



HAL
open science

Learning structured approximations of operations research problems

Axel Parmentier

► **To cite this version:**

Axel Parmentier. Learning structured approximations of operations research problems. 2021. hal-03281894v1

HAL Id: hal-03281894

<https://hal.science/hal-03281894v1>

Preprint submitted on 8 Jul 2021 (v1), last revised 27 Jan 2023 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Learning structured approximations of operations research problems.

Axel Parmentier

CERMICS, Ecole des Ponts, Marne-la-Vallée, France

axel.parmentier@enpc.fr

July 8, 2021

Abstract

The design of algorithms that leverage machine learning alongside combinatorial optimization techniques is a young but thriving area of operations research. If trends emerge, the literature has still not converged on the proper way of combining these two techniques or on the predictor architectures that should be used. We focus on operations research problems for which no efficient algorithms are known, but that are variants of classic problems for which ones efficient algorithm exist. Elaborating on recent contributions that suggest using a machine learning predictor to approximate the variant by the classic problem, we introduce the notion of structured approximation of an operations research problem by another. We provide a generic learning algorithm to fit these approximations. This algorithm requires only instances of the variant in the training set, unlike previous learning algorithms that also require the solution of these instances. Using tools from statistical learning theory, we prove a result showing the convergence speed of the estimator, and deduce an approximation ratio guarantee on the performance of the algorithm obtained for the variant. Numerical experiments on a single machine scheduling and a stochastic vehicle scheduling problem from the literature show that our learning algorithm is competitive with algorithms that have access to optimal solutions, leading to state-of-the-art algorithms for the variant considered.

Keywords. Machine learning for combinatorial optimization, structured approximation, approximation algorithm, stochastic vehicle scheduling problem, single machine scheduling problem.

1 Introduction

[Bengio et al. \[2021\]](#) survey the young but thriving literature on operations research algorithms that leverage machine learning. Part of the literature uses machine learning to improve existing algorithms. For instance, recent contributions learn to tune solvers hyperparameters

or to take heuristic decision within OR algorithms, and notably branching decision in mixed integer linear programming (MILP) solvers. A growing literature also aims at building entirely new algorithms with machine learning at their core [Bello et al., 2017, Kool et al., 2018, Vinyals et al., 2015]. Bengio et al. [2021] call them end-to-end learning algorithms. They are typically in two steps. A machine learning predictor φ_{θ} extracts relevant information on the instance solved, and a constructive heuristic then rebuild a solution based on the information extracted.

Practically, φ_{θ} is chosen in a parametrized family $\{\varphi_{\theta}: \theta \in \Theta\}$. The goal is to have a flexible algorithm that can be tailored to an application of interest by choosing a suitable θ . The idea is to use the available data to “learn” θ . Practically, this data is used to reformulate the problem of choosing θ as an optimization problem called the *learning problem*. We call the algorithm parametrized by the learned θ the *learned algorithm*. The learning problem is solved only once offline, which allows the use of computationally intensive methods. Parameter θ is then known and the learned algorithm can then be used online to find quickly solution on new instances. Such methods therefore enable to deport offline most of the computing time.

On many classic operations research problems, these end-to-end learning based algorithms are still largely outperformed by efficient algorithms on which the community has converged after decades of research, because the constructive heuristic does not exploit the structure of the problem as well as the specialized algorithm. However, such machine learning algorithms remain appealing because they are flexible. Indeed, if the specialized operations research algorithm are generally very efficient on the problem they are designed for, they often cannot be applied to variants. On the contrary, constructive heuristics are easy to adapt to many variants of an operations research problem because they are simple. And the learning problem enables to customize them for a specific variant, making them reasonably efficient.

The author recently proposed a structured learning approach [Parmentier, 2021] with the objective of building algorithms as generic as the constructive heuristics and as efficient as the specialized algorithm. Suppose that we are interested in solving a hard operations research problem

$$\min_{x \in \mathcal{X}(\Gamma^h)} f^h(x; \Gamma^h) \tag{h}$$

that is variant of a classic operations research problem

$$\min_{x \in \mathcal{X}(\Gamma^e)} f^e(x; \Gamma^e) \tag{e}$$

for which a practically efficient algorithm \mathcal{A}^e is available in the literature. The letters h and e stand for hard and easy. Γ^h and Γ^e respectively denote the instances considered. We indicate the instance Γ in the definition of the objective function and the set of admissible solution because several instances of the same problem will be considered simultaneously in the learning algorithm. The approach proposed is illustrated on Figure 1. A machine learning predictor φ_{θ} is used to encode the hard problem instance Γ^h into an instance of Γ^e . Algorithm \mathcal{A}^e then finds an optimal solution x^e . Finally, a decoding algorithm ψ is

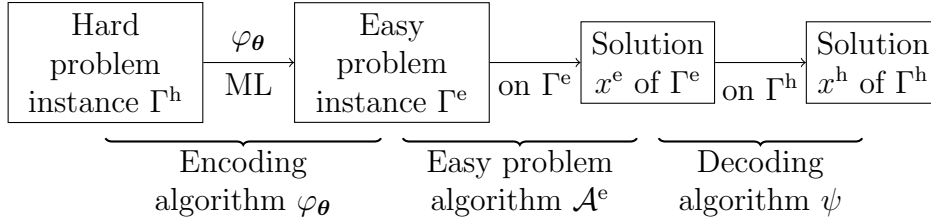


Figure 1: ML to approximate hard problems by well-solved ones

used to rebuild a solution x^h for Γ^h from x^e . The encoder φ_θ is a statistical predictor and its computation is very fast. The easy problem (e) is chosen because even large instances are tractable with a (potentially advanced) algorithm \mathcal{A}^e from the literature. For instance, \mathcal{A}^e can involve the resolution of a tractable MILP formulation with a solver. Finally, the decoder ψ is not mandatory and is typically a descent heuristic. Hence, the learned algorithm is sufficiently fast to be applied online (on a single instance). And it can be applied many times (~ 1000 times) to several instances during the CPU intensive but offline learning phase.

Previous contributions [Parmentier, 2021, Parmentier and T’Kindt, 2021] show the practical advantage of this approximation paradigm: appropriate φ_θ and (e) retain most of the structure of (h) and make the hard problem instance tractable with \mathcal{A}^e . And their numerical experiments show that the learned algorithm is practically efficient and that it meets the scaling challenge [Bengio et al., 2021, Section 6.2]. But they underline two challenges that must be met to make the approach work. First, we must design φ_θ in such a way that the solution returned has a small objective $f^h(x^h, \Gamma^h)$. And second, we must formulate the learning problem and propose a learning algorithm. The initial contributions illustrate on specific applications how to build and learn approximations, but do not provide a generic method to address these challenges. In this paper, we formalize the notion of structured approximation of one problem by another, propose generic methods to design and learn such approximations, provide theoretical guarantees on their performances, and show their practical performance through numerical experiments.

We make the following contributions:

1. We formalize the notion of *structure* of a problem in order to introduce the notion of *structured approximation* of an operations research problem (h) by another one (e). Practically, such an approximation of (h) is given by an easy problem (e) sharing the same structure, as well as an encoder φ_θ and a decoder ψ that satisfy several properties.
2. We provide classifier architectures that mutualize the use of several generalized linear models within φ_θ , which enables to obtain flexible models with a parameter vector θ of moderate dimension (≤ 100).
3. We introduce a formulation of the learning problem and algorithms to solve it. We expect them to work on any structured approximation such that the dimension of θ is moderate (≤ 100) and the solution pipeline can be applied several hundred to several

thousand times during the learning phase. Contrarily to previous formulations of the learning problem which required a training set containing instances of (h) as well as an optimal (or at least a good) solution of each instance, this formulation requires only instances of (h) but no solution. This is a practical advantage because no algorithm for the hard problem is required.

4. We demonstrate the practical performance of our learning algorithm on the applications of the literature [Parmentier, 2021, Parmentier and T’Kindt, 2021]: Our learned algorithm (which has no access to optimal solutions of instances in the training set) matches the performance of those of the literature (which have access to optimal solutions), and are therefore state-of-the-art heuristics for the problems considered.

To the best of our knowledge, the literature on end-to-end learning algorithms for combinatorial optimization problems has focused on the practical efficiency of these heuristics and no theoretical guarantees on the learning algorithms have been proposed. More precisely, convergence results for the estimator of θ they use may exist in the statistical learning literature, but the consequences of these results in terms guarantees on the optimality gap of the solution returned by the learned algorithm have not been studied. Since we propose structured approximations, it is natural to wonder if and at which speed our learning algorithm converges towards the best approximation.

5. Leveraging tools from statistical learning theory, we prove the convergence of the learning algorithm toward the approximation with the best expected loss, and an upper bound on the convergence speed that does not depend on the diversity instances structures.
6. We also prove that, if φ_θ is sufficiently regular and there exists a parameter $\tilde{\theta}$ such that the easy problem objective approximate well the hard problem objective, then with high probability on the training sample, the learned algorithm is an approximation algorithm for the hard problem (h) whose approximation ratio improves with the size of the training set.

The paper is organized as follows. Section 2 introduces the easy and hard problems of the literature [Parmentier, 2021, Parmentier and T’Kindt, 2021] that will be used as running examples in the paper. Section 3 formalizes the notion of structured approximation and provide a practical guide on how to build such approximations. Section 5 details the theoretical analysis of the learning algorithm and the convergence to an approximation algorithm. Finally, Section 6 provides extensive numerical results showing the performance of the learning algorithm on instances from the literature.

2 Easy and Hard problems used as running examples

This section introduces two hard problems (h) considered in previous contributions [Parmentier, 2021, Parmentier and T’Kindt, 2021], as well as the easy problem (e) used to

approximate them. We will use them as running examples of structured approximation and as benchmark in the numerical experiments. The problem description follows these papers.

2.1 Stochastic vehicle scheduling problem.

Hard problem: stochastic vehicle scheduling problem Let V be a set of tasks that should be operated using vehicles. For each task v in V , we suppose to have a scheduled begin time t_v^b in \mathbb{Z}_+ and a scheduled end time t_v^e in \mathbb{Z}_+ . We suppose $t_v^e > t_v^b$ 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 $t_{(u,v)}^{\text{tr}}$. Task v can be operated after task u using the same vehicle if

$$t_v^b \geq t_u^e + t_{(u,v)}^{\text{tr}}. \quad (1)$$

We introduce the digraph $D = (V, A)$ with vertex set $V = T \cup \{o, d\}$ where o and d are dummy 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 $c(P, \Gamma^h)$ 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.

$$\min_z \sum_{P \in \mathcal{P}_{od}} c(P; \Gamma^h) z_P, \quad (2a)$$

$$\text{s.t.} \sum_{P \ni v} z_P = 1, \quad \forall v \in V(\Gamma^h) \setminus \{o, d\}, \quad (2b)$$

$$z_P \in \{0, 1\}, \quad \forall P \in \mathcal{P}_{od}(\Gamma^h), \quad (2c)$$

Let Ω be a set of scenarios. For each task v , we have a random begin time ξ_v^b and a random end time ξ_v^e , and for each arc (u, v) , we have a random travel time $\xi_{(u,v)}^{\text{tr}}$. Hence, $\xi_v^b(\omega)$, $\xi_v^e(\omega)$, and $\xi_{(u,v)}^{\text{tr}}(\omega)$ 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 $\xi_o^e = 0$ and $\xi_d^b = +\infty$.

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

$$\tau_P = \begin{cases} 0, & \text{if } P \text{ is the empty path in } o, \\ \xi_v^e + \max(\tau_Q + \xi_a^{\text{tr}} - \xi_v^b, 0), & \text{if } P = Q + a \text{ for some path } Q \text{ and arc } a. \end{cases} \quad (3)$$

Equation (3) 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 $\tau_Q + \xi_a^{\text{tr}}$ with delay $\max(\tau_Q + \xi_a^{\text{tr}} - \xi_v^b, 0)$. The total delay Δ_P along a path P is therefore defined recursively by

$$\Delta_P = \begin{cases} 0, & \text{if } P \text{ is the empty path in } o, \\ \Delta_Q + \max(\tau_Q + \xi_a^{\text{tr}} - \xi_v^b, 0), & \text{if } P = Q + a \text{ for some path } Q \text{ and arc } a. \end{cases} \quad (4)$$

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

$$c(P) = c^{\text{veh}} + c^{\text{del}} \mathbb{E}(\tau_P) \quad (5)$$

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.

Easy problem: usual vehicle scheduling problem The usual vehicle scheduling problem can also be formulated as (2), the difference being that now the path can be decomposed as the sum of the arcs cost

$$\bar{c}_P = \sum_{a \in P} \bar{c}_a \quad \text{with} \quad \bar{c}_a \in \mathbb{R}. \quad (6)$$

It can be reduced to a flow problem on D and efficiently solved using flow algorithms or linear programming. In Equation (6) and in the rest of the paper, we use an overline to denote quantities corresponding to the easy problem.

2.2 Single machine scheduling problem.

Hard problem: $1|r_j|\sum_j C_j$. n jobs must be processed in 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} . A solution is a schedule $s = (j_1, \dots, j_n)$, i.e., a permutation of $[n]$ that gives the order in which jobs are processed. Using the convention $C_{j_0} = 0$, the completion time of jobs in s are defined as

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

The objective is to find a solution minimizing $\sum_j C_j$. This problem is strongly NP-hard.

Easy problem: $1||\sum_j C_j$. The easy problem is obtained when there is no release time, and only jobs processing times \bar{p}_j . Jobs completion times are therefore given by

$$\bar{C}_{j_i} = \bar{C}_{j_{i-1}} + \bar{p}_{j_i}.$$

Again, we use an overline to denote quantities of the easy problem. An optimal schedule is obtained using the shortest processing time first (SPT) rule, that is, by sorting the jobs by increasing p_j .

3 Structured approximations of Operations Research problems

We can now introduce the notion of structured approximation of a problem by another, and illustrate it on the examples of the previous section.

3.1 Instances structure and parameters.

To that end, we begin with a definition of the structure of an operations research problem. Consider a generic operations research problem

$$\min_{x \in \mathcal{X}(\mathcal{E})} f(x, \Gamma) \quad \text{with} \quad \Gamma = (\mathcal{E}, \boldsymbol{\varrho}).$$

Problem	Structure \mathcal{E}	Parameters $\boldsymbol{\varrho}$
Shortest path	Digraph $D = (V, A)$	Arc costs c_a
Spanning tree	Graph $G = (V, E)$	Edge weights w_a
$1 r_j \sum_j C_j$	Tasks $([n])$	$(r_j, p_j)_{j \in [n]}$
$1 \sum_j C_j$	Tasks $([n])$	$(p_j)_{j \in [n]}$

Table 1: Structure and parameters of several operations research problems.

An instance Γ is a pair $(\mathcal{E}, \boldsymbol{\varrho})$, with \mathcal{E} the *structure* of Γ and $\boldsymbol{\varrho}$ its vector of *parameters*. The structure is a finite collection $\mathcal{E} = \{E_k: k \in K\}$ containing several finite sets E_k called *substructures*. For each k in K , we have a *parameter space* $R_k \subseteq \mathbb{R}^{d_k}$. And for each k in K and $e \in E_k$, we have a vector of parameters $\boldsymbol{\rho}_e \in R_k$. And we denote by $\mathcal{R}(\mathcal{E})$ the set $\prod_{k \in K} R_k^{E_k}$ to which belongs the parameter $\boldsymbol{\varrho}$ of any instance with structure \mathcal{E} . We suppose that the *set of solutions* $\mathcal{X}(\mathcal{E})$ of Γ depends only on its structure. This does not prevent us from modeling parameter dependent constraints within the *objective function* f , that is equal to $+\infty$ when such a constraint is violated.

For instance, if the problem considered is a *b-flow* problem, an instance of the problem is given by a digraph $D = (V, A)$, a vector $(b_v)_v \in \mathbb{R}^V$ giving the inflows-outflows at each vertex v , and vectors $(\ell_a)_{a \in A} \in \mathbb{R}_+^A$, $(u_a)_{a \in A} \in \mathbb{R}_+^A$, and $(c_a)_{a \in A} \in \mathbb{R}^A$ giving arcs lower capacities, upper capacities, and costs. In that case, we can use $K = \{1, 2\}$, $E_1 = V$, $E_2 = A$, $\boldsymbol{\rho}_v = (b_v)$, and $\boldsymbol{\rho}_a = (\ell_a, u_a, c_a)$.

The substructures index set K and parameter spaces $R_k \subseteq \mathbb{R}^{d_k}$ are properties of the problem: They do not change from one instance to another. For the *b-flow* example, $R_1 = \mathbb{R}$ while $R_2 = \mathbb{R}_+^2 \times \mathbb{R}$. On the contrary, the sets E_k change and typically have different cardinal from one instance to another. When the instance is not clear from context, we use the notation $E_k(\Gamma^h)$. The number of vertices and of arcs of a *b-flow* problem change from one instance to another. Table 1 provides the structures \mathcal{E} and parameters $\boldsymbol{\varrho}$ of some classic operations research problems.

Consider the stochastic vehicle scheduling problem of Section 2.1. The structure \mathcal{E} is given by the digraph $D = (V, A)$. And the parameters by $\boldsymbol{\rho}_v^h = (\xi_v^b, \xi_v^e)$ and $\boldsymbol{\rho}_a^h = \xi_a^{\text{tr}}$, which we can store as vectors containing the value taken by these random variables for each scenario. We approximate it by the usual vehicle scheduling problem on the same structure $D = (V, A)$, but where vertices no more have parameters, and arcs have parameters $\boldsymbol{\rho}_a^e = (\bar{c}_a)$. The structure \mathcal{E} of the scheduling problem $1|r_j|\sum_j C_j$ of Section 2.2 is the set of jobs $[n]$. The parameter vector of a job j is $\boldsymbol{\rho}_j^h = (r_j, p_j)$. The easy problem $1||\sum_j C_j$ has the same structure, but the parameter of a job is then $\boldsymbol{\rho}_j^e = (\bar{p}_j)$.

3.2 Structured approximations

We suggest approximating hard problem instances Γ^h with structure \mathcal{E} and parameters $\boldsymbol{\varrho}^h$ by instances of an easy problem with the same structure \mathcal{E} , the same solutions $\mathcal{X}(\mathcal{E})$ but

simpler parameters $\boldsymbol{\varrho}^e$ and objective function.

$$\text{Approximate } \min_{x \in \mathcal{X}(\mathcal{E})} f^h\left(x, \underbrace{(\mathcal{E}, \boldsymbol{\varrho}^h)}_{\Gamma^h}\right) \text{ by } \min_{x \in \mathcal{X}(\mathcal{E})} f^e\left(x, \underbrace{(\mathcal{E}, \boldsymbol{\varrho}^e)}_{\Gamma^e}\right) \quad (7)$$

Let \aleph^h and \aleph^e denote the sets of instances of (h) and (e). In practice, we use the encoder

$$\begin{aligned} \varphi_{\boldsymbol{\theta}} : \quad \aleph^h &\longrightarrow \aleph^e \\ \Gamma^h = (\mathcal{E}, \boldsymbol{\varrho}^h) &\longmapsto (\mathcal{E}, \boldsymbol{\varrho}^e) \quad \text{with } \boldsymbol{\varrho}^e = \tilde{\varphi}_{\boldsymbol{\theta}}(\Gamma^h) \in \mathcal{R}^e(\Gamma^h) \end{aligned} \quad (8)$$

and want to learn the predictor $\tilde{\varphi}_{\boldsymbol{\theta}}$ giving the parameters $\boldsymbol{\varrho}^e$ of the approximation $\Gamma^e = (\mathcal{E}, \boldsymbol{\varrho}^e)$ of an instance $\Gamma^h = (\mathcal{E}, \boldsymbol{\varrho}^h)$.

We now explain how to build $\tilde{\varphi}_{\boldsymbol{\theta}}$. The main difficulty comes from the fact that the size of the output $\varphi_{\boldsymbol{\theta}}(\Gamma^h)$ changes from one instance to another. We handle it by using a predictor $\varphi_{\boldsymbol{\theta}_k, k}$ for each substructure E_k in \mathcal{E} .

$$\begin{aligned} \varphi_{\boldsymbol{\theta}_k, k} : \quad P_k &\rightarrow R_k \\ (e, \Gamma^h) &\mapsto \boldsymbol{\rho}_e^e \quad \text{with } P_k = \left\{ (e, \Gamma^h) : e \in E_k(\Gamma^h), \Gamma^h \in \aleph^h \right\} \end{aligned} \quad (9)$$

The predictor $\varphi_{\boldsymbol{\theta}_k, k}$ takes in input an instance $\Gamma^h = (\mathcal{E}, \boldsymbol{\varrho}^h)$ and an element e of E_k , and returns the parameter $\boldsymbol{\rho}_e^e$ of e in the instance $\varphi_{\boldsymbol{\theta}}(\Gamma^h)$. The full predictor is obtained by applying the substructure predictors to each element of the structures.

$$\tilde{\varphi}_{\boldsymbol{\theta}}(\Gamma^h) = \boldsymbol{\varrho}^e \quad \text{with } \boldsymbol{\varrho}^e = \left((\varphi_{\boldsymbol{\theta}_k, k}(e, \Gamma^h))_{e \in E_k} \right)_{k \in K} \quad (10)$$

and its parameter vector $\boldsymbol{\theta} = (\boldsymbol{\theta}_k)_{k \in K} \in K$ is the concatenation of the substructures predictors parameters. In summary, we handle variable size output by using a shared predictor for all the elements of a substructure E_k . We can now define a *structured approximation* of a problem (h) by a problem (e) to be an encoder $\varphi_{\boldsymbol{\theta}}$ such that Equations (7), (8), (9), and (10) are satisfied.

Figure 2 illustrates the full approximation on our two running examples. In learning algorithms, it is handfull to consider all the elements of substructures together. We therefore denote by $e \in \mathcal{E}$ the fact that $e \in E_k$ for some $k \in K$, and by $\bar{\varphi}_{\boldsymbol{\theta}}(e, \Gamma^h)$ the quantity $\varphi_{k, \boldsymbol{\theta}_k}(e, \Gamma^h)$.

3.3 Building structured approximations: Encoding instances information

A second difficulty comes from the fact that $\varphi_{\boldsymbol{\theta}}$ takes in input instances Γ^h of different sizes, while machine learning predictors generally take in input vectors of fixed size. We therefore need to define, for each substructure $k \in K$, an encoding of the information on (e, Γ^h) for e in E_k .

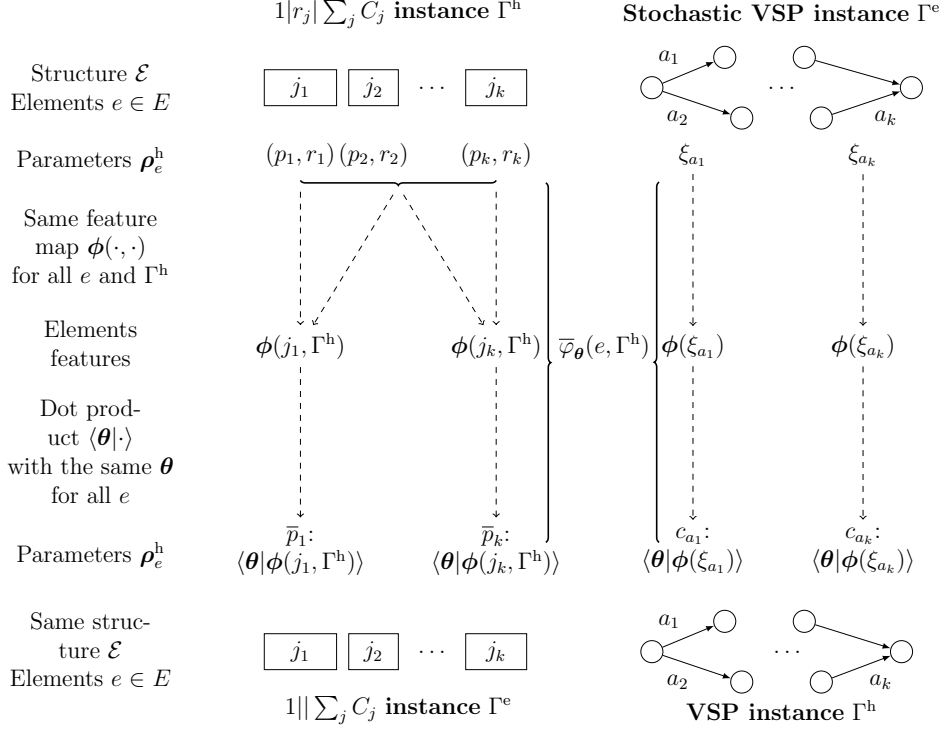


Figure 2: Examples of encoding using linear predictors $\bar{\varphi} : (e, \Gamma^h) \mapsto \langle \theta | \phi(e, \Gamma^h) \rangle$

Generalized linear model A simple approach is to use a generalized linear model. In such an approach, for each substructure k , we build a feature mapping

$$\phi_j : (e, \Gamma^h) \longrightarrow \phi_j(e, \Gamma^h)$$

that maps (e, Γ^h) to a feature space F_k , which will typically be \mathbb{R}^{f_k} . The predictor then makes a linear combination of the components of $\phi_i(e, \Gamma^h)$.

$$\rho_e^e = \varphi_{\theta_k, k}(e, \Gamma^h) = \theta_k^T \phi(e, \Gamma^h) \quad (11)$$

In Equation (11), the parameter θ_k can be seen as a matrix in $\mathbb{R}^{d_k \times f_k}$. When ρ_e^e is in \mathbb{R} , it is a simple dot product $\langle \theta_k | \phi(e, \Gamma^h) \rangle$, which is the case for our running examples illustrated on Figure 2.

On a given problem, we still need to define the different features f in $[f_k]$. The simplest approach is to define $\phi_f(e, \Gamma^h)$ as a function of ρ_e^e only. For instance, on the stochastic VSP, the parameter ρ_a^e of an arc $a = (u, v)$ is the random travel time ξ_a^{tr} . We can for instance take as a feature $\phi(a, \Gamma^h)$ its median. Such features that consider solely an element parameter ρ_e^h are unfortunately blind to the relations between the different elements of an instance. For instance, on the stochastic VSP, the slack $\xi_v^b - (\xi_u^e + \xi_a^{\text{tr}})$ between u and v , i.e., the time that a vehicle operating u and then v if there was no delay propagating gives a much more interesting information on how promising a is, and we can take its quantiles as features [Parmentier, 2021].

Encoding information on an element as part of an instance We now introduce two generic techniques that enable to take such relations between an element and the rest of the structure into account. The first technique enables to *compare* ρ_e^h to $\rho_{e'}^h$ for $e' \in E_k$. It defines a statistic $f : \mathbb{R}^{d_e} \rightarrow \mathbb{R}$, and consider $f(\rho_e^h)$ as a realization of the random variable

$$\begin{aligned} F : E_k &\rightarrow \mathbb{R} \\ e' &\mapsto f(\rho_{e'}^h) \end{aligned}$$

and take some relevant statistics on the realization $F(e)$ of F , such as the value of the cumulative distribution function of F in $f(\rho_j^h)$. For instance, when considering a job j of $1/|r_j| \sum_j C_j$ with parameter $\rho_j^h = (r_j, p_j)$, we can define $f(\rho_j^h) = r_j + p_j$, we obtain as feature the rank (divided by n) of feature j in the schedule where we sort the jobs by increasing $r_j + p_j$, a statistic known to be interesting and used in dispatching rules.

The second technique is to *explore the role of e in the solution of a very simple optimization problem*. A natural way of building features is to run a fast heuristic on the instance Γ^h and seek properties of e 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.

Features free encoding Of course, more advanced model could be used instead of generalized linear models, and we believe that using kernel methods or deep neural networks to build these classifiers $\varphi_{\theta_k, k}$ is a promising direction. The design of deep neural networks adapted to combinatorial optimization problems is indeed an active area of research [Bengio et al., 2021, Joshi et al., 2019, Vinyals et al., 2015]. When the structure is a graph, using a graph convolutional network [Kipf and Welling, 2017] seems natural. However, the learning algorithm proposed in the next section is tailored for model with parameter θ of moderate dimension, typically ≤ 100 . As deep neural networks typically have orders of magnitude more parameters, they would require another learning algorithm.

3.4 Discussion

The structured approximations introduced in this section have two main properties. First, they can be applied on instances of very different size. And second, elements E_k of a given structure share the same predictors $\varphi_{\theta_k, k}$, and hence the same parameters θ_k . This enables to handle large instances with moderate size parameter vector θ , as we illustrated in Section 3.3. Let us now see how these properties can be exploited in the learning phase.

4 Learning structured approximations without optimal solutions

Having defined these structured approximations, we now focus on the problem of learning such approximations. Given a training set composed of representative instances, the learning problem aims at finding a parameter θ such that $\varphi_\theta(\Gamma^h)$ is a good approximation of Γ^h .

4.1 Learning problem and regularized learning problem

Let $\Gamma_1^h, \dots, \Gamma_n^h$ be our *training set* composed of n instances of the learning problem. We place ourselves in the setting of Section 3: $\Gamma^h = (\mathcal{E}, \boldsymbol{\varrho}^h)$ and $\Gamma^e = (\mathcal{E}, \boldsymbol{\varrho}^e)$ with $\boldsymbol{\varrho}^e = \boldsymbol{\varphi}_\theta(\Gamma^h)$ have the same set of solutions $\mathcal{X}(\mathcal{E})$. Without loss of generality, we suppose that $f(x; \Gamma^h) \geq 0$ for all instances $\Gamma^h = (\mathcal{E}, \boldsymbol{\varrho}^h)$ and feasible solution $x \in \mathcal{X}(\mathcal{E})$. We also suppose to have a mapping $u : \mathcal{E} \mapsto u(\mathcal{E}) > 0$ that is a coarse estimation of the value of an optimal solution of an instance with structure \mathcal{E} . We define the loss function as the weighted cost of the easy problem solution as a solution of the hard problem.

$$\ell(\boldsymbol{\theta}, \Gamma^h) := \frac{1}{u(\mathcal{E})} \min \left\{ f^h(x; \Gamma^h) : x \in \arg \min_{x \in \mathcal{X}(\mathcal{E})} f^e(x; \boldsymbol{\varphi}_\theta(\Gamma^h)) \right\}. \quad (12)$$

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

$$\min_{\boldsymbol{\theta} \in \Theta} \frac{1}{n} \sum_{i=1}^n \ell(\boldsymbol{\theta}, \Gamma_i^h). \quad (13)$$

The instances in the training set may be of different size, leading to solutions costs which different order of magnitudes. The weight $\frac{1}{u(\mathcal{E})}$ enables to avoid giving too much importance to large instances.

When the approximation is flexible and the training set is small, the solution of (13) 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 of $\boldsymbol{\theta}$: We want the solution returned to be good even if we use $\boldsymbol{\theta} + \mathbf{Z}$ instead of $\boldsymbol{\theta}$, where \mathbf{Z} is a small perturbation. Practically, we assume that \mathbf{Z} is a standard Gaussian, $\sigma > 0$ is a real number, and we define the *perturbed loss*

$$\ell^{\text{pert}}(\boldsymbol{\theta}, \Gamma^h) = \mathbb{E}_{\mathbf{Z}} \ell(\boldsymbol{\theta} + \sigma \mathbf{Z}, \Gamma^h). \quad (14)$$

The *regularized learning problem* is then formulated as follows.

$$\min_{\boldsymbol{\theta} \in \Theta} \frac{1}{n} \sum_{i=1}^n \ell^{\text{pert}}(\boldsymbol{\theta}, \Gamma_i^h) \quad (15)$$

Remark 1. If we are ready to increase the CPU time, the perturbation of $\boldsymbol{\theta}$ by \mathbf{Z} can also be used to increase the quality of the solution returned by our solution pipeline. We can draw several samples \mathbf{Z}_i of \mathbf{Z} , apply the solution pipeline with $\boldsymbol{\theta} + \sigma \mathbf{Z}_i$ instead of $\boldsymbol{\theta}$, 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. \triangle

4.2 Algorithms to solve the learning problem

Proposition 1. *Given $\Gamma^h = (\mathcal{E}, \boldsymbol{\varrho}^h)$, if $\boldsymbol{\theta} \mapsto \bar{\varphi}_\theta(e, \Gamma^h)$ is piecewise linear for any e in \mathcal{E} , $\boldsymbol{\rho}^h$ and Γ^h , and $\boldsymbol{\varrho}^e \mapsto f^e(x, (\mathcal{E}, \boldsymbol{\varrho}^e))$ is piecewise linear for all x, \mathcal{E} , then the objective of (13) is piecewise constant in Γ^h .*

Proof. Since the composition of two piecewise linear functions is piecewise linear, $\boldsymbol{\theta} \mapsto f^e(x; \mathcal{E}, \varphi_{\boldsymbol{\theta}}(\Gamma^h))$ is piecewise linear. Hence, there exists a partition of the space into a finite number of polyhedra such that the set $\left\{ f^h(x; \Gamma^h) : x \in \arg \min_{x \in \mathcal{X}(\mathcal{E})} f^e(x; \varphi_{\boldsymbol{\theta}}(\Gamma^h)) \right\}$ is constant on each polyhedron. The definition of $\ell(\boldsymbol{\theta}, \Gamma^h)$ then ensures that $\boldsymbol{\theta} \mapsto \ell(\boldsymbol{\theta}, \Gamma^h)$ is piecewise constant on the interior of each polyhedron of the partition, and lower semi-continuous, which gives the result. \square

Proposition 1 is bad news from an optimization point of view. We need a derivative free optimization algorithm that uses a moderate amount of function evaluations and does not rely on “slope”. We therefore suggest using either a heuristic that searches the state space such as the DIRECT algorithm [Jones et al., 1993], or a Bayesian optimization algorithm that builds a global approximation of the objective function and use it to sample the space in $\boldsymbol{\theta}$ 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 (13). Since the convolution product of two functions is as regular as the most regular of the two functions, $\boldsymbol{\theta} \mapsto \ell^{\text{pert}}(\boldsymbol{\theta}, \Gamma^h)$ is C^∞ in that case. In theory, it can therefore be solved using a stochastic gradient descent. We tested it and obtained poor results, which 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. We obtained much better results by solving a sample average approximation of this perturbed learning problem using the algorithms of Section 4.2.

4.3 Practical remarks for a generic implementation

Perturbation strength. Section 5 provides a closed formula to set the perturbation strength σ . We observed in the numerical experiments that using no perturbation or $\sigma = 1$ leads to good results in practice on our problems.

Skipping the bilevel optimization Using a bilevel optimization enables to define $\ell(\boldsymbol{\theta}, \Gamma^h)$ unambiguously even when the easy problem (e) admits several optimal solutions. Since the bilevel optimization is not easy to handle, we use in the numerical experiments the loss

$$\tilde{\ell}(\boldsymbol{\theta}, \Gamma^h) = \frac{1}{u(\mathcal{E})} f^h(\mathcal{A}^e \circ \varphi_{\boldsymbol{\theta}}(\Gamma^h))$$

that takes the solution returned by algorithm \mathcal{A}^e . Its value may therefore depend on \mathcal{A}^e .

Post-processing As an alternative, we can use in the learning problem the solution returned by the solution pipeline after the post-processing ψ instead of the solution of the easy problem. The loss becomes

$$\ell^\psi(\boldsymbol{\theta}, \Gamma^h) := \frac{1}{u(\mathcal{E})} f^h(\psi \circ \mathcal{A}^e \circ \varphi_{\boldsymbol{\theta}}(\Gamma^h)). \quad (16)$$

The resulting learning algorithm may be much more time-consuming depending ψ .

4.4 Comparison to the structured learning formulations

Bengio et al. [2021, Section 3.1] distinguish two approaches to formulate learn a predictor taking some given decisions in a combinatorial optimization context. The *learning by demonstration* approach supposes to have a training set containing instances and good decisions, and trains the predictor to take the same decision. The *learning by experience* approach does not require examples with good decisions, but computes some objective on the result returned with a given parameter, and seeks a parameter leading to a good objective. For instance, when a predictor is used to take branching decision in a branch-and-bound algorithm [Lodi and Zarpellon, 2017], a learning by demonstration approach typically learn to imitate strong branching [Alvarez et al., 2017] while the experience approaches can for instances try to minimize the total running time [Khalil et al., 2016]. With the objective of selecting subroutines in a quadratic optimization solver, Karapetyan et al. [2017] use a learning approach relatively similar to our one, where the relevance of a parametrization is evaluated by launching the solution pipeline

In our structured approximation context, the learning by demonstration approach uses a training set with hard problem (h) instances and their optimal solution, and train the prediction problem to output these solutions. The author [Parmentier, 2021] has previously shown that the learning by demonstration problem can be reformulated as a structured learning problem [Nowozin, 2010]. The most generic algorithm [Parmentier and T’Kindt, 2021] is based on Fenchel Young losses with perturbed differentiable optimizers [Berthet et al., 2020]. It is however less generic than the learning by experience approach proposed in this paper : It requires to have a training set that contains instances Γ^h and their optimal solutions, and that $f^e(x, (\mathcal{E}, \varrho^e))$ is linear in ϱ^e . An advantage of the structured learning approach is that, if $\bar{\varphi}_\theta$ is linear, the learning problem is convex (in θ) and therefore easier to solve. We will see in the numerical experiments that, on examples where all these assumptions are satisfied, we do not lose much by using (13) instead of the structured learning approach.

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 θ is chosen as in Section 4. Two conditions seem necessary to obtain such guarantees. First, it must be possible to approximate the hard problem by the easy one. That is, there must exist a $\tilde{\theta}$ such that an optimal solution of $\varphi_{\tilde{\theta}}(\Gamma^h)$ provides a good solution of Γ^h . And second, when such a $\tilde{\theta}$ exists, our learning problem must be able to find it or another θ' 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” θ when the number of instances in the solution set increases. And then we show that if there exists a $\tilde{\theta}$ such that the expected optimality gap of the solution returned by our solution approach is bounded, then the expected optimality gap for the learned θ is also bounded. For statistical reasons discussed in Section 5.4, we carry this analysis using the regularized learning problem (15).

5.1 Background on learning with perturbed bounded losses

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

$$\ell^{\text{pert}}(y, \boldsymbol{\theta}) = \mathbb{E}[\ell(y, \boldsymbol{\theta} + \sigma Z)] \quad \text{with } \sigma > 0. \quad (17)$$

We suppose that $\ell(\cdot, \boldsymbol{\theta})$ is integrable for all $\boldsymbol{\theta} \in \Theta$. We define the *expected risk* $L(\boldsymbol{\theta})$ and the *expected risk minimizer* $\boldsymbol{\theta}^*$ as

$$\boldsymbol{\theta}^* \in \arg \min_{\boldsymbol{\theta} \in \Theta} L(\boldsymbol{\theta}) \quad \text{with } L(\boldsymbol{\theta}) = \mathbb{E}[\ell^{\text{pert}}(Y, \boldsymbol{\theta})]. \quad (18)$$

Let Y_1, \dots, Y_n be n i.i.d. samples of Y . We define the *empirical risk* $\hat{L}_n(\boldsymbol{\theta})$ and the empirical risk minimizer $\hat{\boldsymbol{\theta}}_n$ as

$$\hat{\boldsymbol{\theta}}_n \in \arg \min_{\boldsymbol{\theta} \in \Theta} \hat{L}_n(\boldsymbol{\theta}) \quad \text{with } \hat{L}_n(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^n \ell^{\text{pert}}(Y_i, \boldsymbol{\theta}). \quad (19)$$

Note that both $L_n(\boldsymbol{\theta})$ and $\hat{\boldsymbol{\theta}}_n$ are random due to the sampling of the training set Y_1, \dots, Y_n . The following result bounds the excess risk incurred when we use $L_n(\boldsymbol{\theta})$ instead of $L(\boldsymbol{\theta})$.

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

$$L(\hat{\boldsymbol{\theta}}_n) - L(\boldsymbol{\theta}^*) \leq C \frac{Md}{\sigma\sqrt{n}} + \sqrt{\frac{2 \log(2/\delta)}{n}} \quad (20)$$

with $C = 48 \int_0^1 \sqrt{-\log x} dx$.

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.

5.2 Learning rate of our structured approximation

In order to apply Theorem 2 to the learning problem of Section 4, we must endow the set \mathbb{N}^h of instances with a distribution. Recall that the loss ℓ and the perturbed loss ℓ^{pert} have been defined in Equations (12) and (14). We assume that the instance Γ^h is a random variable with probability distribution μ on \mathbb{N}^h , and that both $\ell(\cdot, \boldsymbol{\theta})$ and $\ell^{\text{pert}}(\cdot, \boldsymbol{\theta})$ are integrable for all $\boldsymbol{\theta}$. With these definitions, using \mathbb{N}^h as \mathcal{Y} , instances Γ^h as random variables Y , and if we suppose that the training set $\Gamma_1^h, \dots, \Gamma_n^h$ is composed of n i.i.d. samples of Γ^h , we have recast our regularized learning problem (15) as a special case of (19). We can therefore apply Theorem 2 and deduce that the upper bound (20) on the excess risk applies.

This result underlines a strength of the structured approximations of Section 3. Because they enable to use approximations parametrized by $\boldsymbol{\theta}$ whose dimension is small and does

not depend on \mathcal{E} , the bound on the excess risk only depend on the dimension of $\boldsymbol{\theta}$ and not on the structure of the instances used. Hence, *a structured approximation enables to make predictions that generalize (in expectation) on a test set whose instances structures 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 do not appear in the training set.

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

5.3 Approximation ratio of our structured approximation

We now suppose that the approximation ratio of (h) by (e) is small: We suppose that there exists $\hat{\boldsymbol{\theta}} \in \Theta$ and $a > 0$ such that, for all $x \in \mathcal{X}$ and $\Gamma^h \in \mathfrak{N}^h$ such that

$$\frac{\|f^h(x, \Gamma^h) - f^e(x, \varphi_{\boldsymbol{\theta}}(\Gamma^h))\|}{u(\Gamma^h)} \leq a. \quad (21)$$

We suppose that $\Gamma^h \mapsto \mathbb{E}\left[\frac{\min_x f^h(x, \Gamma^h)}{u(\mathcal{E})}\right]$ is integrable on \mathfrak{N}^h . Taking the expectation gives a bound on the expected loss $\mathbb{E}\ell(\hat{\boldsymbol{\theta}}, \Gamma^h)$. We would like to apply Theorem 2 to deduce a bound on the approximation ratio with $\hat{\boldsymbol{\theta}}_n$. This is not possible in general because the perturbation may lead to a perturbed expected loss $L(\boldsymbol{\theta}^*)$ much larger than a . However, the following theorem shows that this work if the structured predictor used is sufficiently regular in $\boldsymbol{\theta}$.

Theorem 3. *Suppose that there exists $\tilde{\boldsymbol{\theta}}$ such that (21) is satisfied, the predictor $\bar{\varphi}_{\boldsymbol{\theta}}(e, \Gamma^h)$ is such that $\boldsymbol{\theta} \mapsto \bar{\varphi}_{\boldsymbol{\theta}}(e, \Gamma^h)$ is κ^φ Lipschitz, and $\boldsymbol{\varrho}^e \mapsto f^e(\boldsymbol{\theta}, (\mathcal{E}, \boldsymbol{\varrho}^e))$ is κ^e Lipschitz. Then, under the hypotheses of Theorem 2, with probability at least $1 - \delta$ (on the sampling of the training set)*

$$\underbrace{L(\hat{\boldsymbol{\theta}}_n) - \mathbb{E}\left[\frac{\min_x f^h(x, \Gamma^h)}{u(\mathcal{E})}\right]}_{\text{Perturbed prediction optimality gap}} \leq \underbrace{C \frac{Md}{\sigma\sqrt{n}} + \sqrt{\frac{2 \log(2/\delta)}{n}}}_{\text{Training set error}} + \underbrace{2a}_{\text{Approximation error}} + \underbrace{2\sigma\kappa^e\kappa^\varphi\sqrt{d}\mathbb{E}\left[\frac{\sqrt{|\mathcal{E}|}}{u(\mathcal{E})}\right]}_{\text{Perturbation error}}$$

where $|\mathcal{E}|$ denotes the number of elements $\sum_i |E_k|$ in the structure \mathcal{E} .

Theorem 3 gives a natural way of setting the strength σ of the perturbation: The bound is minimized when we use

$$\sigma = \sqrt{\frac{CM\sqrt{d}}{\sqrt{n}2\kappa^e\kappa^\varphi\mathbb{E}\left[\frac{\sqrt{|\mathcal{E}|}}{u(\mathcal{E})}\right]}}.$$

Note that the hypothesis that $\varphi_{\boldsymbol{\theta}}(e, \Gamma^h)$ is Lipschitz is satisfied when $\overline{\varphi_{\boldsymbol{\theta}}(e, \Gamma^h)}$ is the dot product $\langle \boldsymbol{\theta} | \boldsymbol{\phi}(e, \Gamma^h) \rangle$ and the feature map is bounded $|\boldsymbol{\phi}(e, \Gamma^h)| \leq \kappa^\varphi$. Let us recall that $u(\mathcal{E})$ is an upper bound on the cost of an instance with structure \mathcal{E} . On many operations

research problems, we there expect to scale linearly with $|\mathcal{E}|$. In this case, the perturbation error is therefore in $O\left(\mathbb{E}[1/\sqrt{|\mathcal{E}|}]\right)$ and its impact on the gap decreases with the size of the instance. Remark that $\sqrt{|\mathcal{E}|}$ and $u(\mathcal{E})$ both measure the size of the instance.

Theorem 3 is an immediate consequence of Theorem 2 and the following lemma.

Lemma 4. *Under the hypotheses of Theorem 3,*

$$L(\boldsymbol{\theta}^*) - \mathbb{E} \left[\frac{\min_x f^h(x, \Gamma^h)}{u(\mathcal{E})} \right] \leq 2a + 2\sigma\kappa^e\kappa^\varphi\sqrt{d}\mathbb{E} \left[\frac{\sqrt{|\mathcal{E}|}}{u(\mathcal{E})} \right].$$

Proof. Given an instance $\Gamma^h = (\mathcal{E}, \boldsymbol{g}^h)$, we have $\varphi_{\boldsymbol{\theta}}(\Gamma^h) = (\bar{\varphi}_{\boldsymbol{\theta}}(e, \Gamma^h))_{e \in \mathcal{E}}$ and

$$\begin{aligned} \|\varphi_{\tilde{\boldsymbol{\theta}}}(\Gamma^h) - \varphi_{\tilde{\boldsymbol{\theta}} + \sigma\mathbf{Z}}(\Gamma^h)\| &= \left\| (\bar{\varphi}_{\tilde{\boldsymbol{\theta}}}(e, \Gamma^h) - \varphi_{\tilde{\boldsymbol{\theta}} + \sigma\mathbf{Z}}(e, \Gamma^h))_{e \in \mathcal{E}} \right\| \\ &\leq \left\| (\sigma\kappa^\varphi\|\mathbf{Z}\|)_{e \in \mathcal{E}} \right\| \leq \|\mathbf{Z}\|\sqrt{|\mathcal{E}|}\sigma\kappa^\varphi. \end{aligned}$$

We have

$$\begin{aligned} &|f^h(x, \Gamma^h) - f^e(x, \varphi_{\tilde{\boldsymbol{\theta}} + \sigma\mathbf{Z}}(\Gamma^h))| \\ &\leq |f^h(x, \Gamma^h) - f^e(x, \varphi_{\tilde{\boldsymbol{\theta}}}(\Gamma^h))| + |f^e(x, \varphi_{\tilde{\boldsymbol{\theta}}}(\Gamma^h) - f^e(x, \varphi_{\tilde{\boldsymbol{\theta}} + \sigma\mathbf{Z}}(\Gamma^h))| \\ &\leq au(\Gamma^h) + \kappa^e \|\varphi_{\tilde{\boldsymbol{\theta}}}(\Gamma^h) - \varphi_{\tilde{\boldsymbol{\theta}} + \sigma\mathbf{Z}}(\Gamma^h)\| \\ &\leq au(\Gamma^h) + \sigma\kappa^e\kappa^\varphi\|\mathbf{Z}\|\sqrt{|\mathcal{E}|}. \end{aligned}$$

Let $x \mapsto f_1(x)$ and $x \mapsto f_2(x)$ be functions from compact set K to \mathbb{R} , and let x_1^* and x_2^* be respectively minima of f_1 and f_2 . If we have $|f_1(x) - f_2(x)| \leq \beta$ for all x , then $f_1(x_2^*) - f_1(x_1^*) \leq 2\beta$. It therefore follows from the previous inequality that

$$u(\mathcal{E})\ell(\boldsymbol{\theta} + \mathbf{Z}, \Gamma^h) - \min_{x \in \mathcal{X}(\mathcal{E})} f^h(x, \Gamma^h) \leq 2au(\Gamma^h) + 2\sigma\kappa^e\kappa^\varphi\|\mathbf{Z}\|\sqrt{|\mathcal{E}|}.$$

Since \mathbf{Z} is a standard Gaussian, we get $\mathbb{E}\|\mathbf{Z}\| \leq \sqrt{d}$ (see Equation (23) in Appendix A), and the results follows by dividing the previous equality by $u(\Gamma^h)$ and taking the expectation. \square

Remark 3. In Theorem 3 and Lemma 4, we don't need the constant a and κ^e and κ^φ to be uniform in Γ^h . It then suffices to replace them by their expectation in the bounds. \triangle

5.4 Influence of the perturbation

Since we have made very few assumptions on f^h , f^e , and φ , we do not have control on the size of the family of functions $\{\ell_{\boldsymbol{\theta}}: \boldsymbol{\theta}\}$, 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 $\boldsymbol{\theta}$ does the job,

but comes at a double cost in Theorem 3: 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 θ . 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 7 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 f^h , f^e , and φ with the objective of making the perturbation facultative in the proof.

6 Numerical experiments

This section tests the performance of our algorithms on the stochastic vehicle scheduling problem and on the $1|r_j|\sum_j C_j$ scheduling problem of Section 2. We use the same encoding φ_θ , easy problem solution algorithm, and decoding ψ as in previous contributions [Parmentier, 2021, Parmentier and T'Kindt, 2021]. The only difference is that, instead of using the learning by demonstration approaches proposed in these papers, we use the learning by experience approach of this paper. All the numerical experiments have been performed on a Linux computer running Ubuntu 20.04 with an Intel® Core™ i9-9880H CPU @ 2.30GHz \times 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.

6.1 Stochastic VSP

6.1.1 Setting: Features, post-processing, and instances

For the numerical experiments on the stochastic VSP, we use the exact same settings as in our previous work [Parmentier, 2021]. We use the same linear predictor with a vector ϕ containing 23 features. And we do not use a post-processing 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 \checkmark 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 [Parmentier, 2021]. The validation set, which is used in the learning by demonstration approach, is not used on the learning by experience 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

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

Table 2: Instances considered for the stochastic VSP.

learning by demonstration approach and exact solvers cannot handle larger instances. This is no more a constraint with the learning by experience 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 (13) and the regularized learning problem (15). We use the number of tasks $|V|$ as $u(\Gamma^h)$. 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 [Jones et al., 1993] implemented in the `nlopt` library [Johnson], and the Bayesian optimization algorithm as it is implemented in the `bayesopt` library [Martinez-Cantin]. 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 (13) and ℓ^{pert} for the regularized problem. The next one provides the training 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 in 1000 iterations. Using Bayesian optimization may however be useful with a smaller iteration budget. Figure 6.1.2 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 θ 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 introduced in Section 6.2.2.

6.1.3 Algorithm performance on test set

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

Table 3: Performance of the DIRECT [Johnson] and Bayesian optimization [Martinez-Cantin] on the learning problems (13) and (15) for the stochastic VSP.

Learning problem θ			Moderate				Large				Huge				All			
Obj	Train. set	Pert	T^{avg}	$\frac{T^{\text{max}}}{T^{\text{avg}}}$	δ^{avg}	δ^{max}	T^{avg}	$\frac{T^{\text{max}}}{T^{\text{avg}}}$	δ^{avg}	δ^{max}	T^{avg}	$\frac{T^{\text{max}}}{T^{\text{avg}}}$	δ^{avg}	δ^{max}	T^{avg}	$\frac{T^{\text{max}}}{T^{\text{avg}}}$	δ^{avg}	δ^{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)

Table 4: Performance of our solution algorithm with different θ on the stochastic VSP test set.

We now evaluate the performance of our solution pipeline with the θ learned. It has been shown [Parmentier, 2021] that, using solution pipeline with the θ 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 [Berthet et al., 2020, Parmentier and T’Kindt, 2021] to obtain a second benchmark.

Table 4 summarizes the results obtained. The first three columns indicate how θ 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} provide the average computing time for our full solution pipeline on the subset of instances considered, which includes the computation of the features and $\varphi_{\theta}(\Gamma^{\text{h}})$, 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 $\frac{T^{\text{max}}}{T^{\text{avg}}}$ gives the maximum value

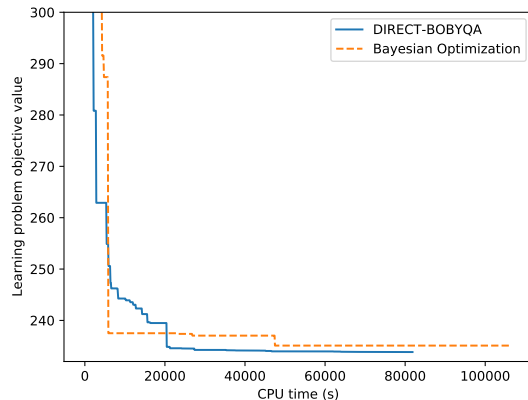


Figure 3: Learning problem objective value evolution as a function of time on stochastic VSP learning problem with perturbed loss ℓ^{pert} on training set “all”.

of this ratio on the subset of instances considered. Since we do not have an exact algorithm for the problem, for each instance Γ^h we compute the gap

$$\frac{c_{\theta} - c^{\text{best}}}{c^{\text{best}}} \quad (22)$$

between the cost c_{θ} of the solution returned by our solution pipeline with the θ evaluated and the cost of the best solution found for these instances using all the algorithms tested. The columns δ^{avg} and δ^{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 θ 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 learning by experience formulation gives slightly weaker performances than the learning by demonstration approach with a Fenchel Young loss when using the same training set.
3. Our learning by experience approach enables 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 turn false 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 [Parmentier and T'Kindt, 2021]. In particular, that paper introduces a vector of 66 features, and a subset of 27 features that leads to better performances. With the 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 the RDI algorithm (RDI \circ LS), and the perturbed versions of the last algorithm, (pert RDI \circ LS) where the solution pipeline is applied with $\theta + \mathbf{Z}$ for 150 different samples of a standard Gaussian \mathbf{Z} , and keep the best solution found.

We use the same generator of instances Γ^h of $1|r_j|\sum_j C_j$ as previous contributions (Della Croce and T'kindt [2002], Parmentier and T'Kindt [2021]). 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

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

[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 [Parmentier and T’Kindt, 2021] 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 learning by experience 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 [Parmentier and T’Kindt, 2021], 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 $u(\Gamma^h)$. It is not an upper bound on the cost, but the cost of the optimal solution scales roughly linearly with $u(\Gamma^h)$. 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 solution pipeline 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

obj	iter	pert	ψ	Tot. CPU	Avg \hat{L}	Best \hat{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.

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

Learning problem θ				Test set results (with several ψ)							
obj	iter	pert	ψ	no ψ		LS		RDI \circ LS		pert RDI \circ LS	
				δ^{avg}	δ^{max}	δ^{avg}	δ^{max}	δ^{avg}	δ^{max}	δ^{avg}	δ^{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 7: Performance of our solution algorithms with different θ on the $1|r_j|\sum_j C_j$ test set.

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 post-processing ψ . In our case, the post-processing 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 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) post-processing 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 \hat{L} and Best \hat{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 θ on the full test set. The first line corresponds to the Fenchel young loss (FYL) of the learning by demonstration approach previously proposed [Parmentier and T’Kindt, 2021], 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

Pred. ψ	θ			Moderate			Large			Huge			
	obj	iter	$ \Omega $	ψ	T^{avg}	δ^{avg}	δ^{max}	T^{avg}	δ^{avg}	δ^{max}	T^{avg}	δ^{avg}	δ^{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.

Table 8: Influence of instances size on the performance of our solution algorithms with different θ on the $1/|r_j| \sum_j C_j$ test set.

the learning problems used to obtained θ 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 post-processing described in Section 6.2.1. Again, we provide the average δ^{avg} and the worse δ^{max} values of the gap (22) 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 loss by experience 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 post-processing tend to improve the performance on the test set with the pipelines that use this preprocessing, and possibly other after. But it decreases the performance on the pipeline which do not use it.

Finally, Table 8 details the results for the fastest (no ψ) and the most accurate one (pert RDI \circ LS) pipeline 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 learning by experience 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.

7 Conclusion

Elaborating on previous contributions, we have introduced the notion of structured approximation of an operations research problem by another, which enables to leverage efficient algorithms of the literature on difficult variants of the problem they are designed for. This notion makes the elaboration of such approximations straightforward, and turn the Fenchel Young loss formulation learning [Parmentier and T’Kindt, 2021] into a generic learning by demonstration approach to fit these approximations. Furthermore, we introduce a generic learning by experience approach to learn these structured approximations. Using arguments at the crossroad of statistical learning theory and approximation algorithm theory, we prove that, provided that the problem can be approximated by the approximations considered, the approximation learned leads to an approximation algorithm. Extensive numerical experiments show that the learning by experience approach matches or outperforms the performance of the learning by demonstration approach on the stochastic vehicle scheduling problem and on the $1|r_j|\sum_j C_j$ problems. Finally, the fact that no solution of the instances in the training set is needed enables to use a more diversified training set, which leads to algorithms that scale better on large instances.

Several possible directions are possible to improve algorithms based on structured approximations. Features free approximations, and constraints approximations, would extend the scope of possible applications. Algorithms tailored to the specific structure of the learning problem may improve its resolution and give theoretical guarantees. Or alternative formulation of the learning problem may improve its tractability. Such contributions could improve the quality of the approximation ratio of the resulting algorithm.

Acknowledgements

I am grateful to Yohann de Castro and Julien Reynier for their help on Section 5, and to Vincent T’Kindt for his help on the scheduling problem.

References

- Alejandro Marcos Alvarez, Quentin Louveaux, and Louis Wehenkel. A Machine Learning-Based Approximation of Strong Branching. *INFORMS Journal on Computing*, 29(1): 185–195, January 2017. doi: 10.1287/ijoc.2016.0723.
- Irwan Bello, Hieu Pham, Quoc Le, Mohammad Norouzi, and Samy Bengio. Neural Combinatorial Optimization with Reinforcement Learning. In *International Conference on Learning Representations*, February 2017.
- Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d’horizon. *European Journal of Operational Research*, 290(2):405–421, April 2021. doi: 10.1016/j.ejor.2020.07.063.

- Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis R. Bach. Learning with differentiable perturbed 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.
- Olivier Bousquet, Stéphane Boucheron, and Gábor Lugosi. Introduction to Statistical Learning Theory. In *Advanced Lectures on Machine Learning*, volume 3176, pages 169–207. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. ISBN 978-3-540-23122-6 978-3-540-28650-9. doi: 10.1007/978-3-540-28650-9_8.
- F. Della Croce and V. T'kindt. A recovering beam search algorithm for the one-machine dynamic total completion time scheduling problem. *Journal of the Operational Research Society*, 53:1275–1280, 2002.
- Steven G. Johnson. The NLOpt nonlinear-optimization package. URL <http://github.com/stevengj/nlopt>. Accessed on 2021-07-04.
- D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181, October 1993. doi: 10.1007/BF00941892.
- Chaitanya K. Joshi, Thomas Laurent, and Xavier Bresson. An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem. *arXiv:1906.01227 [cs, stat]*, October 2019.
- Daniel Karapetyan, Abraham P. Punnen, and Andrew J. Parkes. Markov Chain methods for the Bipartite Boolean Quadratic Programming Problem. *European Journal of Operational Research*, 260(2):494–506, July 2017. ISSN 0377-2217. doi: 10.1016/j.ejor.2017.01.001.
- Elias B Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to Branch in Mixed Integer Programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, page 8, 2016.
- Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations, ICLR '17*, 2017. URL <https://openreview.net/forum?id=SJU4ayYgl>.
- Wouter Kool, Herke van Hoof, and Max Welling. Attention solves your tsp, approximately. *Statistics*, 1050:22, 2018.
- Andrea Lodi and Giulia Zarpellon. On learning and branching: A survey. *TOP*, 25(2): 207–236, July 2017. doi: 10.1007/s11750-017-0451-6.
- Ruben Martinez-Cantin. BayesOpt: A Bayesian Optimization Library for Nonlinear Optimization, Experimental Design and Bandits. page 5.

- Sebastian Nowozin. Structured Learning and Prediction in Computer Vision. *Foundations and Trends® in Computer Graphics and Vision*, 6(3-4):185–365, 2010. doi: 10.1561/06000000033.
- Axel Parmentier. Learning to Approximate Industrial Problems by Operations Research Classic Problems. *Operations Research*, April 2021. doi: 10.1287/opre.2020.2094.
- Axel Parmentier and Vincent T’Kindt. Learning to solve the single machine scheduling problem with release times and sum of completion times. *arXiv:2101.01082 [cs, math]*, January 2021.
- Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS’15, pages 2692–2700, Cambridge, MA, USA, December 2015. MIT Press.
- Michael M. Wolf. Mathematical Foundations of Supervised Learning. https://www-m5.ma.tum.de/foswiki/pub/M5/Allgemeines/MA4801_2018S/ML_notes_main.pdf, 2018.

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 [Bousquet et al., 2004]. The lecture notes of [Wolf, 2018] contain detailed proofs.

We place ourselves in the setting of Section 5.1. Let \mathcal{F} be the family of functions $\{Y \mapsto \ell(Y, \boldsymbol{\theta}) : \boldsymbol{\theta} \in \Theta\}$. The *Rademacher complexity* of \mathcal{F} is

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

where the σ_i are i.i.d. Rademacher variables. The following well-known result bounds the excess risk based on the Rademacher complexity.

Proposition 5. *With probability at least $1 - \delta$, we have*

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

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 Y_i by its values for the training set used Y_1, \dots, Y_n .

$$\hat{\mathcal{R}}_n(\mathcal{F}) = \mathbb{E} \left[\sup_{\boldsymbol{\theta} \in \Theta} \frac{1}{n} \sum_{i=1}^n \sigma_i \ell(Y_i, \boldsymbol{\theta}) | Y_1, \dots, Y_n \right]$$

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

Given n instances Y_1, \dots, Y_n and the corresponding distribution $\hat{\mu}_n$ on \mathcal{Y} , the pseudometric $L_2(\hat{\mu}_n)$ on \mathcal{F} is the L_2 norm induced by $\hat{\mu}_n$ on \mathcal{F}

$$\|\ell(\cdot, \boldsymbol{\theta}) - \ell(\cdot, \boldsymbol{\theta}')\|_{2, \hat{\mu}_n} = \sqrt{\frac{1}{n} \sum_{i=1}^n (\ell(Y_i, \boldsymbol{\theta}) - \ell(Y_i, \boldsymbol{\theta}'))^2}$$

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

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

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

Proposition 6. (*Dudley's theorem*) *Let \mathcal{F} be a family of mapping from \mathcal{Z} to $[-1, 1]$, then*

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

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 \mathbf{Z} be a centered standard Gaussian vector on \mathbb{R}^d . It is well known that

$$\mathbb{E}(\|\mathbf{Z}\|) \leq \sqrt{d}. \quad (23)$$

Indeed, applying $u \leq (1+u^2)/2$ with $u = \sqrt{\frac{1}{d} \sum Z_i^2}$ gives $\frac{1}{\sqrt{d}}\|\mathbf{Z}\| \leq \frac{1}{2}(1 + \frac{1}{d} \sum_{i=1}^d Z_i^2)$. Taking the expectation and using $\mathbb{E}(Z_i^2) = 1$ gives (23).

Lemma 7. *Let $g : \mathbb{R}^d \rightarrow [0, 1]$ be an integrable function, \mathbf{Z} a standard normal random variable, $\sigma > 0$ a positive real number, and $G(\boldsymbol{\theta}) = \mathbb{E}g(\boldsymbol{\theta} + \sigma\mathbf{Z})$. Then $\boldsymbol{\theta} \mapsto G(\boldsymbol{\theta})$ is $\frac{\sqrt{d}}{\sigma}$ -Lipchitz.*

Proof. Let h be the density of $Y = \sigma\mathbf{Z}$. We have

$$G(\boldsymbol{\theta}) = \int h(\mathbf{y})g(\mathbf{y} + \boldsymbol{\theta}) = \int h(\mathbf{y} - \boldsymbol{\theta})g(\mathbf{y})$$

By dominated convergence, we have

$$\nabla G(\boldsymbol{\theta}) = - \int \nabla h(\mathbf{y} - \boldsymbol{\theta})g(\mathbf{y}) = - \int \nabla h(\mathbf{y})g(\mathbf{y} + \boldsymbol{\theta})$$

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

$$\|\nabla G(\boldsymbol{\theta})\| \leq \int \|\nabla h(\mathbf{y})\| = \int \left\| \frac{\mathbf{y}}{\sigma^2 \sqrt{2\pi\sigma}} e^{-\frac{\|\mathbf{y}\|^2}{2\sigma^2}} \right\| = \frac{\mathbb{E}(\|\mathbf{Y}\|)}{\sigma^2} = \frac{1}{\sigma} \mathbb{E}(\|\mathbf{Z}\|) \leq \frac{\sqrt{d}}{\sigma}$$

which gives the result. □

Given an arbitrary instance Y in \mathcal{Y} , Lemma 7 applied with $g = \ell(Y, \cdot)$ gives

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

Hence

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

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

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

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

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

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

Proposition 6 then gives

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

with $C = 48 \int_0^1 \sqrt{-\log x} dx$. Remark that the bound on $\hat{R}_n(\mathcal{F})$ we obtain does not depend on the sample Y_1, \dots, Y_n , and is therefore also valid for $R_n(\mathcal{F}) = \mathbb{E}(\hat{R}_n(\mathcal{F}))$. Proposition 5 then gives Theorem 2.