BATCH LEARNING SDDP FOR LONG-TERM HYDROTHERMAL PLANNING

Daniel Ávila, Anthony Papavasiliou, Nils Löhndorf

REPRINT | 3221

UCLouvain

CORE | LIDAM
Louvain Institute of Data Analysis and Modeling in economics and statistics

CORE

Voie du Roman Pays 34, L1.03.01

B-1348 Louvain-la-Neuve

Tel (32 10) 47 43 04

Email:lidam-library@uclouvain.be

https://uclouvain.be/en/research-institutes/lidam/core/core-reprints.html

This article has been accepted for publication in IEEE Transactions on Power Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPWRS.2023.3246724

Batch Learning SDDP for Long-Term Hydrothermal Planning

Daniel Ávila, Anthony Papavasiliou, *Senior Member, IEEE*, and Nils Löhndorf

***Abstract*—We consider the stochastic dual dynamic programming (SDDP) algorithm - a widely employed algorithm applied to multistage stochastic programming - and propose a variant using experience replay - a batch learning technique from reinforcement learning. To connect SDDP with reinforcement learning, we cast SDDP as a Q-learning algorithm and describe its application in both risk-neutral and risk-averse settings. We demonstrate the superiority of the algorithm over conventional SDDP by benchmarking it against PSR's SDDP software using a large-scale instance of the long-term planning problem of interconnected hydropower plants in Colombia. We find that SDDP with batch learning is able to produce tighter optimality gaps in a shorter amount of time than conventional SDDP. We also find that batch learning improves the parallel efficiency of SDDP backward passes.**

***Index Terms*—Dynamic programming, parallel algorithms, stochastic optimal control, hydroelectric-thermal power generation, SDDP**

I. Introduction

STOCHASTIC dual dynamic programming (SDDP) is a popular technique for optimizing power system operations. Industrial applications include: long-term hydro-thermal planning [1]–[5], it application to power systems of various countries, e.g., Brazil [6] and Norway [7]; risk management in hydro-thermal scheduling [8]; emission constraints in hydro-thermal planning [9]; short-term dispatch [10]; demand response [11]; optimal power flow [12]; gas storage valuation [13]; hydropower expansion planning [14]; and dairy farm operations [15]. While SDDP has exhibited superior performance in solving large-scale instances of difficult optimization problems, the algorithm often also fails to converge due to computational limitations, e.g., [16], despite being able to converge theoretically [17]. Failure to converge is a problem of practical relevance in short-term planning, where solutions need to be available within a certain period in time. Parallel computing has been proposed to circumvent this problem [4], [18]–[20], as it is has shown a lot of promise for solving unit commitment problems [21], but as shown in [22], parallelization does not necessarily overcome the convergence challenges faced by the SDDP algorithm.

We propose to use experience replay - a batch learning technique that is popular within the reinforcement learning framework - to improve SDDP convergence as well as its parallel efficiency. We show how batch learning makes better use of parallel computations than conventional SDDP, and we compare it with a commercial implementation of SDDP - the PSR SDDP algorithm, on a long-term hydro-thermal scheduling. We argue that our findings open the door for a number of relevant applications in short-term planning, which are to be explored further in future work.

A. SDDP Challenges

Stochastic dual dynamic programming (SDDP) is a scalable algorithm for solving multistage stochastic programming problems. Since the seminal work of [1], SDDP has captivated the interest of the stochastic programming community and achieved widespread adoption in several industrial applications [2]–[5], [13], [15].

The SDDP algorithm works by finding supporting hyperplanes that provide tight outer approximations of the value functions of the dynamic programming formulation in regions of the state space that can be reached by the optimal policy. While the gap between the simulated cost from following the incumbent policy and the approximated cost-to-go value closes quickly for many problems, there exist a number of counter-examples where convergence stalls and the gap does not close in a reasonable amount of time. For example, gaps of nearly 22% are reported for a widely studied instance of the long-term planning problem of the Brazilian power system [16].

In order to speed up convergence, several approaches have been proposed in the extant literature. Such methods include cut selection techniques for removing redundant hyperplanes [5], [23]–[25], regularization techniques for selecting better trial points during the forward pass [26], forward sampling schemes that exploit problem structure [27], [28], as well as parallel computing [4], [18]–[20]. While all of these approaches may improve the convergence of SDDP to some extent, their individual performance is never compared to an independent benchmark. We address this shortcoming by comparing our implementation against PSR's SDDP[1] implementation on a challenging instance of the long-term hydro-thermal planning problem.

B. Parallel computing

Parallel schemes are a natural choice for countering the computational complexity of SDDP [4]. The extant literature mostly discusses parallel Monte Carlo sampling during the forward and backward passes of the algorithm [4], [18], [19], [30]. This view is somewhat limited, as we argue in the present paper. Experimental evidence suggests [3], [22], [23] that an obvious downside of increasing the number of parallel forward passes is that it often leads to an accumulation of trial points

[1]PSR is a consulting firm based in Rio de Janeiro that has pioneered the commercialization of the SDDP algorithm for hydro-thermal planning [29]

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Power Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPWRS.2023.3246724

that are similar, which in turn leads to an accumulation of redundant cuts that hardly improve the approximation but severely slow down the convergence of the algorithm.

Furthermore, past research has shown that different techniques tailored for exploiting synchronous and asynchronous parallel computing present a behaviour that is dependent on the problem and may not hold for long runs of the algorithms [20], [22]. These results have allowed us to identify that forward exploration is not sufficient for creating a highly parallelizable algorithm, but that the focus should rather be on the backward pass. This motivates the idea of being accurate during backward passes, a notion which has connections with certain ideas originating from the reinforcement learning framework.

C. Batch learning

Reinforcement learning (RL) is an area of machine learning that aims at training computational (robotic) agents in order to enable them to reach decisions in a dynamic and uncertain environment, so that these agents can maximize their rewards.

Reinforcement learning distinguishes between model-free and model-based methods. In model-free RL, the objective is to train an agent by observing its interaction with an environment for which no model exists. In model-based RL, the objective is to train an agent that interacts with a model of the environment. Since stochastic programming provides us with a model of the environment, we focus on model-based methods.

Conventional RL algorithms apply a sequential strategy that updates a decision policy as soon as new information arrives. It has been found that it can be better to delay and batch these updates so as to avoid costly matrix multiplications when updating gradients or simply to reduce noise [31]. This so-called batch learning has emerged as an attractive alternative that often outperforms other reinforcement algorithms [31]–[33].

Experience replay [32], a batch learning technique, resamples states and actions that have been visited in previous iterations, thereby replacing the old belief regarding expected cost associated with those state-action pairs with new information. In this way, the algorithm decreases the delay to revisit previously explored states, which may have a significant impact on the decision policy [32], [33]. Googles DeepMind algorithm uses experience replay in combination with Q-learning as a strategy for obtaining human-level performance in a series of Atari games [34].

We propose to use batch learning and experience replay in order to speed up learning and thereby the convergence of SDDP. The proposed algorithm applies experience replay during the backward pass, where experiences correspond to the trial points of previous iterations. The updates of these trial points can be batched and parallelized during the backward pass. We address the increase in computational burden due to the accumulation of trial points with parallel computing.

D. Organization & Contributions

In this work, (i) we introduce a novel variant of SDDP, which we refer to as *Batch Learning SDDP (BL-SDDP)* and show its connection to reinforcement learning (RL); (ii) we compare the algorithm to a widely used commercial SDDP implementation distributed by PSR Inc.; (iii) we demonstrate its suitability for parallel computing; (iv) we test its performance in both, risk neutral and averse settings.

In Section II, we formulate multistage stochastic programming problems as Markov decision processes (MDPs). We cast SDDP as a reinforcement learning algorithm, similar to Q-learning. In Section III, we introduce BL-SDDP. BL-SDDP uses experience replay, a batch learning technique from reinforcement learning, as a novel approach for accelerating the convergence of the algorithm. We describe a novel parallel scheme for BL-SDDP in Section IV. In Section V, we benchmark the new algorithm, not only against our own implementation of SDDP, but also against the commercial PSR SDDP software. The benchmark is performed against a high-dimensional, real-world instance of a hydro-thermal planning problem.

II. Problem Formulation

The SDDP algorithm is guaranteed to converge finitely [17]. Nevertheless, empirically SDDP has been observed to struggle in certain (difficult) problems. This can be related to how SDDP constructs approximations of the value function. To cope with this, we resort to batch learning through experience replay, a technique from the reinforcement learning framework, which has demonstrated to improve the learning process.

In this section, we discuss the connection between SDDP and reinforcement learning, so as to motivate our algorithmic developments. The section is divided into three subsections. The first subsection provides a brief introduction to Markov Decision Processes (MDP). The second subsection provides a link between multistage stochastic programs and MDP. In particular, in lemma 2.1, we establish that multistage stochastic programs which underlie SDDP can be cast as MDPs, which in turn underlie reinforcement learning. Last subsection presents Lemma 2.2, which demonstrates that SDDP is a reinforcement learning algorithm. This development allows us to access a wealth of algorithms from reinforcement learning that have delivered impressive performance in applications outside of power systems optimization, thereby bringing these ideas to the power systems community.

A. Markov decision processes

A Markov decision processes (MDP) is defined by the tuple $(\mathcal{S}_t, \mathcal{A}_t, C_t, P)$, where $\mathcal{S}_t$ is the set of states, $\mathcal{A}_t$ is the set of actions, $C_t : \mathcal{S}_t \times \mathcal{A}_t \to \mathbb{R}$ is the reward function of an agent, and $P(s_{t+1}|a_t, s_t)$ is the probability of transitioning to state $s_{t+1} \in \mathcal{S}_t$ if we are in state $s_t \in \mathcal{S}_t$ and select action $a_t \in \mathcal{A}(s_t)$. The set $\mathcal{A}(s_t)$ indicates the set of feasible actions when in state s_t. A policy $\pi = (\pi_1, \pi_1, \ldots)$ is a vector of functions, where each function maps states to actions[2], namely $\pi_t : \mathcal{S}_t \to \mathcal{A}_t$. We focus on finite horizon models with a time horizon T. Further details on the definition of MDPs can

[2]The literature presents more general policies, where a state is mapped to a probability measure over the set of actions. Instead, we focus on deterministic policies in this paper.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Power Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPWRS.2023.3246724

be found in [35]–[37]. The objective in MDPs is to obtain a policy $\pi \in \Pi$ that minimizes the expected cost over the decision-making horizon: $\min_{\pi\in\Pi} \mathbb{E}\Big[\sum_{t=1}^{T} C_t(s_t^\pi, a_t^\pi)\Big]$.

B. Multistage stochastic programming and MDP

We motivate the connections between multistage stochastic linear programs and MDPs using the familiar example of a two-stage hydrothermal scheduling problem.

$$\begin{aligned}
&\min\ C\cdot g_1 + \text{VOLL}\cdot ls_1 + \mathbb{E}[C\cdot g_2(\xi_2) + \text{VOLL}\cdot ls_2(\xi_2)]\\
&s.t.\ q_1 + g_1 + ls_1 = L_1\\
&\quad x_1 = X_0 + A_1 - q_1\\
&\quad q_2(\xi_2) + g_2(\xi_2) + ls_2(\xi_2) = L_2\\
&\quad x_2(\xi_2) = x_1 + A_2(\xi_2) - q_2(\xi_2)\\
&\quad g_t(\xi_t) \le \bar{G}\\
&\quad x_t(\xi_t) \le \bar{X}\\
&\quad g_t(\xi_t), x_t(\xi_t), q_t(\xi_t) \ge 0,\ \text{for all } \xi_t \in \Omega_t
\end{aligned}$$

Here, g_t is the power generated by thermal units at a marginal cost C. The thermal generators can produce up to $\bar{G}$ units of power per period. The system can shed load at a high cost $VOLL$, and ls_t is the amount of power that is curtailed from consumers. The variable q_t is the power generated from hydro units, generated at zero cost. The variable x_t represents the amount of available energy in the hydro reservoir at the end of period t. The hydro reservoir can store a maximum amount $\bar{X}$ of energy, and has as initial condition X_0. The system is subject to uncertain natural inflows represented by A_t. We assume that there are finitely many outcomes Ω_t. Note that there is a single realization $\Omega_1 = \{\xi_1\}$ in the first stage. Using standard MDP notation, we can write the problem as follows.

- The set of states are defined as

$$\begin{aligned}
&\mathcal{S}_1 = \{(X_0, \xi_1)\}\\
&\mathcal{S}_2 = \Big\{(x_1, \xi_2) :\ \text{exists } q_1, g_1 \text{ s.t. } x_1 = X_0 + A_1 - q_1\\
&\qquad\qquad q_1 + g_1 + ls_1 = L_1\\
&\qquad\qquad g_1 \le \bar{G}, x_1 \le \bar{X}, g_1, x_1, q_1 \ge 0, \xi_2 \in \Omega_2\Big\}
\end{aligned}$$

- The actions for a state $s_t = (x_{t-1}, \xi_t)$ is defined as

$$\begin{aligned}
\mathcal{A}_t(s_t) = \{(x_t, q_t, g_t, l_t) :\ & q_t + g_t + ls_t = L_t\\
& x_t = x_{t-1} + A_t(\xi_t) - q_t\\
& g_t \le \bar{G}, x_t \le \bar{X}, g_t, x_t, q_t \ge 0\}
\end{aligned}$$

We use a_t to refer to an action.

- The reward function is given by

$$C_t(s_t, a_t) = C\cdot g_1 + VOLL\cdot ls_t$$

- The probability of transitioning to state s_2 while being in state $s_1 = (X_0, \xi_1)$ and selecting action $a_t = (x_t, q_t, g_t, l_t)$ is defined as:

$$P(s_{t+1}|s_t, a_t) = \begin{cases} P(\xi_2|\xi_1) & s_2 = (x_t, \xi_{t+1})\\ 0 & \text{otherwise}\end{cases}$$

These definitions allow us to translate the multistage stochastic program into the MDP framework.

Now let us consider the general case of a multistage stochastic linear program over T stages, given by:

$$\begin{aligned}
&\min_{\substack{B_1x_0 + A_1x_1 + D_1y_1 = b_1\\ x_1, y_1 \ge 0}} u_1^T x_1 + v_1^T y_1 + \mathbb{E}\Bigg[\\
&\min_{\substack{B_2x_1 + A_2x_2 + D_2y_2 = b_2\\ x_2 \ge 0}} u_2^T x_2 + v_2^T y_2 + \mathbb{E}\Bigg[\cdots +\\
&\mathbb{E}\Bigg[\min_{\substack{B_Tx_{T-1} + A_Tx_T + D_Ty_T = b_T\\ x_T \ge 0}} u_T^T x_T + v_T^T y_T\Bigg]\Bigg]\Bigg]
\end{aligned} \tag{MSP-P}$$

For each stage, we have vectors u_t, v_t, b_t as well as matrices B_t, A_t, D_t that constitute the stochastic data process $\xi_t = (u_t, v_t, b_t, B_t, A_t, D_t)$. We assume that $u_1, v_1, b_1, B_1, A_1, C_1$ are deterministic, and that we are given an initial condition x_0. To avoid notational clutter, we drop the dependence of B_t on ω_t that indicates stochasticity of the data process. Let us assume that, at each stage, there are finitely many outcomes Ω_t, and that the data process follows a Markov chain. We therefore consider that we can define transition matrices $P(\xi_{t+1}|\xi_t)$.

Following the standard MDP description [37], we cast the MSP as an MDP by defining a tuple $(\mathcal{S}_t, \mathcal{A}_t, C_t, P)$ in the following way. To simplify the description we define the following feasibility set:

$$\begin{aligned}
\text{Feas}_t(x_{t-1}, \xi_t) = \{(x_t, y_t) \in \mathbb{R}^n :\ & \exists\, x_t, y_t \ge 0,\\
& B_t(\xi_t)x_{t-1} + A_t(\xi_t)x_t + D_t(\xi_t)y_t = b_t(\xi_t)\}
\end{aligned}$$

- **States:** The set of states is defined recursively as:

$$\begin{aligned}
&\mathcal{S}_1 = \{(x_0, \xi_1)\}\\
&\mathcal{S}_t = \Big\{(x_{t-1}, \xi_t) : \xi_t \in \Omega_t \text{ and } (x_{t-1}, y_{t-1}) \in\\
&\qquad \text{Feas}_{t-1}(x_{t-2}, \xi_{t-1}) \text{ for some } (x_{t-2}, \xi_{t-1}) \in \mathcal{S}_{t-1}\Big\}
\end{aligned}$$

- **Actions:** For each state $s_t = (x_{t-1}, \xi_t)$, the feasible actions are defined as $\mathcal{A}_t(s_t) = \text{Feas}_t(x_{t-1}, \xi_t)$. To improve readability, we define $a_t = (x_t, y_t)$ to refer to an action.
- **Reward:** The reward function is given by $C_t(s_t, a_t) = u_t(\xi_t)^T x_t + v_t(\xi_t)^T y_t$.
- **Dynamics:** The probability of transitioning to state s_{t+1} while being in state $s_t = (x_{t-1}, \xi_t)$ and selecting action $a_t = (x_t, y_t)$ is as follows:

$$P(s_{t+1}|s_t, a_t) = \begin{cases} P(\xi_{t+1}|\xi_t) & s_{t+1} = (x_t, \xi_{t+1})\\ 0 & \text{otherwise}\end{cases}$$

Furthermore, the state transition equation can be expressed as $(x_t, \xi_{t+1}) = f_t(s_t, a_t, \xi_{t+1})$.

Remark II.1. *Note that problem MSP-P is a linear program. Consequently, the optimal action is at the vertex of the feasibility set. Therefore, we can always restrict the actions to such a set, and have finitely many actions and states.*

With these definitions, we can pose the MDP problem as one of finding a policy that minimizes the expected reward, $\min_{\pi\in\Pi} \mathbb{E}\Big[\sum_{t=1}^{T} C_t(s_t^\pi, a_t^\pi)\Big]$. Let us refer to this problem,

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Power Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPWRS.2023.3246724

with the presented choice of $(\mathcal{S}_t, \mathcal{A}_t, C_t, P)$, as MDP-P. To link reinforcement learning with stochastic dual dynamic programming, we establish the following relationship.

Lemma II.2. *The problems MDP-P and MSP-P are equivalent.*

Proof. Let us consider a feasible solution of problem MSP-P. Furthermore, we can ask such a solution to be a vertex of the polyhedra defining the linear programs. Then, for every t and $\xi_t \in \Omega_t$, we have feasible values $x_t(\xi_t), y_t(\xi_t)$ and an objective cost

$$u_1^T x_1 + v_1 y_1 + \mathbb{E}\Big[u_2^T x_2 + v_2^T y_2 + \mathbb{E}\Big[\cdots + \mathbb{E}\Big[u_T^T x_T + v_T^T y_T\Big]\Big]\Big]$$

Note that we can define a policy π_t such that for $s_t = (x_{t-1}(\xi_{t-1}), \xi_t)$ we have $\pi_t(s_t) = (x_t(\xi_t), y_t(\xi_t))$. Moreover, under such a policy,

$$\mathbb{E}\Big[\sum_{t=1}^{T} C_t(s_t^\pi, a_t^\pi)\Big] = u_1^T x_1 + v_1 y_1 + \mathbb{E}\Big[u_2^T x_2 + v_2^T y_2 + \mathbb{E}\Big[\cdots + \mathbb{E}\Big[u_T^T x_T + v_T^T y_T\Big]\Big]\Big]$$

Therefore, every vertex solution of MSP-P gives a policy π which, when evaluated in MDP-P, yields the same cost. Thus, MDP-P $\leq$ MSP-P. Similarly, given any policy π, we can define a feasible solution for MSP-P which, when evaluated, yields the same result as when evaluating the policy, and so MDP-P $\geq$ MSP-P. □

C. Stochastic dual dynamic programming as a reinforcement learning algorithm

The optimal expected reward of problem MDP-P is often calculated with the help of the value functions of the problem, defined as: $V_t^\pi(s_t) = \mathbb{E}\Big[\sum_{n=t}^{T} C_n(s_n^\pi, a_n^\pi)\Big| s_t\Big]$. The value functions, after applying an expectation operator, are denoted as $\mathcal{V}_{t+1}^\pi(s_t, a_t) = \mathbb{E}\Big[V_{t+1}^\pi(s_{t+1})|s_t, a_t\Big]$. We further define Q-factors as

$$\begin{aligned} Q_t^\pi(s_t, a_t) &= \mathbb{E}\Big[\sum_{n=t}^{T} C_n(s_n^\pi, a_n^\pi)\Big| s_t, a_t\Big] \\ &= C_t(s_t, a_t) + \mathcal{V}_{t+1}^\pi(s_t, a_t) \end{aligned}$$

The last equality allows us to express the Q-factors in terms of the value functions. As we describe subsequently, the SDDP algorithm proceeds as an iterative procedure to approximate the Q-factors, using for this purpose supporting hyperplanes in order to approximate $\mathcal{V}_{t+1}^*(s_t, a_t)$ and then using the Bellman optimality equation in order to approximate the value functions of preceding stages.

As discussed in [16], the optimal expected value functions $\mathcal{V}_{t+1}^*(s_t, a_t)$ (which correspond to the expected cost-to-go functions in stochastic programming terminology) can be outer-approximated by a piecewise linear function approximation $\mathcal{V}_{t+1}(s_t, a_t)$. Such an approximation is obtained through supporting hyperplanes $\mathcal{H}$, commonly referred to as cuts. These cuts are computed by calculating the subgradient of the expected value function.

Following a similar structure as double-pass algorithms [35], we can describe SDDP as a reinforcement learning algorithm that can be used to tackle problem MDP-P.

Lemma II.3. *SDDP is a reinforcement learning algorithm used to solve MDP-P.*

Proof. The Q-factors satisfy

$$Q_t^*(s_t, a_t) = C_t(s_t, a_t) + \mathcal{V}_{t+1}^*(s_t, a_t)$$

Moreover, $\mathcal{V}_{t+1}^*$ can be approximated by building a supporting hyperplane around x_t^n [16], where $(x_t^n, y_t^n) = a_t^n$ is a trial action. Thus, we can consider an update rule that adds a supporting hyperplane around x_t^n to the current approximation of the value function, $\mathcal{V}_{t+1}$, and updates the Q-factor as $Q_t(s_t, a_t) = C_t(s_t, a_t) + \mathcal{V}_{t+1}(s_t, a_t)$. Building such a supporting hyperplane requires a model-based scheme. A detailed exposition on how the supporting hyperplane is built can be found in [16]. Let us express this update rule as $Q_t = U(a_t, Q_{t+1})$. We can now formulate the SDDP algorithm following a similar structure as the double-pass algorithms:

Algorithm 1 SDDP

INPUT: Provide an initialization of the Q-factors $Q_t^0(s_t, a_t)$ for $s_t \in \mathcal{S}_t$, $a_t \in \mathcal{A}_t(s_t)$ and $t = 1, \cdots, T$.

for $n = 1, \cdots, N$

1) **Forward Pass**: Initialize at state s_1. **for** $t = 1, \cdots, T$
   (1.1) Find the decision using the current Q-factors.
$$a_t^n \in \underset{a_t \in \mathcal{X}(s_t^n)}{\arg\min}\ Q_t^{n-1}(s_t^n, a_t)$$
   (1.2) Take action a_t^n and transition to state s_{t+1}^n.
2) **Backward Pass**: **for** $t = T, \cdots, 1$
   (2.1) Update Q_t^{n-1} using the update rule.
$$Q_t^n = U_t(a_t^n, Q_{t+1}^n)$$

return Q_t^N estimates for $t = 1, \cdots T$

□

III. Batch Learning SDDP (BL-SDDP)

The motivation of Batch Learning SDDP is to exploit parallel computing for proposing a novel and effective approach to perform backward passes, aiming at back-propagating the information accurately across stages.

The relevance of the backward pass can be understood as follows. Poor value function approximations in the last stage, which SDDP will build at early steps, will produce even looser cuts for the previous stage, with approximation errors increasing as we move backwards in time stages. The BL-SDDP algorithm addresses this drawback by allowing previously visited trial actions or experiences to have access to the value function updates carried out in later stages. The idea of re-visiting previous experiences, a technique known as experience replay, has gained popularity in the reinforcement

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Power Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPWRS.2023.3246724

learning literature due to its success in accelerating learning speed [32], [34], [38].

This section presents a novel application of the experience replay framework in the context of SDDP as a corollary of the results presented in section 2. The first subsection presents the experience replay scheme, and the second subsection introduces our novel batch learning SDDP algorithm.

A. Experience replay

The experience replay framework, introduced first by [32], aims to update Q_t using previously computed states and actions, commonly referred to as experiences. The motivation behind this is that updating a state-action pair (s_t, a_t) at stage t may affect some preceding states s_{t-1}. Nevertheless, this information will not back-propagate until state s_{t-1} is re-visited. Furthermore, states preceding s_{t-1} will need to be re-visited after the update of s_{t-1} before being able to see the update carried out in the upper layers. Therefore, the back-propagation of information is not possible unless states are re-visited. Given an arbitrary update rule $Q_t = U_t(s_t, a_t, s_{t+1}, Q_{t+1})$, the experience replay algorithm, which has been adapted for the finite horizon setting, can be described as follows [32], [34], [38].

Algorithm 2 Experience Replay

INPUT: Initialization of the Q-factors $Q_t^0(s_t, a_t)$ for $s_t \in \mathcal{S}_t$, $a_t \in \mathcal{A}_t(s_t)$ and $t = 1, \cdots, T$. For each stage, provide a set of experiences $M_t = \{(s_t^n, a_t^n, s_{t+1}^n) : n = 1, \ldots, N\}$ and let K be the batch size $K \leq |M_t|$.

for $t = T, \cdots, 1$

1) Retrieve a subset $\{(s_t^k, a_t^k, s_{t+1}^k) : k = 1, \ldots, K\} \subset M_t$.
2) **for** $k = 1, \cdots, K$ update Q_t^0 using the update rule.

$$Q_t^1 = U_t(s_t^k, a_t^k, s_{t+1}^k, Q_{t+1}^1)$$

return Q_t^1 estimates for $t = 1, \cdots T$

Note that this process can be repeated iteratively. That is to say, a set of experiences is collected, the experience replay algorithm is applied, afterwards more experiences are collected, and the experience replay algorithm is applied again. In the literature, when K equals the total set of experiences, the method is usually referred to as a full-batch update, whereas when K is less than the total size it is referred to as a mini-batch update.

B. BL-SDDP description

As presented in section II-C, the SDDP algorithm can be described as a type of double-pass algorithm, of the sort that can be encountered in the reinforcement learning literature (lemma II.3). As a consequence, we can apply known techniques for MDP algorithms, such as the experience replay scheme. This leads to the Batch Learning SDDP algorithm described as follows.

Algorithm 3 BL-SDDP

INPUT: Initialization of the Q-factors $Q_t^0(s_t, a_t)$ for $s_t \in \mathcal{S}_t$, $a_t \in \mathcal{A}_t(s_t)$ and $t = 1, \cdots, T$. Let K be the batch size, Z be the number of collected experiences before applying experience replay. Let $M = \emptyset$ be the set of experiences.

for $n = 1, \cdots, N$

1) Apply SDDP, collecting up to Z experiences, and add them to the replay memory M.
2) Consider a batch of K experiences of M. Apply experience replay on the K experiences.

return Q_t^1 estimates for $t = 1, \cdots T$

Note that, as we are using the update rule based on supporting hyperplanes, the replay memory simply requires $M = \{a_t^n : n = 1, \ldots, N\}$. The proposed scheme can also be seen as an update of cuts, which is carried out for a batch of K cuts every Z SDDP iterations. Note that the procedure can be combined with other schemes. For instance, during step 1 the forward sampling procedure could be changed to the one presented in [27], [28] in order to adjust the sampling to the problem structure.

The BL-SDDP algorithm can be described using the flow chart shown in Fig. 1. The algorithm commences by performing usual SDDP iterations, collecting the trial actions obtained during the forward passes. These trial actions are added to the replay memory. The procedure continues until Z new trial actions are added to the replay memory. Next we proceed to the experience replay scheme. This step receives as an input the replay memory, which is a collection of trial actions, and builds a cut for a batch of these trial actions. As an output of this step, we obtain cuts around a batch of trial actions, which can then be used as an input for SDDP.

Relative to standard SDDP, the improved performance of BL-SDDP stems from the fact that it approximates the value functions more diligently in the backward pass. It thus prevents the back-propagation of approximation errors of value functions at later stages of the problem from "contaminating" the approximations of value functions at earlier stages of the algorithm. This comes at the cost of increased computational effort at each backward pass relative to standard SDDP. We propose resorting to parallel computing in order to cope with this increased computational burden, and thus to combine the most appealing attributes of experience replay and SDDP into a single and highly parallelizable algorithmic procedure.

IV. SDDP Parallelization Strategies

The parallel SDDP literature presents parallelization schemes that are mostly focused on increasing the number of Monte Carlo forward samples and distributing these samples among the available processors [4], [18], [19], [30]. Nevertheless, these strategies may fail to scale properly as the number of Monte Carlo samples increases [3], [22], [23].

A. Standard SDDP parallelization

The common SDDP parallelization framework [4], [18], [19], [30] can be described as follows:

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Power Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPWRS.2023.3246724

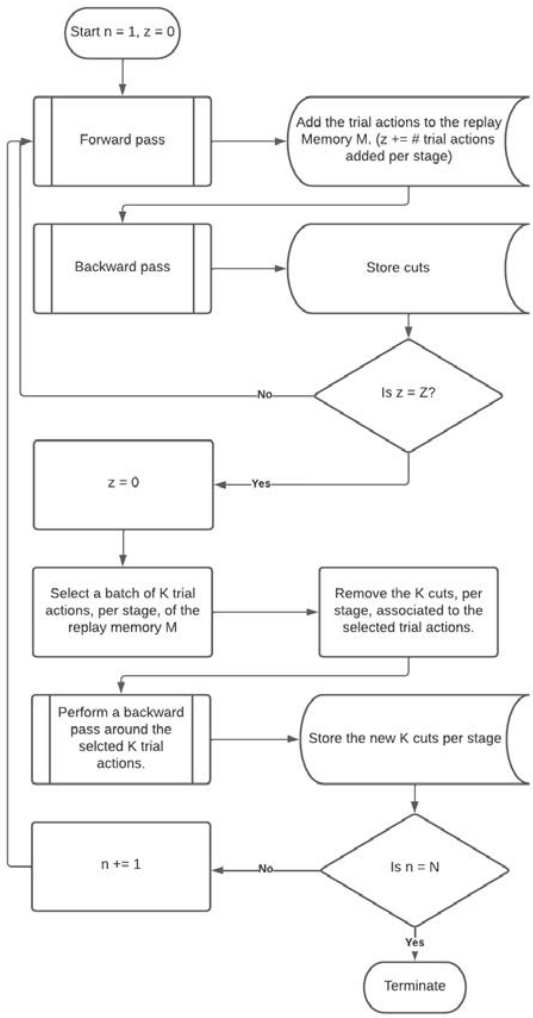

Fig. 1. Flow chart describing the BL-SDDP scheme. The algorithm commences by performing usual SDDP iterations, until Z trial actions or experiences are collected. A batch of K trial actions is selected and the cuts around these trial actions are updated.

- Forward pass: The forward pass consists of N Monte Carlo samples. Each processor computes a different sample, thereby producing trial points $x_1^n, \cdots, x_T^n$, for $n = 1, \cdots, N$.
- Backward pass: At stage t, the n-th processor generates a cut for the expected value functions of stage $t-1$ at point x_{t-1}^n.

In Fig. 2 we depict this procedure graphically over a lattice. The x-axis represents the elapsed computing time. In the forward pass the red CPU draws a sample implied by the lattice distribution. The red CPU then proceeds to solve the node problems and thus produces trial actions. The red CPU then proceeds to the backward pass. During the pass, the cuts are built around the trial actions found by the red CPU. The blue CPU follows a similar sequence of steps.

The communication between processors in this scheme has

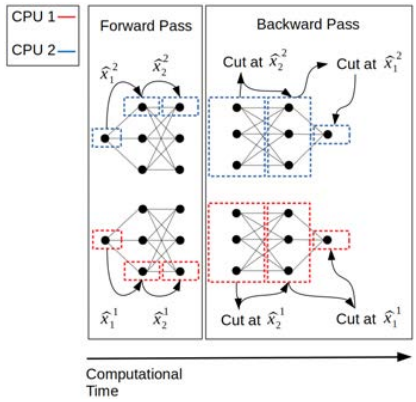

Fig. 2. Standard parallel SDDP Scheme. The forward pass consists of several sample paths, which are distributed among the available processors. The backward pass distributes the trial points that are obtained in the forward pass among the processors.

commonly been synchronous, and is discussed in a number of publications [1], [4], [18]–[20]. Synchronization is included at each stage of the backward pass. Thus, at the end of each stage the cuts are shared among the processors.

The literature has proposed certain variants in the communication between processors. In [19] the authors present a relaxation of the synchronization points, whereby a processor waits for only a subset of processors before proceeding with the next stage of the backward pass. The authors describe the benefits of their proposal relative to a fully synchronous version. In [30] the authors propose an asynchronous version, whereby a processor does not wait for other processors to compute a cut before proceeding to the next stage in the backward pass. Nevertheless, the advantages of such an approach are not clearly demonstrated. As past research has shown, different choices of Monte Carlo sampling may produce different results in the performance of the algorithm [3], [23]. Moreover, as observed by [20], an increase in Monte Carlo samples has the disadvantage of increasing the workload of the problem. At each iteration, the algorithm requires computing more samples, thereby resulting in instances where more CPUs imply a greater workload.

The parallelization strategy described in [20] differs notably from the aforementioned parallel schemes, which are based on increasing the Monte Carlo samples in order to exploit parallelism. In [20], the authors propose a scheme whereby a processor is attached to a stage and builds cuts for the given stage. Thus, there is no actual backward pass. Instead, all the stages of the backward pass are calculated simultaneously. The authors report advantages relative to the standard parallel SDDP scheme for certain instances. Other instances present a similar behaviour to the standard SDDP algorithm.

As the parallel scheme described in Fig. 2 is the most commonly adopted strategy for parallelizing SDDP, we use it as a benchmark for the parallel strategy that we develop in the next subsection.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Power Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPWRS.2023.3246724

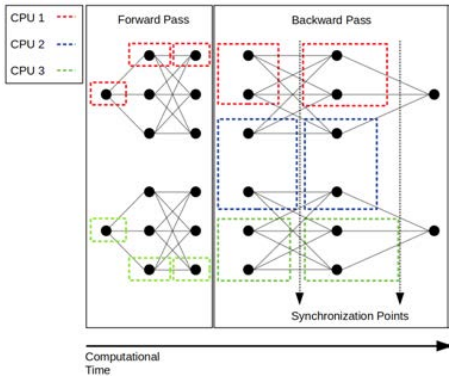

Fig. 3. Parallelization of the BL-SDDP algorithm. The SDDP iterations are parallelized by considering a small number of samples in the forward pass, which are then distributed among the available processors. In the backward pass, the problems at each stage are distributed among the processors.

B. Parallelization of BL-SDDP

We propose a novel parallelization scheme for SDDP based on the BL-SDDP algorithm that we propose in section III. The developed BL-SDDP algorithm is divided into two parts. The first part is a common SDDP run. The second part employs the experience replay framework. We propose a synchronous parallel computing strategy for each one of these parts.

• **SDDP parallelization.** As we have discussed, the standard parallelization strategy for SDDP can result in inferior performance. In order to avoid a possible deterioration in performance, our proposed parallelization proceeds by fixing a small number of samples in the forward pass. We illustrate the parallelization strategy through the example depicted in Fig. 3. The forward pass consists of 2 samples. Each processor computes one sample. Note that the blue CPU remains idle at this point. There is a synchronization point at the end of the forward pass. During the backward pass, at every stage, there are 6 problems, since there are 2 samples and 3 nodes per stage in the lattice. These 6 problems are then distributed among the available processors. Once the problems of the stage have been computed, the processors synchronize, the cuts are built, and the obtained cuts are shared among the available processors. Note that, as presented in the picture, the scheme has several synchronization points.

• **BL parallelization**. Let us assume that the replay memory is given by $M_t = \{a_t^n : n = 1, \ldots, N\}$ where $a_t^n = (x_t^n, y_t^n)$ is an action. Let us introduce the parallelization through the example shown in Fig. 4. Let us assume we have a batch of size 5. The algorithm starts by selecting a batch of 5 experiences (trial actions), for each stage, among the collected experiences in the replay memory. These selected experiences are distributed among the available processors. Then, proceeding backwards in time, each processor updates the Q-factors around the corresponding actions. That is to say, each processor builds supporting hyperplanes for the expected value functions around the experiences that it receives. Note that, at the end of each stage, the processors synchronize in order to receive the cuts obtained by other processors. This procedure naturally scales up with the introduction of additional CPUs, since additional processors imply that each processor will have a smaller set of experiences.

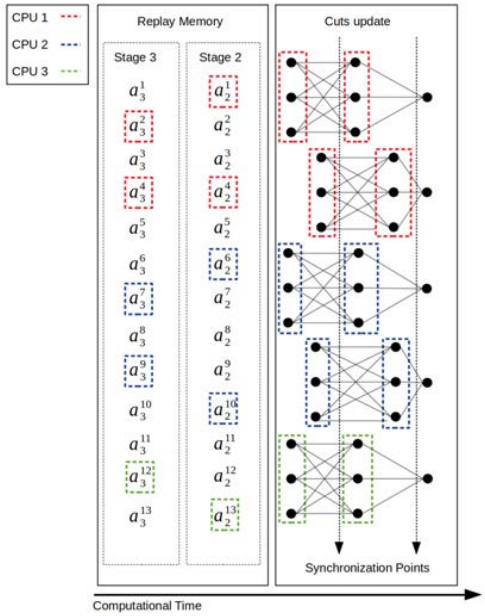

Fig. 4. Parallelization of the BL-SDDP algorithm. The BL steps are parallelized by distributing the trial actions of the replay memory among the available processors. The processors update the cuts around those trial actions. .

V. Case Studies

We carry out computational experiments over a realistic instance of a long-term planning problem for a network of hydropower plants in Colombia. We further analyze the impact of the methodology on an instance of long-term hydro-thermal planning from Brazil that is well-know for its difficulty. Our experimental results can be summarized as follows: (i) The BL-SDDP algorithm is able to produce tighter gaps in less time, compared to the PSR SDDP commercial implementation. (ii) The BL-SDDP parallel scheme is better suited for parallel computing, and responds more favorably to an increase in parallel computing capacity than standard SDDP. (iii) The superior performance of BL-SDDP can be observed in both risk-neutral and risk-averse formulations of multistage stochastic programming.

We proceed by briefly introducing the test cases analyzed in our paper. The SDDP literature has focused extensively on hydrothermal scheduling problems due to their practical relevance [1], [4], [16], [23], [39]. The objective of the problem is to determine optimal storage levels for the hydro reservoirs of a power system under inflow uncertainty, while respecting operational constraints and satisfying demand, in such a way that the total expected operational costs of the system are minimized. The mathematical description of the problem can be found in the Appendix.

The first considered test case is an instance of the Colombian power system that has been provided by PSR. The case study comprises 32 thermal plants and 42 hydro plants, 25

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Power Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPWRS.2023.3246724

of which have storage capacity. The case study considers a river network that consists of a 54-dimensional inflow vector. Since transmission network data was not available to us, it is not considered. The instance considers long-term planning and exhibits a time horizon of 10 years and monthly steps, i.e., 120 stages in total. Inflow uncertainty is modeled using PSR's SDDP software that fits a periodic autoregressive (PAR) model to historical inflow data. We follow the state space expansion discussed in [16] who propose to add equality constraints into the problem. Historical inflow data from 1937 to 2019 is used to calibrate a PAR(1) model. This inflow model is then used in PSR's SDDP model as well as our own implementations, so as to arrive to a meaningful comparison. Uncertainty is represented by drawing a sample of 100 realizations of the error terms of the PAR model. The result is a high-dimensional multi-stage stochastic program. The test case is analyzed in subsections A to D.

The introduction of a transmission network increases the complexity of the problem. Therefore, we consider an additional test case, described in [16], which includes a transmission network. The case study considers an instance of the Brazilian power system, where the reservoirs have been aggregated into equivalent energy reservoirs [40]. The instance has 4 energy equivalent reservoirs. The transmission network is formulated as a transportation network. The model spans a time horizon of 10 years, again in monthly time increments. 100 uncertainty realizations are considered per stage. In [16], the users use this instance to study risk-neutral and risk-averse formulations and find that SDDP struggles to close the optimality gap in the presence of time-dependent inflows. Similar observations have been made in [39]. In addition to analyzing the impact of a transmission network, this test case allows us to test the methodology under two uncertainty assumptions: time series modelling and Markov Chain modelling. The last subsection analyzes this case study.

Our algorithms are implemented in Julia v0.6 [41] and JuMP v0.18 [42]. The chosen linear programming solver is Gurobi 8. In order to avoid confusion in the subsequent sections regarding which codes are being compared, the following nomenclature is adopted. **PSR SDDP:** This scheme refers to the PSR software. The language used to code PSR SDDP is FORTRAN and uses the XPRESS solver. **SDDP:** Refers to our base Julia SDDP implementation. **BL-SDDP:** Refers to our proposed algorithm, which we describe in section III.

A. Comparison of BL-SDDP to PSR SDDP

The present subsection aims at comparing the performance of the BL-SDDP algorithm against the PSR SDDP commercial software. The PSR SDDP commercial software is developed for solving hydrothermal scheduling problems. We conduct our comparison on the basis of an instance of the Colombian power system that has been provided to us by PSR. The experimental results for this subsection were obtained on an Intel Core i5-6198DU CPU with 2.30GHz and 8 GB of RAM, using a single CPU.

Both codes are run with the exact same parameters. Specifically, each SDDP iteration consists of 20 samples for the forward pass. Each stage consists of 100 uncertainty realizations, namely 100 nodes per stage. Regarding the BL-SDDP algorithm, we perform batch updates every 5 SDDP iterations. The batch size corresponds to a full-batch update, namely all the experiences are used for the update. The time horizon is defined to be equal to 120 stages.

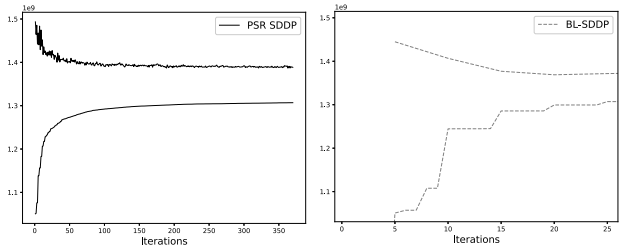

Fig. 5. Lower and Upper bound evolution over iterations for the Colombian hydrothermal test case.. Left panel presents the PSR SDDP software while right panel presents the BL-SDDP algorithm.

Fig. 5 presents the convergence behavior for each of the two algorithms. Panel (a) presents the evolution of the lower and upper bound over iterations for the PSR software, while panel (b) presents the evolution for our BL-SDDP code. The upper bound estimate presented in panel (a) is the one reported by the PSR software. In our approach, in order to provide reliable estimates, we estimate the upper bound every 5 iterations by simulating our current policy over a large collection of samples, concretely 4000 inflow samples. Note that the PSR SDDP software reaches a steady state behaviour after approximately 150 iterations, at which point the difference between the lower and upper bound remains relatively constant. On the other hand, the BL-SDDP approach is able to reduce the difference between upper and lower bound significantly after each batch update.

Table I presents the total run time and the reported gap after 34 hours of run time. Note that the BL-SDDP algorithm is able to produce a better gap. The table also presents the mean cost of an out-of-sample evaluation of both policies. Concretely, 2000 out-of-sample inflow samples are generated using PSR software. The policies are then tested against these inflows. As we can observe, the improved gap of the BL-SDDP algorithm also results in a superior out-of-sample performance relative to the policy generated by the PSR SDDP commercial software.

TABLE I
COMPARISON OF THE POLICIES AFTER 34 HOURS OF RUN TIME.

	PSR SDDP	BL-SDDP
Reported Gap (%)	6.2	4.2
Time (h)	34 (1 CPU)	34 (1 CPU)
Out-Of-Sample Inflows ($)	1.437e9	1.39e9

Interestingly, such a superior performance in computing tighter optimality gaps holds even though our base SDDP implementation is considerably slower as compared to the PSR SDDP implementation. Concretely, PSR SDDP requires approximately about 30 minutes in order to compute 20 iterations

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Power Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPWRS.2023.3246724

while our base SDDP implementation requires approximately 3 hours in order to compute the same number of iterations.

The superior performance of the BL-SDDP algorithm in computing tighter optimality gaps, despite the less optimized performance of the subproblem solvers, can be understood in terms of the amount of "work" that each approach is performing, and in particular the number of linear programs that both approaches are solving. For ease of exposition, let us ignore the forward pass as it comprises a very small part of the computational effort. The problems solved when performing the backward pass can be described as follows:

(i) PSR SDDP: Evaluating a single trial point involves solving, at each stage, 100 LPs. Since the time horizon is equal to 120 stages, this amounts to a total of $119 \cdot 100$ problems. Every iteration considers 20 forward samples. Thus, 20 trial points are generated per iteration. Consequently, a total of $100 \cdot 119 \cdot 20$ problems are solved per iteration. Over 370 iterations, this amounts to a total of $100 \cdot 119 \cdot 20 \cdot 370 = 88,060,000$ problems.

(ii) BL-SDDP: The algorithm performs a total of 25 usual SDDP iterations. As explained in the previous paragraph, this results in $100 \cdot 119 \cdot 20 \cdot 25 = 5950000$ problems after the execution of the 25 usual SDDP iterations. In this case, we have to add the problems that are solved when performing the full-batch updates. After N iterations, a total of $N \cdot 20$ trial points need to be solved. As we mention previously, for each trial point a total of $119 \cdot 100$ LPs need to be solved. Thus, performing a full-batch update after N iterations would require solving $119 \cdot 100 \cdot 20 \cdot N$ LPs. As the full-batch update is performed every 5 iterations, this means $N = 5, 10, 15, 20, 25$. Thus, the full-batch update step throughout the entire execution of the algorithm requires solving $100 \cdot 119 \cdot 20 \cdot (5 + 10 + 15 + 25) = 17,850,000$ problems. Adding the full-batch update LPs and the LPs of the usual SDDP iterations results in a total of $23,800,000$ problems, which represents 27% of the problems that the PSR software is solving. In short, the difference is due to the fact that the BL-SDDP algorithm avoids over-exploring and thus avoids redundant computation in extra iterations.

As a consequence of the aforementioned observation, we additionally note that the expected value function approximations for the BL-SDDP algorithm are considerably lighter. Concretely, PSR SDDP performs 370 iterations before terminating, thus the expected value function approximation consists of approximately $370 \cdot 20 = 7,400$ cuts per stage. Instead, the BL-SDDP code requires approximately $25 \cdot 20 = 500$ cuts per stage.

B. Comparison of parallel BL-SDDP to parallel SDDP

The present subsection aims at analyzing the BL-SDDP algorithm by evaluating it against the standard SDDP scheme. For this purpose, we resort to the same base Julia implementation. The computational work is performed on the Lemaitre3 cluster of UCLouvain, which is hosted at the Consortium des Equipements de Calcul Intensif (CECI). The cluster, where the algorithms are run, consists of 80 compute nodes with two 12-core Intel SkyLake 5118 processors at 2.3 GHz and 95 GB of RAM (3970MB/core), interconnected with an OmniPath network (OPA-56Gbps).

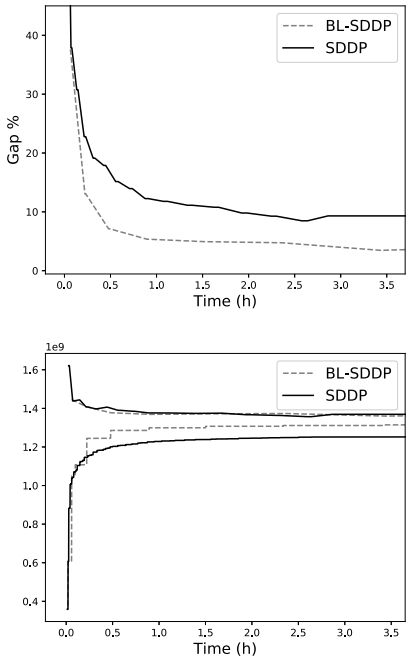

Fig. 6. Convergence evolution for the Colombian hydrothermal test case using 20 CPUs

Figure 6 present the convergence evolution when using 20 CPUs. The algorithms are set up to compute, at each iteration, 20 samples in the forward pass. The BL-SDDP algorithm performs full-batch updates every 5 iterations. The figures demonstrate that the BL-SDDP algorithm produces considerably tighter gaps throughout the execution of the algorithm.

Increasing the number of CPUs produces a similar behaviour. Fig. 7 presents the scaling of the algorithm with respect to an increasing number of CPUs. Panel (a) of the figure presents the elapsed time until achieving a certain target optimality gap in the y-axis. Panel (b) presents the obtained gap after a fixed run time in the y-axis. As we can observe, the BL-SDDP scheme is able to attain a considerable improvement relative to the standard SDDP scheme.

Fig. 8 presents the parallel efficiency results for both test cases. We can observe that both SDDP and BL-SDDP achieve a parallel efficiency approximately equal to 0.8 for the hydrothermal test case.

The literature discusses changes to the forward pass in order to improve exploration of the state space [27], [28]. As described in Section III, the BL-SDDP scheme can be easily combined with these exploration schemes. However, initial tests performed by the authors that compare and combine BL-SDDP with the schemes described in [27] and [28] showed only small improvements in performance, so that we decide not to further explore this topic. It should be noted that the scheme presented in [27] appears to be better suited for problems with many realizations per stage but only few stages, which is not the case for the studied problem instances.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Power Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPWRS.2023.3246724

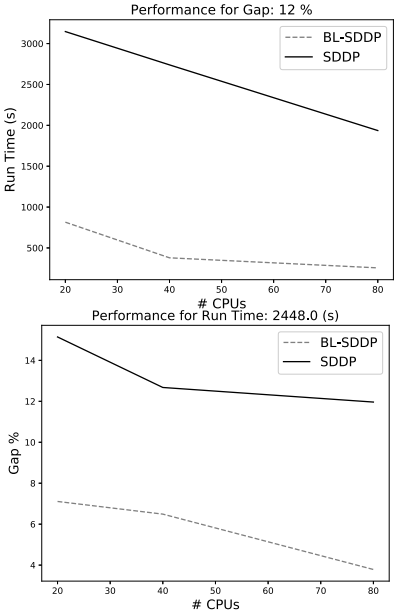

Fig. 7. CPU increase for the Colombian hydrothermal test case.

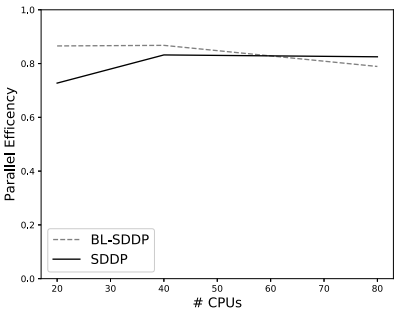

Fig. 8. Parallel efficiency for the Colombian hydrothermal problem.

C. Comparison of the value functions

In order to illustrate the differences between the value functions calculated by SDDP and BL-SDDP, we compare the expected value functions obtained through SDDP and BL-SDDP after running both codes for the same amount of time. The comparison is performed as follows. Let f_t, g_t denote the expected value function approximation obtained by BL-SDDP and SDDP respectively at stage t. At stage t, we consider the trial points obtained by SDDP and evaluate them at both f_t and g_t. The relative difference, for each trial point, is computed as

$$\frac{f_t(x) - g_t(x)}{\max\{|f_t(x)|, |g_t(x)|\}} \cdot 100$$

The relative difference at each stage is then averaged among the trial points. Note that this procedure should in principle lead to an advantage for SDDP, because we are using the trial points where the SDDP cuts are calculated.

Fig. 9 presents the results of this procedure. The x axis corresponds to the stage number and the y axis corresponds to the relative difference. The first observation is that the relative difference is a positive number for almost all stages. This

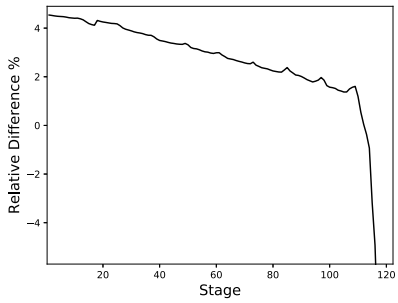

Fig. 9. Comparison of expected value functions for the Colombian hydrothermal test case. The x axis represents the current stage. The y axis represents the relative difference between the expected value functions obtained through SDDP and BL-SDDP.

indicates that the BL-SDDP algorithm is computing tighter cuts. Moreover, negative values are only encountered in few stages at the end of the time horizon. Sometimes the differences are significant. For example, in the hydrothermal test case, the relative difference in the last stage is approximately equal to -20%. However, this is expected: in the last stages the cuts produced by SDDP are tight at the trial point. Since the gap comparison is performed at the trial points obtained by SDDP, it is expected that SDDP performs better at these points (even if these points do not correspond to where an optimal policy would have "landed" at the given stage). The second observation is that the relative difference becomes greater as we move backwards in the number of stages. SDDP builds a cut for the previous stage using the current stage approximation of the expected value function approximation. A poor approximation will result in an even poorer approximation for the previous stages, thereby resulting in a back-propagation of errors. BL-SDDP is less susceptible to such an effect as the batch update results in expected value function approximations that are calculated at a large collection of trial points and therefore high-quality approximations of the expected value functions.

D. Batch Choices

The present subsection aims at providing a brief study on the effect of different batch choices. We consider the following rules for selecting a batch.

- F Corresponds to a full batch update.
- R Corresponds to a random batch.
- B Let C_i correspond to a cut calculated around x_i. Let δ_i be defined as the distance between the cut C_i and the value function at point x_i, namely:

  $$\delta_i = V(x_i) - C_i(x_i)$$

  The batch corresponds to the smallest δ_i: the best cuts.
- W Using the same notation as in previous item, the batch corresponds to the highest δ_i. These are the worst cuts.

The results are presented in Fig. 10. As in the previous experiments, 20 samples are considered in the forward pass. The batch update is performed every 5 iterations for the

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Power Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPWRS.2023.3246724

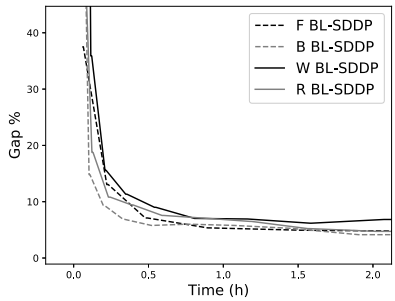

Fig. 10. Batch choices for BL-SDDP applied to the Colombian hydrothermal test case. The batch size is set up to 50%, except for the F BL-SDDP that performs a full-batch update.

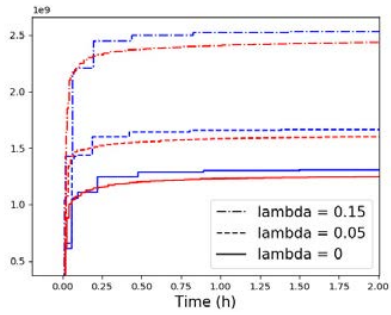

Fig. 11. Risk averse BL-SDDP applied to the Colombian hydrothermal test case. The blue color corresponds to the BL-SDDP code, while the red color corresponds to SDDP.

hydrothermal test case. The chosen batch size for batch choices R, B and W is 50%. Fig. 10 shows that the W and R strategies tend to behave the worst. On the other hand, there is a significant improvement in the B strategy at the early steps, as compared to the F strategy. Nevertheless, in the long run, both strategies behave similarly. We highlight that several alternative strategies for batch selection can be considered. Such a detailed investigation is left for future research.

E. Risk-Averse SDDP

The objective of a risk-averse model is to avoid risky decisions that would lead to high costs for certain unfavorable scenarios. We consider the risk measure [16]

$$\rho_t[Z] = (1-\lambda)\mathbb{E}_t[Z] + \lambda \text{AV@R}_\alpha[Z]$$

where $\lambda \in [0,1]$ is a weighting parameter and $\text{AV@R}_\alpha[Z]$ is the Average Value-at-Risk (also referred to as Conditional Value-at-Risk). It is defined as

$$AV@R_\alpha[Z] = V@R_\alpha[Z] + \alpha^{-1}\mathbb{E}_t[Z - V@R_\alpha[Z]]_+$$

Intuitively, $\text{AV@R}_\alpha[Z]$ is the expected value given that Z is greater than the $(1-\alpha)$-quantile. The minimization then aims at minimizing the costs found at the tail of the distribution. As stated in [16], the SDDP algorithm can be adapted to handle such a measure. Nevertheless, the algorithm only provides a lower bound approximation. Thus, stabilization criteria are used for deciding on convergence [16]. We are then interested in examining whether the BL-SDDP algorithm provides better lower bound estimates in a shorter amount of time.

Fig. 11 presents the results when employing the risk measure introduced in section II-C for different choices of weighting parameters $\lambda \in [0,1]$. As we can observe, the BL-SDDP method, which corresponds to the blue color, is consistently able to achieve a better lower bound relative to the standard SDDP method, which corresponds to the red color.

F. BL-SDDP on models with transmission network

The present subsection aims at testing the BL-SDDP algorithm on a known hard instance of the Brazilian interconnected power system, which is described in [16]. In particular, this allows us to test the effectiveness of BL-SDDP in a system with a transmission network. Inflow uncertainty is modeled as a geometric periodic autoregressive (GPAR) process [16]. Two scenarios are considered: (i) following [16], inflows are modeled as decision variables which entails additional balance equations in the optimization problem. We refer to this setting as the time series approach; (ii) following [39], the GPAR process is discretized to a Markov chain using vector quantization. We refer to this setting as the Markov Chain approach.

Figure 12 presents the convergence evolution when using 20 CPUs. The algorithms use 20 forward samples and, in the case of BL-SDDP, a full batch update is performed every 5 iterations. The upper panel presents the results for the serial independence case while the lower panel presents the results for the Markov chain case. In both cases, BL-SDDP is able to provide a notable improvement as compared to standard SDDP. As in [5], we observe that convergence of the time series approach is slower than with the Markov chain approach.

We additionally tested the effectiveness of the BL-SDDP algorithm on a realistic economic dispatch model, with a horizon of one year, for the pan-European power system. The test case considers an instance of the European Resource Adequacy Assessment (ERAA) [43] which includes 56 zones and 109 lines. The installed capacity consists of nuclear, coal, lignite, gas, oil, PV, wind and hydro. A detailed description of the capacity mix per zone can be found in [44]. The lattice has 92 stages, where each stage corresponds to 4 days of operation in 12 blocks per day time resolution, and 35 nodes per stage. After 6 hours of computation, BL-SDDP attains an optimality gap of approximately 5%, whereas the optimality gap of standard SDDP is approximately 100%. This observation indicates that our findings for the simpler Brazilian network topology carry over to power systems with more complex topologies.

VI. Conclusions

In this work, we introduce a novel variant of the SDDP algorithm that integrates ideas from reinforcement learning. The new algorithm exhibits superior performance when compared

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Power Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPWRS.2023.3246724

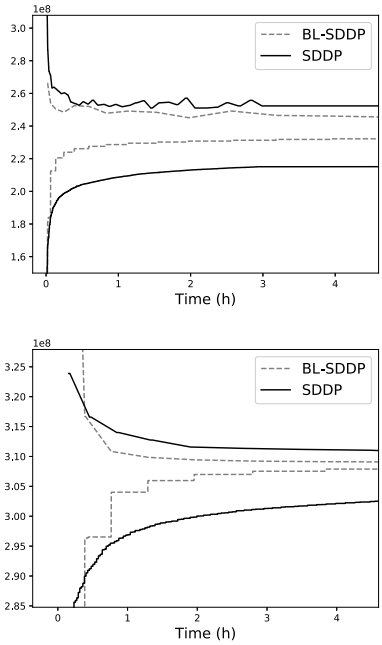

Fig. 12. Convergence evolution for the Brazilian hydrothermal test case using 20 CPUs. Upper panel shows the case with serial independence. Lower panel shows the Markov Chain setting.

with a commercial-grade SDDP implementation on a challenging problem from power systems planning. Our research opens a path for investigating whether SDDP could benefit from other reinforcement learning techniques. In particular, techniques for selecting past experiences may further improve the performance of the algorithm. We found that BL-SDDP benefits more from parallel computing than conventional SDDP. Nevertheless, as the algorithm requires synchronization, it remains susceptible to well-known synchronization issues [45].

Acknowledgments

Computational resources have been provided by the Consortium des quipements de Calcul Intensif (CCI), funded by the Fonds de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under Grant No. 2.5020.11 and by the Walloon Region. This project has received funding from the European Research Council (ERC) under the European Union Horizon 2020 research and innovation programm (grant agreement No. 850540). The authors would also like to thank Joaquim Garcia, from PSR, for providing the hydrothermal test case which is considered in this paper, as well as Mauricio Junca, from los Andes university in Colombia, for his valuable input.

Hydrothermal Scheduling Problem

The problem aims at minimizing the operational costs of thermal plants and curtailed demand by determining optimal water levels in the hydro reservoirs. Let $\mathcal{V}_{t+1}(v_t, \xi_t)$ denote the expected value function. Each stage is described in terms of the value function $V_t(v_{t-1}, \xi_t)$, which is defined by solving the following problem:

$$
\begin{aligned}
V_t(v_{t-1}, \xi_t) = \min \; & \sum_{n \in \mathcal{G}} C_n \cdot g_{n,t} + \text{VOLL} \cdot ls_t + \mathcal{V}_{t+1}(v_t, \xi_t) \\
s.t. \; & \sum_{n \in \mathcal{H}} P_n \cdot q_{t,n} + \sum_{n \in \mathcal{G}} g_{t,n} + ls_t = L_t \\
& v_t = v_{t-1} + A_t(\xi_t) + M(q_t + s_t) \\
& g_t \leq \bar{G} \\
& v_t \leq \bar{V} \\
& q_t \leq \bar{Q} \\
& g_t, v_t, q_t, s_t \geq 0
\end{aligned}
$$

Nomenclature

Markov Decision Process. $\mathcal{S}_t$: The states for stage t.
$\mathcal{A}_t$: The set of feasible actions at stage t.
C_t: The reward function at stage t.
P: Probability distribution.
π_t: A policy that maps states to actions.
T: The considered time horizon.
V_t: The value function.
$\mathcal{V}_t$: The value function after taking expectation.
$\mathcal{Q}_t$: The Q-factors.
$\mathcal{V}_t^* \backslash \mathcal{Q}_t^*$: The optimal value function $\backslash$ Q-factor.
Multistage stochastic program. x_t : The state variable.
y_t: An optimization variable.
ξ_t: The stochastic data process.
u_t, v_t, b_t: Input data vectors.
B_t, A_t, D_t: Input data matrices.
Ω_t: Uncertainty realizations at stage t.
Hydrothermal scheduling problem. $v_{t,n}$: The storage level of reservoir $n \in \mathcal{H}$.
$q_{t,n}$: Water turbined outflow of reservoir $n \in \mathcal{H}$.
$s_{t,n}$: Spilled volume of water at reservoir $n \in \mathcal{H}$.
$g_{t,n}$: Vector of thermal generated power from $n \in \mathcal{G}$.
ls_t: Variable which accounts for load shedding.
C_n: The generation cost of thermal plant $n \in \mathcal{G}$.
P_n: The energy generation coefficient for the turbined outflow for hydro plant $n \in \mathcal{H}$.
L_t: Load at stage t.
A_t: The inflow vector.
M : Matrix representing the hydrological topology.
$\bar{G}, \bar{V}, \bar{Q}$: Upper limits on the variables.
$\mathcal{H}$: The set of reservoirs.
$\mathcal{G}$: The set of thermal plants.

References

[1] M. V. Pereira and L. M. Pinto, "Multi-stage stochastic optimization applied to energy planning," *Mathematical programming*, vol. 52, no. 1-3, pp. 359–375, 1991.
[2] B. Flach, L. Barroso, and M. Pereira, "Long-term optimal allocation of hydro generation for a price-maker company in a competitive market: latest developments and a stochastic dual dynamic programming approach," *IET generation, transmission & distribution*, vol. 4, no. 2, pp. 299–314, 2010.
[3] V. De Matos, A. B. Philpott, E. C. Finardi, and Z. Guan, "Solving long-term hydro-thermal scheduling problems," Technical report, Electric Power Optimization Centre, University of Auckland, Tech. Rep., 2010.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Power Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPWRS.2023.3246724

[4] R. J. Pinto, C. T. Borges, and M. E. Maceira, "An efficient parallel algorithm for large scale hydrothermal system operation planning," *IEEE Transactions on Power Systems*, vol. 28, no. 4, pp. 4888–4896, 2013.

[5] N. Löhndorf, D. Wozabal, and S. Minner, "Optimizing trading decisions for hydro storage systems using approximate dual dynamic programming," *Operations Research*, vol. 61, no. 4, pp. 810–823, 2013.

[6] B. Gorenstin, N. Campodonico, J. da Costa, and M. Pereira, "Stochastic optimization of a hydro-thermal system including network constraints," *IEEE Transactions on Power Systems*, vol. 7, no. 2, pp. 791–797, 1992.

[7] T. Rotting and A. Gjelsvik, "Stochastic dual dynamic programming for seasonal scheduling in the norwegian power system," *IEEE Transactions on Power Systems*, vol. 7, no. 1, pp. 273–279, 1992.

[8] B. Mo, A. Gjelsvik, and A. Grundt, "Integrated risk management of hydro power scheduling and contract management," *IEEE Transactions on Power Systems*, vol. 16, no. 2, pp. 216–221, 2001.

[9] S. Rebennack, B. Flach, M. V. Pereira, and P. M. Pardalos, "Stochastic hydro-thermal scheduling under co2 emissions constraints," *IEEE Transactions on Power Systems*, vol. 27, no. 1, pp. 58–68, 2011.

[10] A. Papavasiliou, Y. Mou, L. Cambier, and D. Scieur, "Application of stochastic dual dynamic programming to the real-time dispatch of storage under renewable supply uncertainty," *IEEE Transactions on Sustainable Energy*, vol. 9, no. 2, pp. 547–558, 2017.

[11] C. Gérard, D. Ávila, Y. Mou, A. Papavasiliou, and P. Chevalier, "Comparison of priority service with multilevel demand subscription," *IEEE Transactions on Smart Grid*, 2022.

[12] A. Kiszka and D. Wozabal, "Stochastic dual dynamic programming for optimal power flow problems under uncertainty."

[13] N. Löhndorf and D. Wozabal, "Gas storage valuation in incomplete markets," *European Journal of Operational Research*, 2020.

[14] S. Rebennack, "Generation expansion planning under uncertainty with emissions quotas," *Electric Power Systems Research*, vol. 114, pp. 78–85, 2014.

[15] O. Dowson, A. Philpott, A. Mason, and A. Downward, "A multi-stage stochastic optimization model of a pastoral dairy farm," *European Journal of Operational Research*, vol. 274, no. 3, pp. 1077–1089, 2019.

[16] A. Shapiro, W. Tekaya, J. P. da Costa, and M. P. Soares, "Risk neutral and risk averse stochastic dual dynamic programming method," *European journal of operational research*, vol. 224, no. 2, pp. 375–391, 2013.

[17] A. B. Philpott and Z. Guan, "On the convergence of stochastic dual dynamic programming and related methods," *Operations Research Letters*, vol. 36, no. 4, pp. 450–455, 2008.

[18] E. L. da Silva and E. C. Finardi, "Parallel processing applied to the planning of hydrothermal systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 8, pp. 721–729, 2003.

[19] A. Helseth and H. Braaten, "Efficient parallelization of the stochastic dual dynamic programming algorithm applied to hydropower scheduling," *Energies*, vol. 8, no. 12, pp. 14 287–14 297, 2015.

[20] F. D. Machado, A. L. Diniz, C. L. Borges, and L. C. Brandão, "Asynchronous parallel stochastic dual dynamic programming applied to hydrothermal generation planning," *Electric Power Systems Research*, vol. 191, p. 106907, 2021.

[21] I. Aravena and A. Papavasiliou, "Asynchronous lagrangian scenario decomposition," *Mathematical Programming Computation*, pp. 1–50, 2020.

[22] D. Ávila, A. Papavasiliou, and N. Löhndorf, "Parallel and distributed computing for stochastic dual dynamic programming," *Computational Management Science*, pp. 1–28, 2021.

[23] V. L. De Matos, A. B. Philpott, and E. C. Finardi, "Improving the performance of stochastic dual dynamic programming," *Journal of Computational and Applied Mathematics*, vol. 290, pp. 196–208, 2015.

[24] V. Guigues, "Dual dynamic programing with cut selection: Convergence proof and numerical experiments," *European Journal of Operational Research*, vol. 258, no. 1, pp. 47–57, 2017.

[25] V. Guigues and M. Bandarra, "Single cut and multicut sddp with cut selection for multistage stochastic linear programs: convergence proof and numerical experiments," *arXiv preprint arXiv:1902.06757*, 2019.

[26] T. Asamov and W. B. Powell, "Regularized decomposition of high-dimensional multistage stochastic programs with markov uncertainty," *SIAM Journal on Optimization*, vol. 28, no. 1, pp. 575–595, 2018.

[27] C. Donohue and J. Birge, "The abridged nested decomposition method for multistage stochastic linear programs with relatively complete recourse," *Algorithmic Operations Research*, vol. 1, no. 1, pp. 20–30, 2006.

[28] M. Hindsberger and A. Philpott, "Resa: A method for solving multistage stochastic linear programs," *Journal of Applied Operational Research*, vol. 6, no. 1, pp. 2–15, 2014.

[29] PSR, "Sddp - stochastic hydrothermal dispatch with network restrictions." [Online]. Available: https://www.psr-inc.com/softwares-en/?current=p4028

[30] O. Dowson and L. Kapelevich, "SDDP.jl: a Julia package for stochastic dual dynamic programming," *Optimization Online*, 2017. [Online]. Available: http://www.optimization-online.org/DB_HTML/2017/12/6388.html

[31] S. Kalyanakrishnan and P. Stone, "Batch reinforcement learning in a complex domain," in *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, 2007, pp. 1–8.

[32] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, no. 3-4, pp. 293–321, 1992.

[33] S. Lange, T. Gabel, and M. Riedmiller, "Batch reinforcement learning," in *Reinforcement learning*. Springer, 2012, pp. 45–73.

[34] V. Mnih, K. Kavukcuoglu, and e. a. Silver, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[35] W. B. Powell, *Approximate Dynamic Programming: Solving the curses of dimensionality*. John Wiley & Sons, 2007, vol. 703.

[36] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[37] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[38] M. Pieters and M. A. Wiering, "Q-learning with experience replay in a dynamic environment," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2016, pp. 1–8.

[39] N. Löhndorf and A. Shapiro, "Modeling time-dependent randomness in stochastic dual dynamic programming," *European Journal of Operational Research*, vol. 273, no. 2, pp. 650–661, 2019.

[40] N. V. Arvanitidits and J. Rosing, "Composite representation of a multireservoir hydroelectric power system," *IEEE Transactions on Power Apparatus and Systems*, no. 2, pp. 319–326, 1970.

[41] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017. [Online]. Available: https://doi.org/10.1137/141000671

[42] I. Dunning, J. Huchette, and M. Lubin, "Jump: A modeling language for mathematical optimization," *SIAM Review*, vol. 59, no. 2, pp. 295–320, 2017.

[43] "European resource adequacy assessment. 2021 edition." https://www.entsoe.eu/outlooks/eraa/2021, accessed: 2023-01-18.

[44] "European resource adequacy assessment. 2021 edition. annex 1 assumptions." https://eepublicdownloads.azureedge.net/clean-documents/sdc-documents/ERAA/ERAA_Annex_1_Assumptions.pdf, accessed: 2023-01-18.

[45] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Prentice hall Englewood Cliffs, NJ, 1989, vol. 23.

Daniel Ávila is a PhD student at the Center for Operations Research and Econometrics (CORE), at the Universit catholique de Louvain. He received his bachelor and masters degree from the mathematics department of Universidad de los Andes, Bogot, Colombia. He works under the supervision of Professor Anthony Papavasiliou. His research interests include convex optimization, stochastic programming, renewable energy integration in power systems.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Power Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPWRS.2023.3246724

Anthony Papavasiliou (M06-SM20) is an Assistant Professor at the department of Electrical and Computer Engineering at the National Technical University of Athens in Greece. He was formerly an Associate Professor and holder of the ENGIE Chair at the Center for Operations Research and Econometrics at the Universit catholique de Louvain. He works on operations research, electricity market design, and electric power system operations. He is the recipient of the Francqui Foundation research professorship (2018), the ERC Starting Grant (2019), and the Bodossaki Foundation Distinguished Young Scientist award (2021). He has served as an Associate Editor of Operations Research and the IEEE Transactions on Power Systems.

Prof. Dr. Nils Löhndorf is Chairholder in Digital Procurement and Associate Professor at the Department of Economics and Management and the Luxembourg Centre for Logistics and Supply Chain Management within the University of Luxembourg. Prior to joining the University of Luxembourg, he was Assistant Professor at the Vienna University of Economics and Business, where he earned his habilitation. He received his PhD from the University of Vienna in Austria and a MSc in Management from the University of Mannheim in Germany. He has extensive expertise in stochastic programming and its application in management decision-making and has published his research in leading journals such as Operations Research, European Journal of Operational Researchs, and others. He is co-founder of Quantego, a company specializing in multistage stochastic programming. His models for operation, valuation, and trading of energy storage systems are used by several energy companies throughout Europe.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/