OJMO

Open Journal of Mathematical Optimization

Axel Parmentier

Learning structured approximations of combinatorial optimization problems

Volume 6 (2025), article no. 10 (27 pages)

https://doi.org/10.5802/ojmo.43

Article submitted on January 27, 2023, revised on March 26, 2024,
accepted on June 13, 2025.

CENTRE MERSENNE

Open Journal of Mathematical Optimization is a member of the
Centre Mersenne for Open Scientific Publishing
http://www.centre-mersenne.org/
e-ISSN: 2777-5860

Learning structured approximations of combinatorial optimization problems

Axel Parmentier
CERMICS, ENPC, Institut Polytechnique de Paris, Marne-la-Vallée, France
axel.parmentier@enpc.fr

Abstract

Neural networks that include a combinatorial optimization layer can give surprisingly efficient heuristic policies for difficult combinatorial optimization problems. Three questions remain open: which architecture should be used, how should the parameters of the machine learning model be learned, and what performance guarantees can we expect from the resulting algorithms? Following the intuitions of geometric deep learning, we explain why equivariant layers should be used when designing such policies, and illustrate how to build such layers on routing, scheduling, and network design applications. We introduce a learning approach that enables to learn such policies when the training set contains only instances of the difficult optimization problem and not their optimal solutions, and show its numerical performance on our three applications. Finally, using tools from statistical learning theory, we prove a theorem showing the convergence speed of the estimator. As a corollary, we obtain that, if an approximation algorithm can be encoded by the neural network for some parametrization, then the learned policy will retain the approximation ratio guarantee. On our network design problem, our machine learning policy has the approximation ratio guarantee of the best approximation algorithm known and the numerical efficiency of the best heuristic.

Digital Object Identifier 10.5802/ojmo.43

Keywords Decision Focused Learning, Combinatorial Optimization Augmented Machine Learning, Combinatorial Optimization Layer, Generalization Bound, Perturbed Empirical Risk Minimization, Stochastic Vehicle Scheduling, Single Machine Scheduling.

1 Introduction

Optimizing industrial processes is one of the main focus of operations research. Such processes often involve allocating non-secable resources such as machines, vehicles, or workers to tasks. Modeling their optimization therefore naturally leads to combinatorial optimization problems

$$\min_{z \in \mathcal{Z}} C(z), \tag{1}$$

where $\mathcal{Z}$ is a finite but combinatorially large set, and C is a cost function. Consider for instance a company that must operate a set of task using a set of vehicles. The set of feasible solutions $\mathcal{Z}$ is the set of partitions of the set of tasks into routes operated by the different vehicles.

Historically, two assumptions have been made when designing algorithms to solve such problems. First, algorithms should *work well on any instance of the problem*. This might seem surprising, as it is well known that industrial instances have a specific structure: Firms often solve every day pretty similar instances of the same problem. But characterizing this structure is difficult. Defining an algorithm that can handle any instance therefore seems a safe approach. On the theoretical side, this perspective is embodied by complexity theory, which mainly focuses on worst-case complexity. The second assumption is that *algorithms should be scalable to very large instances, even if it necessitates simplifying the problem*. The rationale behind this is the substantial benefits derived from economies of scale in industrial optimization. For example, in vehicle routing problems, managing a larger fleet allows for greater flexibility in task allocation, resulting in reduced travel distances. To exploit these economies of scale, it's essential for algorithms to process large problem instances within a feasible timeframe. This assumption made even more sense a few decades ago, when data on industrial processes was scarce and modelling complex phenomena difficult. In the context of vehicle routing, the primary objectives would typically include minimizing the number of vehicles and fuel consumption, while more intricate goals, such as curtailing delay propagation, would be less prioritized.

Given these assumptions, the operations research community has been developing scalable algorithms for combinatorial optimization problems, particularly those with simpler objectives such as linear ones. And it has been widely successful in this endeavor, as illustrated by the progress of Mixed Integer Linear Programming solvers, which have seen a 540 billion times speedup between 1991 and 2015 [5], and problems with millions of variables and constraints are routinely solved in the industry. Contrary to a few decades ago, practitioners now have access to efficient combinatorial optimization algorithms for a wide range of operations research applications, provided that they are willing to keep the objective simple.

Besides, while data on industrial processes was scarce, it is now abundant. And operations research departments across the industry are now looking for ways to leverage this data to improve their processes, aiming for better performance and resilience. For instance, on our vehicle scheduling problem, one could use data on the traffic to improve the estimation of delays and build routes that are more resilient to traffic variations. Such goals typically challenge the "simple objective" paradigm, as they often require a stochastic optimization approach. Unfortunately, algorithms for stochastic combinatorial optimization problems still scale poorly on most applications.

However, the abundance of data and the progress of machine learning algorithms suggest another perspective. Indeed, machine learning now provides a natural way to capture the structure of industrial problems instances. In order to make it clear, it is more convenient to consider the instance in the definition of the combinatorial optimization problem.

$$\min_{z \in \mathcal{Z}(x)} C(z, x). \tag{Pb}$$

Here x is an instance in an *instance set* $\mathcal{X}$, and $\mathcal{Z}(x)$ denotes the set of feasible solutions of x. The objective function is typically difficult to optimize. Taking a machine learning perspective, we assume that instances come from an unknown distribution μ on $\mathcal{X}$, and that we have access to a training set $(x_1, \dots, x_n)$ of historical instances sampled from this distribution.

The classic operations research approach consist in solving each instance of the problem (Pb) online, i.e., when it is encountered in practice, without taking into account distributional information. We suggest considering instead a family $\mathcal{H}$ of *policies* $h : x \in \mathcal{X} \mapsto z \in \mathcal{Z}(x)$ from instances to solutions, and to *learn* offline the mapping that minimizes the *risk*

$$\min_{h \in \mathcal{H}} \mathbb{E}_{x \sim \mu} C\big(h(x), x\big). \tag{2}$$

By learning offline, we mean that we use the training set $(x_1, \dots, x_n)$ to find a mapping h with low risk. When a new instance x is encountered, we use the learned mapping h to obtain a solution $z = h(x)$, and return it as a solution of the problem (Pb).

This raises several questions. On the practical side, how should we design the family $\mathcal{H}$ of solution mappings? And how should we learn the mapping h? And on the theoretical side, what performance guarantees can we expect for the resulting algorithm?

Neural network with combinatorial optimization layers

In this paper, we build a parametrized family of policies $\mathcal{H} = (h_{\boldsymbol{w}})_{\boldsymbol{w} \in \mathcal{W}}$ with $\mathcal{W} \subset \mathbb{R}^d$ as follows. Since Problem (Pb) is too difficult to be solved online, we replace it by a simpler surrogate optimization problem. To that purpose, we introduce a family of surrogate optimization problems parametrized by a parameter vector $\boldsymbol{\theta}$.

$$\min_{y \in \mathcal{Y}(x)} g(y, x, \boldsymbol{\theta}). \tag{CO-layer}$$

The set of feasible solution $\mathcal{Y}(x)$ is typically equal to $\mathcal{Y}(x)$, or such that there is a natural way to map a solution $y \in \mathcal{Y}(x)$ to a solution $z \in \mathcal{Z}(x)$. We assume that we have an algorithm $\mathcal{A}$ called the *oracle* that is fast enough to solve (CO-layer) for large x and any $\boldsymbol{\theta}$ online. Depending on the application, $\mathcal{A}$ can take from fractions of a second to a few minutes. Our policies just need to recast an instance of (Pb) into an instance of (CO-layer), solve the latter using $\mathcal{A}$, and then decode the solution y of this problem into a solution $z \in \mathcal{Z}(x)$ of the initial problem. Practically, we use the kind of policies $h_{\boldsymbol{w}} = \psi \circ \mathcal{A} \circ \varphi_{\boldsymbol{w}}$ illustrated on Figure 1. First, a *statistical model* $\varphi_{\boldsymbol{w}}$ parametrized by $\boldsymbol{w}$ maps the instance x to a parameter $\boldsymbol{\theta} = \varphi_{\boldsymbol{w}}(x)$. This model is typically a neural network or a generalized linear model. We then solve (CO-layer) with $\mathcal{A}$ to obtain a solution y. Since our combinatorial optimization oracle $\mathcal{A}$ is just a layer in $h_{\boldsymbol{w}}$ when $\varphi_{\boldsymbol{w}}$ is a neural network, we call it a *combinatorial optimization layer*. We finally decode the solution y of this problem into a solution z of the initial problem using a deterministic

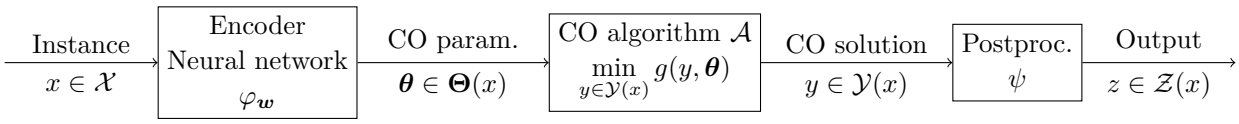


Figure 1 Policy encoded as a neural network with a combinatorial optimization (CO) layer

algorithm ψ, which we call the *decoder*. On our applications, either there is no decoder, i.e., $\psi = \text{Identity}$, or ψ is a local descent heuristic.

Since the postprocessing ψ is assumed deterministic, it does not play a major role on the learning algorithm. For convenience, we bring back the cost on $\mathcal{Y}(x)$. For y in $\mathcal{Y}(x)$, we define

$$c(y, x) = C(\psi(y), x) \quad \text{and} \quad h_{\boldsymbol{w}} = \mathcal{A} \circ \varphi_{\boldsymbol{w}}$$

Examples of relevant settings

Let us now give a few examples of relevant settings for this approach. As we mentioned at the beginning of this introduction, the goal of these methods is to optimize richer combinatorial optimization problems in a data abundant context. For instance, the objective function $C(z, x)$ can be a simulator calibrated on data, which makes it difficult to optimize. This leads to a *combinatorial black-box optimization* problem on a combinatorial set. Replacing the simulator by a tractable surrogate objective is then natural. A variant of this setting is the one where $C(z, x)$ is known and can be optimized, but algorithms do not scale well. It can then be tempting to obtain a *fast heuristic* by replacing the objective function by a simpler one. We detail an application to $1|r_j|\sum_j C_j$, a well known strongly NP-hard single machine scheduling problem in Section 3.1. Since a scheduling can be encoded as a permutation, it is natural to take a sorting problem as CO-layer. We just predict scores θ_j for each job, and then sort them by increasing scores. The statistical model $\varphi_{\boldsymbol{w}}$ takes in input the instance and returns the scores (θ_j), the oracle $\mathcal{A}$ sorts them, and the decoder ψ returns the corresponding permutation, possibly after a local descent heuristic. Parmentier et al. [22] show that this approach gives the best heuristic for this problem on large instances, and we improve upon their results.

Even if there is no simulator, data driven problems are often *stochastic*, with $c(y, x) = \mathbb{E}_\xi[\widetilde{c}(y, x, \xi)]$, where ξ is a random variable. Combinatorial stochastic optimization problems are typically difficult to solve for large instances, and it is therefore tempting to replace them by deterministic surrogate optimization problems, for instance linear ones $\max_{y \in \mathcal{Y}(x)} \boldsymbol{\theta}^\top y$. We consider in this paper the case of a stochastic vehicle scheduling problem, where the goal is to minimize the expected cost of delay. The statistical model $\varphi_{\boldsymbol{w}}$ takes in input the data available on delay and returns for each arc a a parameter θ_a that penalizes delay propagation on this arc – we will see that θ_a it is not the expected delay propagation.

Two stage stochastic optimization model strategic problems where the decision maker must take a decision y before knowing the realization of a random variable ξ. A second decision z_s is taken after the realization of ξ, knowing the scenario s. The two stage aspect makes these problems challenging to solve. We suggest using as surrogate a problem with a single scenario. $\varphi_{\boldsymbol{w}}$ computes the objective θ of the single scenario approximation. Then $\mathcal{A}$ computes the solution y. The decoder ψ then keeps the first stage solution of the approximation rebuilds the second stage solutions for each scenario of the initial problem, and returns $z = (y, (z_s)_s)$. We illustrate this on the minimum weight two stage spanning tree problem.

Learning problem

The goal of the learning problem is to find the parameter $\boldsymbol{w} \in \mathcal{W}$ that leads to the best policy, i.e., that minimizes the *risk*

$$\min_{\boldsymbol{w} \in \mathcal{W}} R(\boldsymbol{w}) \quad \text{where} \quad R(\boldsymbol{w}) = \mathbb{E}_x\big[c(h_{\boldsymbol{w}}(x), x)\big]. \tag{3}$$

Practically, the distribution over instances is unknown. Contributions of the literature use supervised learning approaches, which require a training set $(x_1, z_1), \ldots, (x_n, z_n)$ containing instances of (Pb) and their hard problem solution. A drawback of such an approach is that it requires another solution algorithm for (Pb) to compute the z_i. In this paper, we suppose having only access to a set of instances $x_1, \ldots, x_n$ drawn from the distribution,

and minimize the *empirical risk*

$$\min_{\boldsymbol{w}\in\mathcal{W}} \widehat{R}_n(\boldsymbol{w}) \quad \text{where} \quad \widehat{R}_n(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} c(h_{\boldsymbol{w}}(x_i), x_i). \tag{4}$$

Contributions

In recent years, neural network-based policies with combinatorial optimization layers have achieved notable successes. Concurrently, efficient and versatile supervised learning techniques for their training have been developed. However, the literature is scarce on applications in operations research, the development of effective architectures, and the expected performance guarantees. The goal of this paper is to outline strategies for crafting efficient policies, and present learning algorithms that not only yield practical results but also provide performance guarantees with these architectures. To achieve these guarantees, we transition from a traditional supervised learning model to a risk minimization framework – an approach that, to the best of our knowledge, has not been previously explored in the context of neural networks with combinatorial optimization layers within the domain of operations research. More precisely, we make the following contributions.

1. Choosing a family of models $\mathcal{H}$ with an appropriate architecture, and notably a relevant layer (CO-layer) and a relevant statistical model $\varphi_{\boldsymbol{w}}$ is critical for the performance of the resulting algorithm. We illustrate on the three applications previously mentioned how to choose them, and give practical recipes to choose the right architecture.
2. We show that $\widehat{R}_n(\boldsymbol{w})$ is piecewise constant, which makes it hard to optimize and negatively impacts generalization. That is, $\widehat{R}_n(\boldsymbol{w})$ being small does not guarantee that $R(\boldsymbol{w})$ is small. We therefore use as learning problem a perturbed version of the empirical risk minimization that makes both optimization and generalization easier.
3. We show with extensive numerical experiments on these applications that, despite the non-convexity of the resulting perturbed loss, when the dimension of $\mathcal{W}$ is moderate, i.e., non-greater than 100, solving this problem with a global black-box solver leads to surprisingly efficient policies.
4. Leveraging tools from statistical learning theory, we prove that when the size n of the training set goes to infinity, the risk $R(\boldsymbol{w})$ of the $\boldsymbol{w}$ learned by our algorithm converges to the optimal risk R^*, and provide an upper bound on the convergence speed.
5. We deduce from these statistical learning results that, under some hypotheses on $\mathcal{H}$, the learned algorithm is an approximation algorithm for (Pb). These hypotheses are notably satisfied by our policy for the maximum weight two stage spanning tree problem.

In this paper, we do not try to approximate difficult constraints. We only try to approximate difficult objectives. The paper is organized as follows. Section 2 reviews the literature, while Section 3 details our examples and explain how to build policies. Section 4 formulates the risk minimization problem and introduces algorithms. Section 5 introduces the convergence results and the approximation ratio guarantee. Finally, Section 6 details the numerical experiments.

2 Related works

The interactions between combinatorial optimization and machine learning is an active research area [3]. Combinatorial optimization layers in deep learning belong to the subarea of end-to-end learning methods for combinatorial optimization problems recently surveyed by Kotary et al. [16]. This field can be broadly classified into two subfields. Machine learning augmented combinatorial optimization uses machine learning to take heuristic decisions within combinatorial optimization algorithms. We survey here combinatorial optimization augmented machine learning, which inserts combinatorial optimization oracles within machine learning based policies.

Combinatorial optimization augmented machine learning

Structured learning approaches were the first to introduce these methods in the early 2000s [20] in the machine learning community. They mainly considered maximum a posteriori problems in probabilistic graphical models as combinatorial optimization layers, with applications to computer vision, and sorting algorithms with applications to ranking. They were generally trained using the structured Hinge loss or a maximum likelihood estimator. A

renewed interest for optimization layers in the neural networks has emerged in the last few years in the machine learning community, and notably in continuous optimization layers [1, 6].

We focus here on combinatorial optimization layers. Among these, linear optimization layers have received the most attention. Two challenges must be addressed. First, since the mapping that associated to the objective parameter vector $\boldsymbol{\theta}$ the output y is piecewise constant, and since deep learning networks are generally trained using stochastic gradient descent, meaningful approximations of the gradient must be proposed [24]. Second a loss quantifying the error must be proposed. Blondel et al. [7] address these challenges with an elegant solution based on convex duality: the linear objective is regularized with a convex penalization, which leads to meaningful gradients. Fenchel Young inequality in convex duality then gives a natural definition of the loss function. Berthet et al. [4] have shown that this approach can be extended to the case where a random perturbation is added to the objective instead of a convex regularization. When it comes to integer linear programs, Mandi et al. [17] suggest using the linear relaxation during the learning phase. The author recently introduced the idea of building heuristics for hard combinatorial optimization problems with neural networks with combinatorial optimization layers [21].

Bengio et al. [3] distinguish between two approaches for the learning problems: learning by imitation and learning by experience, which correspond to the supervised learning and risk/regret minimization paradigms in the machine learning literature. We stick to the machine learning terminology in this paper. All the approaches mentioned above use a supervised learning approach. The closest contribution to our risk minimization setting is the smart predict then optimize method of Elmachtoub et al. [12]. It considers the case where there is no decoder and the cost function $c(y, x) = \boldsymbol{\theta}^{*\top} y$ is actually the linear objective of the combinatorial optimization layer $g(y, \boldsymbol{\theta}) = \boldsymbol{\theta}^{\top} y$ for an unknown true parameter $\boldsymbol{\theta}^*$. They propose a generalization of the structured Hinge loss [20, Chapter 6] to that setting.

However, to the best of our knowledge, two aspects of neural networks with combinatorial optimization layers have not been considered in the literature. First, the general risk minimization setting where only instances of (Pb) are available has not been considered. Second, there is no guarantee on the quality of the solution returned by the policy.

Statistical learning bounds for combinatorial optimization

Our convergence theorems rely on statistical learning theory [8]. While there has been some work on statistical learning bounds for structured learning [19], these bounds do not apply to the kinds of policies we consider in this paper, and turning their predictors into tractable policies in our setting is non-trivial. To the best of our knowledge, the only bounds that could apply to our non-perturbed policies settings are those developed by Balcan et al. [2] in the context of data-driven algorithm design. However, they would require additional assumption on c, g, and φ_w. Beyond making the learning problem easier and opening the door to more efficient policies, perturbations enable using statistical learning theory tools based on smoothness that do not require such assumptions.

3 Designing neural networks with combinatorial optimization layers

This section explains how to build neural networks with combinatorial optimization layers. We start by stating the three problems mentioned in the introduction, and describing the policies used for each of these. The first two ones were previously introduced by the author, and their description follows the papers that introduced them [21, 22]. Using these policies as examples, we then give practical recipes to build such policies. The third application is treated more in depth in the recipes, and readers can safely skip Sections 3.1 and 3.2 if they want to read on a single application.

3.1 Single machine scheduling problem.

Problem

m jobs must be processed on a single machine. Jobs cannot be interrupted once launched. Each job has a processing time p_j, and a release time r_j in $\mathbb{R}$. That is, job j cannot be started before r_j, and once started, it takes p_j to complete it. A solution is a permutation $z = (j_1, \dots, j_m)$ of $[m]$ that gives the order in which jobs are processed, where $[m] = \{1, \dots, m\}$. Using the convention $C_{j_0} = 0$, the completion time of jobs are defined as

$$C_{j_i} = \max(r_j, C_{j_{i-1}}) + p_{j_i}.$$

The objective is to find a solution minimizing $\sum_{j=1}^{m} C_j$. An instance x is therefore composed of a number of jobs m, processing times p_j, and release times r_j for each job $j \in [m]$. And $\mathcal{Z}(x)$ is the set of feasible schedules, i.e., the set of permutations of $[m]$. This problem is known as $1|r_j|\sum_j C_j$ in the scheduling literature and is strongly NP-hard [23, Chapter 3].

Combinatorial optimization layer

When we drop the release times r_j, we get an easier problem known as $1||\sum_j C_j$ in the literature. They therefore suggest using it as combinatorial optimization layer, and only need to predict jobs processing times θ_j. Jobs completion times are therefore given by

$$\overline{C}_{j_i} = \overline{C}_{j_{i-1}} + \theta_{j_i}.$$

Again, we use an overline to denote quantities of the easy problem. Note that $\theta_j \neq p_j$ in general. An optimal schedule, i.e., an optimal permutation y, is obtained using the shortest processing time first (SPT) rule, that is, by sorting the jobs by increasing p_j. The parameter space $\boldsymbol{\Theta}(x)$ is therefore $\mathbb{R}^m$, and $\mathcal{Y}(x)$ is equal to $\mathcal{Z}(x)$, the set of permutations of $[m]$.

Decoder

Since $\mathcal{Y}(x) = \mathcal{Z}(x)$, we could use the identity as decoder. In our numerical experiments, we use two alternative decoders: the identity or a local descent heuristic.

Encoder

We define the feature map $\boldsymbol{\phi} : (j, x) \mapsto \boldsymbol{\phi}(j, x) \in \mathbb{R}^d$ where d does not depend on x. The encoder $\varphi_{\boldsymbol{w}}$ then applies in parallel the same linear model $\boldsymbol{\phi}(j, x) \mapsto \boldsymbol{w}^\top \boldsymbol{\phi}(j, x)$ to each job.

$$\varphi_{\boldsymbol{w}}(x) = (\theta_j)_j \quad \text{where} \quad \theta_j = \boldsymbol{w}^\top \boldsymbol{\phi}(j, x).$$

Section 3.4 details how to build $\boldsymbol{\phi}$.

3.2 Stochastic vehicle scheduling problem.

Problem

Let V be a set of tasks that should be operated using vehicles. For each task v in V, we assume we have a scheduled start time t_v^{b} in $\mathbb{Z}_+$, and a scheduled end time t_v^{e} in $\mathbb{Z}_+$. We suppose $t_v^{\mathrm{e}} > t_v^{\mathrm{b}}$ for each task v in V. For each pair of tasks (u, v), the travel time to reach task v from task u is denoted by $t_{(u,v)}^{\mathrm{tr}}$. Task v can be operated after task u using the same vehicle if

$$t_v^{\mathrm{b}} \geq t_u^{\mathrm{e}} + t_{(u,v)}^{\mathrm{tr}}. \tag{5}$$

We introduce the digraph $D = (V, A)$ with vertex set $V = T \cup \{o, d\}$ where o and d are artificial origin and destination vertices. The arc set A contains the pair (u, v) in T^2 if v can be scheduled after task u, as well as the pairs (o, v) and (v, d) for all v in V. An o-d path P represents a sequence of tasks operated by a vehicle. A feasible solution is a partition of V into o-d paths. If we denote by c_P the cost of operating the sequence corresponding to the o-d path P, and by $\mathcal{P}_{od}$ the set of o-d paths, the problem can be modeled as follows.

$$\min_{z} \quad \sum_{P \in \mathcal{P}_{od}} c_P z_P, \tag{6a}$$

$$\text{s.t.} \quad \sum_{P \ni v} z_P = 1, \quad \forall\, v \in V \backslash \{o, d\}, \tag{6b}$$

$$z_P \in \{0, 1\}, \quad \forall\, P \in \mathcal{P}_{od}, \tag{6c}$$

Up to now, we have described a generic vehicle scheduling problem. Let us now define our stochastic vehicle scheduling problem by giving the definition of c_P. Let Ω be a set of scenarios. For each task v, we have a random start time ξ_v^{b} and a random end time ξ_v^{e}, and for each arc (u, v), we have a random travel time $\xi_{(u,v)}^{\mathrm{tr}}$. Hence,

$\xi^{\mathrm{b}}_v(\omega)$, $\xi^{\mathrm{e}}_v(\omega)$, and $\xi^{\mathrm{tr}}_{(u,v)}(\omega)$ are respectively the beginning time of v, end time of v, and travel time between u and v under scenario ω in Ω. We define $\xi^{\mathrm{e}}_o = 0$ and $\xi^{\mathrm{b}}_d = +\infty$.

Given an o-v path P, we define recursively the end-time τ_P of P as follows.

$$\tau_P = \begin{cases} 0, & \text{if } P \text{ is the empty path in } o, \\ \xi^{\mathrm{e}}_v + \max(\tau_Q + \xi^{\mathrm{tr}}_a - \xi^{\mathrm{b}}_v, 0), & \text{if } P = Q + a \text{ for some } o\text{-}u \text{ path } Q \text{ and arc } a = (u, v). \end{cases} \tag{7}$$

Equation (7) models the fact that a task can be operated by a vehicle only when the vehicle has finished the previous task: The vehicle finishes Q at τ_Q, and arrives in v at $\tau_Q + \xi^{\mathrm{tr}}_a$ with delay $\max(\tau_Q + \xi^{\mathrm{tr}}_a - \xi^{\mathrm{b}}_v, 0)$. The total delay Δ_P along a path P is therefore defined recursively by

$$\Delta_P = \begin{cases} 0, & \text{if } P \text{ is the empty path in } o, \\ \Delta_Q + \max(\tau_Q + \xi^{\mathrm{tr}}_a - \xi^{\mathrm{b}}_v, 0), & \text{if } P = Q + a \text{ for some path } Q \text{ and arc } a. \end{cases} \tag{8}$$

Finally, we define the cost of an o-d path P as

$$c_P = c^{\mathrm{veh}} + c^{\mathrm{del}} \mathbb{E}(\tau_P) \tag{9}$$

where c^{veh} in $\mathbb{Z}_+$ is the cost of a vehicle and c^{del} in $\mathbb{Z}_+$ is the cost of a unit delay. Practically, we use a finite set of scenarios Ω, and compute the expectation as the average on this set. An instance x is therefore composed of a set of tasks V, the corresponding digraph D, a set of scenarios Ω, and delays $\xi^{\mathrm{b}}_v(\omega)$, $\xi^{\mathrm{e}}_v(\omega)$, and $\xi^{\mathrm{tr}}_a(\omega)$ for each task v, arc a, and scenario $\omega \in \Omega$.

The practical difficulty of the solving (6) comes from the cost of delay. Indeed, while the problem is easy when c_P is a sum of arc costs along P (see next paragraph), this is clearly not the case here. The delay propagation equations (7) are non-linear, and the expectation $\mathbb{E}(\tau_P)$ therefore turns it into a challenging stochastic optimization problem.

Combinatorial optimization layer

The usual vehicle scheduling problem can also be formulated as (6), the difference being that now the path can be decomposed as the sum of the arcs cost

$$\bar{c}_P = \sum_{a \in P} \theta_a \quad \text{with} \quad \theta_a \in \mathbb{R}. \tag{10}$$

In Equation (10) and in the rest of the paper, we use an overline to denote quantities corresponding to the surrogate optimization problem (CO-layer). It can be reduced to the following flow problem and efficiently solved using flow algorithms or linear programming.

$$\begin{aligned} \min_{y} \ & \sum_{a \in A} \theta_a y_a \\ \text{s.t.} \ & \sum_{a \in \delta^-(v)} y_a = \sum_{a \in \delta^+(v)} y_a, \quad \forall\, v \in V \backslash \{o, d\} \\ & y_a \in \{0, 1\}, \quad \forall\, a \in A. \end{aligned}$$

Our parameter space $\boldsymbol{\Theta}(x)$ is therefore $\mathbb{R}^A$, and $\mathcal{Y}(x)$ is the set of feasible solutions of (11).

Decoder

The decoder ψ only needs to reconstruct the paths P such that $z_P = 1$ from the flow solution $(y_a)_{a \in A}$.

Encoder

The encoder $\varphi_{\boldsymbol{w}}$ takes in input an instance x, i.e., tasks and delay scenarios, and returns a vector of parameters $\boldsymbol{\theta}$ in $\mathbb{R}^{|A|}$. We therefore introduce a feature map $\boldsymbol{\phi} : (a, x) \mapsto \boldsymbol{\phi}(a, x) \in \mathbb{R}^d$ that takes in input an instance x and an arc a in A, and returns a vector in $\mathbb{R}^d$ with d independent of x. The encoder then applies in parallel the same linear model to each arc.

$$\varphi_{\boldsymbol{w}}(x) = (\theta_a)_{a \in A} \quad \text{where} \quad \theta_a = \boldsymbol{w}^\top \boldsymbol{\phi}(a, x).$$

3.3 Minimum weight two stage spanning tree problem.

Problem

Let $G = (V, E)$ be an undirected graph, and S be a finite set of scenarios. The objective is to build a spanning tree on G of minimum cost on a two stage horizon. Building edge e in the first stage costs $c_e \in \mathbb{E}$, while building it in the second stage under scenario s costs $d_{es} \in \mathbb{R}$. The decision maker does not know the scenario s when he chooses which first stage edges to build. Denoting $\mathcal{T}$ the set of spanning trees, we can formulate the problem as

$$\min_{E_1,(E_s)_s} \left\{ \sum_{e\in E_1} c_e + \frac{1}{|S|} \sum_{s\in S} \sum_{e\in E_s} d_{es} \colon E_1 \cap E_s = \emptyset \text{ and } (V, E_1 \cup E_s) \in \mathcal{T} \text{ for all } s \text{ in } S \right\}. \tag{12}$$

When we restrict ourselves to $c_e \leq 0$ and $d_{es} \leq 0$, we obtain the two stage maximum weight spanning tree. Escoffier et al. [13] show that this restriction is APX-complete, i.e., it belongs to the class of problem for which there exists a constant factor approximation algorithm, and it is as difficult as any problem in the class. They introduce a 2-approximation algorithm for the maximization problem, which translates into a 1/2-approximation algorithm for the minimization problem.

Combinatorial optimization layer

Note that an optimal solution of the single scenario version of the problem

$$\min_{E,E_2} \left\{ \sum_{e\in E_1} \bar{c}_e + \sum_{e\in E_2} \bar{d}_e \colon E_1 \cap E_2 = \emptyset \text{ and } (V, E_1 \cup E_2) \in \mathcal{T} \right\} \tag{13}$$

is a minimum weight spanning tree on G with edge weights $\min(\bar{c}_e, \bar{d}_e)$. It can therefore be easily solved using Kruskal's algorithm. We therefore use (13) as combinatorial optimization layer (CO-layer). Hence, we have $\boldsymbol{\theta} = (\bar{c}_e, \bar{d}_e)_{e\in E}$ and $\boldsymbol{\Theta}(x) = \mathbb{R}^{2E}$.

Decoder

Our decoder ψ rebuilds a solution z of (12) from a solution $y = (\overline{E}_1, \overline{E}_2)$ of (13). We keep $\overline{E}_1$ as the first stage solution. It then suffices to seek, for each scenario s, the best second stage solution given $\overline{E}_1$. That is, to compute a minimum weight spanning tree with edges costs d_s among those containing E_1. This problem is easy because, given a graph G, edge weights, and a forest F, Kruskal's algorithm can be adapted to find a minimum weight spanning tree on G among those containing F. We then compare this solution to the optimal solution $E_1 = \emptyset$ and return the best of the two as z.

Encoder

We define a feature map $\boldsymbol{\phi} : ((e, \texttt{stage}), x) \mapsto \boldsymbol{\phi}\big((e, \texttt{stage}), x\big)$ that takes in input an instance x and an edge e in E, and a `stage` in $\{1, 2\}$, and returns a feature vector in $\mathbb{R}^d$ with d independent of x. We then define

$$\boldsymbol{\theta} = (\bar{c}_e, \bar{d}_e)_{e\in E} \quad \text{where} \quad \begin{cases} \bar{c}_e = \boldsymbol{w}^\top \boldsymbol{\phi}\big((e, 1), x\big), \\ \bar{d}_e = \boldsymbol{w}^\top \boldsymbol{\phi}\big((e, 2), x\big). \end{cases}$$

3.4 Recipes for building efficient policies architectures

Let us now explain how to build our neural networks with combinatorial optimization layers.

Combinatorial Optimization layer and decoder

The choice of the combinatorial optimization layer and the decoder are rather application-dependent. Two practical aspects are important. First, we must have a practically efficient algorithm to solve (CO-layer). Second, it must be easy to turn solutions of (CO-layer) into solutions of (Pb). That is, either the solutions of (CO-layer) and (Pb) coincide, or we must have a practically efficient algorithm h that turns a solution of (CO-layer) into a solution of (Pb).

Table 1 Two stage spanning tree features of edge $e = (u, v)$.

Feature description	$\boldsymbol{\phi}\big((e,1),x\big)$	$\boldsymbol{\phi}\big((e,2),x\big)$
First stage cost	c_e	0
Second stage average cost	0	$\sum_s d_{es}/\lvert S\rvert$
Quantiles of second stage cost	0	$Q\big[(d_{es})_s\big]$
Quantiles of neighbors first stage cost	$Q\big[(c_{e'})_{e'\in\delta(u)\cup\delta(v)}\big]$	0
Quantiles of neighbors second stage cost	0	$Q\big[(d_{e's})_{e'\in\delta(u)\cup\delta(v),s\in S}\big]$
"Is edge in first stage MST ?"	$\mathbb{1}^{\mathrm{MST}}\big(e,(c_e)_{e\in E}\big)$	0
Quantiles of "Is edge in second stage MST quantile ?"	0	$Q\Big[\Big(\mathbb{1}^{\mathrm{MST}}\big(e,(b_{es})_{e\in E}\big)\Big)_{s\in S}\Big]$
Quantiles of "Is first stage edge in best stage MST quantile ?"	$Q\Big[\Big(\mathbb{1}^{\mathrm{MST}}\big(e,(b_{es})_{e\in E}\big)\Big)_{\substack{s\in S\\ \text{and } c_e\le d_{es}}}\Big]$	0
Quantiles of "Is second stage edge in best stage MST quantile ?"	0	$Q\Big[\Big(\mathbb{1}^{\mathrm{MST}}\big(e,(b_{es})_{e\in E}\big)\Big)_{\substack{s\in S\\ \text{and } c_e> d_{es}}}\Big]$

Note: MST stands for Minimum Weight Spanning Tree, $b_{es} = \min(c_e, d_{es})$, $\mathcal{Q}[\boldsymbol{a}]$ gives the quantiles of a vector $\boldsymbol{a}$ seen as a sampled distribution, and $\mathbb{1}^{\mathrm{MST}}(e,(\widetilde{c}_e)_e)$ is equal to 1 if e is in the minimum spanning tree for edge weights $(\widetilde{c}_e)_e$. Given a vertex v, $\delta(v)$ is the set of edges incident to v.

Structure of x and generalized linear model

When building a policy, we typically want it to be able to address instances of very different size. Let us illustrate this on our two stage minimum weight spanning tree problem. Instances may have 20 or 1000 edges. It means that the graph G used in the combinatorial optimization layer (13) depends on the instance x of (12), and hence the parameter $\boldsymbol{\theta}$ belongs to the set $\mathbb{R}^{2E}$ which also depends on x. This is the reason why, in our policy, the set of solutions $\mathcal{Y}(x)$ and the parameter space $\boldsymbol{\Theta}(x)$ both depend on x. On the contrary, since we want to use the same model and hence the same $\varphi_{\boldsymbol{w}}$ on different instances, the space $\mathcal{W}$ does not depend on x. This raises the question of *how to build statistical model $\varphi_{\boldsymbol{w}}$ whose output dimension depends on the input dimension.* More generally, such an approach can work only if (CO-layer) retains most of the "structure" of (Pb).

Unfortunately, statistical models generally output vectors of fixed size. Let us pinpoint a practical way of addressing this difficulty with a generalized linear model. Let $\mathcal{I}(x)$ be the *structure* x, i.e., the set of dimensions i of $\boldsymbol{\Theta}(x)$. For instance, the structure of the single machine scheduling problem is $\mathcal{I}(x) = [m]$, the structure of the stochastic vehicle scheduling problem is $\mathcal{I}(x) = A$, and the structure of the minimum weight two stage spanning tree is $\big\{(e, \texttt{stage})\colon e \in E,\ \texttt{stage} \in \{1,2\}\big\}$.

As we did in all our examples, we suggest defining a feature mapping

$$\boldsymbol{\phi} : (i,x) \longmapsto \boldsymbol{\phi}(i,x)$$

that associates to an instance and a dimension i in $\mathcal{I}(x)$ a feature vector $\boldsymbol{\phi}(i,x)$ describing the main properties of dimensions i of instance x. We then define

$$\varphi_{\boldsymbol{w}} : x \longmapsto \boldsymbol{\theta} \quad \text{with} \quad \boldsymbol{\theta} = (\theta_i)_{i\in\mathcal{I}(x)} \quad \text{and} \quad \theta_i = \langle \boldsymbol{w} | \boldsymbol{\phi}(i,x)\rangle,$$

where $\langle \cdot \,|\, \cdot \rangle$ denote the usual scalar product. In summary, $\varphi_{\boldsymbol{w}}$ can output parameters $\boldsymbol{\theta}$ whose dimension depends on x because it applies in parallel for all i in $\mathcal{I}(x)$ the same predictor $(i,x) \mapsto \langle \boldsymbol{w} | \boldsymbol{\phi}(i,x)\rangle$ to predict the value of θ_i.

Encoding information on an element as part of an instance

Relevant features for the single machine scheduling and the stochastic vehicle scheduling problems are available in previous contributions [22, 21]. Table 1 provides details on the features used for the maximum weight two-stage spanning tree problem in our numerical experiments.

The simplest features $\phi(i,x)$ only take into account information on i. This is for instance the case of the first three ones in Table 1. The third one notably shows that *quantiles enables to encode information on a distribution in a fixed size vector.* The fourth feature is slightly more advanced and uses information on elements related to i, neighbors in the graph in that case.

The two next techniques show how to encode information on an element as part of an instance. One can *compare dimension i to the other ones in $\mathcal{I}(x)$*. To that purpose, we define a statistic $\alpha : (i,x) \mapsto \alpha(i,x)$, and

consider $f(\boldsymbol{\rho}_e)$ as a realization of the random variable

$$\begin{aligned}\mathcal{A} : \mathcal{I}(x) &\longrightarrow \mathbb{R} \\ i &\longmapsto \alpha(i, x)\end{aligned}$$

and take some relevant statistics on the realization $\mathcal{A}(i)$ of $\mathcal{A}$, such as the value of the cumulative distribution function of F in $\alpha(i, x)$. We illustrate it on the scheduling problem as we do not use it in Table 1. When considering a job j of $1|r_j|\sum_j C_j$ with parameter (r_j, p_j), if we define $\alpha(j, x) = r_j + p_j$, we obtain as feature the rank (divided by m) of feature j in the schedule where we sort the jobs by increasing $r_j + p_j$, a statistic known to be interesting in the scheduling literature and used in dispatching rules.

Finally, we can *explore the role of i in the solution of a very simple optimization problem.* A natural way of building features is to run a fast heuristic on the instance x and seek properties of i in the resulting solution. For instance, the preemptive version of $1|r_j|\sum_j C_j$, where jobs can be stopped, is easy to solve. Statistics such as the number of times job j is preempted in the optimal solution can be used as features. The last four features in Table 1 illustrate this technique.

Equivariant layers

A lesson from geometric deep learning [9] is that good neural network architectures should respect the symmetries of the problem. Let $\mathcal{S}$ be a symmetry of the problem. A layer ς in a neural network is said to be equivariant with respect to $\mathcal{S}$ if $\varsigma(\mathcal{S}(x)) = \mathcal{S}(\varsigma(x))$. In our combinatorial optimization setting, there is one natural symmetry. The solution predicted y should not depend on the indexing of the variables used in the combinatorial optimization problem : Given a permutation of these variables in the instance x, the solution y should be the permuted solution. Combinatorial optimization layers are naturally equivariant with respect to this symmetry. The generalized linear model above is a simple example of equivariant layer.

4 Learning by minimizing the perturbed empirical risk

We now focus on how to learn policies with a combinatorial optimization layer. Given a training set composed of representative instances, the learning problem aims at finding a parameter $\boldsymbol{w}$ such that the output $z = h_{\boldsymbol{w}}(x)$ of our policy has a small cost. As we mentioned in the introduction, the literature focuses on the supervised learning setting. In that case, the training set $(x_1, \overline{y}_1), \ldots, (x_n, \overline{y}_n)$ contains instances and target solutions of the prediction problem (CO-layer), the learning problem can be formulated as

$$\min \frac{1}{n} \sum_{i=1}^{n} \ell(\boldsymbol{\theta}_i, \overline{y}_i) \quad \text{where} \quad \boldsymbol{\theta}_i = \widetilde{\varphi}_{\boldsymbol{w}}(x_i).$$

Here, $\ell(\boldsymbol{\theta}, y_i)$ represents a loss function. This loss function, $\ell(\boldsymbol{\theta}, \overline{y})$, assesses the discrepancy between the output of (CO-layer) and $\overline{y}$. Ideally, it should be 0 if $\overline{y} \in \arg\min_{y \in \mathcal{Y}(x)} g(y, \boldsymbol{\theta})$, and positive otherwise. Furthermore, it should be easy to optimize in $\boldsymbol{\theta}$, which typically involves being convex in $\boldsymbol{\theta}$, and admitting tractable (stochastic) gradients in $\boldsymbol{\theta}$. Examples from the literature include the structured Hinge loss [20] or the Fenchel–Young losses [4]. The SPO+ loss proves successful when the training set contains target $\boldsymbol{\theta}_i$ instead of target y_i [12].

In this paper, we focus on the risk minimization setting, where the training set $(x_1, \ldots, x_n)$ contains instances but not their solutions.

4.1 Learning problem and regularized learning problem

Let $x_1, \ldots, x_n$ be our *training set* composed of n instances of (Pb). Without loss of generality, we suppose that $c(y, x) \geq 0$ for all instances x and feasible solution $y \in \mathcal{Y}(x)$. We also suppose that we have a mapping $u : x \mapsto u(x) \geq 0$ that is a coarse estimate of the absolute value of an optimal solution of x. We define the loss function as the cost of the easy problem solution as a solution of the hard problem, reweighed by $\frac{1}{u(x)}$ to ensure that the loss corresponding to instances of different size are of the same order of magnitude.

$$\ell(\boldsymbol{w}, x) := \frac{1}{u(x)} \max\left\{ c\big(y, x\big) : y \in \operatorname*{arg\,min}_{\widetilde{y} \in \mathcal{Y}(x)} g\big(\widetilde{y}, \varphi_{\boldsymbol{w}}(x)\big) \right\}. \tag{14}$$

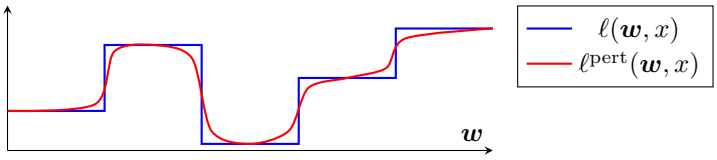


Figure 2 Effect of the smoothing on the loss function.

The *learning problem* consists in minimizing the expected loss on the training set

$$\min_{\boldsymbol{w}\in\mathcal{W}} \frac{1}{n}\sum_{i=1}^{n} \ell(\boldsymbol{w}, x_i). \tag{15}$$

Problem (15) differs from the informal version (4) in the introduction in two ways. First, we use the worse optimal solution of the surrogate problem in the loss (14) to avoid depending on the choice of $\mathcal{A}$. Second, we use the weight $\frac{1}{u(x)}$ in the loss function to ensure that the loss corresponding to instances of different size are of the same order of magnitude.

When the approximation is flexible and the training set is small, the solution of (15) may overfit the training set, and lead to poor performance on instances that are not in the training set. In that case, the usual technique to avoid overfitting is to regularize the problem. One way to achieve this is to make the prediction "robust" with respect to small perturbations: We want the solution returned to be good even if we use $\boldsymbol{w}+\boldsymbol{\eta}$ instead of $\boldsymbol{w}$, where $\boldsymbol{\eta}$ is a small perturbation. Practically, we assume that $\boldsymbol{\eta}$ is a standard Gaussian, $\sigma > 0$ is a real number, and we define the *perturbed loss*

$$\ell^{\text{pert}}(\boldsymbol{w}, x) = \mathbb{E}_{\boldsymbol{\eta}}\left[\frac{1}{u(x)}\max\left\{c\big(y, x\big) : y \in \operatorname*{arg\,min}_{\tilde{y}\in\mathcal{Y}(x)} g\big(\tilde{y}, \varphi_{\boldsymbol{w}+\sigma\boldsymbol{\eta}}(x)\big)\right\}\right]. \tag{16}$$

This perturbation can be understood as a regularization of the surrogate problem [4]. The *regularized learning problem* is then formulated as follows.

$$\min_{\boldsymbol{w}\in\mathcal{W}} \frac{1}{n}\sum_{i=1}^{n} \ell^{\text{pert}}(\boldsymbol{w}, x) \tag{17}$$

4.2 Algorithms to solve the learning problem

Figure 2 illustrates informally the effect of the perturbation on the loss function as detailed in the following results.

▶ **Proposition 1.** *If $\boldsymbol{w} \mapsto \varphi_{\boldsymbol{w}}(x)$ and $\boldsymbol{\theta} \mapsto g(y, \boldsymbol{\theta})$ are piecewise linear for all y in $\mathcal{Y}(x)$, then the objective of (15) is piecewise constant in $\boldsymbol{w}$.*

Proof. Since the composition of two piecewise linear functions is piecewise linear, $\boldsymbol{w} \mapsto g(y, \varphi_{\boldsymbol{w}}(x))$ is piecewise linear. Let $(\mathcal{W}_k)_k$ be a partition of $\mathcal{W}$ such that $\boldsymbol{w} \mapsto g(y, \varphi_{\boldsymbol{w}}(x))$ is linear on each $\mathcal{W}_k$. Let us fix k, we have $g(y, \varphi_{\boldsymbol{w}}(x)) = \boldsymbol{w}^\top A_k y$ for some matrix A_k on $\mathcal{W}_k$. Since the set of optimal solutions of a linear program is piecewise constant in its objective vector (constant on the interior of the normal cone of each vertex of the polyhedron), there is a partition of $\mathcal{W}_k$ into polyhedra such that $\boldsymbol{w} \mapsto \arg\min_{y\in\mathcal{Y}(x)} \boldsymbol{w}^\top A_k y$ is constant on each polyhedron. This concludes the proof. ◀

Proposition 1 is bad news from an optimization point of view. We need a black-box optimization algorithm that uses a moderate amount of function evaluations, does not rely on "slope" (due to null gradient), and takes a global approach (due to non-convexity). We therefore suggest using either a heuristic that searches the state space such as the DIRECT algorithm [15], or a Bayesian optimization algorithm that builds a global approximation of the objective function and uses it to sample the areas in the space of $\boldsymbol{w}$ that are promising according to the approximation. The numerical experiments evaluate the performance of these two kinds of algorithms.

Let us now consider the regularized learning problem (17). Since the convolution product of two functions is as smooth as the most smooth of the two functions, $\boldsymbol{w} \mapsto \ell^{\text{pert}}(\boldsymbol{w}, x)$ is C^∞. It can therefore be minimized using a stochastic gradient descent [10]. On our applications, and using a generalized linear model, we obtain

better results by solving a sample average approximation of this perturbed learning problem using the heuristics mentioned above. This is not so surprising because in that case, the objective of the learning problem is composed of several plateaus with smooth transition inbetween, which is not much easier to solve in practice. Stochastic gradient descent is the method of choice when using a large neural network as $\varphi_{\boldsymbol{w}}$.

4.3 Practical remarks for a generic implementation

Perturbation strength

Section 5 provides a closed formula to set the perturbation strength σ. However, since it involves a constant that is intractable in practice, hyperparameter tuning is helpful.

Skipping the bilevel optimization

Using a bilevel optimization enables to define $\ell(\boldsymbol{w}, x)$ unambiguously even when the easy problem (CO-layer) admits several optimal solutions. Since the bilevel optimization is not easy to handle, we use in practice the loss

$$\widetilde{\ell}(\boldsymbol{w}, x) = \frac{1}{u(x)} c\big(\mathcal{A} \circ \varphi_{\boldsymbol{w}}(x)\big)$$

that takes the solution returned by the algorithm $\mathcal{A}$ we use for (CO-layer). Its value may therefore depend on $\mathcal{A}$.

Decoder

On many applications, the decoder h is time-consuming, and there exists an alternative decoder $\widetilde{h}$ that is much faster, even if the resulting solution z may have a larger cost. A typical example is our $1|r_j|\sum_j C_j$ application, where $\mathcal{Y}(x) = \mathcal{Z}(x)$, and the decoder is only a local descent. The decoder is therefore not mandatory, and we could use $\widetilde{h} = \text{Identity}$. In that context, using $\widetilde{h}$ instead of h during the learning phase leads to a much faster learning algorithm, while not necessarily hurting the quality of the $\boldsymbol{w}$ learned.

Sampling based policy

If we are ready to increase the execution time, the perturbation of $\boldsymbol{w}$ by $\boldsymbol{\eta}$ can also be used to increase the quality of the solution returned by our policy. We can draw several samples $\boldsymbol{\eta}_i$ of $\boldsymbol{\eta}$, apply the policy with $\boldsymbol{w} + \sigma\boldsymbol{\eta}_i$ instead of $\boldsymbol{w}$, and return the best solution found across the samples at the end. We provide numerical results with this perturbed algorithm on the $1|r_j|\sum_j C_j$ problem in Section 6.

5 Learning rate and approximation ratio

This section introduces theoretical guarantees on the average optimality gap of the solution returned by the learned algorithm when $\boldsymbol{w}$ is chosen as in Section 4. Two conditions seem necessary to obtain such guarantees. First, it must be possible to approximate the initial problem by the surrogate one. That is, there must exist a $\widetilde{\boldsymbol{w}}$ such that an optimal solution of (CO-layer) with $\boldsymbol{\theta} = \varphi_{\widetilde{\boldsymbol{w}}}(x)$ provides a good solution of x. And second, when such a $\widetilde{\boldsymbol{w}}$ exists, our learning problem must be able to find it or another $\boldsymbol{w}'$ that leads to a good approximation. Our proof strategy is therefore in two steps. First, we show that the solution of our learning problem converges toward the "best" $\boldsymbol{w}$ when the number of instances in the solution set increases. And then we show that if there exists a $\widetilde{\boldsymbol{w}}$ such that the expected optimality gap of the solution returned by our solution approach is bounded, then the expected optimality gap for the learned $\boldsymbol{w}$ is also bounded. For statistical reasons discussed at the end of Section 5.2, we carry this analysis using the regularized learning problem (17).

5.1 Background on learning with perturbed bounded losses

Let ξ be a random variable on a space Ξ and $\mathcal{W}$ a non-empty compact subset of $\mathbb{R}^d$, $\mathcal{W} \subseteq \mathbb{B}_\infty(M)$ where $\mathbb{B}_\infty(M)$ is the $\|\cdot\|_\infty$ ball of radius M on $\mathbb{R}^d$. Let $\ell : \Xi \times \mathbb{R}^d \to [0, 1]$ be a loss function, we define the perturbed loss as

$$\ell^{\text{pert}}(\bar{\xi}, \boldsymbol{w}) = \mathbb{E}_{\boldsymbol{\eta}}\big[\ell(\bar{\xi}, \boldsymbol{w} + \sigma\boldsymbol{\eta})\big] \quad \text{with} \quad \sigma > 0, \tag{18}$$

where $\boldsymbol{\eta}$ is a standard Gaussian random variable, and $\bar{\xi}$ is a realization of ξ, and $\mathbb{E}_{\boldsymbol{\eta}}$ denotes the expectation with respect to $\boldsymbol{\eta}$. We suppose that $\ell(\,\cdot\,, \boldsymbol{w})$ is integrable for all $\boldsymbol{w} \in \mathcal{W}$. We define the *expected risk* $R(\boldsymbol{w})$ and the *expected risk minimizer* $\boldsymbol{w}^*$ as

$$\boldsymbol{w}^* \in \operatorname*{arg\,min}_{\boldsymbol{w}\in\mathcal{W}} R(\boldsymbol{w}) \quad \text{with} \quad R(\boldsymbol{w}) = \mathbb{E}_{\xi}\big[\ell^{\text{pert}}(\xi, \boldsymbol{w})\big], \tag{19}$$

where $\mathbb{E}_{\xi}$ denotes the expectation with respect to ξ. Let $\xi_1, \ldots, \xi_n$ be n i.i.d. samples of ξ. We define the *empirical risk* $\widehat{R}_n(\boldsymbol{w})$ and the empirical risk minimizer $\widehat{\boldsymbol{w}}_n$ as

$$\widehat{\boldsymbol{w}}_n \in \operatorname*{arg\,min}_{\boldsymbol{w}\in\mathcal{W}} \widehat{R}_n(\boldsymbol{w}) \quad \text{with} \quad \widehat{R}_n(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell^{\text{pert}}(\xi_i, \boldsymbol{w}). \tag{20}$$

The minima exist in (19) and (20) because $\mathcal{W}$ is compact, $\boldsymbol{w} \mapsto \ell^{\text{pert}}(\xi, \boldsymbol{w})$ is $\frac{\sqrt{d}}{\sigma}$-Lipschitz for all ξ thanks to Lemma 10 in Appendix, and $R(\boldsymbol{w})$ and $\widehat{R}_n(\boldsymbol{w})$ are therefore $\frac{\sqrt{d}}{\sigma}$-Lipchitz. Note that both $\widehat{R}_n(\boldsymbol{w})$ and $\widehat{\boldsymbol{w}}_n$ are random due to the sampling of the training set $\xi_1, \ldots, \xi_n$. The following result bounds the excess risk incurred when we use $\widehat{R}_n(\boldsymbol{w})$ instead of $R(\boldsymbol{w})$.

▶ **Theorem 2.** *Suppose that $\mathcal{W} \subseteq \mathbb{B}_\infty(M)$ where $\mathbb{B}_\infty(M)$ is the $\|\cdot\|_\infty$ ball of radius M on $\mathbb{R}^d$. Given $0 < \boldsymbol{\delta} < 1$, with probability at least $1-\delta$, we have the following bound on the excess risk.*

$$R(\widehat{\boldsymbol{w}}_n) - R(\boldsymbol{w}^*) \leq 24\sqrt{\pi}\frac{Md}{\sigma\sqrt{n}} + \sqrt{\frac{2\log(2/\delta)}{n}}. \tag{21}$$

We believe that Theorem 2 is in the statistical learning folklore, but since we did not find a proof, we provide one based on classical statistical learning results in Appendix A.

Learning rate of our structured approximation

Let us now apply Theorem 2 to the learning problem of Section 4. Let us first detail the risk in this context. Recall that the loss ℓ and the perturbed loss ℓ^{pert} have been defined in Equations (14) and (16). In that context, the risks (19) and (20) coincide with those in Equations (3) and (4) in the introduction when $h_{\boldsymbol{w}}$ maps x to a y in $\arg\max\Big\{c\big(y,x\big) \colon y \in \arg\min_{\widetilde{y}\in\mathcal{Y}(x)} g\big(\widetilde{y}, \varphi_{\boldsymbol{w}+\sigma\boldsymbol{\eta}}(x)\big)\Big\}$. The perturbed version corresponds to a $h_{\boldsymbol{w}}$ that is non-deterministic as its output depends on $\boldsymbol{\eta}$.

In order to apply Theorem 2, we need to precise the distribution of x. We assume that the instance x is a random variable with probability distribution μ on $\mathcal{X}$, and that both $\ell(\,\cdot\,, \boldsymbol{w})$ and $\ell^{\text{pert}}(\,\cdot\,, \boldsymbol{w})$ are integrable for all $\boldsymbol{w}$. With these definitions, using $\mathcal{X}$ as Ξ, instances x as random variables ξ, and if we suppose that the training set $x_1, \ldots, x_n$ is composed of n i.i.d. samples of x, we have recast our regularized learning problem (17) as a special case of (20). We can therefore apply Theorem 2 and deduce that the upper bound (21) on the excess risk applies.

This result underlines a strength of the architectures of Section 3.4. Because they enable to use approximations parametrized by a vector $\boldsymbol{w}$ whose dimension is small and does not depend on x, the bound on the excess risk only depends on the dimension of $\boldsymbol{w}$ and not on the size of the instances used. Hence, *a policy with these architectures enables making predictions that generalize (in expectation) on a test set whose instances structures $\mathcal{I}(x)$ are not necessarily present in the training set.* This is confirmed experimentally in Section 6, where the structures of the instances in the test set of the stochastic VSP (the graph D) do not appear in the training set.

▶ Remark 3. Theorem 2 does not take into account the fact that, practically and in our numerical experiments, we use a sample average approximation on $\boldsymbol{\eta}$ of the perturbed loss instead of the true perturbed loss.

5.2 Approximation ratio of our structured approximation

Let $\widetilde{d}(x) = |I(x)|$ and $c^*(x) = \min_{y\in\mathcal{Y}(x)} c(y,x)$ be the cost of an optimal solution of (Pb).

▶ **Theorem 4.** *Suppose that for all x in $\mathcal{X}$ (outside a negligible set for the measure on $\mathcal{X}$),*

1. *$\varphi_{\boldsymbol{w}}(x) = \big(\langle \boldsymbol{w} | \boldsymbol{\phi}(i,x)\rangle\big)_{i\in\mathcal{I}(x)}$ and $\|\boldsymbol{\phi}(i,x)\|_2 \leq \kappa_{\boldsymbol{\phi}}$,*
2. *and there exists $\widetilde{\boldsymbol{w}}$, $a > 0$, $b > 0$, and $\beta \in \{1,2\}$ such that, for any $\boldsymbol{p} \in \mathbb{R}^{\widetilde{d}(x)}$,*

$$c(y,x) - c^*(x) \leq a u(x) + b\|\boldsymbol{p}\|_\beta \quad \textit{for any } y \in \operatorname*{arg\,min}_{\widetilde{y}\in\mathcal{Y}(x)} g\big(\widetilde{y}, \varphi_{\widetilde{\boldsymbol{w}}}(x) + \boldsymbol{p}\big)$$

Then, under the hypotheses of Theorem 2, with probability at least $1-\delta$ (on the sampling of the training set)

$$\underbrace{R(\widehat{\boldsymbol{w}}_n) - \mathbb{E}\left[\frac{c^*(x)}{u(x)}\right]}_{\text{Perturbed prediction optimality gap}} \leq \underbrace{24\sqrt{\pi}\frac{Md}{\sigma\sqrt{n}} + \sqrt{\frac{2\log(2/\delta)}{n}}}_{\text{Training set error}} + \underbrace{a}_{\substack{\text{Appro-}\\ \text{ximation}\\ \text{error}}} + \underbrace{b\sigma\kappa_{\boldsymbol{\phi}}\sqrt{d}\mathbb{E}\left[\frac{[\widetilde{d}(x)]^{1/\beta}}{u(x)}\right]}_{\text{Perturbation error}}$$

Before proving the theorem, let us make some comments. First, we explain why the hypotheses are meaningful. The first hypothesis only assumes that the model is linear, and that, with probability 1 on the choice of x in $\mathcal{X}$ the features are bounded. Such a hypothesis is reasonable as soon as we restrict ourselves to instances whose parameters are bounded. Let us recall that $u(x)$ is a coarse upper bound on $c^*(x)$. When $u(x) = c^*(x)$ and $\boldsymbol{p} = 0$, the second hypothesis only means that our non-perturbed policy with parameter $\widetilde{\boldsymbol{w}}$ is an approximation algorithm with ratio $1 + a$. With a $\boldsymbol{p} \neq 0$, the second hypothesis is stronger: It also ensures that this approximation algorithm guarantee does not deteriorate too fast. Later in this section, we prove that this hypothesis is satisfied for our running example.

Approximation ratio guarantee

Using $u(x) = c^*(x)$ makes clear the fact that *Theorem 4 provides an approximation ratio guarantee in expectation.* Furthermore, it gives *a natural way of setting the strength* σ of the perturbation: The bound is minimized when we use

$$\sigma_n = \sqrt{\frac{24\sqrt{\pi}M\sqrt{d}}{\sqrt{n}b\kappa_{\boldsymbol{\phi}}\mathbb{E}\left[\frac{[\tilde{d}(x)]^{1/\beta}}{u(x)}\right]}}. \tag{22}$$

Using this optimal perturbation and $\delta = \frac{1}{n}$, the upper bound on $R(\widehat{\boldsymbol{w}}_n) - \mathbb{E}\left[\frac{c^*(x)}{u(x)}\right]$ is in

$$a + O\Big(n^{-1/4}\big(1+\log(n)\big)\Big) \xrightarrow[n\to+\infty]{} a.$$

In other words, *in the large training set regime the learned $\widehat{\boldsymbol{w}}_n$ recovers the approximation ratio guarantee a of $\widetilde{\boldsymbol{w}}$.*

Large instances

Theorem 4 always provides guarantees when $\widetilde{d}(x)$ is bounded on $\mathcal{X}$. However, it may fail to give guarantees when $\widetilde{d}(x)$ is unbounded on $\mathcal{X}$ since $\mathbb{E}\left[\frac{[\tilde{d}(x)]^{1/\beta}}{u(x)}\right]$ may not be finite. Since $u(x)$ is a coarse upper bound on $c^*(x)$, the term $\mathbb{E}\left[\frac{[\tilde{d}(x)]^{1/\beta}}{u(x)}\right]$ remains finite when $\widetilde{d}(x)$ is unbounded only if the cost of an optimal solution $c^*(x)$ grows at least as fast as the number of parameters of the instance $\widetilde{d}(x)$ to the power $1/\beta$. From that point of view, the single machine scheduling problem $1|r_j|\sum_{C_j}$ is ideal. Indeed, in that case $\widetilde{d}(x)$ is the number of job, and $c^*(x) \sim [\widetilde{d}(x)]^2$, hence $\left[\frac{[\tilde{d}(x)]^{1/\beta}}{u(x)}\right]$ becomes smaller and smaller when the size of x increases. On the two stage spanning tree problem, the situation is slightly less favorable. Indeed, $\widetilde{d}(x)$ is equal to twice the number of edges. Since an optimal spanning tree contains $|V|-1$ edges, we expect $c^*(x)$ to be of the order of magnitude of $\widetilde{d}(x)$ on sparse graphs (graphs such that $|E| \sim |V|$, like grids for instance), and $\sqrt{\widetilde{d}(x)}$ on dense graphs (graph such that $|E| \sim |V|^2$, like complete graphs). Later in this section, we prove that the second hypothesis is satisfied for the two stage spanning tree problem with $\beta = 1$. Hence, Theorem 4 gives an approximation ratio guarantee for sparse graphs. The situation is roughly the same for the stochastic vehicle scheduling problem. A typical example where $\mathbb{E}\frac{[\tilde{d}(x)]^{1/\beta}}{u(x)}$ may not be finite is the shortest path problem on a dense graph. On many applications such as finding an optimal journey on a public transport system, the number of arcs tends to remain bounded, say ≤ 10, while the number of arcs in the graph $\widetilde{d}(x)$ grows with the size of the instance.

Optimal resolution of the learning problem

Theorem 4 applies for the optimal solution $\widehat{\boldsymbol{w}}_n$ of the learning problem. We leave it to future work to design an exact algorithm that guarantees that the $\boldsymbol{w}$ returned is within an optimality gap γ with the optimal solution. We would then obtain a variant of Theorem 4 proving the approximation ratio result for the $\boldsymbol{w}$ returned, with an additional term in γ in the upper bound taking into account the optimality gap.

Influence of the perturbation

Since we have made very few assumptions on c, g, and $\varphi_{\boldsymbol{w}}$, we do not have control on the size of the family of functions $\{\ell_{\boldsymbol{w}} \colon \boldsymbol{w}\}$, This family may be very large, and therefore able to fit any noise, which would lead to slower learning rate. Without additional assumptions, we therefore need to regularize the family. In particular, we need to smooth the piecewise constant loss (Proposition 1). As we have seen in this section, perturbing $\boldsymbol{w}$ does the job, but comes at a double cost in Theorem 4: a perturbation error, and larger than hoped training set error in $O(d/\sqrt{n})$. The term in $O(d/\sqrt{n})$ is slightly disappointing because the proof techniques used in statistical learning theory typically lead to bounds in $O(\sqrt{d}/\sqrt{n})$. This is for instance the case for the metric entropy method used to prove Theorem 2 when the gradient of the loss is Lipschitz in $\boldsymbol{w}$. The Gaussian perturbation restores the Lipschitz property for the perturbed loss, but it comes at the price of an additional $\sqrt{d}$ in the bound derived by the metric entropy method, as can be seen in the proof of Lemma 10 in Appendix A. Designing a learning approach that avoids the additional $\sqrt{d}$ term is an interesting open question. An alternative would be to make more assumptions on c, g, and φ with the objective of making the perturbation optional.

Proof of Theorem 4. Theorem 2 and the definition of $\boldsymbol{w}^*$ give

$$R(\widehat{\boldsymbol{w}}_n) - \mathbb{E}\left[\frac{c^*(x)}{u(x)}\right] = \underbrace{R(\widehat{\boldsymbol{w}}_n) - R(\boldsymbol{w}^*)}_{\leq C\frac{Md}{\sigma\sqrt{n}} + \sqrt{\frac{2\log(2/\delta)}{n}}} + \underbrace{R(\boldsymbol{w}^*) - R(\widetilde{\boldsymbol{w}})}_{\leq 0} + R(\widetilde{\boldsymbol{w}}) - \mathbb{E}\left[\frac{c^*(x)}{u(x)}\right]$$

We therefore need to upper bound $R(\widetilde{\boldsymbol{w}}) - \mathbb{E}\big[\frac{c^*(x)}{u(x)}\big]$ by $a + b\sigma\kappa_{\boldsymbol{\phi}}\sqrt{d}\mathbb{E}\big[\frac{[\tilde{d}(x)]^{1/\beta}}{u(x)}\big]$. We have

$$\begin{aligned}
\big\|\varphi_{\tilde{\boldsymbol{w}}+\sigma\boldsymbol{\eta}}(x) - \varphi_{\tilde{\boldsymbol{w}}}(x)\big\|_\beta &= \Big\|\sigma\big(\langle\boldsymbol{\eta}|\boldsymbol{\phi}(i,x)\big)_{i\in\mathcal{I}(x)}\Big\|_\beta \\
&\leq \Big\|\sigma\big(\|\boldsymbol{\eta}\|_2\|\boldsymbol{\phi}(i,x)\|_2\big)_{i\in\mathcal{I}(x)}\Big\|_\beta \\
&\leq \Big\|\sigma\big(\|\boldsymbol{\eta}\|_2\kappa_{\boldsymbol{\phi}}\big)_{i\in\mathcal{I}(x)}\Big\|_\beta = \sigma\kappa_{\boldsymbol{\phi}}\|\boldsymbol{\eta}\|_2[\widetilde{d}(x)]^{1/\beta}
\end{aligned}$$

Let y be in $\arg\min_{y\in\mathcal{Y}(x)} g\big(y, \varphi_{\tilde{\boldsymbol{w}}+\sigma\boldsymbol{\eta}}(x)\big)$. The second hypothesis of the theorem and the previous inequality give

$$c(y,x) - c^*(x) \leq au(x) + b\kappa_{\boldsymbol{\phi}}\|\boldsymbol{\eta}\|_2[\widetilde{d}(x)]^{1/\beta}.$$

Since $\boldsymbol{\eta}$ is a standard Gaussian, we get $\mathbb{E}\|\boldsymbol{\eta}\| \leq \sqrt{d}$ (Equation (28) in Appendix A), and the result follows by dividing the previous equality by $u(x)$ and taking the expectation. ◀

5.3 Existence of a $\widetilde{w}$ with an approximation ratio guarantee

In this section, we prove that the hypotheses of Theorem 4 are satisfied for the maximum weight two stage spanning tree problem. We then give a criterion which ensures that these hypotheses are satisfied.

Maximum weight two stage spanning tree

In this section, we restrict ourselves to maximum weight spanning tree instances, that is, instances of the minimum weight spanning tree with $c_e \leq 0$ and $d_{es} \leq 0$ for all e in E and s in S. Given an instance, let $I(x) = \big((e, \texttt{stage}) \colon e \in E, \texttt{stage} \in \{1,2\}\big)$, leading to $\bar{c}_e = \big\langle \boldsymbol{w}|\boldsymbol{\phi}\big((e,1),x\big)\big\rangle$ and $\bar{d}_e = \big\langle \boldsymbol{w}|\boldsymbol{\phi}\big((e,2),x\big)\big\rangle$. We define a feature

$$\boldsymbol{\phi}(e,1) = c_e \quad \text{and} \quad \boldsymbol{\phi}(e,2) = \frac{1}{|S|}\sum_{s\in S} d_{es},$$

and define $\widetilde{\boldsymbol{w}}$ to be equal to 1 for this feature and 0 otherwise. The following proposition shows that the second hypothesis of Theorem 4 is then satisfied with $a = 1/2$ and $b = 1$.

▶ **Proposition 5.** *For any instance x of the maximum weight spanning tree problem, we have*

$$c(y,x) - c^*(x) \leq \frac{1}{2}|c^*(x)| + \|\boldsymbol{p}\|_1 \quad \textit{for any } y \in \operatorname*{arg\,min}_{y'\in\mathcal{Y}(x)} g\big(y, \varphi_{\tilde{\boldsymbol{w}}}(x) + \boldsymbol{p}\big).$$

Our proof shows that the results stands for $a = \frac{|S|-1}{2|S|-1}$ when $|S|$ is upper-bounded by M. Combined with Theorem 4, Proposition 5 ensures that, when the training set is large, the learned $\widehat{w}_n$ has the approximation ratio guarantee proved by Escoffier et al. [13] for the two stage maximum weight spanning tree. The proof of Proposition 5 is an extension of the proof of [13, Theorem 6] to deal with non-zero perturbations $\boldsymbol{p}$.

Proof of Proposition 5. The proof will use the following well known result

▶ **Lemma 6.** *Let $x \mapsto f_1(y)$ and $y \mapsto f_2(y)$ be functions from compact set K to $\mathbb{R}$, and let y_1^* and y_2^* be respectively minima of f_1 and f_2. If we have $|f_1(y) - f_2(y)| \leq \gamma$ for all y, then $f_1(y_2^*) - f_1(y_1^*) \leq 2\gamma$.*

We fix an instance x. Let us first introduce some solutions of interest. Given a $\boldsymbol{\theta}$, let us denote by $\overline{y}(\boldsymbol{\theta}) = (\overline{E}_1(\boldsymbol{\theta}), \overline{E}_2(\boldsymbol{\theta}))$ the result of the prediction problem (13). Let $\overline{z}(\boldsymbol{\theta}) = (\overline{E}_1(\boldsymbol{\theta}), (\overline{E}_s(\boldsymbol{\theta})))$ with $\overline{E}_s(\theta) = \overline{E}_2(\boldsymbol{\theta})$. Let $\widehat{z}(\boldsymbol{\theta}) = (\overline{E}_1(\boldsymbol{\theta}), (\widehat{E}_s(\boldsymbol{\theta})))$ where $E_s(\boldsymbol{\theta})$ is the optimal second stage decision for scenario s when the first stage decision is $E_1(\boldsymbol{\theta})$

$$\widehat{E}_s(\boldsymbol{\theta}) \in \arg\min \left\{ \sum_{e \in E_s} d_{es} \colon E_s \subseteq E,\ E_s \cap \overline{E}_1(\boldsymbol{\theta}) = \emptyset,\ (V, E_s \cup \overline{E}_1(\boldsymbol{\theta})) \in \mathcal{T} \right\}. \tag{23}$$

We denote by $z^{\emptyset} = (\emptyset, (E_s^{\emptyset}))$ the solution with no first stage: $(V, E_s^{\emptyset})$ is a minimum weight spanning tree for second stage weights $(d_{es})_s$ of scenario s. And finally, we denote by $z(\boldsymbol{\theta}) = (E_1, (E_s)_{s \in S})$ the solution returned by our policy, which is the solution of minimum cost among $\widehat{z}(\boldsymbol{\theta})$ and $z^{\emptyset}$.

Let $z^* = (E_1^*, (E_s^*))$ be an optimal solution of (12), and $\overline{y}^{s,*} = (E_1^*, E_s^*)$ be the solution of (13) obtained by taking E_1^* as first stage solution and E_s^* as second stage solution.

First, consider solution $\overline{z} = (\overline{E}_s, (\overline{E}_s)_s)$ of (12) such that the second stage solution $\overline{E}_s$ is identical and equal to $\overline{E}_2$ for all scenarios s in S, and denote by $\overline{y} = (\overline{E}_1, \overline{E}_2)$ the solution of (13) obtained by taking $\overline{E}_1$ as first stage solution and $\overline{E}_2$ as second stage solution. It follows from the definition of $\widetilde{\boldsymbol{w}}$ that

$$C(\overline{z}) = g(\overline{y}, \widetilde{\boldsymbol{\theta}}) \quad \text{where} \quad \widetilde{\boldsymbol{\theta}} = \varphi_{\widetilde{\boldsymbol{w}}}(x). \tag{24}$$

Second, note that, for any $\boldsymbol{\theta}$, $y = (E_1, E_2)$, and $\boldsymbol{p}$ in $\mathbb{R}^{\tilde{d}(x)}$, we have

$$|g(y, \theta) - g(y, \theta + \boldsymbol{\eta})| = |\langle \boldsymbol{\eta} | y \rangle| = \sum_{e \in E_1} p_{e1} + \sum_{e \in E_2} p_{e2} \leq \|\boldsymbol{p}\|_1 \tag{25}$$

where the last inequality comes from the fact that y is the indicator vector of a tree.

Let s be scenario in S. We have

$$\begin{aligned}
C(\widehat{z}(\widetilde{\boldsymbol{\theta}} + \boldsymbol{\eta})) &\leq C(\overline{z}(\widetilde{\boldsymbol{\theta}} + \boldsymbol{\eta})) && \text{Optimal second stage} \\
&= g(\overline{y}(\widetilde{\boldsymbol{\theta}} + \boldsymbol{\eta}), \widetilde{\boldsymbol{\theta}}) && \text{Equation (24)} \\
&\leq g(\overline{y}(\widetilde{\boldsymbol{\theta}}), \widetilde{\boldsymbol{\theta}}) + 2\|\boldsymbol{p}\|_1 && \text{Equation (25) + Lemma 6} \\
&\leq g(\overline{y}^{s,*}, \widetilde{\boldsymbol{\theta}}) + 2\|\boldsymbol{p}\|_1 && \text{Optimality of } \overline{y}(\widetilde{\boldsymbol{\theta}}) \\
&= \sum_{e \in \overline{E}_1^*} c_e + \frac{1}{|S|} \sum_{e \in E_s^*} \sum_{s' \in S} d_{es'} + 2\|\boldsymbol{p}\|_1 \\
&\leq \sum_{e \in \overline{E}_1^*} c_e + \frac{1}{|S|} \sum_{e \in E_s^*} d_{es} + 2\|\boldsymbol{p}\|_1 && d'_{es} \leq 0 \text{ for all } e, s'.
\end{aligned}$$

Furthermore, since $(V, E_s^{\emptyset})$ is a minimum spanning tree with $(d_{es})_e$ edge weights,

$$C(z^{\emptyset}) = \frac{1}{|S|} \sum_{s \in S} \sum_{e \in E_s^{\emptyset}} d_{es} \leq \frac{1}{|S|} \sum_{s \in S} \left[\sum_{e \in E_1^*} \underbrace{d_{es}}_{\leq 0} + \sum_{e \in E_s^*} d_{es} \right] \leq \frac{1}{|S|} \sum_{s \in S} \sum_{e \in E_s^*} d_{es}.$$

Summing the two previous inequalities, we get

$$\begin{aligned}
C(z(\boldsymbol{\theta})) &= \min(C(\widehat{z}(\widetilde{\boldsymbol{\theta}} + \boldsymbol{\eta})), C(z^{\emptyset})) \\
&\leq \frac{|S| C(\widehat{z}(\widetilde{\boldsymbol{\theta}} + \boldsymbol{\eta})) + (|S| - 1) C(z^{\emptyset})}{2|S| - 1} \\
&\leq \frac{|S| \left(\sum_{e \in E_1^*} c_e + 2\|\boldsymbol{p}\|_1 \right) + \sum_s \sum_{e \in E_s^*} d_{es}}{2|S| - 1} \\
&= \frac{|S|}{2|S| - 1} (C(z^*) + 2\|\boldsymbol{p}\|_1) \leq \frac{1}{2} C(z^*) + \|\boldsymbol{p}\|_1
\end{aligned}$$

Since $C(z^*) \leq 0$, we get $C(z(\boldsymbol{\theta})) - C(z^*) \leq \frac{1}{2}|C(z^*)| + \|\boldsymbol{p}\|_1$, which is the result searched. ◀

Remark that we have proved the stronger bound $c(y, x) - c^*(x) \leq \frac{|M|-1}{2|M|-1}|c^*(x)| + \|\boldsymbol{p}\|_1$ when $|S|$ is upper bounded by M on $\mathcal{X}$.

Objective function approximation

Let us finally highlight that the hypotheses of Theorem 4 are satisfied when $g(y, \varphi_{\boldsymbol{w}}(x)$ is a good approximation of $\boldsymbol{\theta} \mapsto g(y, \boldsymbol{\theta})$ for some $\boldsymbol{w}$.

▶ **Lemma 7.** *Suppose that $\boldsymbol{\theta} \mapsto g(y, \boldsymbol{\theta})$ is κ_g Lipschitz in $\|\cdot\|_\beta$, and there exists $\widetilde{\boldsymbol{w}} \in \mathcal{W}$ and $\alpha > 0$ is such that, for all $x \in \mathcal{X}$ and $y \in \mathcal{Y}(x)$, we have*

$$\frac{|c(y, x) - g(y, \varphi_{\widetilde{\boldsymbol{w}}}(x))|}{u(x)} \leq \alpha. \tag{26}$$

Then the second hypothesis of Theorem 4 is satisfied with $a = 2\alpha$ and $b = 2\kappa_g$.

Proof. For any y in $\mathcal{Y}(x)$, we have

$$\begin{aligned}
&|c(y, x) - g(x, \varphi_{\widetilde{\boldsymbol{w}}}(x) + \boldsymbol{p})| \\
&\leq |c(y, x) - g(x, \varphi_{\widetilde{\boldsymbol{w}}}(x))| + |g(x, \varphi_{\widetilde{\boldsymbol{w}}}(x)) - g(x, \varphi_{\widetilde{\boldsymbol{w}}}(x) + \boldsymbol{p})| \\
&\leq \alpha u(x) + \kappa_g \|\boldsymbol{p}\|_\beta
\end{aligned}$$

The results therefore follows from Lemma 6 and the previous inequality. ◀

6 Numerical experiments

Experiments design

This section tests the performance of our algorithms on the stochastic vehicle scheduling problem, the $1|r_j| \sum_j C_j$ scheduling problem of Section 3, and the maximum weight spanning tree problem. The performance of our policies depends on the policy architecture and the learning algorithm. Our goal is to show that our policies trained by regret minimization lead the state-of-the-art algorithms for these problems. Previous contributions have introduced architectures for the two first problems and shown that they lead to state-of-the-art algorithms when trained by supervised learning [21, 22]. We therefore use these as benchmarks. Since Fenchel Young losses have become the state-of-the-art method for supervised structured learning, we use it as a benchmark and refer the readers to the following papers for an introduction [7, 4, 10]. Our goal for these applications is to show that risk minimization leads to similar or better performance than supervised learning without the need of a supervised learning dataset. We also discuss the choice of different black-box solvers for the learning problem on these applications. Our goal on the maximum weight spanning tree problem is to show that our policy gives the performance of the best heuristics for the problem while providing the approximation ratio guarantee. All the numerical experiments have been performed on a Linux computer running Ubuntu 20.04 with an Intel® Core™ i9-9880H CPU @ 2.30GHz × 16 processor and 64 GiB of memory. All the learning problem algorithms are parallelized: The value of the loss on the different instances in the training set are computed in parallel. The prediction problem algorithms are not parallelized. The code to reproduce the numerical experiments is open source[1].

[1] `https://github.com/axelparmentier/MaximumWeightTwoStageSpanningTree.jl`, `https://github.com/axelparmentier/LearningStructuredApproximationsOfCOproblemsCPP`

Table 2 Instances considered for the stochastic VSP.

Size	$\lvert V\rvert$	$\lvert\Omega\rvert$ 50	100	200	500	1000	Train (small)	Train (moderate)	Train (all)	Val	Test
	50	✓	✓	✓	✓	✓	10	5	1	2	8
Moderate	75	✓	✓	✓	✓	✓		5	1	2	8
	100	✓	✓	✓	✓	✓		5	1	2	8
	200	✓	✓	✓	✓	✓		5	1	2	8
Large	500	✓	✓	✓	✓	✓			1	2	8
	750	✓	✓	✓	✓	✓			1	2	8
	1000	✓	✓	✓	✓	✓			1	2	8
Huge	2000	✓								2	8
	5000	✓								2	8

6.1 Stochastic VSP

6.1.1 Setting: Features, decoder, and instances

For the numerical experiments on the stochastic VSP, we use the exact same settings as in our previous work [21]. We use the same linear predictor with a vector $\boldsymbol{\phi}$ containing 23 features. And we do not use a decoder ψ. The easy problem is solved with `Gurobi 9.0.3` using the LP formulation based on flows.

We also use the same instance generator. This generator takes in input the number of tasks $|V|$, the number of scenarios $|\Omega|$ in the sample average approximation, and the seed of the random number generator. We say that an instance is of moderate size if $|V| \leq 100$, of large size if $100 \leq |V| \leq 750$, and of huge size if $1000 \leq |V|$. Table 2 summarizes the instances generated. The first two columns indicate the size of the instances. A ✓ in the next five columns $|\Omega|$ indicates that instances with $|\Omega|$ scenarios are generated for instance size $|V|$ considered. The last five columns detail the composition of the different sets of instances: Three training sets, one validation set (Val), and a test set (Test). The table can be read as follows: The training set (small) contains $10 \times |\{50, 100, 200, 500, 1000\}| = 50$ instances, each of these having 50 tasks in V, but no larger instances. The test set contains instances of all size. For instance, it contains $8 \times |\{50, 100, 200, 500, 1000\}| = 40$ instances of size 50 and 8 instances of size 5000. For the largest sizes, we use only instances with 50 scenarios for memory reasons: The instances files already weigh several gigabytes.

The small training set, the validation set, and the test set are identical to those previously used [21]. The validation set, which is used in the learning by demonstration approach, is not used on the risk minimization approach, since we do not optimize on classifiers hyperparameters. This previous contribution considers only the "small" training set, with 50 instances with 50 tasks, it uses a learning by demonstration approach and exact solvers cannot handle larger instances. This is no longer a constraint with the risk minimization approach proposed in this paper. We therefore introduce two additional training sets: one that contains 100 instances of moderate size, and one containing 35 instances of all sizes. These training sets are relatively small in terms of number of instances, but they already lead to significant learning problem computing time and good performance on the test set.

6.1.2 Learning algorithm

On each of the three training sets, we solve the learning problem (15) and the regularized learning problem (17). We use the number of tasks $|V|$ as $u(x)$. It is not an upper bound on the cost, but the cost of the optimal solution scales almost linearly with $|V|$. In both case, we solve the learning problem on the L_∞ ball of radius 10. For the regularized learning problem, we use a perturbation strength of intensity $\sigma = 1$, and we solve the sample averaged approximation of the problem with 100 scenarios. We evaluate two heuristic algorithms: The DIRECT algorithm [15] implemented in the `nlopt` library [14], and the Bayesian optimization algorithm as it is implemented in the `bayesopt` library [18]. We run each algorithm on 1000 iterations, which means that they can compute the objective function 1000 times. Both algorithms are launched with the default parameters of the libraries. In particular, the Bayesian optimization algorithm uses the anisotropic kernel with automatic relevance determination `kSum(kSEARD,kConst)` of the library.

Table 3 summarizes the result obtained with both algorithms. The first column contains the loss used: ℓ for the non-regularized problem (15) and ℓ^{pert} for the regularized problem. The next one provides the training

Table 3 Performance of the DIRECT [14] and Bayesian optimization [18] on the learning problems (15) and (17) for the stochastic VSP.

Learning problem			DIRECT		Bayes Opt	
Obj.	Train. set	pert	CPU time (hh:mm:ss)	Obj	CPU time (days, hh:mm:ss)	Obj
ℓ	small	–	0:01:20	290.39	0:09:06	287.48
	moderate	–	0:10:56	256.39	0:18:35	259.88
	all	–	2:56:40	231.66	3:56:18	235.04
ℓ^{pert}	small	100	0:11:52	286.11	0:37:22	287.95
	moderate	100	1:58:50	258.76	2:09:17	259.13
	all	100	22:44:03	233.84	1 day, 5:35:30	235.10

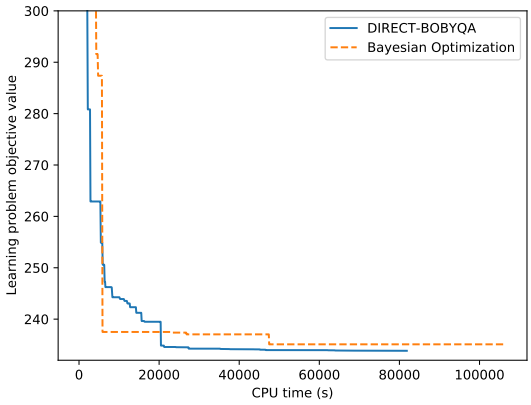


Figure 3 Learning problem objective value evolution as a function of time on stochastic VSP learning problem with perturbed loss ℓ^{pert} on training set "all".

set used. And the third column provides the number of samples used in the sample average approximation of the perturbation. The next four columns give the total computing time for the 1000 iterations and the value of the objective of the learning problem obtained at the end using the DIRECT algorithm and the Bayesian optimization algorithm.

The DIRECT algorithm approximates the value of the function based on a division of the space into hypercubes. At each iteration, the function is queried in the most promising hypercube, and the result is used to split the hypercube. The algorithm leverages a tractable lower bound to identify the most-promising hypercube and the split with few computations. Hence, the algorithm is very fast if the function minimized is not computationally intensive. The Bayesian optimization algorithm builds an approximation of the function minimized: It seeks the best approximation of the function in a reproducing kernel Hilbert space (RKHS) given the data available. At each iteration, it minimizes an activation function to identify the most promising point according to the model, evaluate the function at that point, and updates the approximation based on the value returned. Each of these steps are relatively intensive computationally. Hence, if the function minimized is not computationally intensive, the algorithm will be much slower than the DIRECT algorithm. This is what we observe on the first line of Table 3. Furthermore, in `bayesopt`, the DIRECT algorithm of `nlopt` is used to minimize the activation function. Our numerical experiments tend to indicate that the approximation in a RKHS does not enable to find a better solution than the simple exploration with DIRECT after 1000 iterations. Using Bayesian optimization may however be useful with a smaller iteration budget. Figure 3 provides the evolution of the objective function along time for the Bayesian optimization algorithm and the DIRECT algorithm on the learning problem corresponding to the last line of Table 3.

Since the DIRECT algorithm gives the best performance on most cases, we keep the $\boldsymbol{w}$ returned by this algorithm for the numerical experiments on the test set. The performance of the DIRECT algorithm could be improved using the optimization on the seed that will be introduced in Section 6.2.2.

Table 4 Performance of our solution algorithm with different $\boldsymbol{w}$ on the stochastic VSP test set.

Learning problem $\boldsymbol{w}$			Moderate				Large				Huge				All			
Obj	Train. set	Pert	T^{avg}	$\frac{T^{\max}}{T^{\text{avg}}}$	δ^{avg}	$\delta^{\max}$	T^{avg}	$\frac{T^{\max}}{T^{\text{avg}}}$	δ^{avg}	$\delta^{\max}$	T^{avg}	$\frac{T^{\max}}{T^{\text{avg}}}$	δ^{avg}	$\delta^{\max}$	T^{avg}	$\frac{T^{\max}}{T^{\text{avg}}}$	δ^{avg}	$\delta^{\max}$
CRF	small	–	0.03	2.14	9.47%	20.47%	1.21	1.47	2.58%	6.77%	27.69	1.09	1.27%	1.88%	5.74	2.14	5.13%	20.47%
FYL	small	100	0.03	2.02	1.67%	4.23%	0.97	1.52	0.70%	2.10%	19.20	1.15	0.26%	1.06%	4.04	2.02	1.01%	4.23%
ℓ	small	–	0.03	2.33	4.37%	10.35%	0.82	1.47	3.65%	5.56%	20.42	1.26	3.29%	4.90%	4.21	2.33	3.88%	10.35%
	moderate	–	0.03	1.77	**0.31%**	3.24%	0.86	1.39	1.09%	2.92%	17.48	1.13	2.85%	6.18%	3.67	1.77	1.10%	6.18%
	all	–	0.03	1.56	0.52%	**1.98%**	0.84	1.33	**0.07%**	**0.86%**	18.10	1.14	**0.07%**	**0.66%**	3.78	1.56	**0.25%**	**1.98%**
ℓ^{pert}	small	100	0.03	1.81	2.90%	6.71%	0.84	1.53	2.55%	4.48%	16.29	1.99	2.04%	3.61%	3.44	1.99	2.59%	6.71%
	moderate	100	0.03	1.98	1.56%	5.00%	0.86	1.42	0.77%	2.12%	16.79	1.09	0.90%	1.82%	3.54	1.98	1.11%	5.00%
	all	100	0.03	1.63	1.05%	3.57%	0.87	1.45	1.10%	2.83%	18.08	1.30	1.16%	2.22%	3.79	1.63	1.09%	3.57%

The best results are in bold. CRF = Conditional Random Field

6.1.3 Algorithm performance on test set

We now evaluate the performance of our policy with the $\boldsymbol{w}$ learned. It has been shown [21] that, using the policy with the $\boldsymbol{w}$ learned by the structured learning approach with a conditional random field (CRF) loss on the small training set gives a state-of-the-art algorithm for the problem (the paper uses the maximum likelihood terminology instead of CRF loss). We therefore use it as a benchmark of the problem. We have also introduced a new learning by demonstration approach on the problem: We implement the Fenchel Young loss (FYL) structured learning approach [22, 4] to obtain a second benchmark.

Table 4 summarizes the results obtained. The first three columns indicate how $\boldsymbol{w}$ has been computed: They provide the loss minimized as objective of the learning problem (Obj), the training set used, and for the approaches that use a perturbation, the number of scenarios used in the sample average approximation (SAA).

The next columns provide the results on the test set. These columns are divided into four blocks giving results on the subsets moderate, large, huge instances of the test set and on the full test set. On each of these subsets of instances, we provide four statistics. The statistic T^{avg} provides the average computing time for our full policy on the subset of instances considered, which includes the computation of the features and $\varphi_{\boldsymbol{w}}(x)$, and the resolution of the easy problem with the LP solver (no decoding ψ is used). Most of this time is spent in the LP solver. Then, for each instance in the training set, we compute the ratio of the computing time for the instance divided by the average computing time for all the instances of the test set with the same number of tasks $|V|$. Indeed, we expect instances with the same $|V|$ to be of comparable difficulty. The column $\frac{T^{\max}}{T^{\text{avg}}}$ gives the maximum value of this ratio on the subset of instances considered. Since we do not have an exact algorithm for the problem, for each instance x we compute the gap

$$\frac{c_{\boldsymbol{w}} - c^{\text{best}}}{c^{\text{best}}} \tag{27}$$

between the cost $c_{\boldsymbol{w}}$ of the solution returned by our policy with the $\boldsymbol{w}$ evaluated and the cost of the best solution found for these instances using all the algorithms tested. The columns δ^{avg} and $\delta^{\max}$ respectively provide the average and the maximum value of this gap on the set of instances considered. The two first lines provide the result obtained with the learning by demonstration benchmarks, and the next six ones obtained with the $\boldsymbol{w}$ obtained with the learning algorithms of Table 3.

We can conclude from these experiments that:

1. When using the learning by demonstration approach, the Fenchel Young loss leads to better performances than the conditional random field loss.
2. Our risk minimization approach gives slightly weaker performances than the supervised learning approach with a Fenchel Young loss when using the same training set.
3. Our risk minimization approach enables us to use a more diversified training set, which enables it to outperform all the previously known approaches. The more diversified the training set, the better the performance.
4. The regularization by perturbation used tends to decrease the performance of the algorithm. This statement may no longer hold if we optimized the strength of the perturbation using a validation set.

6.2 Single machine scheduling problem $1|r_j|\sum_j C_j$

6.2.1 Setting

We use the exact same setting as the previous contribution on this problem [22]. In particular, that paper introduces a vector of 66 features, and a subset of 27 features that leads to better performances. With the

Table 5 Size of the $1|r_j|\sum_j C_j$ instances in the subsets of the test set.

	Subsets of instances		
	Moderate	Large	Huge
Size n of instances in subset	$\{50, 75, 100, 150\}$	$\{200, 300, 500, 750\}$	$\{1000, 1500, 2000, 3000\}$

Table 6 Learning algorithm results on $1|r_j|\sum_j C_j$.

obj	iter	pert	ψ	Tot. CPU	Avg $\widehat{L}$	Best $\widehat{L}$
ℓ	1000	–	–	0:05:46	36.71	35.19
ℓ	2500	–	–	0:12:31	36.64	35.29
ℓ^{pert}	1000	100	–	9:09:06	36.76	35.27
ℓ^{pert}	2500	100	–	20:45:52	36.69	35.25
ℓ	1000	–	LS	0:52:54	35.13	35.06
ℓ	2500	–	LS	1:47:42	35.12	35.06
ℓ^{pert}	1000	100	LS	3 days, 15:14:42	35.12	35.06
ℓ^{pert}	2500	100	LS	7 days, 15:24:08	35.11	35.06

Tot. CPU is given in days, hh:mm:ss.

objective of testing what our learning algorithm can do on a larger dimensional problem, we focus ourselves on the problem with 66 features. And we also use the four kinds of decoding algorithms in that paper: no decoding (no ψ), a local search (LS), the same local search followed by release date improvement (RDI) algorithm (RDI $\circ$ LS), and the perturbed versions of the last algorithm, (pert RDI $\circ$ LS) where the policy is applied with $\boldsymbol{w} + \boldsymbol{\eta}$ for 150 different samples of a standard Gaussian $\boldsymbol{\eta}$, and keep the best solution found. RDI is a classic heuristic for scheduling problems, which is more time-consuming but more efficient than the local search.

We use the same generator of instances as previous contributions [11, 22]. For a given instance with n jobs, processing times p_j are drawn at random following the uniform distribution $[1; 100]$ and release dates r_j are drawn at random following the uniform distribution $[1; 50.5\, n\rho]$. Parameter ρ enables to generate instances of different difficulties: We consider $\rho \in \{0.2, 0.4, 0.6, 0.8, 1.0, 1.25, 1.5, 1.75, 2.0, 3.0\}$. For each value of n and ρ, N instances are randomly generated leading for a fixed value of n to $10N$ instances. For the learning by demonstration approach, we use the same training set as [22] with $n \in \{50, 70, 90, 110\}$ and $N = 100$, leading to a total of 4000 instances. We do not use larger instances because we do not have access to optimal solutions for larger instances. For the risk minimization approach, we use $n \in \mathcal{N} := \{50, 75, 100, 150, 200, 300, 500, 750, 1000, 1500, 2000, 3000\}$ and $N = 20$, leading to 2400 instances. In the test set, we use a distinct set of 2400 instances with $n \in \mathcal{N}$ and $N = 20$. This test set is almost identical to the one used in the literature [22], the only difference being that instances with $n = 2500$ have been replaced by instance with $n = 3000$ to get a more balanced test set. Table 5 shows how we have partitioned this test set by number of jobs n in the instances, to get sets of instances of moderate, large, and huge size.

6.2.2 Learning algorithm

We use $n(n+1)$ as $u(x)$. It is not an upper bound on the cost, but the cost of the optimal solution scales roughly linearly with $u(x)$. We draw lessons from the stochastic VSP and use only a diverse training set of 4000 instances of all size in the training set. And because our policy for $1|r_j|\sum_j C_j$ is much faster than the one for the stochastic vehicle scheduling problem, we can use a larger training set. And we introduce two new perspectives. First, the DIRECT algorithm uses a random number generator. We observed that its performance is very dependent on the seed of the random number generator, and that using a larger number of iterations does not necessarily compensate for the poor performance that would come from a bad seed. We therefore launch the algorithm with 10 different seeds, each time with a 1000 iterations budget, and report the best result. Second, as underlined in Section 4.3, it can be natural to use the loss ℓ^{ψ} where, instead of using the output of the easy problem, we use the output of the decoder ψ. In our case, the decoder is in two steps: first the local search, second the RDI heuristic. Since RDI is time-consuming, using it would lead to very large computing times on the training set used. We therefore take the solution at the end of the local search.

Table 7 Performance of our solution algorithms with different $\boldsymbol{w}$ on the $1|r_j|\sum_j C_j$ test set.

Learning problem $\boldsymbol{w}$				Test set results (with several ψ)							
				no ψ		LS		RDI ∘ LS		pert RDI ∘ LS	
obj	iter	pert	ψ	δ^{avg}	$\delta^{\max}$	δ^{avg}	$\delta^{\max}$	δ^{avg}	$\delta^{\max}$	δ^{avg}	$\delta^{\max}$
FYL	–	–	–	1.81%	8.57%	1.10%	6.88%	0.07%	3.41%	0.02%	0.46%
ℓ	1000	–	–	0.63%	24.53%	0.34%	4.59%	0.06%	1.65%	0.02%	1.53%
ℓ	2500	–	–	1.08%	21.19%	0.30%	6.33%	0.07%	1.71%	0.04%	1.71%
ℓ^{pert}	1000	100	–	0.83%	23.61%	0.38%	3.64%	0.06%	1.65%	0.03%	1.37%
ℓ^{pert}	2500	100	–	0.75%	19.54%	0.33%	3.98%	0.06%	1.69%	0.02%	1.37%
ℓ	1000	–	LS	10.51%	54.67%	0.02%	1.30%	0.01%	1.12%	0.01%	1.12%
ℓ	2500	–	LS	10.16%	55.70%	0.02%	1.30%	0.01%	1.12%	0.01%	1.12%
ℓ^{pert}	1000	100	LS	10.54%	55.22%	0.03%	2.26%	0.02%	2.26%	0.02%	2.26%
ℓ^{pert}	2500	100	LS	10.51%	53.64%	0.03%	2.26%	0.02%	2.26%	0.02%	2.26%

Table 6 summarizes the results obtained. The first column indicate if the perturbed loss or the non-perturbed loss has been used. The second indicates the number of iterations of DIRECT used. The third column indicates the number of scenarios used in the sample average approximation when the perturbed loss is used. And "–" (resp. LS) in the fourth column indicates if no (resp the local search) decoder has been applied to the solution used in the loss. The column Tot. CPU then provides the total CPU time of the 10 runs of DIRECT with different seeds. Finally, the columns Avg $\widehat{L}$ and Best $\widehat{L}$ give respectively the average and the best loss value of the best solution found by DIRECT algorithm on the 10 seeds used.

We can conclude from these results that optimizing on the seed seems a good idea. We also observe that the loss function after the local search is smaller, which is natural given that the local search improves the solution found by the easy problem.

6.2.3 Algorithm performance on test set

Table 7 summarizes the results obtained with the different $\boldsymbol{w}$ on the full test set. The first line corresponds to the Fenchel young loss (FYL) of the learning by demonstration approach previously proposed [22], and serves as a benchmark. The next eight ones correspond to the parameters obtained solving the learning problem described in this paper with the settings of Table 6. The first four columns describe the parameters of the learning problems used to obtained $\boldsymbol{w}$ and are identical to those of Table 6. The next columns indicate the average results on the full test set for the four kind of decoder described in Section 6.2.1. Again, we provide the average δ^{avg} and the worse δ^{max} values of the gap (27) between the solution found by the algorithm and the best solution found by all the algorithms.

Two conclusions can be drawn from these results:

1. The solution obtained with our risk minimization approach tend to outperform on average those obtained using the Fenchel Young loss, but tend to have a poorer worst case behavior.
2. Using the loss with decoder tend to improve the performance on the test set with the policies that use this preprocessing, and possibly other after. But it decreases the performance on the policy which do not use it.

Finally, Table 8 details the results for the fastest (no ψ) and the most accurate policy (pert RDI ∘ LS) on the subsets of instances of moderate, large, and huge size. In addition to the gaps, the average computing time T^{avg} is provided. Again, we can observe that:

3. Because the risk minimization approach enables to use a diversified set of instances in the training set, it outperforms the learning by demonstration approach on large and huge instances.

6.3 Maximum weight two stage spanning tree

Let us now consider the performance of our policy on the maximum weight two stage spanning tree. All the algorithms are implemented in `julia`.

Training, validation and test sets

We use instances on square grid graphs of width {10, 20, 30, 40, 50, 60}, i.e., with $|V|$ in {100, 400, 900, 1600, 2500, 3600}. First stage weights are uniformly sampled on the integers in $\{-20, \dots, 0\}$. Second stage weights

Table 8 Influence of instances size on the performance of our solution algorithms with different $\boldsymbol{w}$ on the $1|r_j|\sum_j C_j$ test set.

Pred.	$\boldsymbol{w}$				Moderate			Large			Huge		
ψ	obj	iter	$\lvert\Omega\rvert$	ψ	T^{avg}	δ^{avg}	$\delta^{\max}$	T^{avg}	δ^{avg}	$\delta^{\max}$	T^{avg}	δ^{avg}	$\delta^{\max}$
no ψ	FYL	–	–	–	0.01	1.13%	8.57%	0.40	1.80%	6.06%	102.82	2.50%	6.47%
	ℓ	1000	–	–	0.01	1.38%	24.53%	0.20	0.39%	2.33%	25.03	0.11%	0.54%
	ℓ	2500	–	–	0.01	2.59%	21.19%	0.15	0.52%	3.61%	22.85	0.14%	0.96%
	ℓ^{pert}	1000	100	–	0.01	1.51%	23.61%	0.25	0.66%	4.27%	31.95	0.33%	2.23%
	ℓ^{pert}	2500	100	–	0.01	1.25%	19.54%	0.18	0.59%	3.46%	15.91	0.41%	2.00%
	ℓ	1000	–	LS	0.01	10.19%	54.67%	0.07	10.64%	51.23%	2.04	10.70%	46.99%
	ℓ	2500	–	LS	0.01	10.31%	55.70%	0.07	10.24%	50.63%	2.20	9.93%	43.87%
	ℓ^{pert}	1000	100	LS	0.01	10.28%	55.22%	0.07	10.64%	49.59%	2.47	10.70%	46.04%
	ℓ^{pert}	2500	100	LS	0.01	10.01%	53.64%	0.07	10.63%	49.27%	2.45	10.90%	46.36%
pert RDI $\circ$ LS	FYL	–	–	–	0.36	0.02%	0.46%	2.58	0.02%	0.24%	208.72	0.02%	0.16%
	ℓ	1000	–	–	0.48	0.05%	1.53%	2.49	0.02%	0.62%	50.98	0.00%	0.10%
	ℓ	2500	–	–	0.50	0.09%	1.71%	2.44	0.02%	0.34%	45.37	0.00%	0.10%
	ℓ^{pert}	1000	100	–	0.49	0.05%	1.37%	2.61	0.02%	0.66%	65.53	0.00%	0.11%
	ℓ^{pert}	2500	100	–	0.48	0.05%	1.37%	2.54	0.02%	0.61%	40.28	0.00%	0.10%
	ℓ	1000	–	LS	0.48	0.03%	1.12%	2.34	0.01%	0.27%	14.06	0.00%	0.02%
	ℓ	2500	–	LS	0.49	0.03%	1.12%	2.32	0.00%	0.14%	14.19	0.00%	0.01%
	ℓ^{pert}	1000	100	LS	0.48	0.05%	2.26%	2.36	0.01%	0.27%	14.44	0.00%	0.05%
	ℓ^{pert}	2500	100	LS	0.49	0.05%	2.26%	2.39	0.01%	0.27%	14.59	0.00%	0.05%

T^{avg} is given in seconds.

are uniformly sampled on the integers in $\{-K, \dots, 0\}$ with $K \in \{10, 15, 20, 25, 30\}$. Finally, instances have 5, 10, 15 or 20 second stage scenarios. Our training set, validation set, and test set contain 5 instances for each grid width, weight parameter K, and number of scenarios. The training, validation and test sets therefore each contain 600 instances.

Bounds and benchmarks

On each instance of the training, validation, and test set, we solve the Lagrangian relaxation problem using a subgradient descent algorithm for 50,000 iterations, which provides a lower bound on an optimal solution. We also run a Lagrangian heuristic based on the final value of the duals.

We use three benchmarks to evaluate our algorithms: the Lagrangian heuristic, the approximation algorithm of [13], and our policy trained using supervised learning using a Fenchel Young loss to reproduce the solution of the Lagrangian heuristic. Note that since the approximation algorithm, the policy learned using in a supervised way, and the policy learned by risk minimization are all instances of our policy, they take roughly the same time. On the contrary the Lagrangian heuristic requires to solve the 50,000 iterations of the subgradient descent algorithm, and is therefore 4 order of magnitude slower.

Hyperparameters tuning

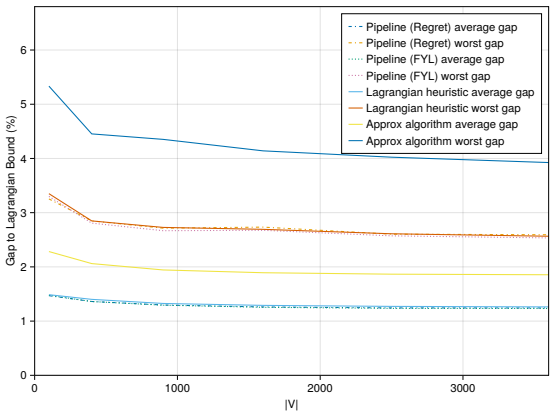
We use a sample average approximation of our perturbed loss ℓ^{pert} with 20 scenarios. Figure 4.a provides the average value of the gap between the solution returned by our policy and the Lagrangian lower bound on the validation set for the model learned with different value of ε. Based on these results, we use $\varepsilon = 0.001$.

Results

Figure 4.b illustrates the average and worst gap with respect to the Lagrangian bound obtained on the test set for our policy and the different benchmarks. Whether it is trained using supervised learning or regret minimization, our policy is able to match the performance of the Lagrangian heuristic, while being four orders of magnitude faster. These three algorithms significantly outperform the approximation algorithm. In summary, our policy trained by risk minimization enables to retrieve the performance of the best algorithms as well as the theoretical guarantee of the approximation algorithm.

ε	Gap
0.0e+00	2.8%
1.0e-04	2.7%
3.0e-04	2.7%
1.0e-03	2.7%
3.0e-03	2.7%
1.0e-02	2.7%
3.0e-02	3.9%
1.0e-01	5.5%
3.0e-01	59.5%

(a)



(b)

Figure 4 Maximum weight spanning tree. (a) Hyperparameters tuning on the validation dataset for the model learned on the training set. (b), Gap with respect to the Lagrangian relaxation bound as a function of the number of vertices $|V|$ with $\varepsilon = 0.001$.

7 Conclusion

We have focused on heuristic algorithms for hard combinatorial optimization problems based on machine learning policies that use a simpler combinatorial optimization problem as a layer in a neural network. Previous contributions in the literature required training sets with instances and their optimal solutions to train such policies. We have shown that the solutions are not necessarily needed, and we can learn such policies by risk minimization if we formulate the learning problem as a risk minimization problem. This widens the potential applications of such methods since it removes the need of an alternative algorithm for the hard problem to build the training set. Furthermore, even when such an algorithm exists, it may not be able to handle large instances. The risk minimization can therefore use larger instances in its training set, and can take into account the effect of potential decoders. These two ingredients enable to scale better on large instances. Finally, we have shown that, if an approximation algorithm can be encoded in the policy with a given parametrization, then the parametrization trained by risk minimization retains the approximation guarantee while giving a more efficient algorithm in practice.

Future contributions may focus on providing richer statistical models in the neural network, which would require to adapt the learning algorithm. Furthermore, the approximation ratio guarantee could be extended to more general settings.

Acknowledgements

I am grateful to Yohann de Castro and Julien Reygner for their help on Section 5, and to Vincent T'Kindt for his help on the scheduling problem. This work has been partially supported by the "Intelligence artificielle pour le transport aérien" chaire between Air-France and École Nationale des Ponts et Chaussées.

A Proof of Theorem 2

A.1 Background on Rademacher complexity and metric entropy method

This section introduces some classical tools of statistical learning theory [8]. The lecture notes of [25] contain detailed proofs.

We place ourselves in the setting of Section 5.1. Let $\mathcal{F}$ be the family of functions $\left\{\xi \mapsto \ell(\xi, \boldsymbol{w})\colon \boldsymbol{w} \in \mathcal{W}\right\}$. The *Rademacher complexity of* $\mathcal{F}$ *is*

$$\mathcal{R}_n(\mathcal{F}) = \mathbb{E}_{\xi_i,\sigma_i}\left[\sup_{\boldsymbol{w}\in\mathcal{W}} \frac{1}{n}\sum_{i=1}^{n} \sigma_i \ell(\xi_i, \boldsymbol{w})\right]$$

where the σ_i are i.i.d. Rademacher variables, i.e., variables equal to 1 with probability $1/2$, and to -1 otherwise. The following well-known result bounds the excess risk based on the Rademacher complexity.

▶ **Proposition 8.** *With probability at least* $1-\delta$*, we have*

$$R(\widehat{\boldsymbol{w}}_n) - R(\boldsymbol{w}^*) \leq 4\mathcal{R}_n(\mathcal{F}) + \sqrt{\frac{2\log(2/\delta)}{n}}.$$

The metric entropy method enables to bound the Rademacher complexity. The *empirical Rademacher complexity* of $\mathcal{F}$ is obtained when we replace the expectation over ξ_i by its values for the training set used $\xi_1, \ldots, \xi_n$.

$$\widehat{\mathcal{R}}_n(\mathcal{F}) = \mathbb{E}\left[\sup_{\boldsymbol{w}\in\mathcal{W}} \frac{1}{n}\sum_{i=1}^n \sigma_i \ell(\xi_i, \boldsymbol{w}) | \xi_1, \ldots, \xi_n\right]$$

and we have $\mathcal{R}_n(\mathcal{F}) = \mathbb{E}[\widehat{\mathcal{R}}_n(\mathcal{F})]$.

Given n instances $\xi_1, \ldots, \xi_2$ and the corresponding distribution $\widehat{\mu}_n$ on Ξ, the pseudometric $L_2(\widehat{\mu}_n)$ on $\mathcal{F}$ is the L_2 norm induced by $\widehat{\mu}_n$ on $\mathcal{F}$

$$\|\ell(\,\cdot\,, \boldsymbol{w}) - \ell(\,\cdot\,, \boldsymbol{w}')\|_{2,\widehat{\mu}_n} = \sqrt{\frac{1}{n}\sum_{i=1}^n \left(\ell(\xi_i, \boldsymbol{w}) - \ell(\xi_i, \boldsymbol{w}')\right)^2}$$

We denote by $B_{\varepsilon, L_2(\widehat{\mu}_n)}(\ell(\,\cdot\,, \boldsymbol{w}))$ the ball of radius ε centered in $\ell(\,\cdot\,, \boldsymbol{w})$. The set covering number of $\mathcal{F}$ with respect to $L_2(\widehat{\mu}_n)$ is

$$N(\varepsilon, \mathcal{F}, L_2(\widehat{\mu}_n)) = \min\left\{m\colon\ \exists\, \{\boldsymbol{w}_1, \boldsymbol{w}_m\} \subseteq \mathbb{R}^d, \mathcal{F} \subseteq \bigcup_{j=1}^m B_{\varepsilon, L_2(\widehat{\mu}_n)}(\ell(\,\cdot\,, \boldsymbol{w}_m))\right\}.$$

The following result bounds the empirical Rademacher complexity from the covering number.

▶ **Proposition 9** (Dudley's theorem)**.** *Let* $\mathcal{F}$ *be a family of mapping from* $\boldsymbol{\eta}$ *to* $[-1, 1]$*, then*

$$\widehat{\mathcal{R}}_n(\mathcal{F}) \leq 12\int_0^\infty \sqrt{\frac{\log N(\varepsilon, \mathcal{F}, L_2(\widehat{\mu}_n))}{n}} \mathrm{d}\varepsilon$$

A.2 Proof of Theorem 2

The proof is as follows. We show that the Gaussian perturbation turns any bounded function in a Lipschitz function. Hence, the perturbed loss is Lipschitz. This implies an upper bound on the covering number, and Dudley's theorem enables to conclude.

Let $\boldsymbol{\eta}$ be a centered standard Gaussian vector on $\mathbb{R}^d$. It is well known that

$$\mathbb{E}(\|\boldsymbol{\eta}\|) \leq \sqrt{d}. \tag{28}$$

Indeed, applying $u \leq (1+u^2)/2$ with $u = \sqrt{\frac{1}{d}\sum \boldsymbol{\eta}_i^2}$ gives $\frac{1}{\sqrt{d}}\|\boldsymbol{\eta}\| \leq \frac{1}{2}(1 + \frac{1}{d}\sum_{i=1}^d \boldsymbol{\eta}_i^2)$. Taking the expectation and using $\mathbb{E}(\boldsymbol{\eta}_i^2) = 1$ gives (28).

▶ **Lemma 10.** *Let* $g : \mathbb{R}^d \to [0, 1]$ *be an integrable function,* $\boldsymbol{\eta}$ *a standard normal random vector on* $\mathbb{R}^d$*,* $\sigma > 0$ *a positive real number, and* $G(\boldsymbol{w}) = \mathbb{E}g(\boldsymbol{w} + \sigma\boldsymbol{\eta})$*. Then* $\boldsymbol{w} \mapsto G(\boldsymbol{w})$ *is* $\frac{\sqrt{d}}{\sigma}$*-Lipchitz.*

Proof. Let h be the density of $\widetilde{\boldsymbol{\eta}} = \sigma\boldsymbol{\eta}$. We have

$$G(\boldsymbol{w}) = \int h(\boldsymbol{z})g(\boldsymbol{z} + \boldsymbol{w}) = \int h(\boldsymbol{z} - \boldsymbol{w})g(\boldsymbol{z})$$

By dominated convergence, we have

$$\nabla G(\boldsymbol{w}) = -\int \nabla h(\boldsymbol{z} - \boldsymbol{w})g(\boldsymbol{z}) = -\int \nabla h(\boldsymbol{z})g(\boldsymbol{z} + \boldsymbol{w})$$

From there, using the facts that $|g(\boldsymbol{w})| \leq 1$ and $\boldsymbol{\eta}$ is a standard Gaussian, we get

$$\|\nabla G(\boldsymbol{w})\| \leq \int \|\nabla h(\boldsymbol{z})\| = \int \|\frac{\boldsymbol{z}}{\sigma^2(\sqrt{2\pi}\sigma)^n} e^{-\frac{\|\boldsymbol{z}\|^2}{2\sigma^2}}\| = \frac{\mathbb{E}(\|\widetilde{\boldsymbol{\eta}}\|)}{\sigma^2} = \frac{1}{\sigma}\mathbb{E}(\|\boldsymbol{\eta}\|) \leq \frac{\sqrt{d}}{\sigma}$$

which gives the result. ◀

Given an arbitrary element ξ in Ξ, Lemma 10 applied with $g = \ell(\xi, \cdot)$ gives

$$|\ell(\xi, \boldsymbol{w}) - \ell(\xi, \boldsymbol{w}')| \leq \frac{\sqrt{d}}{\sigma} \|\boldsymbol{w} - \boldsymbol{w}'\|_2$$

Hence

$$\|\ell(\,\cdot\,, \boldsymbol{w}) - \ell(\,\cdot\,, \boldsymbol{w}')\|_{2,\widehat{\mu}_n} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \big(\ell(\xi_i, \boldsymbol{w}) - \ell(\xi_i, \boldsymbol{w}')\big)^2} \leq \frac{\sqrt{d}}{\sigma} \|\boldsymbol{w} - \boldsymbol{w}'\|.$$

As a consequence, if $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m$ is an $\frac{\varepsilon\sigma}{\sqrt{d}}$ covering of $\mathcal{W}$ endowed with the Euclidean norm, then $\ell(\,\cdot\,, \boldsymbol{w}_1), \ldots, \ell(\,\cdot\,, \boldsymbol{w}_m)$ is an ε covering of $\mathcal{F}$. Hence, if $\mathcal{W}$ is contained in the Euclidean ball of radius M, we get

$$N(\varepsilon, \mathcal{F}, L_2(\widehat{\mu}_n)) \leq N\left(\frac{\varepsilon\sigma}{\sqrt{d}}, \mathcal{W} = \mathbb{B}^d(M), \|\cdot\|_2\right) \leq \left(\frac{M\sqrt{d}}{\varepsilon\sigma}\right)^d$$

for $\varepsilon \leq \frac{M\sqrt{d}}{\sigma}$ and $N(\varepsilon, \mathcal{F}, L_2(\widehat{\mu}_n)) = 1$ otherwise. And we obtain

$$\log N(\varepsilon, \mathcal{F}, L_2(\widehat{\mu}_n)) \leq d\big(\log(M\sqrt{d}/\sigma) - \log\varepsilon\big)$$

for $\varepsilon \leq \frac{M\sqrt{d}}{\sigma}$ and $\log N(\varepsilon, \mathcal{F}, L_2(\widehat{\mu}_n)) = 0$ otherwise.

Proposition 9 then gives

$$\widehat{\mathcal{R}}_n(\mathcal{F}) \leq 12 \int_0^{\frac{M\sqrt{d}}{\sigma}} \sqrt{d\frac{\log(\frac{M\sqrt{d}}{\sigma}) - \log\varepsilon}{n}}\mathrm{d}\varepsilon = 12\sqrt{\frac{d}{n}} \int_0^{\frac{M\sqrt{d}}{\sigma}} \sqrt{-\log\left(\frac{\varepsilon}{M\sqrt{d}/\sigma}\right)}\mathrm{d}\varepsilon = \frac{C}{4}\frac{Md}{\sigma\sqrt{n}}$$

with $C = 48\int_0^1 \sqrt{-\log t}\mathrm{d}t$. It is well known that $\int_0^1 \sqrt{-\log t}\mathrm{d}t = \frac{\sqrt{\pi}}{2}$, we get that $C = 24\sqrt{\pi}$. Note that the bound on $\widehat{\mathcal{R}}_n(\mathcal{F})$ we obtain does not depend on the sample $\xi_1, \ldots, \xi_n$, and is therefore also valid for $\mathcal{R}_n(\mathcal{F}) = \mathbb{E}\big(\widehat{\mathcal{R}}_n(\mathcal{F})\big)$. Proposition 8 then gives Theorem 2.

References

1 Brandon Amos and J. Zico Kolter. OptNet: Differentiable Optimization as a Layer in Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 136–145. PMLR, 2017.

2 Maria-Florina Balcan, Dan DeBlasio, Travis Dick, Carl Kingsford, Tuomas Sandholm, and Ellen Vitercik. How Much Data Is Sufficient to Learn High-Performing Algorithms? Generalization Guarantees for Data-Driven Algorithm Design. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 919–932. ACM Press, 2021.

3 Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *Eur. J. Oper. Res.*, 290(2):405–421, 2021.

4 Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis Bach. Learning with Differentiable Pertubed Optimizers. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

5 Dimitris Bertsimas, Angela King, and Rahul Mazumder. Best Subset Selection Via a Modern Optimization Lens. *Ann. Stat.*, 44(2):813–852, 2016.

6 Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert. Efficient and Modular Implicit Differentiation. In *NIPS'22: 36th International Conference on Neural Information Processing Systems*, pages 5230–5242, 2022.

7 Mathieu Blondel, André F. T. Martins, and Vlad Niculae. Learning with Fenchel–Young Losses. *J. Mach. Learn. Res.*, 21: article no. 35 (69 pages), 2020.

8 Olivier Bousquet, Stéphane Boucheron, and Gábor Lugosi. Introduction to Statistical Learning Theory. In *Advanced Lectures on Machine Learning*, volume 3176 of *Lecture Notes in Computer Science*, pages 169–207. Springer, 2004.

9 Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. `https://arxiv.org/abs/2104.13478`, 2021.

10 Guillaume Dalle, Léo Baty, Louis Bouvier, and Axel Parmentier. Learning with Combinatorial Optimization Layers: A Probabilistic Approach. `https://arxiv.org/abs/2207.13513`, 2022.

11 Federico Della Croce and Vincent T'kindt. A recovering beam search algorithm for the one-machine dynamic total completion time scheduling problem. *J. Oper. Res. Soc.*, 53(11):1275–1280, 2002.

12 Adam N. Elmachtoub and Paul Grigas. Smart "Predict, Then Optimize". *Manag. Sci.*, 68(1):9–26, 2021.

13 Bruno Escoffier, Laurent Gourvès, Jérôme Monnot, and Olivier Spanjaard. Two-Stage Stochastic Matching and Spanning Tree Problems: Polynomial Instances and Approximation. *Eur. J. Oper. Res.*, 205(1):19–30, 2010.

14 Steven G. Johnson. The NLopt Nonlinear-Optimization Package. `http://github.com/stevengj/nlopt`, Accessed on 2021-07-04.

15 D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian Optimization without the Lipschitz Constant. *J. Optim. Theory Appl.*, 79(1):157–181, 1993.

16 James Kotary, Ferdinando Fioretto, Pascal Van Hentenryck, and Bryan Wilder. End-to-End Constrained Optimization Learning: A Survey. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, pages 4475–4482. International Joint Conferences on Artificial Intelligence Organization, 2021.

17 Jayanta Mandi, Peter J Stuckey, Tias Guns, et al. Smart predict-and-optimize for hard combinatorial optimization problems. In *Proceedings of the AAAI Conference on Artificial Intelligence 34(02)*, pages 1603–1610, 2020.

18 Ruben Martinez-Cantin. BayesOpt: A Bayesian Optimization Library for Nonlinear Optimization, Experimental Design and Bandits. *J. Mach. Learn. Res.*, 15:3915–3919, 2014.

19 Alex Nowak-Vila, Francis Bach, and Alessandro Rudi. A general theory for structured prediction with smooth convex surrogates. `https://arxiv.org/abs/1902.01958`, 2019.

20 Sebastian Nowozin. Structured Learning and Prediction in Computer Vision. *Found. Trends Comput. Graph. Vision*, 6(3-4):185–365, 2010.

21 Axel Parmentier. Learning to Approximate Industrial Problems by Operations Research Classic Problems. *Oper. Res.*, 70(1):606–623, 2022.

22 Axel Parmentier and Vincent T'kindt. Structured learning based heuristics to solve the single machine scheduling problem with release times and sum of completion times. *Eur. J. Oper. Res.*, 305(3):1032–1041, 2023.

23 Michael L Pinedo. *Scheduling. Theory, algorithms, and systems*. Springer, 4th edition, 2012.

24 Marin Vlastelica Pogančić, Anselm Paulus, Vit Musil, Georg Martius, and Michal Rolinek. Differentiation of Blackbox Combinatorial Solvers. In *International Conference on Learning Representations*, 2020. `https://openreview.net/forum?id=BkevoJSYPB`.

25 Michael M. Wolf. Mathematical Foundations of Supervised Learning, 2018. `https://www-m5.ma.tum.de/foswiki/pub/M5/Allgemeines/MA4801_2018S/ML_notes_main.pdf`.