, i.e. solve the underlying optimisation problem more effectively.

Such problems are encountered in load shifting [10], where the aim is to create an energy-aware day-head schedule based on predicted hourly energy prices.

Traditional approaches to this problem consist of two phases: 1) use machine learning to estimate the problem parameters, and 2) optimise over the estimated parameters. However, Grimes et al. [10] and Mathaba et al. [16] have separately shown that in energy-aware scheduling, learning accurate values of the parameters by minimising the mean-square error of the predictions, a commonly used metric in machine learning, does not necessarily lead to solutions of better quality for the optimisation problem. The reason is that not all errors on estimated energy prices have an equal effect on the optimisation problem, but the machine learning algorithm does not take this into account.

The challenge is to incorporate information from the optimisation problem into the learning. The main difficulty is that learning techniques typically assume that the loss function is convex. However, given the combinatorial optimisation component, this is no longer holds. It is also not differentiable as this would require computing the gradient over the *argmax* of the optimisation problem. Intuitively, such a gradient would capture the direction in which the predicted values should change to lead to a solution that is closer to the true solution obtainable under perfect knowledge.

The discussed methods have been evaluated on combinatorial problems which are solvable in polynomial time. However, we direct our attention to *NP-hard* combinatorial problems, i.e. difficult problems for which the existance of a polynomial algorithm is not known. We investigate their use on the *knapsack*, a fundamental combinatorial problem, both unit-weighted (polynomially solvable) and weighted (NP-hard). The knapsack was chosen as it has a simple constraint, yet captures a difficult combinatorial optimisation problem, suitable for exploring the use of prediction + optimisation techniques in constraint optimisation.

This work falls into the wider research theme of combining machine learning and constraint optimisation [18]. Most research has focussed on using machine learning to improve the solving process, e.g. algorithm selection and hyperparameter optimisation [14] and using machine learning to improve MIP solvers [4]. This is different from our setting, where the aim is to develop machine learning algorithms specifically designed for use with combinatorial optimisation problems where the parameters, e.g. profit values for items in the knapsack problem, are estimated with machine learning rather than given precisely. In terms of modeling, constraint acquisition [1] uses machine learning techniques to learn structural constraints from data, while other works are concerned with finding the most likely parameters of given hard constraints [20]. Closely related, as predictions are used in the objective, is the emerging topic of constructive machine learning [22, 7], where the goal is to learn to synthesize structured objects from data, e.g. by interactively learning the preferences of a user and searching for the most preferred object. In contrast, our work is concerned with learning the weights of the objective on a per-instance basis. This has been done with a two-phase approach in practice, e.g. in energy-aware load shifting [16, 10].

We formalise the problem of minimising the *expected regret* and draw the relation to stochastic optimisation. We then investigate multiple approaches to formulate the machine learning problem for the knapsack: indirectly - a two-stage

method (predict then optimise); directly - by incoporating the optimisation objective as the loss function; and semi-directly - using domain-specific knowledge of the optimisation problem in the loss function, but without requiring to solve the constraint optimisation problem at each step.

To summarise, our contributions are as follows:

- We formalise the problem formulation in terms of *regret*;
- We investigate the relation between regret and surrogate loss functions;
- We propose two semi-direct methods based on appropriate semi-direct loss functions specifically designed for the knapsack problem;
- We empirically evaluate different strategies for prediction+optimisation on the knapsack problem with artificial and real-life energy-price data. Contrary to previous work, we show that direct methods do not outperform simple two-stage approaches on these benchmarks, demonstrating the difficulty that arises when NP-hard problems are introduced.

1 Formalisation

A combinatorial optimisation problem is a tuple $CSP(X, D, C, o)$, where X is the set of variables, D the domain of each variable, C a set of constraints over subsets of the variables, and o an objective function over X that needs to be maximized. A CSP is often a parameterized representation of a class of problems.

For example, the knapsack problem consists of selecting a number of items from a set, such that the total value is maximized. Each item has a weight and the sum of the weights of the selected items may not exceed a given capacity threshold. Let there be n items, then the knapsack problem can be formalised as a CSP with $X = \{x_1, \ldots, x_n\}$ variables, domain $D(x_i) = \{0, 1\}, \forall i$ representing whether an item is in or out and constraint set $C = \{\sum_i w_i x_i \leq c\}$ and objective $o = \sum_i v_i x_i$. Let $V = (v_1, \ldots, v_n)$ be the set of profits, $W = (w_1, \ldots, w_n)$ the set of weights and c the capacity. Any given assignment of parameters (V, W, c) is an instance of the parameterized knapsack CSP.

In the above example, we may not know the profits V of the items in advance, but we may have attributes $\boldsymbol{a}$ such as what temperature it is, how popular the items are, whether they are in promotion etc. At the same time, we have a set of historical $\{(\boldsymbol{a}, V)\}$ data of tuples, with values for the same attributes as well as the (post-hoc) profits of the items that day. This historical data can be used to predict the most likely item profits V given today's attributes.

More formally, let us define θ as the set of parameters of a CSP. Then, in a prediction + optimisation setting, this consists of two disjoint sets $\theta = \theta^p \cup \theta^y$ where the θ^p parameters are assumed given and the θ^y parameters will need to be predicted. A problem instance hence does not consist of a tuple (θ^p, θ^y), but rather a tuple $(\theta^p, \boldsymbol{a})$ with $\boldsymbol{a}$ the attributes of this problem instance. From a set of historical data $\{(\boldsymbol{a}, \theta^y)\}$ one can then *learn* a function f that outputs an estimated set of parameters $f(\boldsymbol{a}) \approx \theta^y$ such that one obtains the problem parameters $(\theta^p, f(\boldsymbol{a}))$ of the parameterized CSP.

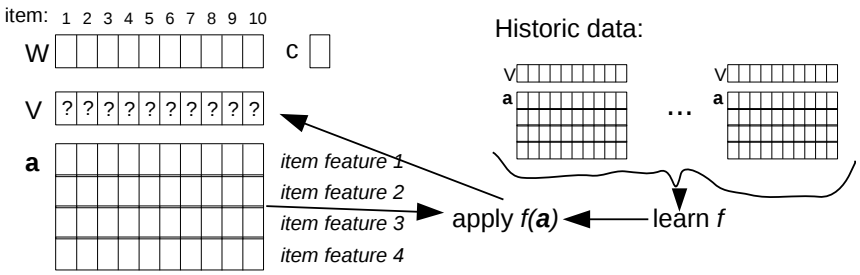

Fig. 1. Schematic figure of the components for a knapsack with 10 items, weights W, capacity c, unknown profits V and 4 attributes per item.

In the knapsack case we consider here, $\theta^p = (W, c)$ consists of the weights and the capacity, while $\theta^y = V$ are the item profits. Figure 1 shows a graphical example where it is assumed that the attributes $\boldsymbol{a}$ consist of 4 features per item (the columns), with $\boldsymbol{a}$ the union of them (one long sequence).

Solution quality In a standard machine learning setting, one assumes a training set $\{(\boldsymbol{a}, \theta^y)\}_{train}$ and an independent test set $\{(\boldsymbol{a}, \theta^y)\}_{test}$. One can then evaluate the quality of a model f trained on $\{(\boldsymbol{a}, \theta^y)\}_{train}$ by measuring how it performs on $\{(\boldsymbol{a}, \theta^y)\}_{test}$ through a loss function $loss(f(\boldsymbol{a}), \theta^y)$ over instances $(\boldsymbol{a}, \theta^y)$. Following the risk minimisation framework [23], the goal of machine learning is then to minimise the expected loss:

$$\mathbb{E}[loss(f(\boldsymbol{a}), \theta^y)]$$

In our prediction + optimisation setting, the predictions are merely *intermediary results* and the true goal is to minimise the error of the optimisation procedure when using the predictions:

$$\mathbb{E}[loss(f(\boldsymbol{a}), \theta^y, \theta^p)]$$

Such a loss is called the *task* loss in [5]. A natural task loss in our setting is to consider the *regret* of using the predictions rather than using the (apriori unknown) actual profits, that is, the difference in *true* solution quality obtained when optimising with the predictions as opposed to the real profits. Let $s(\theta^y, \theta^p)$ be an optimal solution to a (θ^y, θ^p) parameterized CSP, where we write $s(\theta^y)$ when θ^p is clear from the context. We can formalise the regret of f for a single instance $(\boldsymbol{a}, \theta^y, \theta^p)$ as:

$$regret(f(\boldsymbol{a}),\theta^y, \theta^p) = o(s(\theta^y, \theta^p), \theta^y, \theta^p) - o(s(f(\boldsymbol{a}), \theta^p), \theta^y, \theta^p) \qquad (1)$$

where $o(X, \theta^y, \theta^p)$ denotes the value of the CSP's objective function $o()$ with solution X when using θ^y and θ^p.

In case of knapsack:

$$regret(f(\boldsymbol{a}),V,(W,c)) = V * s(V,(W,c)) - V * s(f(\boldsymbol{a}),(W,c)) \tag{2}$$

Observe how, when computing the quality $o(X^y, \theta^y, \theta^p)$ and $o(X^f, \theta^y, \theta^p)$, in both cases the real parameters θ^y are used, e.g. profits V in case of knapsack. The regret hence quantifies the difference in the objective value when using the estimated profits compared to the ideal case, i.e. using perfect information.[5]

The goal of prediction + optimisation is hence to devise prediction and optimisation methods such that the expected regret is minimised, i.e.

$$\mathbb{E}[regret(f(\boldsymbol{a}), \theta^y, \theta^p)]$$

We assume that the parameters θ^p are independent of $\boldsymbol{a}$ and fixed, e.g. the weights in case of knapsack. The **learning problem** for a given θ^p is hence that of learning an f:

$$\min_f \mathbb{E}[regret^{\theta^p}(f(\boldsymbol{a}), \theta^y)]$$

where we write the θ^p in superscript to make clear that it is constant.

The main difficulty here is that one function evaluation of $regret()$ requires solving a discrete optimisation problem. This is different from the usual setting in machine learning, where the loss is a (differentiable) function rather than the result of a set of non-decomposable discrete optimisation problems.

2 Relation to stochastic optimisation

Prediction+optimisation is a setting in which one has a large sample of data from an unknown distribution, one feature vector from that distribution and an optimisation problem where we want to minimise expected regret. We discuss the setting in a stochastic optimisation problem (see e.g. [21]).

Assume, as we have done so far, that the stochasticity is only in the parameters of the objective function of the optimisation problem. Let this function be denoted by $o(X^y, \theta^y)$. It can be written as a stochastic problem as follows:

$$\max_X \mathbb{E}[o(X, \theta^y)] \ s.t. \ C(X)$$

This is a simple stochastic problem, and as the uncertainty is only on the objective, not the constraints, there are no second stage decisions or recourse. The goal is to find one X that is good in expectation, over some distribution of θ^y.

One key difference in prediction+optimisation, is that we assume the presence of a single feature vector $\boldsymbol{a}$ of observed variables which are correlated with

[5] Note the different problem where profits are known and the weights are learned is more complicated, since we may need some form of recourse mechanism to repair inconsistent decisions X^f.

the parameters θ^y. Hence, the knowledge of $\boldsymbol{a}$ changes the probability distribution over θ^y that we should optimise over. Indeed, we should optimise over the conditional distribution:

$$\max_X \mathbb{E}[o(X, \theta^y)|\boldsymbol{a}] \ s.t. \ C(X)$$

When given a finite sample of observations $\{(\boldsymbol{a'}, \theta^y)\}$ we can empirically approximate it by the following:

$$\max_X \sum_{(\boldsymbol{a'}, \theta^y)} o(X, \theta^y) * P[(\boldsymbol{a'}, \theta^y)|\boldsymbol{a}] \ s.t. \ C(X)$$

that is, a probability weighted sum of the objective. The remaining question is now: what is the value of $P[(\boldsymbol{a'}, \theta^y)|\boldsymbol{a}]$, namely the probability of an historical θ^y with features $\boldsymbol{a'}$ given the feature vector $\boldsymbol{a}$? Stochastic optimisation does not provide an answer, as it assumes that probability of each scenario is given.

From a machine learning point of view, we can take inspiration from case-based reasoning and more specifically *nearest neighbor* methods that use distance information. More specifically, we can replace the probability $P[(\boldsymbol{a'}, \theta^y)|\boldsymbol{a}]$ by the (inverse of the) Euclidean distance between $\boldsymbol{a}$ and $\boldsymbol{a'}$.

In fact, the k-nearest neighbor (k-NN) classification method [3], takes the k nearest neighbors and assign them a probability of $1/k$ while assigning all other instances a probability of 0. Let $kn(\boldsymbol{a})$ denote k-$nearest(\{(\boldsymbol{a'}, \theta^y)\}, \boldsymbol{a})$, that is, the k instances most near to $\boldsymbol{a}$. The predicted value is then the weighted average over the samples: $\theta^f = \sum_{(\boldsymbol{a'}, \theta^y) \in kn(\boldsymbol{a})} \frac{1}{k} * \theta^y$. A weighted ($k$) nearest neighbors weights each instance by the inverse distance $\theta^f = \sum_{(\boldsymbol{a'}, \theta^y) \in kn(\boldsymbol{a})} \frac{1}{d(\boldsymbol{a'}, \boldsymbol{a})} * \theta^y$. When θ^y is a list, this is done for each component individually.

In our stochastic problem, we can also use the inverse distance as probability estimate, leading to the following:

$$\max_X \sum_{(\boldsymbol{a'}, \theta^y) \in kn(\boldsymbol{a})} o(X, \theta^y) * \frac{1}{d(\boldsymbol{a'}, \boldsymbol{a})} \ s.t. \ C(X)$$

When θ^y is a vector of values (as we assume), and the objective is *linear* wrt θ, then we can write the objective in its decomposed form $o(X, \theta^y) = \sum_i o_i(X) * \theta^y_i$. E.g. in case of knapsack where $o_i(X) = X_i$ and $\theta^y_i = v_i$ we have $o(X, \theta^y) = X * V = \sum_i X_i * v_i$. We can then do the following rewriting:

$$\sum_{(\boldsymbol{a'}, \theta^y) \in kn(\boldsymbol{a})} o(X, \theta^y) * \frac{1}{d(\boldsymbol{a'}, \boldsymbol{a})} \tag{3}$$

$$= \sum_{(\boldsymbol{a'}, \theta^y) \in kn(\boldsymbol{a})} \sum_i o_i(X) * \theta^y_i * \frac{1}{d(\boldsymbol{a'}, \boldsymbol{a})} \tag{4}$$

$$= \sum_i o_i(X) * \big(\sum_{(\boldsymbol{a'}, \theta^y) \in kn(\boldsymbol{a})} \frac{1}{d(\boldsymbol{a'}, \boldsymbol{a})} * \theta^y_i \big) \tag{5}$$

$$= \sum_i o_i(X) * \theta^f_i \tag{6}$$

where $\theta_i^f = \sum_{(\boldsymbol{a}', \theta^y) \in kn(\boldsymbol{a})} \frac{1}{d(\boldsymbol{a}', \boldsymbol{a})} * \theta_i^y$, the prediction of the distance-weighted k-nearest neighbor method for component i.

Hence, in the simple stochastic setting corresponding to prediction+optimisation where the objective is linear, doing a *stochastic* optimisation over multiple distance-weighted scenarios, coincides with one standard *deterministic* optimisation over the distance-weighted kNN predictions θ^f.

3 Machine learning formulations

As shown in the previous section, stochastic optimisation is not sufficient for solving a prediction + optimisation problem. Thus, we now turn to machine learning methods. We consider three different learning approaches:

Indirect methods use a standard learning method and loss function that is independent of the optimisation problem;

Direct methods do the learning using a convex surrogate of the regret function as loss function, which requires solving the optimisation problem repeatedly;

Semi-direct methods, which we introduce in this paper for knapsack, that use a convex surrogate of the regret which takes key properties of the optimisation problem into account but which does not require repeatedly solving it.

3.1 Indirect learning formulations

Following the formalisation of the problem, the most natural setting is to consider the learning as a problem of mapping the feature vector $\boldsymbol{a}$ to the list of values θ^y. This is known as **multi-output regression** [2] and many standard regression methods can be extended to multi-output regression, where the loss function is the sum of the losses of each individual prediction:

$$\mathbb{E}[loss_{mo}(f(\boldsymbol{a}), \theta^y)] = \mathbb{E}[\frac{1}{n} \sum_i loss(f(\boldsymbol{a})_i, \theta_i^y)]$$

The assumption of multi-output regression, compared to pointwise regression, is that the values should be learned with respect to other items in the same group. For example, because there are correlations between the values that are always present but can get lost when predicting them independently.

Standard regression can also be used, when a set of attributes $\boldsymbol{a}_i$ is given for *each* item i, e.g. $\boldsymbol{a} = \cup_i \boldsymbol{a}_i$. Ideally, any correlation present between items can be indirectly accounted for through the values of the attributes, for example that sales of icecream products rise with hot weather.

The data is in a form where existing regression methods can be used to estimate the $f(\boldsymbol{a}_i)$'s. The expected loss is the average loss over all items:

$$\mathbb{E}[loss(f(\boldsymbol{a}_i), \theta_i^y)]$$

Regression will predict the values, and good predictions should lead to good optimisation solutions. However, predictions are estimates that have errors, and the errors can interact in the optimisation. This is not captured in the loss functions of regression.

Learning-to-rank Following the observation of [10] that for load-shifting problems the *correlation* between the rankings of the predictions and the true values is indicative for the optimisation quality, we may opt to learn to *rank* the items consistently. This is studied in machine learning as learning-to-rank.

The learning-to-rank problem has its roots in information retrieval [15]. In this setting, one assumes a set of *queries*, such as keywords entered in a search engine, and a set of *documents* that need to be ranked according to their relevance with the query. A relevance grade is given to each query-document pair. The goal is to rank the documents in decreasing order of relevance.

Learning-to-rank methods do this by learning a function f over a feature vector representing a query+document instance $f(\boldsymbol{a}_i)$. Typically some features are related to the query and some to the document. The value the function returns has no connection to the actual value v_i other than if $v_i > v_j$ then it should be that $f(\boldsymbol{a}_i) > f(\boldsymbol{a}_j)$. The loss function is a 0-1-like loss defined over each pair of instances where one has a higher relevance than the other:

$$\mathbb{E}[\sum_{(i,j),v_i>v_j} loss(I_{\{f(\boldsymbol{a}_i)>f(\boldsymbol{a}_j)\}}, 1)] \tag{7}$$

where $I_{\{.\}}$ is 1 if the condition inside is true, and 0 otherwise. Furthermore, ranking can be decomposed into pairs and hence it is assumed the elements only interact in pairwise ways. An optimisation problem typically trades off different decision variables to each other. This can go beyond pairwise interactions, so a pairwise loss is just another problem-independent surrogate.

One can use learning to rank methods in a prediction + optimisation setting by treating each optimisation instance $(\boldsymbol{a}, \theta^y)$ as one query with $|\theta^y|$ documents, each with relevance θ^y_i. Thus, in our setting, for a given knapsack instance (query), we wish to order the items (documents) by profit (relevance).

SVMRank . In [12], the author considers learning a linear function over feature vectors, aiming to learn a ranking function as defined in Eq. 7. However computing the optimal coefficients for the resulting linear function is NP-hard. As an alternative, the problem can be approximated[12]:

$$min\ \frac{1}{2}\overrightarrow{w} \cdot \overrightarrow{w} + K \sum \xi_{i,j,k} \tag{8}$$

$$\text{where } \overrightarrow{w} \cdot \overrightarrow{a_{i,j}} \geq \overrightarrow{w} \cdot \overrightarrow{a_{i,k}} + 1 - \xi_{i,j,k} \qquad q_i \in Q, \forall (j,k) \in q_i \tag{9}$$

$$\xi_{i,j,k} \geq 0, \tag{10}$$

where the first term in Eq. 8 is the regularisation term, Q is the set of queries with each query being a partial order, $\xi_{i,j,k}$ is the error variable for the *j-th* and *k-th* item in the *i-th* query, K is a trade-off coefficient between regularisation and training-error, and $a_{i,j}$ is the feature vector for the *j-th* item in the *i-th* query. The resulting problem is a convex quadratic program and thus the solution can be obtained using generic continuous optimisation methods. Equations 8-10 are solved to obtain the weight vector w for a linear ranking function based on item

attributes, i.e. $f(a) = \sum(w_i * a_i)$, such that $f(a_1) > f(a_2)$ indicates that the first item is more profitable. Intuitively, the resulting function aims at capturing the rankings as faithfully as possible for the historical data: each benchmark defines an ordering of items based on their profitability. We note that specialised techniques have been devised for solving quadratic programs of this particular form [11, 13].

3.2 Direct learning formulations

We consider two direct learning formulations that were recently proposed [8, 24]. These methods use historical data to compute the gradient for combinatorial problems, to minimise the regret by gradient descent.

The aim of *gradient descent* is to compute the minimising point x_{min} for a function f, i.e. $\forall x : f(x_{min}) \leq f(x)$. Starting from an initial point, the algorithm iteratively moves the point towards a local minima according to the direction of its gradient.

Smart Predict then Optimise In this approach [8] the authors aim to directly optimise the regret of an optimisation problem that has a linear objective function. For the case of knapsack, this is $\sum_{(\boldsymbol{a},V)} V * s(V) - V * s(f(\boldsymbol{a}))$ where we recall that $s(L)$ returns the optimal solution when solving the CSP with values L. They derive a clever surrogate loss function using upper bounds motivated by duality, a scaling approximation and a first-order approximation of the optimal cost with respect to the predictions [8]. The resulting surrogate loss for linear objectives is the following:

$$loss_{SPO}(f(\boldsymbol{a}), \theta^y) = (\theta^y - 2f(\boldsymbol{a})) * s(\theta^y - 2f(\boldsymbol{a})) + 2f(\boldsymbol{a}) * s(\theta^y) - \theta^y * s(\theta^y)$$

The authors show that this indeed an upper-bounding surrogate loss to regret, and that it is convex in $f(\boldsymbol{a})$. This means that one can optimise over this loss function, for example using stochastic gradient descent methods as used in neural networks. Detailed information on what the (sub)gradients are in this case is given in [8].

For completeness we note that to avoid degenerate cases, the result of the solving method $s(\cdot)$ used should be a valid upper bound, so in case multiple optimal solutions to $s(\cdot)$ exist, the one with the best regret should be chosen. We avoid this issue for the knapsack case by solving the following greedy relaxation that has a unique optimal solution, namely it orders all items by profitability (v_i/w_i) and selects all items with the highest profitability not yet selected and distributes capacity *equally* among them, this procedure is repeated until no more capacity is left.

Quadratic Programming Task Loss (QPTL) Another approach recently proposed is not limited to linear objectives, but encompasses convex optimisation [24]. To derive gradients they apply the chain rule to a task loss, such as regret, as follows:

$$\frac{dregret(f(\boldsymbol{a}), \theta^y)}{df(\boldsymbol{a})} = \frac{dregret(f(\boldsymbol{a}), \theta^y)}{ds(f(\boldsymbol{a}))} \frac{ds(f(\boldsymbol{a}))}{df(\boldsymbol{a})}$$

The first component is the gradient of the regret with respect to the solution $s(f(\boldsymbol{a}))$. The difficult part is the second part, which requires differentiation over the optimisation. They do this by differentiating through the Karush-Kuhn-Tucker conditions around the optimal point. With X, λ being the primal and dual solutions of solving the convex optimisation problem $s(f(\boldsymbol{a}))$ with linear equalities represented by $BX = c$, and $\hat{f}$ the shorthand for $f(\boldsymbol{a})$ of a specific $\boldsymbol{a}$, one needs to solve the following set of differential equations to obtain the gradients $\frac{dX}{d\hat{f}} = \frac{ds(f(\boldsymbol{a}))}{df(\boldsymbol{a})}$:

$$\begin{bmatrix} \nabla^2_X o(X, \hat{f}) & B^T \\ diag(\lambda)B & diag(BX - c) \end{bmatrix} \begin{bmatrix} \frac{dX}{d\hat{f}} \\ \frac{d\lambda}{d\hat{f}} \end{bmatrix} = \begin{bmatrix} \frac{d\nabla_X o(X, \hat{f})}{d\hat{f}} \\ 0 \end{bmatrix}$$

As described in [24] this can also be applied to linear relaxations of combinatorial optimisation problems. One difficulty is that $\nabla^2_X o(X, \hat{f})$ is always zero for linear programs. The authors suggest to add a weighted quadratic term to the linear program to overcome this, e.g. solve:

$$max\ \hat{f}^T X - \gamma ||X||_2^2\ s.t.\ BX = c, GX \leq h$$

for some small-valued parameter γ. As explained in [24], in this case the differential equations become, with I the identity matrix:

$$\begin{bmatrix} \gamma I & B^T \\ diag(\lambda)B & diag(BX - c) \end{bmatrix} \begin{bmatrix} \frac{dX}{d\hat{f}} \\ \frac{d\lambda}{d\hat{f}} \end{bmatrix} = \begin{bmatrix} I \\ 0 \end{bmatrix}$$

The resulting gradient can be used in the chain rule and backpropagated with stochastic gradient descent [24].

3.3 Semi-direct learning formulations

We now introduce a new class of techniques that are in between indirect and direct methods: *semi-direct methods*. Semi-direct methods do not require solving the optimisation problem repeatedly like direct methods, but do use information from the optimisation problem in the loss function.

Profitability-learning. A first example is in case of weighted knapsacks, one can use regression to **predict the profitability** V_i/W_i of items rather than the profit V_i. The greedy approach to solving weighted knapsacks is to sort them by profitability and iteratively select as many as capacity allows. While we can assume that the weights W are independent of the features $\boldsymbol{a}$, they rescale the values V_i and hence also its errors: errors on items with larger weights (and hence smaller profitability) will be relatively smaller than equal errors on items with smaller weights. In particular for weighted knapsacks we can use learning-to-rank methods on profitability as a surrogate for regret.

Simplifying QPTL When implementing the QPTL approach [24] we observed that for knapsack, with its simple constraint of $WX \leq c$ with $B = W$ the weight vector and c the capacity, the result of solving the linear equations is often non-informative: if the solution maximizes the capacity constraint then $diag(BX - c) = 0$ and $diag(\lambda)B\frac{dX}{d\hat{f}} = 0$ forces the $\frac{dX}{d\hat{f}}$ gradients to zero, or to a below-precision small value. If $diag(BX - c) > 0$ then $\frac{d\lambda}{d\hat{f}}$ may be forced to zero leading to the $\frac{dX}{d\hat{f}}$ having the huge value of $1/\gamma$ Both cases are not meaningful and have to be guarded against, for example by replacing the gradients with a small negative constant.

In fact, we can go as far as cutting out the QP and the solving of the differential equations. The gradient is then simply $\frac{dregret(f(\boldsymbol{a}),\theta^y)}{ds(f(\boldsymbol{a}))}$ which is θ^y for knapsack. The gradient will hence push the predictions in the direction of the true values, *independent of the actual prediction it gives*. While unusual, the motivation is two-fold: 1) the instability of solving the differential equations has this effect in many cases and 2) the magnitude of the gradient for each item is proportional to the true value and hence the gradient updates are also proportional to it and so will the predictions be over time. For linear objective functions, which are scale invariant, that is, the same optimal solution is found when rescaling all weights, these 'proportional to true value' updates are desirable.

Specialising SVMRank for knapsack A key observation is that not all pairs (i, j) in Eq. 7 contribute equally towards minimising the expected regret in the case of the knapsack problem. This is illustrated in the following example.

Example 1. Consider a unit-weighted knapsack problem with four items of true profits $[10, 20, 30, 40]$ and capacity 2, and three ranking functions giving the following ranking values: $f = [-10, 0, 5, 10]$, $g = [10, 0, 30, 20]$, and $h = [1, 3, 2, 4]$. According to Eq. 7, the functions f, g, and h have $0, 2, 1$ violations. The function f, as it has zero violations, captures the ranking perfectly and thus achieves the optimal solution $[0, 0, 1, 1]$. However, function g allows us to obtain the optimal solution as well, despite having two violations, because they do not affect the two highest items being ranked first. In contrast, function h only has one violation, but it misranks two critical items leading to a worse solution $[0, 1, 0, 1]$ with regret $(30 + 40) - (20 + 40) = 10$.

Based on the observation above, we modified the SVMRank objective function (Eq. 8) by adding constant weights $e_{i,j,k}$ to the error variables to account for the differences among error variables:

$$\frac{1}{2}\overrightarrow{w} \cdot \overrightarrow{w} + K \sum e_{i,j,k} \cdot \xi_{i,j,k}.$$

We propose a weight-scheme where items within a query are partitioned based on whether their profitability exceeds a given threshold. Weights for error variables within the same query are set one if the two items are not in the same partition, and zero otherwise. The threshold is taken as the value of the profitability of the least profitable item in the solution given by the linear relaxation

of the query. Formally, let $p_{q_i,j}$ be the profitability for the *j-th* item in the *i-th* query, and T_i be the threshold for the *i-th* query, the weight-scheme is given as:

$$e_{i,j,k} = \begin{cases} 1, (p_{q_i,j} \geq T_i \wedge p_{q_i,k} < T_i) \vee (p_{q_i,j} < T_i \wedge p_{q_i,k} \geq T_i) \\ 0, (p_{q_i,j} \geq T_i \wedge p_{q_i,k} \geq T_i) \vee (p_{q_i,j} < T_i \wedge p_{q_i,k} < T_i) \end{cases}$$

Example 1. (continued) The four items with profits $(p_1, p_2, p_3, p_4) = (10, 20, 30, 40)$ compose a query q. Given capacity 2, the threshold value is set to $T = 30$ and the items are partitioned into $\{1, 2\}$ and $\{3, 4\}$. Therefore, $e_{q,1,2} = 0$ and $e_{q,3,4} = 0$, and the remaining weights are set to one. Note that if all weights were set to one, we would obtain standard SVMRank.

4 Experiments

The aim of the experimental section is to investigate the different machine learning approaches, namely indirect, direct, and semi-direct methods, for the knapsack problem. Even for a *"simple"* problem, such as the knapsack, there are multiple approaches to the solution process that warrant being investigated. We note that the benchmarks and code are available online: `https://github.com/vub-dl/predopt_knapsack`.

Benchmarks and data. We perform experiments with artificially generated datasets and real-life energy price data.

The *artificial datasets* are constructed such that the profits can not be easily learned. Each item is represented by a 2-dimensional attribute vector (i, j), where $i, j \in [0, 360]$. The profit is set to $profit((i, j)) = 10^3 * sin(i) * sin(j)$. This constitutes the initial set of items, which is filtered as follows: to obtain a bijection, if multiple pairs (i, j) map to the same value, we only keep one such pair, e.g. (pi, j) and $(0, j)$ both lead to zero profit and thus only one of these pairs is kept. To ensure each profit value is positive, we add a positive constant to each profit. A weight $w \in \{3, 5, 7\}$ is assigned to each pair (i, j) and its profit is multiplied accordingly, hence preserving the profit-weight ratio. Knapsack instances are generated using these items, such that in each instance there are 16 items of each weight (total of 48 items), and special measures are taken to ensure that the distribution of item ratios is similar for each benchmark. To increase the difficulty of learning, we draw a random integer from $[1, 5]$ for each benchmark and multiply its profits.

The *real-life datasets* contain two years of historical energy price data from the day-ahead market of SEM-O, the Irish Single Electricity Market operator. The data was used in the ICON energy-aware scheduling competition and a number of publications (e.g. see [10, 6]). For every half hour, the data consists of seven calendar features: whether it was a holiday, the day of the week, the week of the year, day, month, year and the half-hour-of-the-day. In addition, it includes three weather features, namely, the estimates of the temperature, windspeed, and CO2 intensity in the Irish city of Cork, and three key energy-related day-ahead forecasts by SEM-O itself: the forecasted wind production, the

forecasted system load, and the forecasted energy price. The goal is to predict the real energy price, as determined post-hoc two days later. The task is hence not to predict energy prices from scratch, but rather learn to use the SEM-O predictions together with weather and calendar information to improve on their predictions. As is common in energy price predictions, it is difficult to derive accurate estimates and due to price swings there is a large variance in the prices and prediction errors. In our benchmarks, we consider that one physical day, consisting of 48 half-hour slots, is the range of one problem instance, i.e. it is used in a day-ahead planning setting and each benchmark contains 48 items.

The data is used in a weighted and unit-weighted setting. The weights are generated as in the artificial data and all benchmark contain 48 items.

Methodology and implementation. The data is divided into training and test sets at a 70% - 30% ratio. On the training data, we perform for each learning method a 5-fold cross-validation grid search over a small range of hyperparameters with regret as a measure to do the selection. We discuss the results of both datasets at 10%, 30% and 50% capacity of the sum of weights. The results are presented in tables, where an entry (x, y) represents the average regret for the training (x) and testing set (y), respectively.

Regarding the implementation, we use *scikit-learn* [19] and *torch* Python libraries, Gurobi as the quadratic linear program solver, and the dedicated knapsack solver of or-tools (`http://developers.google.com/optimization/`).

We investigate solution-quality rather than runtime. Direct methods are always more computationally expensive given their use of optimisation, and implementations have not been optimised in terms of execution time in any case.

Learning methods. We experiment with the methods detailed in Section 3: **(indirect)** *kNN*, k-nearest neighbours regression; *kNN-mo*, k-nearest neighbours multi-output regression; *Ridge*, ridge regression which has shown good accuracy on the energy dataset in the past; *Ridge-mo*, multi-output ridge regression; *SVM-Rank*, SVMrank; **(direct)** *QPTL*, quadratic programming task loss [24]; *SPO*, smart predict then optimise [8]; **(semi-direct)** *Ridge-p*, ridge regression on profitability v_i/w_i rather than profit v_i *QPTL-s*, our quadratic programming task loss simplification as described in Section 3.3; and *SVMRank-s*, our weighted SVMrank modification as described in Section 3.3.

Results. We discuss the results of both datasets in the unit-weighted and weighted case at 10%, 30% and 50% capacity of the total weight to vary the combinatorial component in the problem instances. The results are presented in tables, where an entry (x, y) represents the average regret for the training (x) and testing set (y), respectively. We omit the training results of kNN as it simply stores all training samples in its knowledge base.

Artificial data, unit-weighted, Table 1 (top). Among the indirect methods, the multi-output regression variants are outperformed by the individual learning settings, which is an indication that learning from a joint feature representation (for all items at once) is more difficult then when learning for each item specifically and that the ability to account for correlation in the output does not compensate that. The data is generated to be difficult for pointwise methods, and indeed for

	Indirect					Direct		Semi-direct	
Capacity	kNN	kNN-mo	ridge	ridge-mo	SVMR	SPO	QPTL	QPTL-s	SVMR-s
5 (10%)	(⊥; 1.6)	(⊥; 27.0)	(12.0; 12.0)	(22.0; 28.0)	(1.12; 0.99)	(20.49; 20.36)	(23.01; 19.17)	(24.22; 23.68)	(0.08; 0.41)
15 (30%)	(⊥; 1.5)	(⊥; 66.0)	(11.0; 12.0)	(61.0; 75.0)	(2.29; 2.54)	(30.31; 29.75)	(33.66; 27.62)	(35.34; 34.76)	(0.5; 0.27)
25 (50%)	(⊥; 0.7)	(⊥; 76.0)	(1.1; 0.9)	(52.0; 66.0)	(4.71; 6.07)	(0.38; 0.13)	(0.09; 0.05)	(0.1; 0.0)	(0.04; 0.03)

	Indirect					Direct		Semi-direct		
Capacity	kNN	kNN-mo	ridge	ridge-mo	SVMR	SPO	QPTL	QPTL-s	ridge-p	SVMR-s
25 (10%)	(⊥; 26)	(⊥; 119)	(62; 63)	(93; 127)	(4; 1)	(72; 67)	(62; 73)	(72; 61)	(26; 29)	(2; 0)
75 (30%)	(⊥; 64)	(⊥; 287)	(104; 102)	(218; 282)	(9; 8)	(238; 214)	(206; 235)	(240; 203)	(24; 27)	(1; 1)
125 (50%)	(⊥; 25)	(⊥; 324)	(42; 43)	(259; 347)	(23; 23)	(166; 151)	(274; 318)	(322; 270)	(8; 8)	(6; 8)

Table 1. Regret values for the artificial dataset: unit-weighted (top) and (bottom) weighted. (training, test) values in thousands.

	Indirect					Direct		Semi-direct	
Capacity	kNN	kNN-mo	ridge	ridge-mo	SVMR	SPO	QPTL	QPTL-s	SVMR-s
5 (10%)	(⊥; 88)	(⊥; 99)	(43; 51)	(67; 96)	(39; 44)	(40; 55)	(50; 64)	(50; 64)	(41; 42)
15 (30%)	(⊥; 108)	(⊥; 112)	(59; 67)	(80; 117)	(55; 53)	(59; 72)	(76; 105)	(76; 105)	(55; 51)
25 (50%)	(⊥; 68)	(⊥; 83)	(42; 48)	(51; 89)	(38; 41)	(36; 45)	(49; 70)	(49; 70)	(39; 45)

	Indirect					Direct		Semi-direct		
Capacity	kNN	kNN-mo	ridge	ridge-mo	SVMR	SPO	QPTL	QPTL-s	ridge-p	SVMR-s
25 (10%)	(⊥; 78)	(⊥; 91)	(85; 72)	(86; 88)	(97; 97)	(126; 106)	(177; 142)	(177; 142)	(84; 97)	(88; 89)
75 (30%)	(⊥; 89)	(⊥; 95)	(60; 73)	(76; 97)	(145; 155)	(259; 223)	(260; 237)	(261; 236)	(144; 144)	(143; 155)
125 (50%)	(⊥; 80)	(⊥; 82)	(47; 60)	(56; 84)	(117; 121)	(136; 113)	(236; 191)	(236; 191)	(112; 105)	(124; 126)

Table 2. Regret values for the energy-pricing dataset: unit-weighted (top) and (bottom) weighted.

small and medium capacities, Ridge regression performs poorly while SVMRank does well. The direct and semi-direct methods capture the case of 50% capacity all really well. For tighter capacities, the direct methods do not seem to capture the essential parts of the predictions better. The specialised SVMRank-s on the other hand clearly improves on vanilla SVMRank and ridge.

Artificial data, weighted, Table 1 (bottom). In the weighted case, the weights add more combinatorial effects in the optimisation as well as varying the importance of the prediction errors. Among the indirect methods the rank-learning SVMRank is the clear winner. The semi-direct SVMRank-s modification again improves it and has the best results. The direct methods perform well at 10% capacity but much less at higher capacities where more items are involved. The semi-direct method of learning profitability rather than profit shows very strong results, demonstrating the benefit of including limited information of the optimisation problem.

Energy data, unit-weighted, Table 2 (top). The data is much more noisy in this case which can be seen at the worse results of kNN versus ridge regression. The SVMRank method that learns to rank pairs of items rather than individual values performs better than ridge regression. The direct methods are more effective on this data, especially SPO. The semi-direct SVMRank-s modification is again improving over SVMRank leading to the best results overall. Ranking methods are clearly the best for our unit-weighted sets.

Energy data, weighted, Table 2 (bottom). In contrast to the unit-weighted case, learning the profits with Ridge regression leads to the lowest expected regret. Ranking is less advantagous it seems, as an optimal solution cannot be computed merely by ordering items based on their profit ratios as in the unit-weighted case. In contrast to the artificial data, learning the profitability rather than profit does not guarantee better results either and somewhat disappointingly the direct methods do not gain much by solving the optimisation problem and learning from that at each gradient step.

Note that in some cases there is a difference in regret for test and training. This can be explained by the difficulty of our setting: the machine learning function that minimises regret is highly nonlinear and nondifferentiable, as it depends on a combinatorial optimisation problem.

Discussion Our investigation demonstrate that we have a long way to go to know how best to predict+optimise. Direct methods may be very promising, but non-gradient-descent methods like ridge regression may provide more robust predictions in general. Surprisingly, QPTL-s, which omits the QP part, is even on par or better than QPTL again demonstrating the difficulty of deriving gradients over a combinatorial optimisation problem. The only case when the reverse holds were in simple versions of the knapsack which were close to convex problems. The alternative approach, semi-direct, of including some knowledge of the optimisation problem without having to solve it repeatedly has shown its potential, but this needs to be done for each optimisation problem specifically. The ranking based methods, which have not been considered in prediction+optimisation works so far, perform well here since they do not learn on items individually, but over the relation between pairs of items. But when we examine the most difficult class of instances: weighted energy; it appears that existing indirect methods have the capability to more accurately learn the profits and override the methods which attempt to take into account the actual regret as loss function.

5 Conclusion

We study the prediction+optimisation knapsack problem.We provided insight into the relation to stochastic optimisation and the suitability of different learning methods.We compare indirect methods, standard learning approaches, versus direct methods which combine learning with the optimisation problem, and introduce semi-direct methods which combine learning with the optimisation problem while avoiding solving the optimisation problem using a form of surrogate for regret. We show that direct methods can outpeform alternative indirect methods, however their utility seems limited to cases when the optimisation problem is near convex. Hence, the best approach for prediction+optimisation problems is still an open question. Some of the challenges include further exploiting learning to rank methods, improving direct methods and to explore more automatic ways to create semi-direct approaches to a new optimisation problem.

References

1. Bessiere, C., Koriche, F., Lazaar, N., O'Sullivan, B.: Constraint acquisition. Artificial Intelligence **244**, 315 – 342 (2017). https://doi.org/https://doi.org/10.1016/j.artint.2015.08.001, http://www.sciencedirect.com/science/article/pii/S0004370215001162, combining Constraint Solving with Mining and Learning
2. Borchani, H., Varando, G., Bielza, C., Larrañaga, P.: A survey on multi-output regression. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **5**(5), 216–233 (2015)
3. Cover, T., Hart, P.: Nearest neighbor pattern classification. IEEE Transactions on Information Theory **13**(1), 21–27 (1967)
4. Di Liberto, G., Kadioglu, S., Leo, K., Malitsky, Y.: Dash: Dynamic approach for switching heuristics. European Journal of Operational Research **248**(3), 943–953 (2016)
5. Donti, P.L., Amos, B., Kolter, J.Z.: Task-based end-to-end model learning in stochastic optimization. In: Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017). pp. 5484–5494 (2017)
6. Dooren, D.V.D., Sys, T., Toffolo, T.A.M., Wauters, T., Berghe, G.V.: Multi-machine energy-aware scheduling. EURO J. Computational Optimization **5**(1-2), 285–307 (2017). https://doi.org/10.1007/s13675-016-0072-0, https://doi.org/10.1007/s13675-016-0072-0
7. Dragone, P., Teso, S., Passerini, A.: Pyconstruct: Constraint programming meets structured prediction. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18. pp. 5823–5825. International Joint Conferences on Artificial Intelligence Organization (7 2018)
8. Elmachtoub, A.N., Grigas, P.: Smart "predict, then optimize". Tech. rep. (2017), https://arxiv.org/pdf/1710.08005.pdf
9. Gilmore, P., Gomory, R.E.: The theory and computation of knapsack functions. Operations Research **14**(6), 1045–1074 (1966)
10. Grimes, D., Ifrim, G., O'Sullivan, B., Simonis, H.: Analyzing the impact of electricity price forecasting on energy cost-aware scheduling. Sustainable Computing: Informatics and Systems **4**(4), 276–291 (2014). https://doi.org/http://dx.doi.org/10.1016/j.suscom.2014.08.009, http://www.sciencedirect.com/science/article/pii/S221053791400050X, special Issue on Energy Aware Resource Management and Scheduling (EARMS)
11. Joachims, T.: Making large-scale svm learning practical. Tech. rep., Technical Report, SFB 475: Komplexitätsreduktion in Multivariaten Datenstrukturen, Universität Dortmund (1998)
12. Joachims, T.: Optimizing search engines using clickthrough data. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 133–142. KDD '02, ACM, New York, NY, USA (2002). https://doi.org/10.1145/775047.775067, http://doi.acm.org/10.1145/775047.775067
13. Joachims, T.: Training linear svms in linear time. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 217–226. ACM (2006)
14. Kotthoff, L.: Algorithm selection for combinatorial search problems: A survey. AI Magazine **35**(3), 48–60 (2014), http://www.aaai.org/ojs/index.php/aimagazine/article/view/2460

15. Liu, T.Y., et al.: Learning to rank for information retrieval. Foundations and Trends® in Information Retrieval **3**(3), 225–331 (2009)
16. Mathaba, T., Xia, X., Zhang, J.: Analysing the economic benefit of electricity price forecast in industrial load scheduling. Electric Power Systems Research **116**, 158–165 (2014). https://doi.org/http://doi.org/10.1016/j.epsr.2014.05.008, http://www.sciencedirect.com/science/article/pii/S0378779614001886
17. Matthews, G.: On the partition of numbers. Proceedings of the London Mathematical Society **28**, 486–490 (1897)
18. Passerini, A., Tack, G., Guns, T.: Introduction to the special issue on combining constraint solving with mining and learning. Artif. Intell. **244**, 1–5 (2017). https://doi.org/10.1016/j.artint.2017.01.002, https://doi.org/10.1016/j.artint.2017.01.002
19. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
20. Picard-Cantin, É., Bouchard, M., Quimper, C., Sweeney, J.: Learning the parameters of global constraints using branch-and-bound. In: Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings. pp. 512–528 (2017)
21. Spall, J.: Introduction to Stochastic Search and Optimizatio. Wiley (2003)
22. Teso, S., Passerini, A., Viappiani, P.: Constructive preference elicitation by setwise max-margin learning. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016. pp. 2067–2073 (2016)
23. Vapnik, V.: Principles of risk minimization for learning theory. In: Advances in neural information processing systems. pp. 831–838 (1992)
24. Wilder, B., Dilkina, B., Tambe, M.: Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In: Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19). p. to appear (2019), https://arxiv.org/pdf/1809.05504.pdf