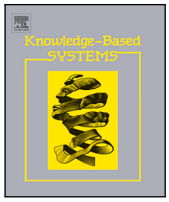
Knowledge-Based Systems 303 (2024) 112383

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys



UNIFY: A unified policy designing framework for solving integrated Constrained Optimization and Machine Learning problems

Mattia Silvestri, Allegra De Filippo *, Michele Lombardi, Michela Milano

Department of Computer Science and Engineering, University of Bologna, Italy

ARTICLE INFO

Keywords:
Machine Learning
Constrained Optimization
Reinforcement Learning
Parameter tuning

ABSTRACT

The integration of Machine Learning (ML) and Constrained Optimization (CO) techniques has recently gained significant interest. While pure CO methods struggle with scalability and robustness, and ML methods like constrained Reinforcement Learning (RL) face difficulties with combinatorial decision spaces and hard constraints, a hybrid approach shows promise. However, multi-stage decision-making under uncertainty remains challenging for current methods, which often rely on restrictive assumptions or specialized algorithms. This paper introduces UNIFY, a versatile framework for tackling a wide range of problems, including multi-stage decision-making under uncertainty, using standard ML and CO components. UNIFY integrates a CO problem with an unconstrained ML model through parameters controlled by the ML model, guiding the decision process. This ensures feasible decisions, minimal costs over time, and robustness to uncertainty. In the empirical evaluation, UNIFY demonstrates its capability to address problems typically handled by Decision Focused Learning, Constrained RL, and Stochastic Optimization. While not always outperforming specialized methods, UNIFY's flexibility offers broader applicability and maintainability. The paper includes the method's formalization and empirical evaluation through case studies in energy management and production scheduling, concluding with future research directions.

1. Introduction

In recent years, the integration of Machine Learning (ML) and Constrained Optimization (CO) techniques has attracted considerable interest. In general, pure CO methods encounter challenges in terms of scalability and robustness, while pure ML methods such as constrained Reinforcement Learning (RL) [1] face difficulties in handling combinatorial decision spaces and hard constraints. A promising research direction involves combining the two approaches at the modeling level, using ML to provide parameters [2] or entire parts [3,4] of CO models. Methods in this class can exploit implicit knowledge (from data) and provide better support for uncertainty compared to classical CO approaches; they also grant a better degree of constraints satisfaction guarantees compared to pure ML methods.

However, while integrated ML and CO methods have expanded the scope of automated decision support, some complex problems remain out reach even for such techniques. This is the case for constrained decision-making problems under uncertainty over multiple stages, which are widespread, but notoriously very challenging to tackle (e.g. production or logistic planning, energy management) and very common in industrial contexts.

Existing approaches often rely on assumptions that narrow their applicability (e.g. to linear or convex optimization problems) or require specialized algorithms [5]; as a result, identifying and implementing a viable technique for a given practical problem can be non trivial, creating a barrier towards industrial adoption. Based on the authors' experience with decision-support companies, constrained optimization solutions relying on deterministic, expert-designed, models tend to be the most frequently employed solution in practice. Available data is typically lightly processed (e.g. to compute averages) or used to train ML models in isolation that then feed parameters to the CO models.

In this perspective, we present a single framework called UNIFY that : (1) is versatile enough to deal with a wide range of problems, including multi-stage decision making under uncertainty; and (2) can be implemented by relying on standard ML and CO components. Starting from the problem knowledge in the form of both a declarative formulation (constraints and an objective function), and data (historical or simulated), the framework defines a *solution policy* by combining a CO problem and an unconstrained ML model. The interface between the two components consists of a set of *virtual parameters* in the CO

* Corresponding author.
E-mail addresses: mattia.silvestri4@unibo.it (M. Silvestri), allegra.defilippo@unibo.it (A. De Filippo), michele.lombardi2@unibo.it (M. Lombardi), michela.milano@unibo.it (M. Milano).

https://doi.org/10.1016/j.knosys.2024.112383
Received 20 May 2024; Received in revised form 28 July 2024; Accepted 12 August 2024
Available online 22 August 2024

problem, whose value is controlled by the ML model to "steer" the decision process.

Any optimization technology can be used for the CO component, while training is performed via standard RL techniques. The resulting policy can queried repeatedly and efficiently to obtain decisions that: (1) are feasible, (2) lead to minimal cost over the considered time horizon, and (3) are robust w.r.t. uncertainty.

The content of this paper builds upon the earlier results by [6] and substantially extends them in terms of both theory, experimentation and generalizability. In addition to introducing UNIFY, we also show how the method can address problems usually tackled via approaches such as Decision Focused Learning [7], Constrained RL [8], Algorithm Configuration [9], and Stochastic Optimization [10], and to extend their functionality. While our approach cannot be expected to exceed the performance of specialized methods, we believe its flexibility makes it considerably more generally applicable and maintainable, and hence appealing from a practical perspective. Moreover, to the best of the authors' knowledge, no approach proposed up to this point can deal with constrained decision-making problems under uncertainty over multiple stages without significant shortcomings, due to difficulties in handling constraints or multi-stage decisions.

The work is structured as follows: in Section 2 we introduce the context and notation for our method, discuss its key idea by means of a simple example, and provide motivation by highlighting potential application domains; a full formalization follows in Section 3. In Section 4 we discuss how UNIFY can be configured to address problems that are typically tackled by existing ML, CO, or hybrid approaches, covering related work for each considered case. In Section 5, we provide an empirical evaluation focused on investigating the flexibility of our approach; we consider two practical case studies, namely an Energy Management System and a Weighted Set Multi-cover Problem with stochastic demands (capturing the structure of a simple production scheduling scenario). Concluding remarks and open research directions are in Section 6.

2. Context and motivation

In its most general form, our method is capable of addressing multi-stage constrained decision problems with clearly defined constraints and cost, but defined in the presence of uncertainty represented by means of data. Since problems in this class provide the main motivation for our approach, here we list and discuss a few notable examples, two of which cover the use cases employed in our experiments. We then proceed to highlight their common characteristics and to define a formal notation.

Energy management systems (EMSs). An EMS requires the allocation of minimum-cost power flows from different energy resources to satisfy energy demands. Based on forecasts of the availability of Distributed Energy Resources (DERs) and user load demand, the EMS schedules power flows among the different sources and decides whether any excess of energy should be sold to the market or stored in the system. This problem has a clear declarative formulation, but it also involves elements of uncertainty (e.g. demands and power generation from DERs). While extensive literature on the topic exists [11,12], solutions deployed in the real-world often disregard uncertainty by assuming determinism (i.e. they treat predictions as perfect) or by being myopic (i.e. disregarding the long-term impact of current decisions). This is preferred in practice to the poor scalability of stochastic constrained optimization, and the difficulty of guaranteeing feasibility with data-driven methods.

Production scheduling. In a production scheduling problem, a factory has to manufacture a set of products to satisfy customer demands. Knowledge about the product demands is usually available in the form of historical data; at production time, true demands may be unknown, in which case only forecasts based on implicit knowledge can be made. Failing to satisfy the demands results in missed profit, or a cost in case additional products need to be purchased from external sources. The manufacturing process can be subject to a number of constraints that are typically well-known and problem-specific. While methods for production planning and scheduling under uncertainty have been studies for decades, practical solutions still often assume determinism or rely on heuristic rules [13], again to preserve scalability and feasibility. In [14], some of the authors of this paper proposed to build an anticipatory approach by relying on an offline parameter tuning phase, a behavior that UNIFY is capable of replicating and extending.

Paired kidney donation. Some organ transplant programs require complex planning stages. As an example, in paired kidney donation incompatible patient–donor pairs are allowed to swap donors; while the approach opens opportunities for new transplants, it also makes it very challenging to determine which transplants to perform. However, when choosing which pairs to consider for surgery, one needs to consider biological compatibility and to balance short-term survival with long-term maximization of the number of transplants. New patients and donors continuously arrive or leave the pool, according to a distribution that can be estimated only via historical data. This setting has received some research attention in the last two decades [15]; due to the challenging nature of the problem and the size of the patient–donor pool, both the literature and practical approaches have focused on myopic solutions. Among the few exceptions, the approach from [16] introduced a degree of anticipativity via parameter tuning, similarly to what would later by done in [14] for EMSs.

Delivery problems under uncertainty. Delivery problems are a vast class of optimization problems from the field of logistics that span from long-range hauling of goods to last-mile delivery and shuttle services. These problems require finding the optimal routes for a set of vehicles in such a way that customer requests are satisfied. Typically, several problem elements are well-known (e.g. road network, vehicle capacity), but many others are subject to uncertainty (e.g. travel times, amount and sometimes the location of the requests). Similarly to the previous cases, historical data can be used to characterize the elements of uncertainty. Routing problems under uncertainty have been extensively studied [17], but scalability in the general case remains a challenge, as the most effective algorithms rely deeply on properties of specific problem classes.

2.1. Key problem elements and notation

The considered examples feature some common elements, which define the broadest class of problem that the UNIFY framework can address. They can be formalized as follows:

(1) *Multiple decision stages*, referred to as a sequence of stage indices $\{k\}_{k=1}^{T}$. The notation T represents the end of the planning horizon, and it might be infinite for decision processes that run indefinitely.
(2) *Observables available at each stage*, referred to as vector of values $x^{(k)}$. These might represent information that is useful for making decisions, for evaluating the impact of past decisions, for making predictions, or for describing the system state.
(3) *Uncertainty*, affecting the values of the observables and represented via a probability distribution P, i.e. $x \sim P$. While we make no specific assumption on the distribution, its properties (e.g., causality or exogeneity) can affect which solution methods can be employed to implement an approach based on UNIFY.

Table 1
Examples of real-world problems that can be tackled with UNIFY and their components.

Problem	Stages	Observables	Uncertainty	Decisions	Constraints	Cost
EMS	Time intervals	Past demands and production	True demands and production	Power flows	Power balance and limits	Power flows cost
Production scheduling	Production days	Customer information, time of the year	True product demands	Molds to use	Molds availability	Manufacture cost
Organ transplant programs	Sets of surgery operations	Patient information	True transplants requests	Accepted transplants requests	Patient–donor compatibility	Patients deceased
Delivery problems	Customers requests	Customer information, hour of the day	Travel time, new requests	Travel route	Time constraints	Traveling time

(4) *Decisions to be taken for each stage*, referred to as vectors of variables $z^{(k)}$. The variables might represent how many items to produce in a day, the power-flows for a 15 min interval in an EMS, which donor–patient pairs to select for surgery in an organ transplant program, etc. We make no assumption on the domain of $z^{(k)}$, which in the general case may even be stage-dependent (e.g. changing donor and patient pools in an organ transplant program).
(5) *Hard constraints for each stage*, which define the feasible values for the decision variables. We represent such constraints as a set, whose definition depends on the values of the observables: formally, we have that $z^{(k)} \in C(x^{(k)})$, with $C(\cdot)$ being a set-valued function. We assume without loss of generality[1] that the feasible set can be defined based on the observables for the current stage. The constraints may require a power balance, or establish a maximal production capacity for a set of products.
(6) *An immediate cost function*, referred to as f, which specifies the cost incurred in stage $k+1$ depending on the decisions at stage k, and on the observable at stages k and $k+1$ (i.e. the previous state and how uncertainty unfolds). Formally, we have that the cost incurred at stage $k+1$ is given by $f(x^{(k)}, x^{(k+1)}, z^{(k)})$. For example, the function may measure the total profit we get at stage $k+1$ depending on the observed demand (in $x^{(k+1)}$), by selling items produced ($z^{(k)}$) and stored (in $x^{(k)}$) at stage k. We assume the goal is to *minimize the expected cost* over all the decision stages.

Knowledge about the constraints and the cost function can typically be obtained *in explicit form* by talking to domain experts, while information about the uncertainty distribution P is typically available *in implicit form*, through collections of historical data.

While all elements can be relevant in the general case, a specific application may introduce simplifying restrictions, e.g., a limited number of stages, a focus on exogenous uncertainty, or a deterministic cost function f. Similarly, not all terms in $x^{(k)}$ might be relevant for defining the constraint set $C(x^{(k)})$ or the cost $f(x^{(k)}, x^{(k+1)}, z^{(k)})$ (see Table 1).

3. The UNIFY framework

We can now proceed to define the general problems that our framework can address. We view a multi-stage decisions process as the repeated application of a solution policy, which can be trained on available data to minimize an expected cost. This setup shifts most of the computation effort to the training phase, thus keeping the inference (execution) process lightweight.

As a key challenge, the policy we seek should output decision vectors that are *feasible* w.r.t. the problem constraints. Such issue is addressed in UNIFY by decomposing the solution policy in two component: a ML model (naturally capable of dealing with uncertainty) and a CO problem (tasked with guaranteeing constraint satisfaction). Communication between the two components occurs via set of parameters in the CO problem formulation, which we refer to as *virtual parameters*. Training can be performed by classical Reinforcement Learning techniques, by treating the CO problem as part of the environment.

Inference and training problem statements. The process of choosing a decision vector $z^{(k)}$ based on the observables $x^{(k)}$ can be viewed as the application of a *policy*. This is defined as a function:

$$\pi : (x; \theta) \mapsto z \in C(x) \tag{1}$$

where x, z, and $C(x)$ are defined as in Section 2. The requirement $z \in C(x)$ implies that any viable policy is expected to *consistently satisfy all the constraints* defined for a single decision stage.

The term θ represents a set of *training parameters* that can be used to adjust the function behavior. These should not be confused with the virtual parameters in our decomposition, which will be formally discussed later in this section. The θ parameters should be chosen to minimize the long-term cost of the decisions, accounting for uncertainty. We can therefore formulate the training problem as:

$$\begin{aligned} &\underset{\theta \in \Theta}{\operatorname{argmin}}\ \mathbb{E}_{\tau \sim P}\left[\sum_{k=1}^{T} \gamma^k f(x^{(k)}, x^{(k+1)}, z^{(k)})\right] \\ &\text{with: } z^{(k)} = \pi(x^{(k)}; \theta) \end{aligned} \tag{2}$$

where f is the cost function for a single stage and τ refers to a trajectory, i.e. a sequence of observables and decisions $\tau = \{x^{(1)}, z^{(1)}, x^{(2)}, z^{(2)}, \ldots, x^{(T)}, z^{(T)}, x^{(T+1)}\}$.

The trajectory probability is given by the distribution P, which in practical applications will likely be approximated via a sample of historical or simulated data. The term T is the End Of Horizon, and γ is a discount factor, analogous to that used in Reinforcement Learning. For an infinite horizon, γ should be strictly lower than 1, and for decision problems over a finite horizon, γ should be equal to 1.[2]

Once training has been performed and an optimal parameter vector θ^* has been found, the decision-making problem can be solved by repeatedly observing $x^{(k)}$, querying $\pi(x^{(k)}; \theta^*)$ to obtain a decision vector $z^{(k)}$, and deploying the decisions to move to the next step.

Decomposed policy formulation. The main challenge when solving Eq. (2) is that of guaranteeing constraint satisfaction, which prevents addressing the problem with classical RL methods. In UNIFY, such difficulty is addressed by reformulating the policy as a composition of two terms:

$$\pi(x; \theta) \equiv g(x, h(x; \theta)) \tag{3}$$

[1] If this is not true, the observables can be redefined so as to capture all relevant information (typically with adverse effects on scalability).

[2] In Eq. (2) γ is raised to the power of k to give higher priority to the costs of stages that are closer in time.

where $h(x;\theta)$ is a ML model and $g(x,y)$ denotes the solution of a Constrained Optimization problem. Both components take as input the observable x. The ML model is the only component in the decomposition whose behavior is affected directly by the training parameters θ. The output of the ML model consists of a parameter vector y for the CO problem formulation, representing for example travel times, demands, costs, or penalties. Due to how training is performed in UNIFY, such parameters are not associated to ground truth values; instead, they are used by the ML model to guide the CO problem towards desirable solutions, and trained (as we will discuss) to minimize the solution cost. For these reason, we refer to y as *virtual parameters*. Formally, the g function is defined as follows:

$$g(x,y) \equiv \underset{z \in \tilde{C}(x,y)}{\operatorname{argmin}} \tilde{f}(x,y,z) \tag{4}$$

where the cost and constraint functions $\tilde{f}(x,y,z)$ and $\tilde{C}(x,y)$ are application-dependent and need to be defined when formulating the decomposition.

Both terms can be derived from the original cost and constraint functions from 2 i.e. f and C, with a few modifications. First, $\tilde{f}$ and $\tilde{C}$ need to depend on the virtual parameter vector y, so that the ML model can alter the optimal solution of Eq. (4). Second, the cost function $\tilde{f}$ does not use the next-stage observables $x^{(k+1)}$ as input. One way to derive $\tilde{f}$ and $\tilde{C}$ consists in treating $x^{(k+1)}$ as the virtual parameters y, e.g. letting uncertain travel times, demands, or costs be provided by the ML model. A second approach involves neglecting the effect of the future observable $x^{(k+1)}$, and introducing artificial costs or penalty terms, which then take the role of the virtual parameters. Both strategies are showcased in our experimental evaluation, while guidelines for these steps are provided below at the end of this Section.

The set $\tilde{C}(x,y)$ should imply the feasibility of the decision vector according to the original constraints. This is necessary for the policy to satisfy Eq. (1) and can be formalized as:

$$z \in \tilde{C}(x,y) \Rightarrow z \in C(x) \tag{5}$$

In practice, the property can be enforced either (1) by retaining the same constraints as the original problem, i.e. $\tilde{C}(x,y) \equiv C(x)$, or 2 by incorporating the virtual parameters in a conservative fashion (e.g. temporal buffers over deadline constraints, or reduction factors over capacity constraints).

Reformulated training problem. With the decomposed policy reformulation, the training problem becomes:

$$\underset{\theta \in \Theta}{\operatorname{argmin}}\ \mathbb{E}_{\tau \sim P}\left[\sum_{k=1}^{T} \gamma^k f(x^{(k)}, x^{(k+1)}, z^{(k)})\right] \tag{6}$$
$$\text{with: } z^{(k)} = g(x^{(k)}, y^{(k)}) \quad \text{and: } y^{(k)} = h(x^{(k)};\theta)$$

Eq. (6) is considerably easier to solve than Eq. (2), since the sources of complexity in the original problem are handled them via distinct, more appropriate techniques: the CO problem (which can be solved via Mathematical Programming or similar techniques) is in charge of ensuring feasibility and exploring a complex decision space; the ML model handles uncertainty and long-term feasibility; both components contribute to cost optimization.

The main challenge when solving Eq. (6) is the fact that $g(x^{(k)}, y^{(k)})$ is defined through an argmin operator. In many practical cases, such as Linear Programs or Combinatorial Problems, the decision vector may change in discrete steps in response to arbitrarily small changes in the virtual parameter vector, thus making $g(x^{(k)}, y^{(k)})$ piecewise constant and non-differentiable. Thankfully, optimizing over functions with these properties is a much better-understood topic, thanks to recent developments in Decision Focused Learning, and decades of research in both Black Box Optimization and Reinforcement Learning. See the surveys by [7,18], and [19] for additional references.

In fact, Eq. (6) can be mapped to a traditional RL problem by a simple change in perspective. At *training time*, the solution of the CO problem can be seen as *part of the environment* and the virtual parameter vector $y^{(k)}$ is viewed as the RL agent "action" for the kth stage. Conversely, at *inference time* the ML model and the CO problem are components of a single policy (as already discussed). The two different viewpoints are depicted in Fig. 1. As a major benefit, this mapping enables one to use *any Reinforcement Learning algorithm* for policy training in UNIFY.

Guidelines for grounding UNIFY. Defining the virtual parameters y, the cost function $\tilde{f}(x,y,z)$, and the constraint function $\tilde{C}(x,y)$ are the major design decisions when grounding our method on a practical use case. Given a use case that we want to tackle with UNIFY, the crucial steps to follow (illustrated in Fig. 2) include: (1) defining the problem components (e.g., observables, decisions, constraints); (2) determining the virtual parameters; (3) employing ML to predict them; (4) understanding how the virtual parameters modify the original optimization problem formulation.

Nevertheless, the primary source of UNIFY's flexibility lies in adopting virtual parameters as an interface between the ML model and the optimization process. Choosing the most suitable ones can, however, be challenging when grounding UNIFY.

The main idea in UNIFY is to extend an optimization approach, such as the one we have just defined, by *allowing an external component to guide its behavior*. Doing this requires introducing a mechanism for enabling communication between such a component and the optimization solver. This is achieved in UNIFY by *adjusting the value of certain parameters* in the optimization problem. Such parameters can be either (1) chosen from those naturally present in the formulation or (2) introduced ad-hoc for this purpose. In both cases, the selected parameters remain interpretable (since they are employed in a symbolic model), but only in a loose sense (since they are determined by a problem-agnostic component): for this reason, we refer to them as *virtual parameters*.

In UNIFY, a Machine Learning model can tune the virtual parameters. In particular, we introduce a ML model h, with training parameter vector θ, whose role is to predict the optimal virtual parameters $y^{(k)}$ at stage k given the current observation $x^{(k)}$.

Our design choice is motivated by a few observations: (1) Machine Learning is naturally well-suited to deal with uncertainty; (2) predictions made by ML models are contextual, meaning that in our case they can change depending on the observed $x^{(k)}$ values; (3) once trained, a ML model can very efficiently perform inference on unseen examples. Alternative options for the external component, such as blackbox optimization, do not provide the same advantages.

Unlike in classical ML tasks such as supervised learning, the ML model should not be trained for maximum accuracy. In fact, on the one hand, the model output consists of *virtual* parameters, which may lack a real-world counterpart, and therefore any ground truth value. On the other hand, our overall goal is not to make accurate predictions but rather to lead to optimal decisions. Therefore, our ML model should in principle be trained to minimize the decision cost of the overall policy. We will discuss how this is done in the UNIFY framework in Section 3.

4. Applicability of UNIFY to different problem classes

The main appeal of our method lies in its versatility and ease of implementation, thus leading to more maintainable decision support systems in practical settings. In this section, we will discuss the applicability of our methods to problem classes usually addressed by other approaches. We will present the problem statement considered by each method, then we will describe how UNIFY can be grounded to achieve analogous results. We also discuss how the resulting groundings compare to related work in each area. For sake of simplicity, we will use for all approaches a notation as close as possible to that of Sections 2 and 3.

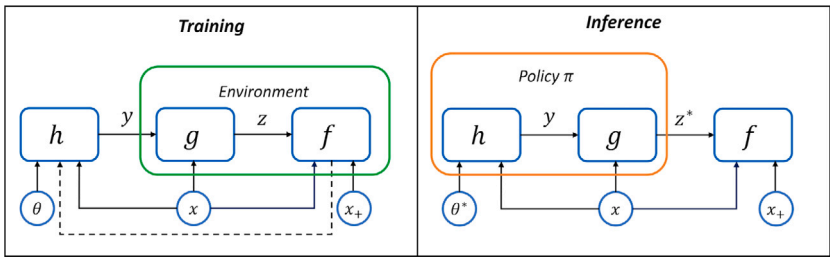


Fig. 1. UNIFY decomposition for the training and inference problems.

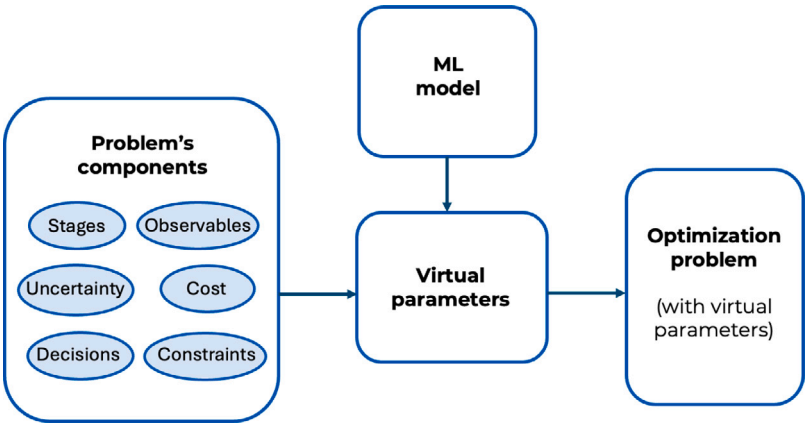


Fig. 2. When using UNIFY, one must define the components of the decision-making problem, the virtual parameters that the ML estimates and how they alter the original problem formulation.

4.1. Decision focused learning (DFL)

Many real world decision problems are addressed via "predict, then optimize" approaches. These involve training a ML model to estimate parameters of optimization problems that are unknown at solution time (e.g. travel times, item or energy demands); then, CO methods are employed as usual to obtain optimal decisions.

Traditionally, supervised learning methods are used to train the ML model. However, it was recently shown [20] that such practice can lead to suboptimal solutions, due to a misalignment between the training and optimization objective (i.e. accuracy and decision cost). Such issue can be addressed by training the ML for minimal decision cost, which typically leads to lower accuracy, but higher solution quality. The approach is known as Decision Focused Learning and has attracted considerable research interest in recent years; the field is well surveyed in [7].

DFL problem statement and a related UNIFY *grounding.* The decision-focused training problem in typical DFL methods can be stated as:

$$\underset{\theta \in \Theta}{\operatorname{argmin}}\ \mathbb{E}_{(x,y^*)\sim P}\left[f(x, y^*, z^*)\right] \tag{7}$$

$$\text{with: } z^* = \underset{z}{\operatorname{argmin}}\{f(x, y, z) \mid z \in C\} \tag{8}$$

$$\text{and: } y = h(x;\theta) \tag{9}$$

where x is the contextual information available to make estimates, y is the ML model estimate of the unknown optimization problem parameters, z^* is the optimal solution of the problem instantiated by the predictions y, y^* are the ground truth problem parameters, and it is assumed that the feasible region C is fixed. While the original approach from [20] was restricted to quadratic programming, subsequent works have tackled linear and combinatorial problems, either in an exact fashion by assuming linear costs and a fixed feasible space [21,22], or in an approximate fashion via continuous relaxations [23]. Both inner and outer relaxations have been employed also to enhance scalability of the training process [24,25], which is a key challenge in DFL since it requires to solve a large number of optimization problem.

Table 2
Grounding UNIFY to address Decision Focused Learning problems, as in Eqs. (7) and (9).

Element	Decision Focused Learning
Number of stages T	$T = 1$, since there is a single stage
Observables $x^{(k)}$	$x^{(1)} \equiv (x, \perp)$, $x^{(2)} \equiv (\perp, y^*)$
Virtual parameters $y^{(k)}$	$y^{(1)} \equiv y$, the estimated parameters
Decisions $z^{(k)}$	$z^{(1)} \equiv z$, the decision vector
Actual cost $f(x^{(k)}, x^{(k+1)}, z^{(k)})$	$f(x, y^*, z^*)$
CO constraints $\tilde{C}(x^{(k)}, y^{(k)})$	C, since the feasible set is fixed
CO cost $\tilde{f}(x^{(k)}, y^{(k)}, z^{(k)})$	$f(x, y, z)$
ML model $h(x^{(k)}; \theta)$	$h(x; \theta)$
Distribution P	P, approximated via the training data

Our framework can be considered a generalization of the DFL setting; in particular, equivalent result can be obtained by using the predicted parameters y in Eqs. (7)–(9) as the virtual parameters. The full list of grounding choices is formally presented in Table 2. Most notably, the ground truth parameters in DFL incorporated in the observable vector in UNIFY: the initial observation vector is $x^{(1)} = (x, \perp)$ and the second stage observable vector is $x^{(2)} = (\perp, y^*)$; the $\perp$ notation is used to denote values that are unused in the respective stage.

Comparison of DFL methods and UNIFY. Analogously to DFL, this grounding of UNIFY enables training for minimal decision costs; benefits and limitations are also the same, i.e. better solution quality compared to "predict, then optimize", but higher training time. As high accuracy is not a primary focus, both methods enable using simpler ML models that are faster to evaluate and easier to verify using formal methods.

While UNIFY employs RL as a general training solution, DFL methods tend to focus on specific types of optimization problems and use dedicated techniques. For example, surrogate losses such as the one from [26] have shown excellent performance, but are applicable only to linear cost functions.

Table 3
Grounding UNIFY to address Constrained RL problems, as in Eqs. (13) and (15).

Element	Constrained RL
Number or stages T	T, unchanged
Observables $x^{(k)}$	$x^{(k)}$, unchanged
Virtual parameters $y^{(k)}$	$y^{(k)}$, unchanged
Decisions $z^{(k)}$	$z^{(k)}$, unchanged
Actual cost $f(x^{(k)}, x^{(k+1)}, z^{(k)})$	$f(x^{(k)}, x^{(k+1)}, z^{(k)})$, unchanged
CO constraints $\tilde{C}(x^{(k)}, y^{(k)})$	$C(x)$
CO cost $\tilde{f}(x^{(k)}, y^{(k)}, z^{(k)})$	$\|z - y^{(k)}\|_2^2$
ML model $h(x^{(k)}; \theta)$	$h(x^{(k)}; \theta)$, unchanged
Distribution P	P, approximated via the training data or simulation

Overall, our approach is considerably more versatile than existing DFL techniques, naturally having the ability to deal with non-linear cost functions, unknown parameters in the problem constraints, and multiple-decision stages. DFL methods are classically restricted to single-decision stages and convex or linear cost functions. Recently, a few DFL approaches that can address two-decision stages (main decisions, plus a recourse action) have emerged [27,28]; such methods can also deal with parameters in the constraints and non-linear cost functions, in the case of [27] just for packing problems, while [28] is capable to addressing Mixed-Integer Linear Programs. Unknown parameters in constraints for Integer Linear Problems are also considered in [29]. However, the proposed method focuses on imitating the optimal solution instead of minimizing the downstream tasks loss. No existing DFL method can deal with more than two decision stages.

4.2. Constrained RL

Constrained RL addresses the task of training a policy to maximize a reward function while adhering to a set of constraints. These constraints may involve safety considerations, natural laws, or limitations on resource availability. Such a setup is common in various real-world scenarios, e.g. self-driving cars, drone operation, battery-powered industrial robots.

Constrained RL problem statement and a related UNIFY *grounding.* The problem of training a constrained RL policy can be formulated as follow:

$$\underset{\theta \in \Theta}{\operatorname{argmin}}\ \mathbb{E}_{\tau \sim P}\left[\sum_{k=1}^{T} \gamma^k f(x^{(k)}, x^{(k+1)}, z^{(k)})\right] \tag{10}$$

$$\text{with: } z^{(k)} = \pi(x^{(k)}; \theta) \tag{11}$$

$$\text{s.t.: } z^{(k)} \in C(x^{(k)}) \tag{12}$$

where τ is a trajectory sampled from a probability distribution P, f is the reward function, π is the RL agent, $z^{(k)}$ and $x^{(k)}$ are respectively the action and observation at timestep k, and $C(x^{(k)})$ is the set of constraints.

A strategy that some RL methods use to enforce constraints consists in projecting the decision vector from a baseline policy in feasible space; the technique is often presented a as a "safety layer" on top of a neural network policy [8], and uses the Euclidean distance to guide the projection. This approach can be replicated in UNIFY by grounding the method as follows:

$$\underset{\theta \in \Theta}{\operatorname{argmin}}\ \mathbb{E}_{\tau \sim P}\left[\sum_{k=1}^{T} \gamma^k f(x^{(k)}, x^{(k+1)}, z^{(k)})\right] \tag{13}$$

$$\text{with: } z^{(k)} = \underset{z}{\operatorname{argmin}}\{\|z - y^{(k)}\|_2^2 \mid z' \in C(x)\} \tag{14}$$

$$\text{and: } y^{(k)} = h(x^{(k)}; \theta) \tag{15}$$

The full set of grounding choices is reported in Table 3. Most notably, the virtual parameter vector $y^{(k)}$ and the decision vector $z^{(k)}$ are in the same space, since in constrained RL the ML agent is still in charge of producing actual decisions.

Comparison of constrained RL methods and UNIFY. UNIFY retains all the key properties of Constrained RL: (1) decisions are feasible by construction w.r.t. constraints defined for single stages; (2) fast inference, since the training cost is payed only once; (3) there is no need for the agent to have access to detailed problem knowledge. In fact, the presented constrained RL problem is similar to our training formulation from Eq. (6).

As a key difference, the ML model is expected to directly output a vector in the decision space, which makes it more difficult for RL method to deal with combinatorial problems; for example, routing problem need to be decomposed in RL by addressing a single transition decision at a time. UNIFY has no such limitation, as it can rely on the CO problem to effectively explore combinatorial space (e.g. the routing or transplant management problems mentioned in Section 2). Moreover, by choosing a design with a limited number of virtual parameters, our method can further simplify the RL training process; in the EMS use case from our experimentation in Section 5, for example, we use a single virtual parameter even if the decision space has a higher dimensionality (multiple power flows). Finally, our method naturally accounts for the decisions costs when computing $z^{(k)}$, rather than using a Euclidean distance as a proxy for the cost of restoring feasibility.

Safety layers are not the only viable for constrained RL. Another strategy, investigated by [30], enforces constraints by projecting *the policy weights*, as in the projected gradient method [31]; this approach improves numerical stability, but incurs a much higher computational cost, due to the large number of parameters typically present in ML models. Most techniques designed for RL can potentially be employed for UNIFY, either directly or with some adaptation. In our experiments we limit ourselves to the Advantage Actor-Critic (A2C) algorithm, leaving this area open for investigation.

4.3. Integrated offline/online optimization

Many real-world problems consist of two distinct phases. During an offline phase, long-term "strategic" decisions are obtained via expensive but accurate approaches. Conversely, during a subsequent online phase, "operational" decisions are scheduled within strict time constraints and usually over multiple steps, requiring computational-efficient but often approximated methods. For example, in vehicle routing problems, we might plan the route in advance and then adjust it when new customers appear. While for many years these two phases have been addressed separately, recently there has been an increasing interest in a tighter integration [32].

Offline/online optimization problem statement and UNIFY *grounding.* Examples include the approach from [16] for the Kidney Exchange Problem, and the more general method from [5], applied to the EMS problem considered in our experimentation. In [5] some of the authors of this papers describe how to control the behavior of an online CO problem by adjusting some of the parameters of its formulation in a offline phase. The approach requires the online problem to be convex, and works by using the Karush–Kuhn–Tucker conditions to encode its behavior as a set of constraints. Scenario-based stochastic optimization is then used in the offline phase to obtain parameter values that are guaranteed optimal, within the limit of the sampling noise. The overall approach is referred to as TUNING and its behavior is schematically depicted in Fig. 3.

Formally, approaches in this class target problems in the form:

$$\underset{y \in Y}{\operatorname{argmin}}\ \mathbb{E}_{\tau \sim P}\left[\sum_{k=1}^{T} \gamma^k f(x^{(k)}, x^{(k+1)}, z^{(k)})\right] \tag{16}$$

$$\text{with: } z^{(k)} = \underset{z}{\operatorname{argmin}}\{\tilde{f}(x^{(k)}, y, z) \mid z \in \tilde{C}(x, y)\} \tag{17}$$

where T is the number of steps in the online phase, f is the actual cost of the decisions made at each step; the $\tilde{f}$ and $\tilde{C}$ terms correspond to a modified cost and constraint function that depend on the parameters being tuned; such parameters can be virtual in nature like in the UNIFY case.

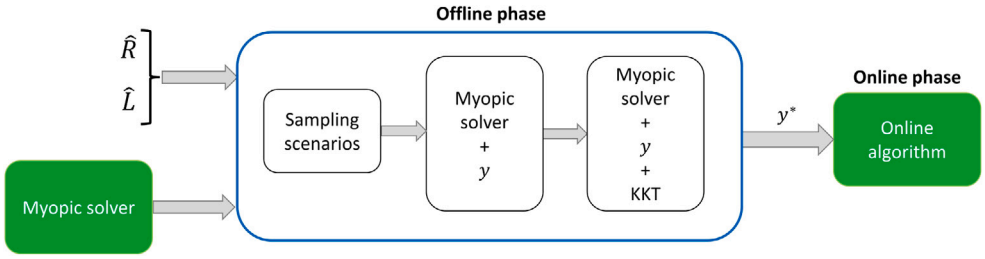


Fig. 3. Schematic view of the TUNING algorithm. Given the forecasted load demands and RES production, and the model of the myopic solver, the offline model is built by sampling many scenarios and adding the virtual cost (y in the picture) to the myopic solver. The model is then augmented with the KKT conditions and its solution, i.e. the optimal y w.r.t. the sampled scenarios, is then employed in the online algorithm.

Table 4
Grounding UNIFY to address Offline/Online Optimization problems, as in Eqs. (16) and (17).

Element	Offline/Online Optimization
Number of stages T	T, unchanged
Observables $x^{(k)}$	$x^{(k)}$, unchanged
Virtual parameters $y^{(k)}$	y, there is a single virtual parameter vector
Decisions $z^{(k)}$	$z^{(k)}$, unchanged
Actual cost $f(x^{(k)}, x^{(k+1)}, z^{(k)})$	$f(x^{(k)}, x^{(k+1)}, z^{(k)})$, unchanged
CO constraints $\tilde{C}(x^{(k)}, y^{(k)})$	$\tilde{C}(x, y)$, unchanged
CO cost $\tilde{f}(x^{(k)}, y^{(k)}, z^{(k)})$	$\tilde{f}(x^{(k)}, y^{(k)}, z)$, unchanged
ML model $h(x^{(k)}; \theta)$	none, y can be trained directly
Distribution P	P, approximated via the training data or simulation

Our method represents a strict generalization of this offline-online setup, and the problem from Eqs. (16) and (17) can be addressed by grounding the approach as in Table 4. Most notably, the virtual parameter vector lacks a stage index, i.e. $y^{(k)} = y$, since it is computed once and for all in the offline phase. Moreover, the ML model is absent or, in other words, the trainable parameters θ are the same as the virtual parameters y.

Comparison of offline/online optimization and UNIFY. When configured in this fashion, our approach matches the behavior of the existing general methods for offline/online integration, but without the requirement that the online optimization problem is convex. Due to its reliance on a ML model, our method can however take advantage contextual information, such as weather or traffic forecasts. The analogy with offline/online optimization also suggest an alternative configuration for UNIFY; in particular, rather then computing the virtual parameters $y^{(k)}$ at every step (as we do in the formalization from Section 3), on finite-horizon problem it is possible to compute them all at once. For example, based on weather forecasts, one may compute virtual parameters for a full day of operation of an Energy Management System. In such a scenario, the actual power flows would still be computed in an online fashion, thus maintaining the ability to satisfy the balance constraints; strategic guidance, however, would be more stable, thanks to the pre-computed virtual parameters. We investigate this approach in our experimentation, in the context of the EMS use case.

4.4. Stochastic optimization

The field of Stochastic Optimization [10] focuses on enhancing robustness in decision-making problems Two-stage stochastic problems have received particular attention in the field. These involve making a set of first-stage decisions before uncertainty is revealed (e.g. assigning customers to routes); once the uncertain elements are observed, however, one can rely on second-stage decisions (often referred to as *recourse actions*) to recover feasibility or to improve the cost.

Two-stage stochastic problem statement and UNIFY *grounding.* Two-stage stochastic optimization problems can be formulated as:

$$\underset{z'}{\operatorname{argmin}} \ \left\{ f_0(x, z') + \mathbb{E}_{x^+ \sim P} \left[f(x, x^+, z') \right] \mid z' \in C'(x) \right\} \tag{18}$$

with: $f(x, x^+, z') = \underset{z''}{\operatorname{argmin}} \{ F(x, x^+, z', z'') \mid z'' \in C''(x, x^+, z') \}$ (19)

where x refers to observable quantities such as know problem parameters (e.g. truck capacities) and correlates for the uncertain elements x^+ (e.g. the time of the day as a correlate for road traffic). The first- and second-stage variables, referred to respectively as z' and z'', must belong to the feasible set returned by $C'(x)$ and $C''(x, x^+, z')$, representing the first- and second-stage constraints. The problem cost includes a deterministic term f_0, plus a term f that is known only once the uncertainty is revealed, i.e. in the second stage; therefore, in the first stage we should minimize its expectation, accounting both for the uncertain elements and for the second-stage decisions.

The UNIFY framework can be used to address two-stage stochastic optimization, by using as virtual parameters y the elements of uncertainty present in the problem (e.g. demands, travel times). The training process in UNIFY will then adjust such parameters to achieve a low expected costs, rather than to correctly estimate their distribution. As a result, the y vector can be viewed as a "virtual scenario", according to which the CO problem will optimize the decisions vector.

Formally, in this setup the $g(x, y)$ function from Eq. (4) takes the form:

$$\underset{z'}{\operatorname{argmin}} \ \underset{z''}{\min} \{ \underbrace{f_0(x, z') + F(x, y, z', z'')}_{\tilde{f}(x, y, (z', z''))} \mid (z', z'') \in C'(x) \times C''(x, y) \} \tag{20}$$

The modified cost function $\tilde{f}$ for the CO problem corresponds to the cost payed under such virtual scenario. The decisions obtained by solving Eq. (20) are then evaluated as usual in UNIFY, according to their actual cost under the revealed uncertainty, i.e. $f_0(x, z') + f(x, x^+, z')$. The complete set of mapping choices is detailed in Table 5.

Comparison of two-stage stochastic optimization method and UNIFY. Traditional techniques for stochastic optimization problems employ Monte Carlo methods to estimate expected values and evaluate constraint satisfaction. In particular, the Sample Average Approximation method [33] is still a staple of the field, at least for complex real-world problem that are not amenable to more specialized methods. The approach tackles Eqs. (18) and (19) by sampling a set of scenarios from the distribution P, then replacing the expectation with a sample mean. This results in optimization models with a single set of first-stage decision variables, but multiple copies of second-stage variables (one per scenario). After a solution is obtained, only the first-stage decisions are retained, since the second-stage decisions in the SAA model are only used for approximating the problem objective.

Scalability is a key challenge in SAA approaches, and it has been mitigated by resorting to Benders decomposition in so-called L-shaped methods [34,35]; as a drawback, these approaches require additional assumptions (e.g. recourse actions computed by Linear Programs). Multi-stage decision problems are frequently approximated as a series of two-stage problems in what are known as online anticipatory algorithms [36], then solved via L-shaped methods. Alternatively, the AM-SAA method [37] directly exploits the connection between multi-stage stochastic optimization and Markov Decision Processes.

Table 5
Grounding UNIFY to address Two-Stage Stochastic Optimization problems, as in Eqs. (18) and (19).

Element	Two-Stage Stochastic Optimization
Number of stages T	$T = 1$, the focus is on the first-stage decisions
Observables $x^{(k)}$	$x^{(1)} = (x, \perp)$, $x^{(2)} = (\perp, x^+)$
Virtual parameters $y^{(k)}$	$y^{(1)}$, representing a "virtual scenario"
Decisions $z^{(k)}$	$z^{(1)} = (z', z'')$, of which only z' is retained
Actual cost $f(x^{(k)}, x^{(k+1)}, z^{(k)})$	$f_0(x, z') + f(x, x^+, z')$
CO constraints $\tilde{C}(x^{(k)}, y^{(k)})$	$C'(x) \times C''(x, y)$
CO cost $\tilde{f}(x^{(k)}, y^{(k)}, z^{(k)})$	$f_0(x, z') + f(x, y, z')$
ML model $h(x^{(k)}; \theta)$	missing in classical stochastic optimization
Distribution P	P, approximated via the training data or simulation

Using UNIFY for two-stage stochastic optimization still allows to obtain robust decisions, but with much better inference-time scalability compared to SAA approaches. This is a consequence of relying on a single "virtual" scenario in the CO problem rather than sampling multiples ones. In our method sampling stil occurs, but only when evaluating the training cost, which enables using parallelization for a faster computation of the expected value. Moreover, UNIFY naturally supports endogenous uncertainty, since at training time the first-stage decisions are known when sampling x^+ ($x^{(2)}$) in the grounding; modeling endogenous uncertainty is typically difficult in SAA approaches, since the scenarios are sampled before the definition of the optimization problem. As a drawback, our approach may over-constrain the decision process, since it can yield only solutions that can be defined based on a single scenario.

4.5. General remarks

An area of research that is related to our method is that of Algorithm Configuration, Parameter Tuning, and more recently Auto ML. These approaches seek to improve the performance of an algorithm over a target distribution, but they focus on adjusting the algorithm behavior (or the structure of a ML pipeline), rather than on changing the parameters of optimization models. For example, the focus might be on choosing branching heuristics, the learning rate in a gradient descent method, or the number of layers and neurons in a feed-forward neural network. We do not cover methods in this class in detail, but a good survey is provided by [9]. Some solutions methods from this area could be employed within UNIFY: for example, the surrogate-based black-box optimization method from [38] or the Reinforcement Learning approach from [39].

A policy-based framework for stochastic optimization was also proposed in [10], but with the aim of providing a unified view over diverse solution approaches. The result is convenient tool for defining a taxonomy or analyzing existing methods. Conversely, our UNIFY is closer in spirit to a single, though very versatile, solution approach that can implemented using standard ML and CO components.

We designed the discussion in this section to clarify the versatility of UNIFY, despite the simplicity of its core idea. We hope our discussion clarifies how the framework *blurs the line between approaches that have been investigated mostly in isolation*, thus highlighting opportunities for cross-fertilization. For example, Decision Focused Learning and Reinforcement Learning share a few key challenges (differentiating black-box or piecewise constant functions), suggesting that many ideas developed for one of the two fields could be adapted to the other. Similarly, drawing ties from Machine Learning to offline/online integration and Stochastic Optimization could open the way for more scalable approaches, while retaining the key advantages of such techniques (e.g. convergence and feasibility guarantees).

Finally, the performance of a UNIFY grounding can be heavily dependent on the choice of the virtual parameters. On the one hand, this is a non-trivial challenge for the method designer, since it requires an understanding of both Machine Learning and Constrained Optimization methods. On the other, by choosing a convenient y vector one can configure *which aspects of the original problem are delegated to the ML model and which ones to the CO problem.* Making y more similar to a vector of decisions makes the approach closer to Reinforcement Learning, and it might be better suited for use cases where declarative models are hard to craft. Using just a few key parameters in y (as we do in our EMS example) goes in the opposite direction, and allows one to capitalize on explicit problem knowledge in cases where this is available. Overall, *choosing the virtual parameters can be thought of as a design handle* that enables partitioning the complexity of the original problem into either a ML or CO module. By managing this decision, it is possible to make sure that each module is used according to its strengths, at the same time compensating for known limitations.

5. Empirical evaluation

Our experimental evaluation is designed to demonstrate the versatility of the UNIFY approach, and how such versatility can compensate for weaknesses in existing approaches. For this reason, we will focus on two use cases where using UNIFY can provide advantages. We will consider an Energy Management System Problem and a Weighted Set Multi-cover Problem with stochastic coverage requirements, which can be seen as a simplified production scheduling problem. Code and dataset to reproduce the results are publicly available.[3]

Specifically, we employ our approach to: (1) replicate the behavior of some of the offline/online integration methods discussed in Section 4; (2) enforce constraint satisfaction in RL; (3) enable DFL in a scenario where existing methods are not applicable; (4) enhance decision robustness in stochastic optimization without using sampling at inference time.

At the same time, we will use our experiments to provide examples of how the method can be grounded by choosing virtual parameters to refactor a decision problem into a ML and a CO component.

5.1. Use cases and UNIFY groundings

Despite representing very different application domains, both these problems involve decision-making under uncertainty, observable quantities that can provide information about uncertainty, and a combination of explicit and implicit knowledge.

5.1.1. Energy management system

The first use case we consider is the EMS. An Energy Management System requires the allocation of minimum-cost power flows from different Distributed Energy Resources and an energy storage unit. We consider the same EMS setup as [40,41], which assumes *exogenous uncertainty* stemming from uncontrollable variations in the planned consumption load and the inclusion of Renewable Energy Sources (RES). This problem is based on real industrial data and settings, also used in research projects.[4] [42]

Given energy prices and the accessibility of DERs, the EMS must determine: (1) the amount of energy to generate; (2) the selection of generators for fulfilling the energy demand; (3) if the energy in excess should be sold in the energy market or stored. Decisions need to be taken and implemented at fixed intervals (e.g. every 15 min), so that *power balance* can be maintained to prevent grid failure. This results in tight restrictions on the response time for any optimization method. Additionally, power flows from/to individual generators and the storage system are subject to capacity constraints. Provided that power balance is maintained, the EMS goal is to minimize the cost over one day of operation since several key pieces of information (e.g. grid energy prices) are provided with a daily frequency.

[3] https://github.com/ai-research-disi/unify
[4] http://www.virtus-csea.it/

In this use case, it is possible to account for the explicit problem elements by relying on declarative optimization. In particular, we can model the cost function and the constraints via the following Linear Program (LP), similarly to what was done by [40]. For stage k we have:

$$\underset{z^{(k)}}{\text{argmin}} \sum_{i=2}^{m} c_i^{(k)} z_i^{(k)} \tag{21}$$

$$\text{s.t.} \sum_{i=1}^{m} z_i^{(k)} = x_{load}^{(k)} \tag{22}$$

$$l_i \le z_i^{(k)} \le u_i \qquad \forall i = 1..m \tag{23}$$

$$0 \le x_{storage}^{(k)} - \eta z_1^{(k)} \le q \tag{24}$$

$$z_i^{(k)} \in \mathbb{R} \qquad \forall i = 1..m \tag{25}$$

The decision variables $z_i^{(k)}$ correspond to the power flows from (for a positive sign) or to (for a negative sign) each power generator, the grid, and the storage system. There is a linear cost associated with every power flow, except for the storage system, which is associated with index 1. The costs change over each interval, but they are known at planning time (since they are communicated one day ahead). The net energy produced must match the observed demand, i.e. $x_{load}^{(k)}$. All flows must satisfy lower and upper physical bounds, i.e. l_i and u_i. We have $l_i \ge 0$ for every unit except for the grid, for which l_i is strictly negative since selling energy is always an option. The energy level in the storage system cannot be negative and cannot exceed its capacity q. The charging rate η depends on the time interval length and the storage system efficiency.

The LP we have presented can provide feasibility guarantees (assuming the physical limits from/to the grid are large enough) and optimize the cost for a single stage. It can also be solved quickly enough that latency constraints are not an issue. However, the model is *totally myopic*: it lacks any mechanism to anticipate future load, energy production, and costs. In particular, the model will never store energy in preparation for price spikes, since within a single stage using energy to satisfy demand or selling to the grid is always more efficient than moving it to the storage system.

In the EMS case, handling control of any of the existing parameters to the external component (e.g. l_i, u_i, η) might lead to the violation of a critical constraint. We can however introduce an ad-hoc parameter that allows the external component to alter the problem solution without affecting short-term feasibility. In particular, we will associate *a virtual cost* $y^{(k)}$ to the storage system, leading to the following modified Linear Program:

$$\underset{z^{(k)}}{\text{argmin}}\ y^{(k)} z_1^{(k)} + \sum_{i=2}^{m} c_i^{(k)} z_i^{(k)} \tag{26}$$

$$\text{s.t. Eq. (22)–(25)} \tag{27}$$

Where we assume without loss of generality that $z_1^{(k)}$ refers to the flow to/from the storage unit. Now, by giving a negative value to $y^{(k)}$ it is possible to provide an incentive for the optimization model to fill the storage system, for example, to prepare for a forthcoming peak in the grid energy price. In other words, while the optimization problem is still largely unchanged (and very easy to solve), it can now exhibit anticipatory behavior by adjusting the value of the virtual parameters.

In particular, we introduce a ML model h, with training parameter vector θ, whose role is to predict the optimal virtual parameters $y^{(k)}$ at stage k given the current observation $x^{(k)}$, i.e. $y^{(k)} = h(x^{(k)}; \theta)$.

To practically demonstrate the advantages of introducing a set of virtual parameters, we present preliminary results on the EMS problem that will be discussed in more detail later in the paper. Fig. 4 compares the output flows from the storage w.r.t. the hour of the day for the myopic solver and UNIFY.

Since it is myopic, the online solver uses all the energy available in the storage at the beginning of the day without ever refilling it. Conversely, by adjusting the virtual costs, UNIFY exhibits anticipatory capabilities, so that the storage system is used also in the later hours of the day.

We consider two versions of this problem, with slightly different grounding choices. In both cases, the CO problem is defined via the Linear Program in Eqs. (26)–(27), i.e. the version where a virtual cost was associated with the storage system. The details of the grounding process are listed in Table 6.

The two versions differ in terms of how the ML model is used at execution (i.e. inference) time. In the first version, referred to as *sequential*, the ML model is provided with up-to-date information at every decision step, which includes forecasts about energy demand and RES production for the entire planning horizon, plus the charge level of the storage system. The ML model output is the virtual cost for the storage system at the current stage, i.e. $y^{(k)}$. In this setup, training is performed via RL by viewing each solution of the CO problem, and subsequent cost evaluation, as an environment interaction, as per Fig. 1.

In the second version, the ML model input includes only information that is available one day ahead, i.e. demand and RES production forecasts. Such approach allows to obtain all virtual parameters ahead of the multi-stage decision process, thus following the same strategy used in offline/online optimization. After the virtual parameters have been obtained, power flow decisions are taken as usual in a sequential fashion. In this setup, training is performed again via RL, but an environment interaction corresponds to the computation of the entire sequence of decisions $\{z^{(k)}\}_{k=1}^{T}$. As a result, the learning task becomes easier, but less capable of adapting to dynamic conditions. Both setups are practically meaningful; additional technical details are provided in Section 5.2.

5.1.2. Weighted set multi-cover

As a second use case, we consider a production scheduling problem. Formally, we assume a factory can manufacture products out of a universe I. Products can be built only in specific combinations, each associated with a different construction cost and represented as sets over I. For our experimental evaluation, we consider a stochastic version of the problem where *the demands d are uncertain* and follow a Poisson distribution with rate λ. Their true value is unknown at production time, but we assume that some correlated information x can be observed, and used to make predictions. The observable information x is described in a rather abstract fashion since this is a synthetic benchmark: additional information in this regard is provided later in this section. Unmet demands can still be satisfied by buying additional products but at a higher cost. Our goal is to meet customer demands (by either manufacturing or buying) while minimizing the total cost.

Formally, this is a two-stage stochastic optimization problem, with recourse actions. From the point of view of a UNIFY grounding, there is a single decision stage; the observables include information available decision making, corresponding to $x^{(1)} = (x, \perp)$; and the information revealed after the decisions have been made, corresponding to $x^{(2)} = (\perp, d)$. The grounding choices are listed in details in Table 7. Notably, most stage superscripts are omitted (since there is a single decision state) and no hard constraints are present. The actual decision cost is given by:

$$f(x, d, z) = \sum_{j \in J} c_j z_j + \sum_{i \in I} p_i \max\left(0, d_i - \sum_{j \in J} a_{ij} z_j\right) \tag{28}$$

where I is the universe, and J is the collection (the indices) of all possible sets. The sets are described via the a_{ij} coefficients, with $a_{ij} = 1$ iff the ith item is contained in the jth set. The c_j coefficient is the cost of manufacturing the jth set, while p_i is the cost of buying one unit of the ith item. The number of units to be bought equals the unmet demand, i.e. it is the feasible assignment of the recourse actions having the lowest cost.

According to the guidelines for selecting virtual parameters outlined in Section 3 and to the grounding process in Section 4.4, the virtual parameters correspond in this case to an element that is already part

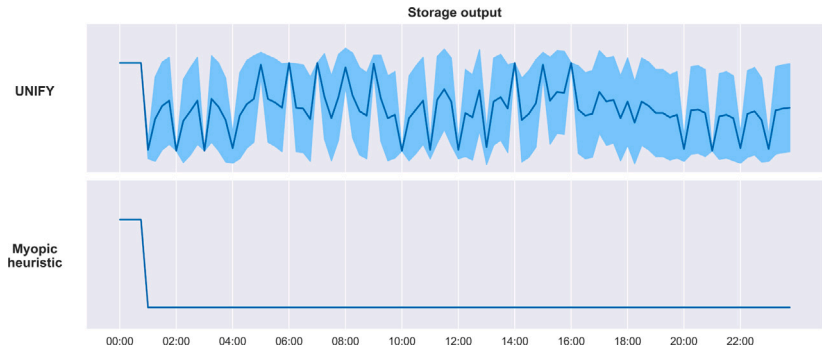


Fig. 4. Mean and standard deviation of the storage output flow on 100 problem instances w.r.t. the hour of the day.

Table 6
EMS grounding for the main problem elements in the approach.

Element	EMS Grounding
Number of stages T	$T = 96$, each stage corresponds to a 15-minute interval over one day
Observables $x^{(k)}$	$x^{(k)} = (x^{(k)}_{storage}, x^{(k)}_{load}, x^{(k)}_{res}, x_{load\,f}, x_{res\,f})$, corresponding to the energy level in the storage system, the RES production and load for stage k, plus forecasts for RES production and load for all stages
Virtual parameters $y^{(k)}$	the virtual cost for storage power flows at stage k
Decisions $z^{(k)}$	power flow from/to each generator and the storage system at stage k
Actual cost $f(x^{(k)}, x^{(k+1)}, z^{(k)})$	cost (or profit, if negative) of generating, buying, or selling energy for stage k, as in
CO constraints $\tilde{C}(x^{(k)}, y^{(k)})$	$\tilde{C}(x^{(k)}, y^{(k)}) = \tilde{C}(x^{(k)})$, since the constraints are independent on the virtual parameters; constraints include the power balance of the energy system, and upper/lower bounds for the power flows Eq. (22)-(25)
CO cost $\tilde{f}(x^{(k)}, y^{(k)}, z^{(k)})$	modified cost function, including a storage-related plan, as in Eq. (26)
ML model $h(x^{(k)}; \theta)$	$h(x^{(k)}, x_{load\,f}, x_{res\,f}; \theta)$ in the *sequential* setup; $h(x_{load\,f}, x_{res\,f}; \theta)$ in the *all-at-once* setup, since only information available one day ahead can be used in that case
Distribution P	approximated via the training data

Table 7
WSMC grounding for the main problem elements in the approach.

Element	WSMC Grounding
Number of stages T	$T = 1$, since there is a single stage
Observables $x^{(k)}$	$x^{(1)} \equiv (x, \perp)$, $x^{(2)} \equiv (\perp, d)$, where x are correlates that can be used to predict the customer demands d
Virtual parameters $y^{(k)}$	$y^{(1)} \equiv y$, virtual demand values
Decisions $z^{(k)}$	$z^{(1)} \equiv (z, s)$, i.e. the number of units of each product set to be manufactured, plus the number of additional products to buy to meet the virtual demands
Actual cost $f(x^{(k)}, x^{(k+1)}, z^{(k)})$	$f(x, d, z^*)$, as per Eq. (28); cost of both manufactured and bought items, with d being the actual demands
CO constraints $\tilde{C}(x^{(k)}, y^{(k)})$	absent (demands can always be satisfied by buying products)
CO cost $\tilde{f}(x^{(k)}, y^{(k)}, z^{(k)})$	$f(x, y, z^*)$, as per Eq. (28); cost of both manufactured and bought items, with y being the virtual demands
ML model $h(x^{(k)}; \theta)$	$h(x; \theta)$, where the input are the observable correlates
Distribution P	P, approximated via the training data

of the problem description. In particular, the y vector corresponds to a set of demands; unlike actual demands, however, these are virtual, since their role is that of guiding the CO problem towards a good solution, and not that of providing an accurate representation of the actual demand distribution. This also means that, after training, the virtual demands y will not necessarily match any statistics of the true demands d (e.g. their expected value, or even a quantile). To offer an initial insight, Fig. 5 presents some results that will be discussed in greater detail later. The image depicts the ground-truth and virtual demands with respect to the input features. It is evident that the scale of the virtual demands differs from that of the ground-truth demands.

We use the virtual demands to formulate a WSMC problem with a single scenario (represented by the virtual demands themselves) and recourse actions (i.e., how much of the items should be bought, rather than manufactured). The CO problem used in UNIFY can be modeled via the following Mixed Integer Linear Program:

$$\min \sum_{j \in J} c_j z_j + \sum_{i \in I} p_i s_i \tag{29}$$

$$\sum_{j \in J} a_{i,j} z_j \geq y_i (1 - w_i) \qquad \forall i \in I \tag{30}$$

$$(w_i = 1) \implies \left(s_i \geq y_i - \sum_{j \in J} a_{i,j} z_j \right) \qquad \forall i \in I \tag{31}$$

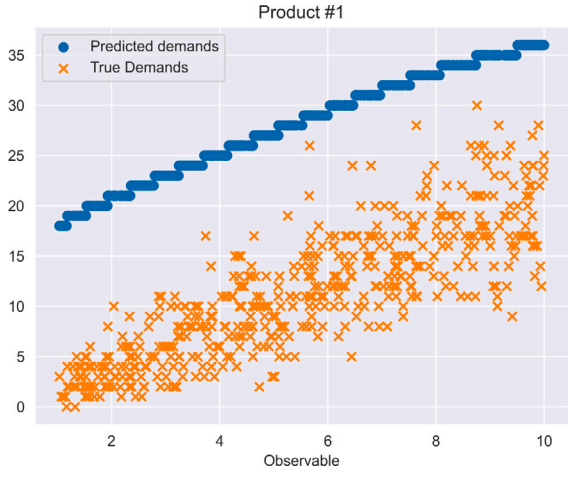


Fig. 5. This figure provides an intuitive understanding of the concept of "virtual demands" in the WSMC: the predicted demands (blue dots) exhibit a different scale compared to the true ones (orange crosses).

$$z_j \geq 0, z_j \in \mathbb{Z} \qquad \forall j \in J \qquad (32)$$

$$w_i \in \{0,1\} \qquad \forall i \in I \qquad (33)$$

$$s_i \geq 0 \qquad \forall i \in I \qquad (34)$$

While the vector of decision variables z specifies how many units of each set should be *manufactured* , the additional vector of variables s represents how many units should be *bought*. As we mentioned, y_i is the virtual demand for the ith item. Eq. (30) specifies that all virtual demands should be met unless the "flag variable" w_i for the respective item is raised, i.e. $w_i = 1$. In this case, the indicator constraint in Eq. (31) forces the s_i variable to be larger than the unmet demand. The objective combines the cost of manufacturing each set (built using the c_j coefficients) with that of buying items (with the p_i coefficients).

The cost function of the CO problem is obtained from Eq. (28) by simply swapping the actual demands d for the virtual ones y; linearizing the max operators that appear in the equations required the introduction of constraints in the MILP, which should be considered for this reason just part of the problem cost representation.

5.2. Technical details on dataset generation

To generate EMS problem instances, we used the same dataset and price values of [6]. From this dataset, we extract electric load demand and photovoltaic production forecasts, as well as upper and lower limits for generating units, and the initial status of storage units. The realizations of uncertain variables are derived from the forecasts by adding noise from a normal distribution, as previously outlined. The dataset contains individual profiles of load demand with a time step of 5 min resolution from 00:00 to 23:00. We aggregate these profiles into 15-minute timestamps and utilize them as forecasted load data. The photovoltaic production data is derived from the same dataset, featuring profiles for various sizes of photovoltaic units but consistent solar irradiance (i.e., identical shape but varying amplitude due to the panel sizes used). Similarly, photovoltaic production serves as a forecast in this scenario as well. As in [6], when evaluating the methods, we randomly selected 100 pairs of load demand and photovoltaic production forecasts.

We generate synthetic WSMC data following a set of guidelines that allow us to obtain realistic instances. For the availability matrix, we follow the procedure described in [43]: every column covers at least one row and every row is covered by at least two columns. Additionally, the availability matrix has a density of 2%, indicating the number of 1 in the matrix. The set costs are randomly generated in the range $[1, 100]$ with a uniform probability distribution. The penalties p are computed as follows: $p_i = \max_{\substack{j \in J \\ a_{i,j}=1}} c_j \cdot 10$

The equation shows that the penalty for the ith product is calculated as the maximum set cost among those covering the ith element, multiplied by 10. This basically ensures that covering an element is always more convenient than receiving a penalty. When using the SAA algorithm of Eqs. (35)–(41), penalties are the same for all the scenarios. The observable variables vectors x and the coefficients a are randomly generated with a uniform probability distribution respectively in the range $[1, 5]$ and $[1, 10]$. We generate 1000 instances, each with 1000 sets and 200 elements, and we uniformly split them between training and test sets.

For our experimentation, the demands are generated according to a Poisson distribution, and we assume the existence of a linear relationship between an observable variable $x \in \mathbb{R}$ and the Poisson rates λ for each product, i.e. $\lambda_j = a_j x \quad \forall j \in N$. In this case, the historical data is represented by a dataset $\{x_i, y_i\}_{i=1,\ldots,m}$, where y are the demands and m is the dataset size.

5.3. Offline/online integration using UNIFY

In this section, we demonstrate that UNIFY can replicate the *offline/online integration approach* based on the idea of tuning virtual parameters, which we discussed in Section 4. Experiments are run on the EMS use case, where we employ a myopic CO solver and introduce an anticipatory behavior by means of a virtual cost associated with the storage unit. Intuitively, by associating a negative cost to storage we can provide an incentive for the CO problem to accumulate energy, in preparation for forthcoming spikes in the grid energy price.

We consider both the sequential and the all-at-once versions of the problem described in Section 5.1. As a reminder, in the sequential formulation, values for the virtual storage cost are generated based on all available information, including the actual load and RES generation observed at the beginning of each decision step. In the all-at-once version, a full schedule for the virtual costs (i.e. a value for $\{y_k\}_{k=1}^{T}$) is generated before the actual decision process starts, i.e. in an "offline" phase. This approach is similar to what is typically done by an Algorithm Configuration method, except that the focus here is on tuning model parameters rather than algorithm parameters. Each decision stage is 15 min long, and the process runs for one full day (i.e. $T = 96$).

We use UNIFY to solve both versions of the problem and refer to the two approaches respectively as UNIFY-ALL-AT-ONCE and UNIFY-SEQUENTIAL. The two methods are equivalent to those we proposed in [6], which indeed can be considered an application-specific grounding of the UNIFY framework. As a baseline, we use the state-of-the-art TUNING approach from [5], which solves the all-at-once version of the problem by relying on a Mathematical Program. Notably, this approach provably converges to the best possible non-clairvoyant solution, as the number of samples used to approximate uncertainty grows. However, scalability issues prevent its usage with a large number of samples. The method also requires the online decision problem to be convex, thus limiting its applicability.

We also compare the performance to that of a clairvoyant solution (referred to as ORACLE), to provide an optimistic reference for the solution quality. This approach is obtained by simply instantiating Eqs. (26)–(27) for all decision stages, replacing all parameters with their actual historical values. Similarly to [6], we compare the methods by ensuring that they have access to the same computation time. Given that the execution time of TUNING is constant, we select this value as the time limit for training the other methods and plot its results as a horizontal line. In Fig. 6, we show the ratio to true optimal w.r.t. the computation time. As also noted in [6], we cannot have clear benefits by leveraging the sequential nature of the problem, possibly due to a suboptimal training solution, so that UNIFY-SEQUENTIAL yields slightly lower-quality solutions compared to UNIFY-ALL-AT-ONCE.

Both the UNIFY approaches performed remarkably well, with UNIFY-ALL-AT-ONCE beating the state-of-the-art TUNING method. Intuitively, the additional scalability provided by our framework enables collecting a much higher variety of samples, which was enough to compensate for the use of a suboptimal approach (Reinforcement Learning) to tackle the training problem. Additionally, UNIFY *does not require convexity for the online problem*, making it more broadly applicable.

5.4. Constraints in RL

In Section 4 we showed how UNIFY can be used to deal with hard constraints and combinatorial decision spaces in Reinforcement Learning. Similarly to safety-layer approaches, UNIFY handles constraints by applying a Constrained Optimization step on top of the output of a ML model. Unlike safety layer approaches, however, the ML model is not in charge of producing a decision vector, but rather of "piloting" the CO solver by adjusting the values of virtual parameters.

For this experiment, we rely again on the EMS benchmark. In this case, any feasible solution policy must satisfy hard constraints at each decision stage, i.e. the flow bounds and the power balance

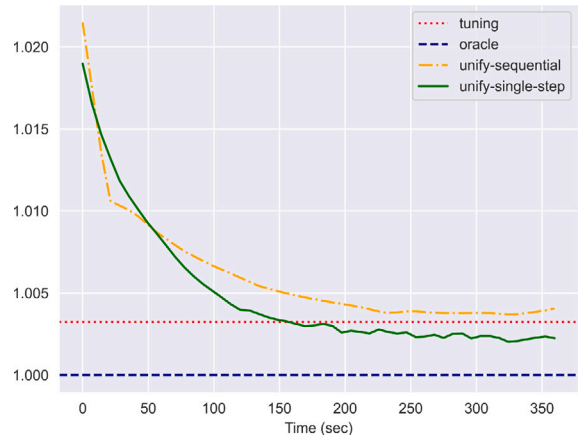


Fig. 6. Ratio to true optimal of TUNING (the state-of-the-art approach) and the UNIFY methods w.r.t. the computational time.

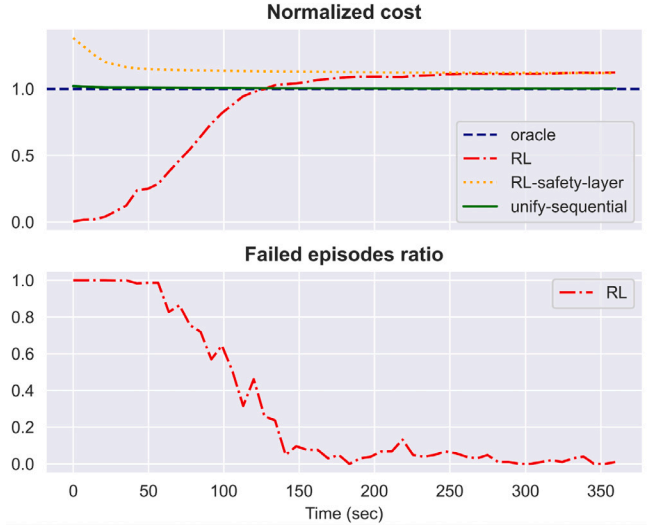


Fig. 7. This figure shows how demanding constraints satisfaction to the downstream solver is able to improve over a full end-to-end RL method and SAFETY-LAYER.

restrictions. We compare the UNIFY-SEQUENTIAL from Section 5.3 with two approaches from the literature, plus the clairvoyant oracle that serves as an optimistic reference. In particular, we train a full end-to-end deep RL algorithm to generate a solution, focusing on learning constraints satisfaction solely from the reward signal (i.e. through reward shaping); we denote this approach as RL. Designing the reward function is a non-trivial task, as it should provide a balance between exploring the feasible space and finding good solutions. Projection-based deep RL algorithms (e.g., Safety Layer) provide an alternative to complete end-to-end methods when handling constraints: we have conducted experiments using a Safety Layer implementation [8] for the EMS and which we will denote as SAFETY-LAYER.

As discussed in Section 4, both UNIFY-SEQUENTIAL and SAFETY-LAYER can be considered instances of our UNIFY framework but they have a crucial distinction: in SAFETY-LAYER, during the projection step, it is possible to fix infeasible decisions. However, this is done in a cost-agnostic manner, and the RL agent must still produce a vector in the same space as the problem decisions. Conversely, in UNIFY-SEQUENTIAL the CO problem is capable of handling, at least partially, much of the problem elements, including the cost and constraints for a single stage. The ML model needs to guide such a problem-specific solver by means of the virtual storage costs, which can be a much simpler task for well chosen virtual parameters.

Table 8
Average optimality gap, single epoch duration and inference time to solve a problem instance on the EMS benchmark. UNIFY-SINGLE-STEP provides the best results and it outperforms the SOTA TUNING method. Conversely, while learning to satisfy the constraints RL-SAFETY-LAYER and RL provide suboptimal solutions. In terms of epoch runtime, RL is the fastest method since it does not involve solving any optimization problem. The methods that rely on the sequential formulation of the problem (RL-SAFETY-LAYER and RL-SEQUENTIAL tend to be slower since we use a larger batch size for them, as we explain in the technical details section.

Method	Opt. gap	Epoch time (s)	Inf. time (s)
UNIFY-SINGLE-STEP	**0.002 ± 0.002**	9.86 ± 5.09	0.171 ± 0.055
UNIFY-SEQUENTIAL	0.004 ± 0.003	19.28 ± 1.57	0.224 ± 0.050
TUNING	0.003 ± 0.002	–	–
RL-SAFETY-LAYER	0.118 ± 0.063	19.16 ± 15.59	0.291 ± 0.078
RL	0.117 ± 0.071	**7.05 ± 1.31**	**0.159 ± 0.039**

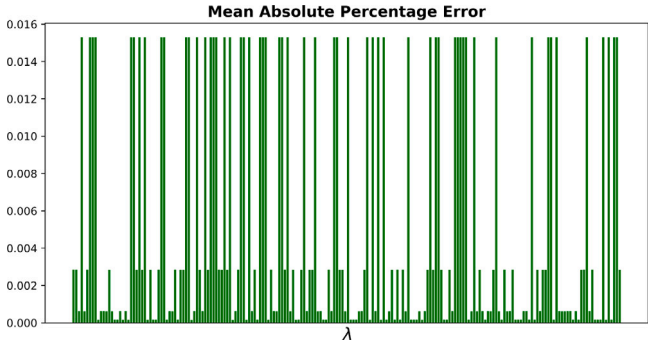


Fig. 8. The Poisson rates MAPE of the ML model employed in the predict-then-optimize approach for each of WSMC item (i.e. product).

In the upper and lower parts of Fig. 7, we respectively show the ratio to true optimal of all the approaches and the frequency of failed episodes of RL caused by constraints violations. In the first stages of training, RL fails to complete a full episode. It then start learning to satisfy the constraints but, in contrast, the cost of the solutions increases. SAFETY-LAYER converges rapidly, but the final solution cost is very similar to the one produced by RL. Also UNIFY-SEQUENTIAL rapidly converges, by also *improving the previous methods by a significant margin.*

As shown in Table 8, all the methods are comparable in terms of runtime during inference. However, RL is the fastest method since it does require to solve an optimization problem but simply to evaluate the cost function.

These experiments show that *Reinforcement Learning can benefit from a policy decomposition that properly balances optimization and learning.*

5.5. Generalization of DFL

In this set of experiments, we investigate whether UNIFY *can be used as a DFL method* compared with a predict-then-optimize method. We performed our analysis on the WSMC previously described. In this UNIFY formulation, the RL policy h is a probabilistic model, i.e. a gaussian distribution: during the estimated demands are sampled from this distribution whereas at inference we take the mean. The estimated demands are then inserted into the optimization model, and the policy π_θ is trained for the minimization of the solution cost. Differently from the UNIFY implementation, applying traditional DFL approaches is challenging due to the non-linear and non-differentiable nature of the cost function. In the predict-then-optimize approach, we train a probabilistic model to maximize the likelihood of the observed demands. We assume that this model has access to the true shape of the underlying distribution, i.e. the Poisson distribution, although the rate remains unknown. We have deliberately chosen this setup to further demonstrate the effectiveness of our UNIFY solution. During the inference step of the evaluation, we used the estimated Poisson rate as prediction.

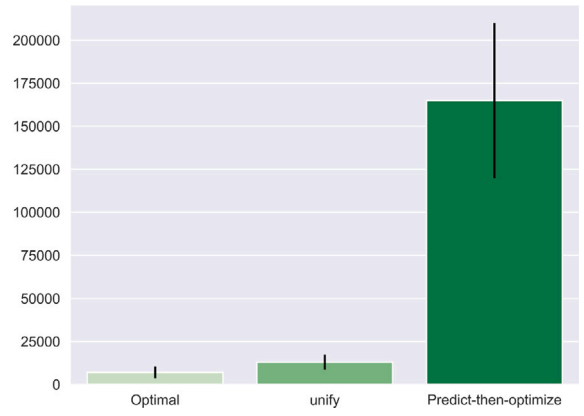


Fig. 9. This figure shows the mean cost and standard deviation for both the UNIFY and the Predict-then-optimize methods w.r.t. the optimal values.

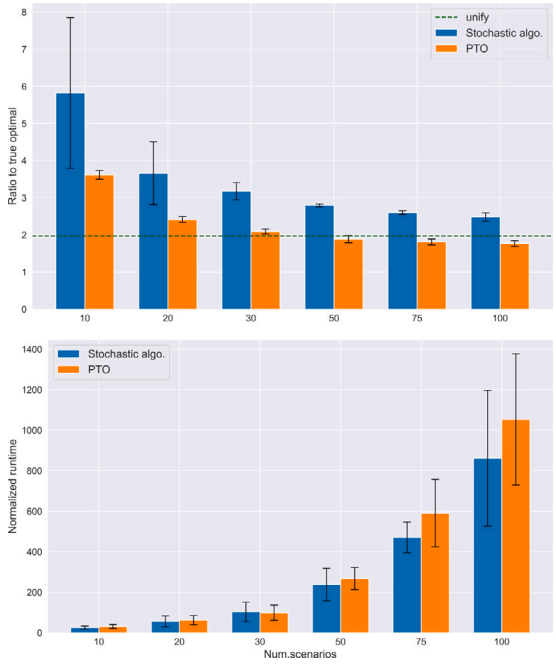


Fig. 10. Results on the WSMC problem. Upper figure: average ratio to true optimal (low is better) w.r.t. the number of sampled scenarios for UNIFY, stochastic algorithm and predict-then-optimize (PTO). UNIFY is plotted as an horizontal line it requires to sample a single virtual scenario. Lower figure: average normalized runtime (w.r.t. UNIFY) for stochastic algorithm and PTO. When increasing the number of scenarios the runtime exponentially increases, limiting the applicability of the methods.

To train and test the methods, we utilized two distinct sets of instances. As evident from Fig. 8, the Mean Absolute Percentage Error (MAPE) is low for each rate λ_i $\forall i \in I$, indicating the ML model is accurate. However, despite this accuracy, as illustrated in Fig. 9, the i.e. the solution cost, that is the true task loss, remains far from optimal. In contrast, the UNIFY implementation which is trained to directly minimize the task loss significantly outperforms the predict-then-optimize method and it is notably closer to the optimal cost. Therefore, we can conclude that UNIFY *provides a valid alternative to traditional DFL approaches*, in cases where they cannot be easily applied.

5.6. Stochastic optimization

Solving stochastic optimization problems can be incredibly challenging. As discussed in Section 4, while SAA methods are commonly

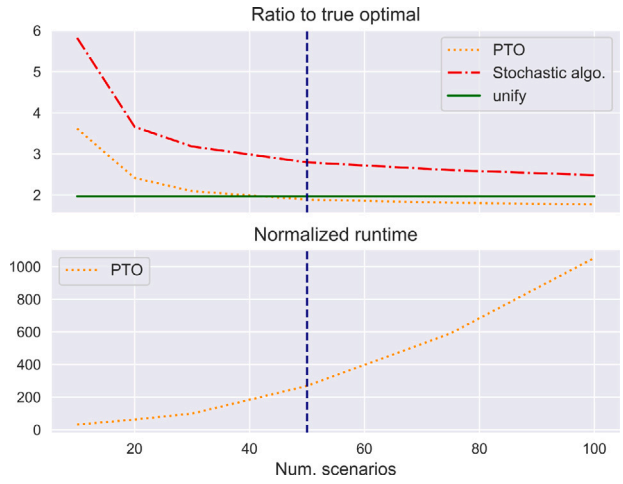


Fig. 11. Ratio to true optimal on the WSMC problem and the solution time w.r.t. the number of scenarios. The blue dashed line highlights the number of scenarios required by predict-then-optimize to beat UNIFY.

used in this domain, they can be computationally demanding. In this section, we demonstrate how UNIFY can *enhance the robustness of the downstream solver* by conducting a series of experiments on the WSMC (see Fig. 10).

As a baseline approach to ensure robustness, we employ the SAA algorithm based on Monte Carlo sampling that relies on the following optimization model:

$$\min \sum_{j\in J} c_j z_j + \frac{1}{|\Omega|} \sum_{\omega\in\Omega} \sum_{i\in I} p_i s_{i,\omega} \tag{35}$$

$$\sum_{j\in J} a_{i,j} z_j \geq y_{i,\omega}(1 - w_{i,\omega}) \qquad \forall i \in I, \omega \in \Omega \tag{36}$$

$$w_{i,\omega} = 1 \implies s_{i,\omega} \geq y_{i,\omega} - \sum_{j\in J} a_{i,j} x_j \qquad \forall i \in I, \omega \in \Omega \tag{37}$$

$$z_j \geq 0 \tag{38}$$

$$w_{i,\omega} \in [0,1] \tag{39}$$

$$s_{i,\omega} \geq 0 \tag{40}$$

$$z, w \in \mathbb{Z} \tag{41}$$

where $\omega \in \Omega$ are the sampled scenarios. If we increase Ω we also increase robustness but, at the same time, we drastically increase the computational complexity and thus reduce scalability.

In detail, we compare three methods:

- **Stochastic Optimization**: Scenarios are sampled directly from the training set. This method lacks contextual information and does not provide instance-dependent scenarios.
- **Predict-then-optimize**: This method provides instance-specific samples by querying an ML model to obtain the rate of the Poisson distribution, and then sampling from it. This approach has an advantage over simple stochastic optimization as it utilizes the ML model, but its robustness improves with an increase in the number of sampled scenarios.
- UNIFY **Implementation**: UNIFY is trained to minimize the expected cost (accounting for the recourse actions) and is inherently more robust. The actual reason is that the "single scenario" used in UNIFY is optimized to lead to the best expected behavior (in our terms, it is a *virtual* scenario, and not any real scenario).

As we mentioned in the previous section, it is worth highlighting that the comparison favors the predict-then-optimize method since

Table 9
Detailed comparison of UNIFY, stochastic algorithm and predict-then-optimize on the WSMC. As one would expected, methods that rely on sampling provide better optimality ratio with the increasing of the number of scenarios, at the price of a higher runtime. Due to computational limits, we managed to experiment with at most 100 scenarios. Predict-then-optimize (PTO) requires at least 50 scenarios to provide a better optimality ratio than UNIFY, at the cost of two orders of magnitude higher runtime. Conversely, even 100 scenarios are not enough for allowing the stochastic algorithm to improve over UNIFY.

Method	Optimality ratio	Runtime w.r.t. UNIFY
UNIFY	1.97 ± 0.08	–
Stochastic algo. (10 scenarios)	5.82 ± 2.03	25.12 ± 8.29
Stochastic algo. (20 scenarios)	3.66 ± 0.85	56.13 ± 27.55
Stochastic algo. (30 scenarios)	3.18 ± 0.23	102.90 ± 47.62
Stochastic algo. (50 scenarios)	2.79 ± 0.04	238.06 ± 80.61
Stochastic algo. (75 scenarios)	2.60 ± 0.05	471.08 ± 75.78
Stochastic algo. (100 scenarios)	2.48 ± 0.11	861.78 ± 334.81
PTO (10 scenarios)	3.62 ± 0.12	31.14 ± 9.83
PTO (20 scenarios)	2.41 ± 0.08	62.25 ± 22.96
PTO (30 scenarios)	2.09 ± 0.07	98.99 ± 37.71
PTO (50 scenarios)	1.89 ± 0.10	268.44 ± 54.90
PTO (75 scenarios)	1.81 ± 0.08	590.85 ± 165.96
PTO (100 scenarios)	$\mathbf{1.77 \pm 0.08}$	1053.32 ± 324.26

it is designed by assuming exact knowledge of the type of probability distribution whereas the UNIFY implementation makes no such assumption.

Results are shown in Fig. 11. In the upper part of the figure, we present the ratio to true optimal of the three methods on a distinct set of instances w.r.t. the number of sampled scenarios. Both the simple stochastic algorithm and predict-then-optimize approaches benefit from increasing the number of sampled scenarios. On the contrary, the implementation of UNIFY is independent of the number of sampled scenarios, as it directly predicts the demand values inserted into the optimization model (i.e. the virtual scenario). Despite the previously mentioned advantage, the predict-then-optimize approach outperforms UNIFY only when utilizing at least $\sim$ 50 scenarios in the downstream stochastic optimization model. In the lower section of the figure, we illustrate the runtime required by the predict-then-optimize method as a multiple of the runtime of UNIFY, w.r.t. to the number of scenarios. To achieve better results, we can clearly see that the predict-then-optimize approach needs more than 200 times the computation of UNIFY. Therefore, it can be concluded that a *smart implementation of* UNIFY *represents a cost-effective alternative to employing SAA methods for enhancing the robustness of the solver when tackling stochastic optimization problems* (see Table 9).

5.7. Experimental settings

Below we introduce some technical details about the experimental settings of Reinforcement Learning environments and hyperparameter settings.

5.7.1. Reinforcement learning environments

In the EMS use case we used the same environment variants as those proposed in [6]. For UNIFY-ALL-AT-ONCE, the observations are the day-ahead photovoltaic generation and electric demand forecasting and the actions are the set of virtual costs $y^{(k)}$ for all the stages $k \in \{1, \dots, n\}$ and thus the policy is a function $\pi : \mathbb{R}^{n\times 2} \to \mathbb{R}^n$. Once the whole set of $y^{(k)}$ is provided, a solution $\{x_g^{(k)}\}_{k=1}^n$ is found by solving the online optimization problem defined in Eq. (21) and the reward is the negative real cost computed as: $-\sum_{k=1}^{T}\sum_{i=2}^{m} c_i^{(k)} z_i^{(k)}$

For UNIFY-SEQUENTIAL the policy π is a function $\pi : \mathbb{R}^{n\times 3+1} \to \mathbb{R}$. At each stage k, the ML model prediction is the virtual cost $y^{(k)}$ and the corresponding online optimization problem is solved. The set of observations is the battery charge $x_{storage}^{(k)}$, the set of forecasts and a one-hot encoding of the stage k. The reward is again the negative real cost but for the only current stage k: $-\sum_{i=2}^{m} c_i^{(k)} z_i^{(k)}$

In the full end-to-end RL approach, the policy provides a $(|m|-1)$-dimensional vector corresponding to the power flows for a single stage. The output of the policy is clipped in the range $[-1, 1]$ and then each decision variable z_i is rescaled in its feasible range $[l_i, u_i]$ for $i \in \{1, \dots, m\}$. Since one of the power flows has no upper bound, we have set its value so that the power balance constraint of Eq. (21) is satisfied, reducing the actions space and making the task easier. We refer to this power flow as z_2. Despite adopting these architectural constraints, the decisions provided by the policy may still be infeasible: the storage constraint of Eq. (21) and the lower bound l_2 can be violated. The reward is non-zero solely for the last stage, calculated as the negative cumulative real cost. Given the solution cost falls within the range $[0, 3000]$, selecting infeasible actions is rewarded with a value of $-10\,000$ to encourage the search for feasible solutions. As the last detail, for all the environments described above the observations are rescaled in the range $[0, 1]$ by dividing by their maximum values.

For the production scheduling use case, the policy is a function $\pi : \mathbb{R} \to \mathbb{Z}^I$ where I is the set of the elements. Since the RL agent outputs real numbers, we convert its actions to the closest integer values. The reward is computed as the negative total cost: $-\sum_{j\in J} c_j \hat{z}_j - \sum_{i\in I} p_i \overline{y}_i$, and $\overline{d}_i = \max\left(0, y_i - \sum_{j\in J} a_{i,j} z_j\right)$ where $\hat{z}$ is the solution found using the predicted demands, $\overline{y}$ is the vector of not satisfied demands and p is the vector of penalties.

5.7.2. Hyperparameters setting

As RL algorithm we chose the Advantage Actor-Critic (A2C)[5] for its robustness and ability to handle continuous action spaces. Since hyperparameter search was outside the scope of the paper, we opted for a fairly standard architecture. The policy is modeled as a Gaussian distribution for each action dimension, parametrized by a feedforward fully-connected neural network with two hidden layers, each of 32 units and a hyperbolic tangent activation function. The critic is again a deep neural network with the same hidden architecture of the policy. Parameter updates are performed using the Adam optimizer. In the EMS use case experiments, we selected a learning rate of 0.01 (larger than usual), as it enhances convergence speed without compromising the final results.

For the predict-then-optimize approach of WSMC problem, we employ a linear regression model: the ground-truth relation is indeed a linear relation and, since the accuracy of the model is high, there is no need to overcomplicate the architecture.

For UNIFY-ALL-AT-ONCE in the EMS use case and the UNIFY implementation for the WSMC, we used a batch size of 100 whereas for UNIFY-SEQUENTIAL, RL and SAFETY-LAYER we preferred a larger batch size of 9600 to have a comparable number of episodes for each training epoch. For the EMS experiments, UNIFY-ALL-AT-ONCE, UNIFY-SEQUENTIAL, RL and SAFETY-LAYER have been trained for respectively 37, 19, 52 and 19 epochs. We chose these values because we want to provide the same computation time required by TUNING algorithm. The UNIFY implementation for the WSMC was trained for 10 000 epochs.

Experiments on the EMS were performed on a laptop with an Intel i7 CPU with 4 cores, 1.5 GHz clock frequency and 16 GB of memory. Experiments on the WSMC were conducted on a AMD EPYC 7272 16-Core Processor with 2.8 GHz clock frequency and 512 GB of memory. Despite the availability of multi-core processors, we do not exploit multi-threading and all experiments were executed on a single core to keep the setup the simplest as possible and simplify reproducibility.

[5] A2C algorithm was implemented with the TensorFlow version of the `garage` [44] library.

6. Concluding remarks

In this paper, we propose UNIFY, a flexible framework for solving CO problems with ML. In a similar spirit to ours, [45] proposes a high-level framework that alternates between solving a CSP and using ML to correct and update the CP model, with an emphasis on Human-in-the-Loop settings. Our framework is much more structured and closer to a direct implementation, with the key idea of decomposing a constrained policy in two modules and using virtual parameter to enable their communication. Our approach provides a versatile solution to address a variety of problem, while relying on standard ML and CO tools for the implementation. Our method also provides unified perspective on various approaches, which we hope will improve cross-fertilization by highlighting previously unrecognized connections. Finally, we conducted an comprehensive experimental evaluation of two practical problems to highlight the advantages of our approach.

We hope this work will also encourage future research in this direction. For instance, RL has been recently used to build a solution for combinatorial optimization problems or inside the search process of already existing solvers [19]. With some effort, it may be possible to reformulate these methods in the UNIFY framework.

While our approach has excellent inference-time scalability, the training process can be computationally expensive; this is due to the limited sample efficiency of RL algorithms [46] and the cost of each interaction with the environment, which in the trivial requires solving a CO problem. We view improving the training-time scalability of UNIFY as a key area for future research, for which actor-critic approaches or sample re-utilization solutions seem particularly promising. The versatility of the UNIFY, and the ability to choose which virtual parameters to use, also allows for some unusual design choices: for example, in problems with a complex cost functions f, one may want to use a simpler

Finally, problems characterized by decision space fluctuations based on observable variables or constraints defined over multiple steps pose challenges for traditional DFL approaches. In future works, we aim to investigate how UNIFY can also address these issues.

CRediT authorship contribution statement

Mattia Silvestri: Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Conceptualization. **Allegra De Filippo:** Writing – review & editing, Writing – original draft, Software, Methodology, Conceptualization. **Michele Lombardi:** Writing – review & editing, Writing – original draft, Supervision, Software, Methodology, Formal analysis, Conceptualization. **Michela Milano:** Writing – review & editing, Supervision, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

We shared the link with code and data in the paper.

Acknowledgments

Research partly supported by European ICT-48-2020 Project TAILOR - g.a. 952215, the Horizon Europe project TUPLES g.a. 101070149, and by PNRR - M4C2 - PE00000013 -“FAIR - Future Artificial Intelligence Research“ - Spoke 8 “Pervasive AI”, funded by the European Commission under the NextGeneration EU program.

References

[1] Y. Liu, A. Halev, X. Liu, Policy learning with constraints in model-free reinforcement learning: A survey, in: Proceedings of IJCAI, 2021, pp. 4508–4515.
[2] J. Kotary, F. Fioretto, P. Van Hentenryck, B. Wilder, End-to-end constrained optimization learning: A survey, 2021, arXiv preprint arXiv:2103.16378.
[3] M. Lombardi, M. Milano, A. Bartolini, Empirical decision model learning, Artificial Intelligence 244 (2017) 343–367.
[4] A. De Filippo, A. Borghesi, A. Boscarino, M. Milano, HADA: An automated tool for hardware dimensioning of AI applications, Knowl.-Based Syst. 251 (2022) 109199.
[5] A. De Filippo, M. Lombardi, M. Milano, Integrated offline and online decision making under uncertainty, J. Artificial Intelligence Res. 70 (2021) 77–117.
[6] M. Silvestri, A. De Filippo, F. Ruggeri, M. Lombardi, Hybrid offline/online optimization for energy management via reinforcement learning, in: Proceedings of CPAIOR, Springer, 2022, pp. 358–373.
[7] J. Mandi, J. Kotary, S. Berden, M. Mulamba, V. Bucarey, T. Guns, F. Fioretto, Decision-focused learning: Foundations, state of the art, benchmark and future opportunities, 2023, arXiv preprint arXiv:2307.13565.
[8] G. Dalal, K. Dvijotham, M. Vecerík, T. Hester, C. Paduraru, Y. Tassa, Safe exploration in continuous action spaces, CoRR abs/1801.08757, 2018, arXiv:1801.08757.
[9] E. Schede, J. Brandt, A. Tornede, M. Wever, V. Bengs, E. Hüllermeier, K. Tierney, A survey of methods for automated algorithm configuration, J. Artificial Intelligence Res. 75 (2022) 425–487.
[10] W.B. Powell, A unified framework for stochastic optimization, European J. Oper. Res. 275 (3) (2019) 795–821.
[11] W.B. Powell, S. Meisel, Tutorial on stochastic optimization in energy—Part I: Modeling and policies, IEEE Trans. Power Syst. 31 (2) (2015) 1459–1467.
[12] W.B. Powell, S. Meisel, Tutorial on stochastic optimization in energy—Part II: An energy storage illustration, IEEE Trans. Power Syst. 31 (2) (2015) 1468–1475.
[13] S.C. Graves, Uncertainty and production planning, in: Planning Production and Inventories in the Extended Enterprise: A State of the Art Handbook, Vol. 1, Springer, 2011, pp. 83–101.
[14] A. De Filippo, M. Lombardi, M. Milano, Methods for off-line/on-line optimization under uncertainty., in: Proceedings of IJCAI, 2018, pp. 1270–1276.
[15] I. Ashlagi, A.E. Roth, Kidney exchange: An operations perspective, Manage. Sci. 67 (9) (2021) 5455–5478.
[16] J. Dickerson, A. Procaccia, T. Sandholm, Dynamic matching via weighted myopia with application to kidney exchange, in: Proceedings of AAAI, Vol. 26, 2012, pp. 1340–1346.
[17] U. Ritzinger, J. Puchinger, R.F. Hartl, A survey on dynamic and stochastic vehicle routing problems, Int. J. Prod. Res. 54 (1) (2016) 215–231.
[18] K.K. Vu, C. d’Ambrosio, Y. Hamadi, L. Liberti, Surrogate-based methods for black-box optimization, Int. Trans. Oper. Res. 24 (3) (2017) 393–424.
[19] N. Mazyavkina, S. Sviridov, S. Ivanov, E. Burnaev, Reinforcement learning for combinatorial optimization: A survey, Comput. Oper. Res. 134 (2021) 105400.
[20] P. Donti, B. Amos, J.Z. Kolter, Task-based end-to-end model learning in stochastic optimization, Adv. Neural Inf. Process. Syst. 30 (2017).
[21] H. Liu, P. Grigas, Online contextual decision-making with a smart predict-then-optimize method, CoRR abs/2206.07316, 2022, arXiv:2206.07316.
[22] M.V. Pogančić, A. Paulus, V. Musil, G. Martius, M. Rolinek, Differentiation of blackbox combinatorial solvers, in: Proceedings of ICLR, 2020.
[23] B. Wilder, B. Dilkina, M. Tambe, Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization, in: Proceedings of AAAI, Vol. 33, 2019, pp. 1658–1665.
[24] J. Mandi, T. Guns, Interior point solving for LP-based prediction+optimisation, in: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), Proceedings of NeurIPS, 2020.
[25] M. Mulamba, J. Mandi, M. Diligenti, M. Lombardi, V. Bucarey, T. Guns, Contrastive losses and solution caching for predict-and-optimize, in: Z. Zhou (Ed.), Proceedings of IJCAI, ijcai.org, 2021, pp. 2833–2840.
[26] A.N. Elmachtoub, P. Grigas, Smart “predict, then optimize”, Manage. Sci. 68 (1) (2022) 9–26.
[27] X. Hu, J.C. Lee, J.H. Lee, Predict+ optimize for packing and covering LPs with unknown parameters in constraints, in: Proceedings of AAAI, Vol. 37, No. 4, 2023, pp. 3987–3995.
[28] X. Hu, J. Lee, J. Lee, Two-stage predict+ optimize for MILPs with unknown parameters in constraints, Adv. Neural Inf. Process. Syst. 36 (2024).
[29] A. Paulus, M. Rolínek, V. Musil, B. Amos, G. Martius, Comboptnet: Fit the right np-hard problem by learning integer programming constraints, in: Proceedings of ICML, PMLR, 2021, pp. 8443–8453.
[30] T.-Y. Yang, J. Rosca, K. Narasimhan, P.J. Ramadge, Projection-based constrained policy optimization, in: Proceedings of International Conference on Learning Representations, 2019.
[31] N. Parikh, S. Boyd, et al., Proximal algorithms, Found. Trends® Optimiz. 1 (3) (2014) 127–239.
[32] A. De Filippo, M. Lombardi, M. Milano, The blind men and the elephant: Integrated offline/online optimization under uncertainty., in: Proceedings of IJCAI, 2020.

[33] A.J. Kleywegt, A. Shapiro, T. Homem-de Mello, The sample average approximation method for stochastic discrete optimization, SIAM J. Optim. 12 (2) (2002) 479–502.
[34] R.M. Van Slyke, R. Wets, L-shaped linear programs with applications to optimal control and stochastic programming, SIAM J. Appl. Math. 17 (4) (1969) 638–663.
[35] G. Laporte, F.V. Louveaux, The integer L-shaped method for stochastic integer programs with complete recourse, Oper. Res. Lett. 13 (3) (1993) 133–142.
[36] P.V. Hentenryck, R. Bent, Online stochastic combinatorial optimization, The MIT Press, 2006.
[37] L. Mercier, P.V. Hentenryck, Amsaa: A multistep anticipatory algorithm for online stochastic combinatorial optimization, in: Proceedings of CPAIOR, Springer, 2008, pp. 173–187.
[38] F. Hutter, L. Xu, H.H. Hoos, K. Leyton-Brown, Algorithm runtime prediction: Methods & evaluation, Artificial Intelligence 206 (2014) 79–111.
[39] A. Biedenkapp, H.F. Bozkurt, T. Eimer, F. Hutter, M. Lindauer, Dynamic algorithm configuration: foundation of a new meta-algorithmic framework, in: Proceedings of ECAI, IOS Press, 2020, pp. 427–434.
[40] D. Aloini, E. Crisostomi, M. Raugi, R. Rizzo, Optimal power scheduling in a virtual power plant, in: Proceedings of IEEE PES ISGT, 2011, pp. 1–7.
[41] A. De Filippo, M. Lombardi, M. Milano, How to tame your anticipatory algorithm, in: Proceedings of IJCAI, 2019, pp. 1071–1077.
[42] S. Bianchi, A. De Filippo, S. Magnani, G. Mosaico, F. Silvestro, Virtus project: a scalable aggregation platform for the intelligent virtual management of distributed energy resources, Energies 14 (12) (2021) 3663.
[43] T. Grossman, A. Wool, Computational experience with approximation algorithms for the set covering problem, European J. Oper. Res. 101 (1) (1997) 81–92.
[44] The garage contributors, Garage: A toolkit for reproducible reinforcement learning research, 2019, https://github.com/rlworkgroup/garage.
[45] C. Bessiere, L. De Raedt, T. Guns, L. Kotthoff, M. Nanni, S. Nijssen, B. O'Sullivan, A. Paparrizou, D. Pedreschi, H. Simonis, The inductive constraint programming loop, IEEE Intell. Syst. 32 (5) (2017) 44–52.
[46] Y. Yu, Towards sample efficient reinforcement learning., in: Proceedings of IJCAI, 2018, pp. 5739–5743.