Hybrid Offline/Online Optimization for Energy Management via Reinforcement Learning

Hybrid Offline/Online Optimization for Energy Management via Reinforcement Learning

Mattia Silvestri, Allegra De Filippo, Federico Ruggeri, and Michele Lombardi

DISI, University of Bologna
{mattia.silvestri4,allegra.defilippo,federico.ruggeri6}@unibo.it
{michele.lombardi2}@unibo.it

Abstract. Constrained decision problems in the real world are subject to uncertainty. If predictive information about the stochastic elements is available offline, recent works have shown that it is possible to rely on an (expensive) parameter tuning phase to improve the behavior of a simple online solver so that it roughly matches the solution quality of an anticipative approach but maintains its original efficiency. Here, we start from a state-of-the-art offline/online optimization method that relies on optimality conditions to inject knowledge of a (convex) online approach into an offline solver used for parameter tuning. We then propose to replace the offline step with (Deep) Reinforcement Learning (RL) approaches, which results in a simpler integration scheme with a higher potential for generalization. We introduce two hybrid methods that combine both learning and optimization: the first optimizes all the parameters at once, whereas the second exploits the sequential nature of the online problem via the Markov Decision Process framework. In a case study in energy management, we show the effectiveness of our hybrid approaches, w.r.t. the state-of-the-art and pure RL methods. The combination proves capable of faster convergence and naturally handles constraint satisfaction.

Keywords: Deep Reinforcement Learning · Offline/Online Optimization· Uncertainty · Constrained Optimization

1 Introduction

Real world constrained decision problems *often mix offline and online elements*. In many cases, a substantial amount of information about the uncertainty (e.g. in the form of historical solutions, event logs or probability distributions) is available before it is revealed, i.e. before the online execution starts. This information generally allows to make both strategic (offline) and operational (online) decisions: in production scheduling, for example, we may devise an initial plan to be revised at run time in case of disruptions; or in Energy Management Systems (EMS) the electrical load should be planned the day ahead, while power flow balance should be maintained hour by hour.

The interplay of these offline and online phases has received attention in the last years [8]. Recent works [9, 8] show that whenever distinct offline and online

phases are present, a tighter integration can lead to substantial improvements in terms of both solution quality and computational costs. In particular, since in many application domains, efficient suboptimal algorithms for online optimization are already available or easy to design (e.g. greedy heuristics or myopic declarative models), such works exploit the available offline information to rely on a (typically expensive) parameter tuning phase to improve the behavior of the online solver, maintaining its original efficiency. [9] is based on the idea of injecting knowledge of a (convex) online approach into an offline solver. This is achieved by formulating the Karush-Kuhn-Tucker (KKT) optimality conditions for the online solver and adding them as constraints in a (offline) Mixed-Integer Programming (MIP) problem. The resulting model can be used to perform (offline expensive) parameter tuning. However, formulating optimality conditions is not trivial and requires operations research expertise. Moreover, KKT conditions introduce non-linearity to the initial model which dramatically reduces scalability. Finally, in this method, the uncertainty is managed by sampling, introducing approximations.

In this paper, we explore the idea of using learning-based approximations to lift this limitation. In particular, we employ Deep Reinforcement Learning (DRL) approaches as black-box solvers to perform (instance-specific) parameter tuning in a simpler integration scheme without requiring convexity for the online optimization problem.

We propose two hybrid approaches that combine both learning and optimization. The first one selects the parameters all at once, while the second approach exploits the sequential nature of the online problem by using the Markov Decision Process (MDP) framework.

Based on an Energy Management System case study, we show the effectiveness of our hybrid approaches, both compared to the (tuning) optimization problem from [7, 9]. To demonstrate the advantages over full RL-based solutions, we have developed and compared RL end-to-end counterparts of our proposed methods. We show that the resulting hybrid approach benefits from powerful learning algorithms and is well suited to deal with operational constraints.

The rest of the paper is organized as follows. In Section 2 we provide a brief introduction on RL. Section 3 describes the proposed case study and the state-of-the-art approach for offline/online optimization grounded on it. Section 4 presents our proposed two hybrid approaches that combine both learning and optimization. Section 5 provides an analysis of results. Section 6 discusses the main approaches proposed in the literature focused on Deep RL, hybrid methods that combine both learning and optimization, and methods for hybrid offline/online optimization. Concluding remarks are in Section 7.

2 Background

Reinforcement Learning (RL) is a paradigm to solve sequential decision-making problems, defined on top of the MDP mathematical framework. Formally, a fully-observable MDP is defined by a tuple (S, A, p, r, γ), where S is the set of

states, A is the set of actions, $p(\cdot|s,a)$ is the probability distribution of next states, $r(\cdot|s,a)$ is the probability distribution of the reward and $\gamma \in [0,1]$, called discount factor, controls the impact of future rewards.

The sequential decision making problem is then cast down to a recurrent process where the RL agent interacts with the environment by performing actions according to its behavior policy $\pi_\theta(a_t|s_t)$. Consequently, these actions provoke the agent's state transitions. Based on the action outcome, a reward signal may be attributed to the RL agent . The learning process is then formulated as the maximization problem of cumulative rewards along state-action trajectories τ, dictated by $\pi_\theta(a_t|s_t)$.

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} \gamma^t r(s_t, a_t) \right] \tag{1}$$

$$p_\theta(\tau) = p_\theta(\boldsymbol{s}_1, \boldsymbol{a}_1, \ldots, \boldsymbol{s}_T, \boldsymbol{a}_T) = p(\boldsymbol{s}_1) \prod_{t=1}^{T} \pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t) p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t) \tag{2}$$

where T is the trajectory time horizon.

RL algorithms can be classified into two main categories: model-free and model-based RL. Model-free RL algorithms try to find the optimal policy π^* such that the expected cumulative discounted reward from the initial state $s_{t=1}$ is maximized. The idea of model-based RL is to learn the model of the environment, i.e. the transition probabilities $p(\cdot|s,a)$, rather than the optimal policy and then use the learned model to choose the optimal actions.

Within the model-free family, Policy Gradient algorithms are widely used when the actions space is continuous. One such an example is REINFORCE [20]: given a parametric policy π_θ, the parameters θ are optimized by gradient ascent to directly maximize $J(\theta)$.

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) \left(\sum_{t=1}^{T} r(s_t, a_t) \right) \right] \tag{3}$$

Policy gradient algorithms are known to suffer from high variance. several non-mutually exclusive solutions can be employed to mitigate this issue, such as baseline subtraction to correctly isolate positive actions. Among the possible baselines, Actor-Critic (AC) methods are particularly effective in reducing variance. Instead of using a state-dependent baseline, one can reduce the variance by computing the advantage of taking an action a_t in state s_t. The advantage is defined as $A(s_t, a_t) = r + \gamma V(s_{t+1}) - V(s_t)$, where $V_\pi(s_t) = \mathbb{E}_\pi [J(\tau)|s = s_t]$ is the value function, r is the reward and s_{t+1} is the next state. Thus, the actor is represented by the policy, whereas the value function acts as the critic.

Modern RL approaches take advantage of deep learning models as powerful tools for representation learning [15]. More precisely, neural networks are employed to approximate the policy π_θ and $V_\theta(\cdot)$. The scientific community usually refers to this research field as Deep Reinforcement Learning (DRL).

3 Problem description

In this section, we present the details of the application scenario of an Energy Management System, and we illustrate the state-of-the-art offline/online approach grounded on it.

3.1 Energy Management Case Study

As a practical use case, we consider an Energy Management System (EMS) that requires allocating the minimum-cost power flows from different Distributed Energy Resources (DERs). The uncertainty stems from uncontrollable deviations from the planned loads of consumption and the presence of Renewable Energy Sources (RES). Based on actual energy prices and on the availability of DERs, the EMS decides: 1) how much energy should be produced; 2) which generators should be used for the required energy; 3) whether the surplus energy should be stored or sold to the energy market. Unlike in most of the existing literature, we acknowledge that in many practical cases [8] *some parameters can be tuned offline*, while the energy balance should be maintained online by managing energy flows among the grid, the renewable and traditional generators, and the storage systems. Intuitively, handling these two phases in an integrated fashion should lead to some benefits, thus making the EMS a good benchmark for our integrated approach.

In our case study, it is desirable to encourage the online heuristic to store energy in the battery system when the prices of the Electricity Market are cheap and the loads are low, in anticipation of future higher users' demand. Storing energy has no profit so the online (myopic) solver always ends up in selling all the energy on the market. However, by defining a *virtual cost* parameter related to the storage system, it is possible to associate a profit (negative cost) to storing energy, which enables addressing this greedy limitation. Then, based on day-ahead RES generation and electric demand forecasts, we can find the optimal virtual costs related to the storage system to achieve better results in terms of solution quality (management costs of the energy system).

3.2 State-of-the-art Offline/Online approach

We refer to the integrated offline/online optimization method proposed in [7, 9] that assumes *exogenous uncertainty*, and that is composed of two macro steps: an offline two-stage stochastic optimization model based on sampling and scenarios; and an online parametric algorithm, implemented within a simulator, that tries to make optimal online choices, by building over the offline decisions. The authors assume that *the online parametric algorithm is based on a convex optimization model*. Based on some configuration parameters of the online model, an offline parameter tuning step is applied. In this way, the authors can take advantage of the convexity of the online problem to obtain guaranteed optimal parameters. In particular, convexity implies that any local minimum must be a global minimum. Local minima can be characterized in terms of the KKT optimality conditions.

Essentially, *those conditions introduce a set of constraints that must be satisfied by any solution that is compatible with the behavior of the online heuristic*. They can exploit this property by formulating the tuning phase as a Mathematical Program that is not a trivial task for every constrained real-world problem.

The online step is composed by a greedy (myopic) heuristic that minimizes the cost and covers the energy demand by manipulating the flows between the energy sources. We underline that this is a typical approach to handle the online optimization of an EMS [1]. The heuristic can be formulated as an LP model:

$$\min \sum_{k=1}^{n} \sum_{g \in G} c_g^k x_g^k \tag{4}$$

$$\text{s.t. } \tilde{L}^k = \sum_{g \in G} x_g^k \tag{5}$$

$$0 \leq \gamma_k + \eta x_0^k \leq \Gamma \tag{6}$$

$$\underline{\mathrm{x}}_g \leq x_g^k \leq \overline{x}_g \tag{7}$$

For each stage k up to n, the decision variables x_g are the power flows between nodes in $g \in G$ and c_g are the associated costs. All flows must satisfy the lower and upper physical bounds $\underline{\mathrm{x}}_g$ and $\overline{x}_g$. Index 0 refers to the storage system and the index 1 to the RES generators. Hence the virtual costs associated with the storage system are c_0^k. The battery charge, upper limit and efficiency are γ, Γ and η. The EMS must satisfy the user demand at each stage k referred to as $\tilde{L}^k$.

The baseline offline problem is modeled via MIP and relies on the KKT conditions to define a model for finding the optimal values of c_0^k for the set of sampled scenarios $\omega \in \Omega$. Such model is given by:

$$\min \frac{1}{|\Omega|} \sum_{\omega \in \Omega} \sum_{g \in G} \sum_{k=1}^{n} c_g^k x_{g,\omega}^k \tag{8}$$

$$\text{s.t. } \tilde{L}_\omega^k = \sum_{g \in G} x_{g,\omega}^k \qquad \forall \omega \in \Omega, \forall k = 1, \cdots, n \tag{9}$$

$$\underline{\mathrm{x}}_g \leq x_{g,\omega}^k \leq \overline{x}_g \qquad \forall \omega \in \Omega, \forall k = 1, \cdots, n \tag{10}$$

$$0 \leq \gamma_\omega^k \leq \Gamma \qquad \forall k = 1, \cdots, n \tag{11}$$

$$\gamma_\omega^{k+1} = \gamma_\omega^k + \eta x_{0,\omega}^k \qquad \forall \omega \in \Omega, \forall k = 1, \cdots, n-1 \tag{12}$$

$$x_{1,\omega}^{k+1} = \hat{R}_k + \xi_{R,\omega}^k \qquad \forall \omega \in \Omega, \forall k = 1, \cdots, n \tag{13}$$

$$\tilde{L}_\omega^{k+1} = \hat{L}_k + y_k + \xi_{L,\omega}^k \qquad \forall \omega \in \Omega, \forall k = 1, \cdots, n \tag{14}$$

$\hat{R}_k$ and $\hat{L}_k$ are the estimated RES production and load, and ξ_R^k and ξ_L^k are the corresponding random variables representing the prediction errors. y_k are optimal load shifts and are considered as fixed parameters. The authors assume that the errors follow roughly a Normal distribution $N(0, \sigma^2)$ and that the variance σ^2 is such that 95% confidence interval corresponds to $\pm 10\%$ of the estimated

value. $\tilde{L}^k_\omega$ is the observed user load demand for stage k of the scenario ω. Equations (12) to (14) model the transition functions.

The above formulation is free to assign variables (as long as the constraints are satisfied), whereas all decisions that are supposed to be made by the heuristic can not rely on future information. We account for this limitation by introducing, as constraints, the KKT optimality conditions for our convex online heuristic. The model achieves integration at the cost of offline computation time, because of the additional variables introduced and the presence of non-linearities.

In the following we show the KKT conditions formulation for the online heuristic in a single scenario:

$$-c^k_g = \lambda^k_\omega + \mu^k_{g,\omega} - \nu^k_{g,\omega} \qquad \forall g \in G \qquad (15)$$

$$\mu^k_{g,\omega}(x^k_{g,\omega} + \overline{x}_g) = 0 \qquad \forall g \in G \qquad (16)$$

$$\nu^k_{i,\omega}(\underline{x}_g - x^t_{g,\omega}) = 0 \qquad \forall g \in G \qquad (17)$$

$$\hat{\mu}^k_\omega(\eta x^k_{0,\omega} + \gamma^k - \Gamma) = 0 \qquad (18)$$

$$\hat{\nu}^k_\omega(\eta x^k_{0,\omega} + \gamma^k) = 0 \qquad (19)$$

$$\mu^k_{g,\omega}, \nu^k_{g,\omega} \geq 0 \qquad \forall g \in G \qquad (20)$$

$$\hat{\mu}^k_\omega, \hat{\nu}^k_\omega \geq 0 \qquad (21)$$

where $\mu^k_{g,\omega}$ and $\nu^k_{g,\omega}$ are the multipliers associated to the physical flow bounds, while $\hat{\mu}^k_\omega$ and $\hat{\nu}^k_\omega$ are associated to the battery capacity bounds. Injecting the conditions in the offline model yields:

$$\min \frac{1}{|\Omega|} \sum_{\omega \in \Omega} \sum_{g \in G} \sum_{k=1}^{n} c^k_g x^k_{g,\omega}$$

s.t. Eq. (9) − (14) – *offline problem constraints* –

Eq. (15) − (21) $\forall \omega \in \Omega, \forall k = 1, \dots n$ – *KKT conditions* –

where the decision variables are $x^k_{g,\omega}$, $\mu^k_{g,\omega}$, $\nu^k_{g,\omega}$, $\hat{\mu}^k_\omega$, $\hat{\nu}^k_\omega$. To those, the authors add the cost c^k_0 associated with the flow from and to the storage system (the only parameter they allow the solver to adjust). This method allows the offline solver to associate a virtual profit for storing energy, which enables addressing the original limitation at no online computational cost.

4 Proposed Methods

Due to the limitations in terms of convexity assumption and scalability presented in Section 3, we devise an alternative to the TUNING approach of [7, 9]. Decision-focused learning approaches are not directly applicable since the cost function employed in the optimization problem takes also into account the (virtual) costs related to the storage system c^k_0, whereas the real cost to be minimized does not. In particular, we propose a hybrid method that employs DRL as a black-box

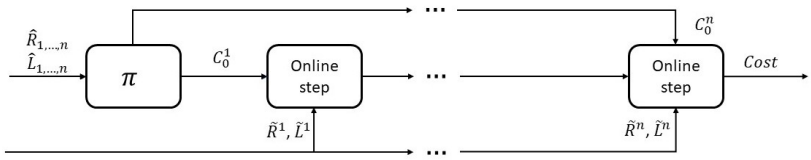

Fig. 1. In the SINGLE-STEP version, the policy π provides the set of $\{C_0^k\}_{k=1}^n$ all at once.

tool to find the optimal c_0^k. The major benefit over the TUNING version of [7, 9] is that *we do no longer require the greedy heuristic to be convex* but we are still able to compensate for its myopic behavior.

In the following sections, we will first describe our RL-based version of the TUNING algorithm. Then, to show the benefits of a hybrid approach that combines both learning and optimization, we will outline an alternative end-to-end RL method that directly provides the power flows.

4.1 RL-based TUNING

We devise two viable ways to formulate the Reinforcement Learning problem. As shown in fig. 1, in the first formulation (referred to as SINGLE-STEP), the policy $\pi : \mathbb{R}^{n\times 2} \to \mathbb{R}^n$ maps the day-ahead photovoltaic generation $\hat{R}^k$ and electric demand forecasting $\hat{L}^k$ to the set of all the virtual costs c_0^k for $k = \{1, \dots, n\}$. Once c_0^k are provided, a solution $\{x_g^k\}_{k=1}^n$ is found solving the online optimization problem defined in Equations (4) to (7) and the reward is the negative real cost computed as:

$$-\sum_{k=1}^{n}\sum_{\substack{g\in G\\ g\neq 0}} c_g^k x_g^k$$

The second formulation (referred to as MDP) exploits the sequential nature of the online step and fits the MDP framework and it is shown in fig. 2. The policy π is a function $\pi : \mathbb{R}^{n\times 3+1} \to \mathbb{R}$. The state s_k keeps track of the battery charge γ^k and it is updated accordingly to the input and output storage flows. At each stage k, the agent's action a_k is the virtual cost c_0^k and the corresponding online optimization problem is solved. Then the environment provides as observations the battery charge γ^k, the set of forecasts $\tilde{R}_{1,\dots,n}$ and $\tilde{L}_{1,\dots,n}$, and a one-hot encoding of the stage k. The reward is again the negative real cost but for the only current stage k:

$$-\sum_{\substack{g\in G\\ g\neq 0}} c_g^k x_g^k$$

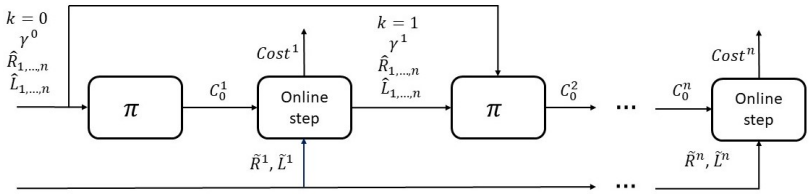

Fig. 2. In the MDP formulation, the agent sets the cost associated to the storage step-by-step, for each stage in the range from 1 to n.

The two formulations have complementary advantages and drawbacks. The single-step version is less prone to find suboptimal behaviors since it is rewarded with the actual real cost at the end of all the optimization steps. Instead, MDP receives a reward for each stage which makes it challenging to find a tradeoff between maximizing both immediate and far-in-time rewards. On the other hand, the task to be learned for MDP is simpler than for SINGLE-STEP because it has only to set one virtual cost at a time rather than deciding them all at once.

4.2 End-to-end RL

As for the hybrid approaches, we have developed both the single episode and sequential versions of the RL problem. Directly providing a feasible solution is extremely hard because the actions must satisfy all the constraints. To simplify the task, we make some architectural choices that allow for reducing the actions space.

For both the formulations, the observations are the same as for the corresponding counterparts described in section 4.1. In the version equivalent to SINGLE-STEP, the output of the policy is a vector of dimension $n \times (|G| - 1)$ corresponding to the power flows x_g^k for each stage k from 1 to n. Since one of the power flows has no upper bound $\overline{x}_g$, we have set its value so that the power balance constraint of eq. (9) is satisfied, reducing the actions space and making the task for RL easier. We refer to this decision variable as x_2^k. In the MDP version, the policy provides a $(|G| - 1)$-dimensional vector corresponding to the power flows for a single stage. The actions are clipped in the range $[-1, 1]$ and then rescaled in their feasible ranges $\left[\underline{x}_g, \overline{x}_g\right]$.

Despite adopting these architectural constraints, the actions provided by the agent may still be infeasible: the storage constraint of eq. (12) and the lower bound $\underline{x}_2$ can be violated. Since the solutions' cost is in the range $[0, 3000]$, the policy network is rewarded with a value of -10000 when infeasible actions are selected to encourage the search for feasible solutions. The full RL version of SINGLE-STEP has the same reward of SINGLE-STEP itself. Unfortunately, this approach never founds a feasible solution during training. This is reasonable since the actions space is huge and the task extremely hard. Due to its poor performance, we do not consider this method for further investigation. In the

MDP version, instead, the reward is non-zero only for the last stage and it is computed as the negative cumulative real cost. In following of the paper, we only consider this full RL method and refer to it as RL.

5 Experimental results

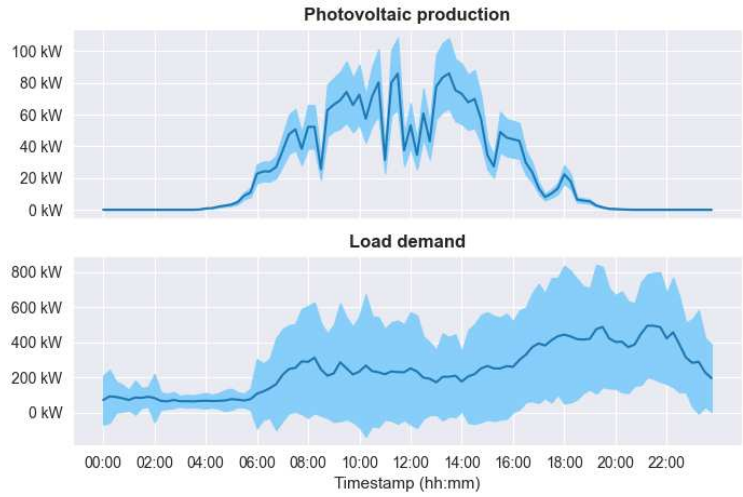

Fig. 3. Mean and standard deviation of the photovoltaic production and load demand forecasts obtained from the Public Dataset.

Training and test of the methods are performed on real data based on a Public Dataset [1]. From this dataset, we assume electric load demand and photovoltaic production forecasts, upper and lower limits for generating units and the initial status of storage units. During training of all the methods with an RL component, $\tilde{R}$ and $\tilde{L}$ are obtained from the forecasts by adding noise from a normal distribution as described in section 3.2. The dataset presents individual profiles of load demand with a time step of 5 minutes resolution from 00:00 to 23:00. We consider aggregated profiles with a timestamp of 15 minutes and use them as forecasted load. The photovoltaic production is based on the same dataset with profiles for different sizes of photovoltaic units but the same solar irradiance (i.e. the same shape but different amplitude due to the different sizes of the panels used). Also in this case photovoltaic production is adopted as forecast.

To assess the variability of the dataset, in fig. 3 we show the mean and standard deviation of photovoltaic production and user load demand regarding

[1] www.enwl.co.uk/lvns

the hour of the day. Photovoltaic production has not a high variance and this is reasonable since it mainly depends on the solar irradiance. On the other hand, the load demand is extremely variable proving the robustness of the benchmark.

The electricity demand hourly prices have been obtained based on data from the Italian national energy market management corporation[2] (GME) in €/MWh. The diesel price is taken from the Italian Ministry of Economic Development[3] and is assumed as a constant for all the time horizon (one day in our model) as assumed in literature [1] and from [11].

In the following, we will refer to the version of TUNING based on perfect information (i.e. without scenario sampling) as ORACLE. For SINGLE-STEP, MDP and RL, we have employed the Advantage Actor Critic (A2C) algorithm [4] since it is robust and it can deal with a continuous actions space. All the code and dataset to reproduce the results are publicly available at the following link [5]. Both training and evaluation were performed on a laptop with an Intel i7 CPU with 4 cores and 1.5 GHz clock frequency.

Since hyperparameter search was outside the scope of this paper, we employ a quite standard architecture. The policy is represented by 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. Parameters are updated using Adam optimizer with a learning rate of 0.01, which is larger than the usual 0.001: we choose this value because it improves the speed of convergence without compromising the final results for our use case. Observations are rescaled in the same range $[0, 1]$ dividing by their maximum values. We have used a batch size of 100 for all the methods but MDP for which we have preferred a larger batch size of 9600 to have a comparable number of episodes for each training epoch.

For the evaluation, we randomly select 100 pairs of load demand and photovoltaic production forecasts, referred to as instances in the following of the section. Each method with a learning component (i.e. SINGLE-STEP, MDP and RL) is trained on each instance individually. Here we focus on probing the effectiveness of the proposed method and we intend to investigate the generalization capabilities in future work.

5.1 Cost value over computation time

We start by comparing the mean cost on the generated realization during each training epoch as a function of the computation time, averaging the results considering the set of 100 instances. Since the optimal cost may be different among the instances, we normalize it by the best value found (i.e. the one provided by

[2] http://www.mercatoelettrico.org/En/Default.aspx
[3] http://dgsaie.mise.gov.it/
[4] A2C algorithm was implemented with the TensorFlow version of the `garage` [5] library.
[5] https://github.com/matsilv/rl-offline-online-opt

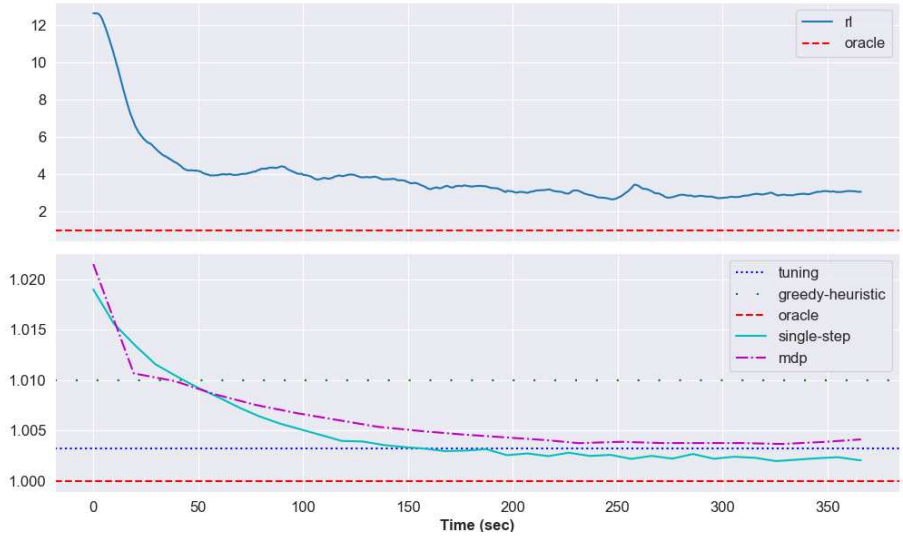

Fig. 4. Cost comparison of the methods w.r.t. the computational time.

ORACLE). To make a fair comparison with TUNING, we train the methods with a learning phase choosing a number of epochs such that the computation time is similar. The mean epoch duration on the 100 instances, the number of epochs and the total computation time required for all the methods are reported in table 1.

Table 1. Mean epoch duration, number of epochs and total duration for the methods.

Method	**Epoch duration (sec)**	**Num. of epochs**	**Total duration (sec)**
SINGLE-STEP	9.85	37	364.45
MDP	19.28	19	366.32
RL	0.33	1085	358.05
ORACLE	-	-	74.13
TUNING	-	-	360.21
HEURISTIC	-	-	0.66

In the upper part of fig. 4, RL is compared to ORACLE: despite the agent is actually minimizing the cost, it is far from being optimal. The results are so poor that we do not make a further comparison with the other methods.

In the lower part of fig. 4, our devised approaches (SINGLE-STEP and MDP) are compared with the greedy heuristic, TUNING and ORACLE. Since there is no learning for these last three methods, the solution found is used as a reference value and a simple horizontal line is plotted. Despite being extremely fast, the greedy heuristic provides considerably worse results than the oracle, due to its myopic behavior. Among our proposed methods, SINGLE-STEP provides better results and a faster convergence; in addition, it also outperforms the state-of-the-art TUNING in almost the same computation time and without requiring a convex online optimization problem. ORACLE finds the optimal solution and it is faster than our proposed methods. On the other hand, it requires perfect information so it is not applicable to real-world problems and here it is only used as a reference value to evaluate the performance of the other methods.

5.2 Decision variables

Next, we proceed by comparing the power flows and storage capacity for each method (shown in fig. 5). Since we introduce a virtual cost related to the battery system, our discussion focuses on storage usage. The end-to-end RL approach is only learning to satisfy the power balance and storage constraints and it does not take smarter actions to reduce the cost. One possible reason for this poor performance is the challenging exploration of the huge actions space. As one would expect, the greedy (and myopic) heuristic uses all the available energy in the storage and does not further charge the battery since it is not directly profitable. The hybrid approaches (SINGLE-STEP and MDP) have similar behaviors and extensively use the storage whereas TUNING focuses on the only hours close to the users demand peaks. The smartest decisions are taken by ORACLE which frequently resorts to the battery system but keeps the storage fully loaded for the first part day when the load demand is low.

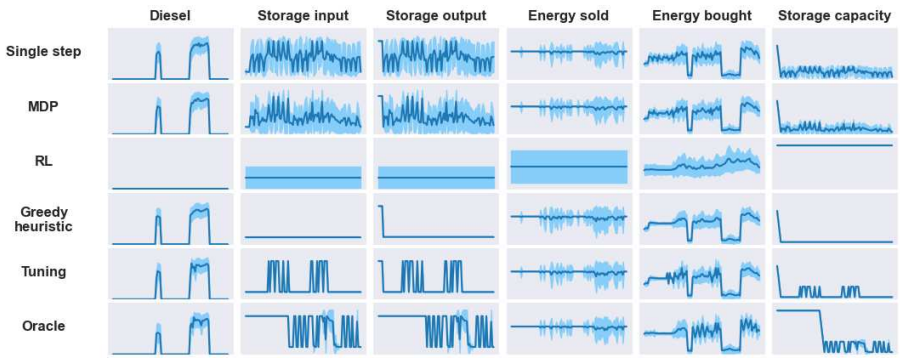

Fig. 5. Mean and standard deviation of the power flows and storage capacity w.r.t. the time for all the described methods.

6 Related Work

In this section we initially describe some recent Deep Reinforcement Learning approaches to solve combinatorial optimization problems. Then we illustrate the predict-then-optimize framework which has several properties in common with the method we have devised. The section ends with a brief overview of hybrid offline/online optimization approaches.

6.1 Deep Reinforcement Learning for Combinatorial Optimization

Recently, there has been an increasing interest in combining learning and optimization [13] with particular emphasis on DRL to solve combinatorial optimization problems [14]. Handcrafted heuristics are often used in place of exact solvers to find high-quality solutions in a reasonable time, but they require expert knowledge to be designed. Instead, DRL can learn its heuristic from a simple reward signal without any supervision. In the following, we describe the state-of-the-art methods adopting the same taxonomy proposed in [14].

Principal learning. In principal learning, the agent directly provides a solution or takes actions that are part of the solution. One of the first attempts to solve Combinatorial Optimization problems with DRL has been made in [2] and mainly addresses the Traveling Salesman Problem. In particular, a Pointer Network [19] iteratively builds a tour by choosing the most probable remaining city at each step. The network is trained with an Actor-Critic algorithm using the negative tour length as a reward. Experimental results show that this method can achieve near-optimal solutions for tours with up to 100 nodes. In [12], the authors improve the results of [2] by replacing the Pointer Network with the Transformer architecture [18]. Similarly, [16] further extends [2] to the family of Vehicle Routing Problem (VRP). In [6], the authors develop a meta-algorithm to solve combinatorial optimization problems defined over graphs. The state is a partial solution and the set of actions is represented by the set of all possible nodes that can be added.

Generally, *DRL approaches have trouble dealing with combinatorial structures*: this issue could be addressed by injecting knowledge of the online solver into the policy itself, either by making the solver part of the environment, or by using Differentiable Programming to embed the online solver in the structure of the deep neural network. In this perspective, here we take advantage of the powerful learning framework provided by RL and rely on Declarative Optimization to deal with operational constraints.

Joint training. Alternatively to principal learning, the policy can be jointly trained with an off-the-shelf solver to improve the solution quality or other performance metrics. For instance, in [3] DRL is employed as a value-selection heuristic to improve Constraint Programming searching strategy. Rather than *constructing* a solution, one can think of using RL to improve an already existing one, similarly to Local Search. For example, NeuRewriter [4] is an Actor Critic

algorithm that learns a solution improvement heuristic by iteratively re-writing part of the solution until convergence is reached. Two policies are learned simultaneously: the region-picking and rule-picking policies. The region-picking policy chooses which part of the solution should be re-written, whereas the rule-picking policy selects the re-writing rule.

Despite the RL algorithm being trained end-to-end with the online solver, our proposed method is different from the approaches described above: *the agent is not directly integrated into a step of the solutions process of a pre-defined solver*. Instead, it applies a parameters tuning phase separated from the optimization step and that guides the solver.

6.2 Predict-then-optimize

Our approach is related to the family of decision-focused learning. Many real-world problems require a predictive model whose predictions are given as input to a combinatorial optimization problem. In decision-focused learning, the training of the predictive model is improved by taking into account the solutions of the optimization problem.

One such example is the Smart "Predict, then Optimize" (SPO) framework [10]: rather than simply minimizing the prediction error, the model is trained to provide estimates such that optimal solutions are found. Training is usually performed in a supervised fashion and the major challenge of this kind of approach is finding a differentiable and computational-efficient loss function, like the SPO+ that was proposed in [10].

Our method differs from decision-focused learning since *we allow for a discrepancy between the true cost that needs to be minimized and the cost function technically employed in the optimization problem*. As an additional benefit, we do not require differentiability on the cost function. This is the reason why we adopt RL rather than a supervised method in the learning stage.

6.3 Hybrid offline/online optimization

Stochastic optimization problems are usually solved via offline or online methods. *Offline* approaches find a robust solution taking into account future uncertainty in advance but they are computationally expensive. On the other side, *online* algorithms take decisions once uncertainty is revealed but the solution quality is strictly affected by the available amount of computation time.

In many real-world cases, a large amount of information about the stochastic variables is available before the uncertainty is revealed. For example, in the energy management case study, historical data about past user demands can be used to model the uncertainty. This motivates the interest in developing hybrid offline/online approaches and taking advantage of both worlds to improve in terms of solution quality and computational cost.

If we model an n-stage stochastic optimization problem as a Markov Decision Process [17] then Dynamic Programming can be seen as a hybrid offline/online

optimization approach. The policy and its corresponding value-function are iteratively improved offline, simulating executions, and then the resulting policy can be efficiently executed online.

When fast but sub-optimal, online algorithms are available (e.g. greedy heuristic), their behavior can be improved via a parameter tuning procedure without introducing additional computational cost during the online phase.

In [7, 9], the authors propose a method to inject knowledge about a convex online solver in the offline problem. In practice, this is achieved by adding the KKT optimality conditions for the online solver as constraints in the offline problem. The method achieves positive results in cost/quality tradeoff by taking advantage of the offline/online integration. However, formulating optimality conditions is not trivial requiring experience and domain knowledge. Moreover, KKT conditions introduce non-linearity to the initial model and dramatically reduce scalability. Finally, in this method, the uncertainty is managed by sampling, introducing approximations.

The major benefit of our learning/optimization hybrid methods over the tuning version of [7, 9] is that *we do no longer require the greedy heuristic to be convex* but we are still able to compensate for its myopic behavior.

7 Conclusions

This paper makes a significant step towards hybrid learning/optimization approaches for offline/online optimization under uncertainty.

We start from a state-of-the-art offline/online optimization method that makes offline parameter tuning by relying on optimality conditions to inject knowledge of a (convex) online approach into an offline solver. Then, we propose two approaches to replace this offline parameter tuning phase, by using DRL as a black-box solver. We present two hybrid methods that combine both learning and optimization: the first one optimizes all the parameters at once, whereas the second approach exploits the sequential nature of the online problem via the MDP framework.

In a case study in energy management, we show the effectiveness of our hybrid approaches w.r.t. the state-of-the-art methods. We also experimentally assess that a full RL-based approach struggles to find feasible solutions and its performance are poor compared to the state-of-the-art and our devised methods. The combination of RL and optimization proves capable of faster convergence and naturally handles constraint satisfaction. In contrast to current state-of-the-art approaches for offline/online optimization, our hybrid method has the potential to generalize: we leave probing generalization as an open question and future research direction.

Acknowledgements

This work has been partially supported by European ICT-48-2020 Project TAILOR (g.a. 952215). We thank professor Michela Milano (University of Bologna) for the valuable discussions.

References

1. Aloini, D., Crisostomi, E., Raugi, M., Rizzo, R.: Optimal power scheduling in a virtual power plant. In: 2011 2nd IEEE PES International Conference and Exhibition on Innovative Smart Grid Technologies. pp. 1–7 (Dec 2011)
2. Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S.: Neural combinatorial optimization with reinforcement learning. arXiv preprint arXiv:1611.09940 (2016)
3. Cappart, Q., Moisan, T., Rousseau, L.M., Prémont-Schwarz, I., Cire, A.: Combining reinforcement learning and constraint programming for combinatorial optimization. arXiv preprint arXiv:2006.01610 (2020)
4. Chen, X., Tian, Y.: Learning to perform local rewriting for combinatorial optimization (2019)
5. garage contributors, T.: Garage: A toolkit for reproducible reinforcement learning research. https://github.com/rlworkgroup/garage (2019)
6. Dai, H., Khalil, E.B., Zhang, Y., Dilkina, B., Song, L.: Learning combinatorial optimization algorithms over graphs. arXiv preprint arXiv:1704.01665 (2017)
7. De Filippo, A., Lombardi, M., Milano, M.: Methods for off-line/on-line optimization under uncertainty. In: IJCAI. pp. 1270–1276 (2018)
8. De Filippo, A., Lombardi, M., Milano, M.: The blind men and the elephant: Integrated offline/online optimization under uncertainty. In: IJCAI (2020)
9. De Filippo, A., Lombardi, M., Milano, M.: Integrated offline and online decision making under uncertainty. Journal of Artificial Intelligence Research **70**, 77–117 (2021)
10. Elmachtoub, A.N., Grigas, P.: Smart "predict, then optimize". Management Science (2021)
11. Espinosa, A., Ochoa, L.: Dissemination document "low voltage networks models and low carbon technology profiles". Tech. rep., University of Manchester (June 2015)
12. Kool, W., Van Hoof, H., Welling, M.: Attention, learn to solve routing problems! arXiv preprint arXiv:1803.08475 (2018)
13. Lodi, A., Zarpellon, G.: On learning and branching: a survey. Top **25**(2), 207–236 (2017)
14. Mazyavkina, N., Sviridov, S., Ivanov, S., Burnaev, E.: Reinforcement learning for combinatorial optimization: A survey. Computers & Operations Research p. 105400 (2021)
15. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. nature **518**(7540), 529–533 (2015)
16. Nazari, M., Oroojlooy, A., Snyder, L.V., Takáč, M.: Reinforcement learning for solving the vehicle routing problem. arXiv preprint arXiv:1802.04240 (2018)
17. Puterman, M.L.: Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons (2014)
18. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems. pp. 5998–6008 (2017)
19. Vinyals, O., Fortunato, M., Jaitly, N.: Pointer networks. arXiv preprint arXiv:1506.03134 (2015)
20. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning **8**(3), 229–256 (1992)