Compromise Policy for Multi-stage Stochastic Linear Programming: Variance and Bias Reduction

Jiajun Xu[1] and Suvrajeet Sen[2*]

[1]Ming Hsieh Department of Electrical and Computer Engineering, University of Southern California, University Park, Los Angeles, 90089, CA, US.

[2*]Daniel J. Epstein Department of Industrial and Systems Engineering, University of Southern California, University Park, Los Angeles, 90089, CA, US.

*Corresponding author(s). E-mail(s): suvrajes@usc.edu;
Contributing authors: jiajunx@usc.edu;

Abstract

This paper focuses on algorithms for multi-stage stochastic linear programming (MSLP). We propose an ensemble method named the "compromise policy", which not only reduces the variance of the function approximation but also reduces the bias of the estimated optimal value. It provides a tight lower bound estimate with a confidence interval. By exploiting parallel computing, the compromise policy provides demonstrable advantages in performance and stability with marginally extra computational time. We further propose a meta-algorithm to solve the MSLP problems based on the in-sample and out-of-sample optimality tests. Our meta-algorithm is incorporated within an SDDP-type algorithm for MSLP and significantly improves the reliability of the decisions suggested by SDDP. These advantages are demonstrated via extensive computations, which show the effectiveness of our approach.

Keywords: Multi-stage Stochastic Programming, Ensemble Model, Variance Reduction, Online Optimization, Distributed Computing, Bias Reduction

1 Introduction

Multistage stochastic programming (MSP) is a natural model for finding sequences of optimal decisions under evolving and uncertain conditions. The decisions are optimized in discrete time, where the decision made in one period should consider uncertainty regarding the future and how current decisions affect the future. This model has been successfully applied in a wide variety of real-world applications, ranging from financial planning [1] to production-inventory planning [2] to electric power systems [3, 4] and others. In addition, the sequential decision-making framework (under uncertainty) of MSP is closely related to dynamic programming, stochastic optimal control [5], and approximate dynamic programming (ADP) [6]. As a result, our contributions in MSP can be extended to these other areas as well.

One particular class of MSP problems is the multistage stochastic linear programming (MSLP) problem, where the objective function, constraints, and system dynamics are linear. MSLP problems are notoriously difficult to solve computationally since they suffer from the well-known "curse of dimensionality". Some such difficulties result from the fact that state and decision spaces are continuous and typically involve multi-dimensional uncertainty (i.e., vector-valued stochastic processes). Because of multi-dimensionality, the space over which the stochastic process evolves grows exponentially in the number of stages (time horizon). In general, the MSLP problem is known to be PSPACE-hard [7, 8]. It requires exponential computational effort in the number of stages to obtain accurate approximations with high probability [9]. These characteristics make such optimization problems challenging to solve computationally.

We note that one of the main sources of difficulty for multi-stage decision-making via MSLP is the inability of standard SDDP to create a multi-stage policy that is reasonably close to being optimal when the stochastic process strays from the data/scenarios that were used to create policies for the decision-problem. These difficulties are not new and have been observed by some authors [10, 11]. To the best of our knowledge, this paper is the first to provide a resolution for this conundrum.

The loss of optimality is only part of the problem. Just as importantly, the (non-optimal) policy suggested by SDDP can lead to large excursions from predicted costs, leading to much greater variability, as illustrated in this paper. Given the widespread popularity of SDDP, it is important to not only identify this issue but also to help mitigate the consequences of certain algorithmic choices commonly adopted in the MSLP literature. Our paper is devoted to providing an overarching replications-based framework intended to produce reliable policies that reduce bias and variance, thus overcoming the difficulties outlined above.

Our approach to obtaining reliable policies is to develop a meta-algorithm to solve multistage stochastic linear programming problems. We build an

ensemble model that can accommodate the most widely-used value function-based algorithms, e.g., (SDDP) [12] and stochastic dynamic linear programming (SDLP) [13]. Our meta-algorithm provides a unified framework to solve MSLP problems and provides reliable estimates of optimality through in-sample and out-of-sample stopping rules. In the process of doing so, we construct a new policy that reduces both variances as well as bias, and is referred to as the "compromise policy". The additional computational effort required to obtain a compromise policy is relatively minimal due to the method's ability to exploit parallel computing. This approach is an extension of the notion of "compromise decisions" which were developed in the context of the Stochastic Decomposition (SD) algorithm [14].

Our meta-algorithm provides a general framework to solve the MSLP problem. It not only incorporates the value-function-based algorithms (e.g., SDDP), but also improves the quality of common out-of-sample stopping rules. There are many stopping rules that have been proposed for multistage stochastic programming. The most common one is proposed in [12], which has been applied in [15, 16]. This stopping rule is equivalent to the hypothesis test, which checks whether we reject the null hypothesis (i.e., the current policy is optimal). Specifically, the rule compares whether the LB and the UB estimates are equal, which is equivalent to testing whether the confidence interval of the optimality gap contains zero. However, as argued by [17], this rule does not guarantee that the SAA problem was solved with reasonable accuracy. Shapiro [17] proposes a new criterion to check whether the upper confidence bound of the UB estimate and the lower confidence bound of the LB estimate is less than a predefined accuracy level. Moreover, [18] requires us to check the rule in [12] first and then calculate the probability of premature termination of the algorithm. If this probability is less than the tolerance, then we stop. The last two rules require additional parameters which need additional tuning to attain greater accuracy. In this paper, we will show that our proposed compromise policy improves all these stopping criteria [12, 17, 18], where it provides a tight LB estimate, as well as a low variance UB estimate.

During the course of this study, we will use a standard hydro-thermal scheduling problem from the literature, and a well-tested open-source implementation of SDDP [15], to illustrate that in comparison with the approach of this paper, the standard (single replication) version of SDDP produces relatively unreliable computational results because of it lacks the variance reduction features we introduce. Furthermore, we propose an online dual dynamic programming (ODDP) sequential sampling algorithm that only requires an oracle to generate the random samples instead of the accurate distribution of the random variables. This approach integrates several aspects of SDDP and SDLP. It allows the new meta-algorithm to integrate ODDP, in which we are able to pause the algorithm for optimality tests and resume if necessary for further refinement of the policy. Finally, we compare the performance of the one replication SDDP algorithm, our meta-algorithm with SDDP, and our meta-algorithm with ODDP. Through extensive computations, using

data with relatively long planning horizons, we demonstrate the effectiveness of our approach. Our contributions are summarized as follows:

- We propose a meta-algorithm for multi-stage stochastic linear programming, which creates a stable and high-quality policy, the compromise policy. Our meta-algorithm can incorporate most of the commonly used value function-based algorithms, such as SDDP. The additional computational time is minimal, while the improved performance is significant.
- We implement a parallel framework for SDDP-type algorithms, which optimizes multiple replications in parallel. We show that, instead of an increasing number of samples, increasing the number of replication is more efficient in identifying the optimal policy.
- We show a variance reduction property of the compromise policy in theory and in experiments. In multi-stage problems, the variability propagates across stages, while the compromise policy reduces the variance in all stages.
- In addition to reducing variance, we show a simultaneous reduction in bias due to the compromise policy, which provides a tighter estimate for the lower bound of the optimal value.
- Finally, we propose an online algorithm (ODDP) for MSLP which is suitable for the streaming-data settings. ODDP does not require users to provide the 'true' distribution of the random variables. Instead, it only requires a simulator of the stochastic phenomena. Moreover, ODDP can automatically determine the sample size required to achieve a given tolerance.

1.1 Structure of this paper

This paper is structured as follows. The second section provides the background of multistage stochastic linear programming (MSLP), which includes the mathematical setting and the model formulation. Section 3 formulates the "ensemble model" of MSLP, which produces a policy called the "Compromise Policy". We solve one example with SDDP and demonstrate the advantage of the compromise policy over the individual policies. Section 4 presents the bias and variance reduction properties of the Compromise Policy. Following this section, we present the details of our meta-algorithm, which is composed of distributed optimization together with aggregated validation steps for bias and variance reduction. We propose the integration of our meta-algorithm with the SDDP-type algorithm, as well as the parallel scheme. Section 5 details our online algorithm, ODDP, which generates accurate function approximation with sampling on-the-fly. Section 6 presents further computational results of our meta-algorithm. Finally, we present the conclusions of our study in the last section.

2 Formulation of MSLP

We consider a risk-neutral multistage stochastic linear programming (MSLP) problem with continuous decision variables. This problem can be regarded as

a multistage dynamic process under uncertainty, where sequential decisions are made at discrete epochs in time. We denote $\mathcal{T} = \{0, \dots, T\}$ as the set containing all decision-making times, where T is finite and is fixed once-and-for-all-time. Therefore, our focus is a finite horizon decision model with $T+1$ stages.

The uncertainty in the system is characterized as an exogenous process (i.e., not dependent on decisions) defined via a sequence of random variables $\{\tilde{\omega}_t\}_{t=0}^{T}$ defined on a filtered probability space $(\Omega, \mathcal{F}, \mathbb{P})$, with ω_t denoting one observation of the random variable $\tilde{\omega}_t$, and the probability space obeys the following relationships:

- $\Omega = \Omega_0 \times \Omega_1 \times \cdots \times \Omega_T$, where outcomes $\omega_t \in \Omega_t$.
- The σ-algebras $\mathcal{F}_t \subseteq \mathcal{F}$ include the scenario data until time t. Thus, we have $\mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \cdots \subseteq \mathcal{F}_T$.

Due to the sequential nature of information evolution, the notation can become a bit overwhelming. To ease this notational burden, we introduce some simplifications.

- For any time stage t, we denote $t+$ and $t-$ as the succeeding and preceding stages, respectively,
- We use the subscript $[t]$ to represent the history of the stochastic process $\{\omega_t\}_{t=0}^{T}$ until the current stage (included), i.e., $\omega_{[t]} = \omega_0, \dots, \omega_t$, and use the subscript (t) to denote the process starting from t to the end of the horizon, i.e., $\omega_{(t)} = \omega_t, \dots, \omega_T$.
- Note that $\omega_{[T]}$ is one element in the sample space of Ω, which is referred to as a scenario path. At any time t, we can only *observe* the exogenous state before t, i.e., $\omega_{[t-1]}$.
- We use $\langle \cdot, \cdot \rangle$ to represent the inner product of vectors, i.e., $\langle u, v \rangle = u^\top v$. If the former variable is a matrix, then it is the product of a transposed matrix and a vector, i.e., $\langle M, v \rangle = M^\top v$.
- The notation $[\![i, j]\!]$ denotes the set of integers between i and j, and both are included.

In many MSLP formulations [12, 16, 19], there are only decision/control variables, which focus on finding optimal decisions. In addition to control variables u_t, we also introduce state variables x_t, which represent historical information. This formulation is more general and applicable to a larger variety of situations, especially those involving dynamic systems. It also builds the bridge between MSLP and reinforcement learning (RL)/stochastic optimal control.

As for modeling choices for MSLP, one might consider one of two information structures: *decision-hazard* and *hazard-decision*. In the former information structure, the agent chooses a control u_t *before* observing a realization of random data $\omega_t \in \Omega_t$ according to a decision-rule $\mu_t(x_t)$. In contrast, under a *hazard-decision* scheme, the control u_t is selected *after* observing ω_t according to a decision-rule $\mu_t(x_t, \omega_t)$.

These two information structures are different modeling choices. The difference arises from the disparity in information at the time of decision-making (i.e., choosing u_t). Take a hydro-thermal scheduling problem with inflow uncertainty as an example. At a given stage, the decision-hazard formulation finds the decision considering all the possible inflow values, while the hazard-decision formulation finds the decision with a given inflow value. In fact, these two types of modeling choices are mutable [20]. For example, a one-stage decision-hazard problem can be decomposed into a deterministic problem followed by a hazard-decision problem. Considering that the decision-hazard formulation is typically much more challenging than the hazard-decision formulation, we study the problem with the decision-hazard scheme in the remainder of this paper. Since these two information structures are mutable, our method can be applied to both cases.

The state variables x_t are defined for any $t \in \mathcal{T}$, where the initial state x_0 is given. With a given state x_t, a control variable u_t, and exogenous information ω_t, we obtain the next state according to the system dynamics as follows:

$$x_{t+} = \mathcal{D}_t\left(x_t, u_t, \omega_t\right) = a_t + A_t x_t + B_t u_t \tag{1}$$

where $\mathcal{D}_t$ is the transition function, and (a_t, A_t, B_t) depend on the exogenous state ω_t. As the randomness data $\omega_{(0)}$ has been observed gradually over time, the states are available, and our decisions should be adapted to this process. The whole process has the form:

$$\begin{aligned}&\text{A given state } (x_0) \rightsquigarrow \text{ decision } (u_0) \rightsquigarrow \text{ observation } (\omega_0) \rightsquigarrow \text{state } (x_1) \rightsquigarrow \cdots \\ &\rightsquigarrow \text{ state } (x_T) \rightsquigarrow \text{ decision } (u_T) \rightsquigarrow \text{ observation } (\omega_T) \rightsquigarrow \text{ state } (x_{T+1})\end{aligned} \tag{2}$$

The state x_t depends on the random information up to time $t-1$, but not on the future observations, which is known as the nonanticipativity property. Given the past observations and decisions, we can calculate the state value x_t explicitly by iteratively applying the system dynamics. Since the states sequence $\{x_t\}_{t=0}^T$ and decisions sequence $\{u_t\}_{t=0}^T$ depend on $\omega_{[t-1]}$, these two sequences can be viewed as stochastic processes as well.

Now consider the following problem:

$$\begin{aligned}&\langle c_0, x_0\rangle + \min\ \langle d_0, u_0\rangle + \mathbb{E}_{\tilde{\omega}_{(0)}}\left[\langle c_1, x_1\rangle + \langle d_1, u_1\rangle + \mathbb{E}_{\tilde{\omega}_{(1)}}[\ldots + \mathbb{E}_{\tilde{\omega}_{T-1}}\left[\langle c_T, x_T\rangle + \langle d_T, u_T\rangle\right]\right] \\ &\quad \text{s.t. } u_t \in \mathcal{U}_t\left(x_t\right) := \{u_t \mid D_t u_t \leq b_t - C_t x_t, u_t \geq 0\} \quad \forall t \in \mathcal{T} \\ &\qquad x_{t+} = \mathcal{D}_t\left(x_t, u_t, \omega_t\right) = a_t + A_t x_t + B_t u_t \quad \forall t \in \mathcal{T}\end{aligned} \tag{3}$$

where we have applied the assumption that the expected terminal cost for the $T+1$ stage is 0. The expectation is taken in a nested form for the stochastic process $\tilde{\omega}_{(t)}$ from the t stage to the end of the horizon. The feasible set of the decision variables $\mathcal{U}_t(x_t)$ is a closed convex set which depends on the state x_t, where (b_t, C_t) and the recourse matrix D_t are fixed. Notice that if (b_t, C_t) includes random variables, we can incorporate the random parts into the system dynamics in Eq. (1) and define an intermediate state, which contains all

the random information. Based on the system dynamics, the stagewise state variable x_t can be explicitly expressed with the initial state x_0, together with all past decisions $u_{[t-1]}$, and exogenous information $\omega_{[t-1]}$.

Our problem can also be defined recursively. For any $t \in \mathcal{T}$, we define

$$\begin{aligned} h_t\left(x_t\right) = \left\langle c_t, x_t \right\rangle + \min \left\langle d_t, u_t \right\rangle + \mathbb{E}\left[h_{t+}\left(\tilde{x}_{t+}\right)\right] \\ \text{s.t. } u_t \in \mathcal{U}_t\left(x_t\right) := \left\{u_t \mid D_t u_t \leq b_t - C_t x_t, u_t \geq 0\right\} \end{aligned} \tag{4}$$

where $\tilde{x}_{t+} = \mathcal{D}_t\left(x_t, u_t, \tilde{\omega}_t\right)$. The function $h_t(\cdot)$ is called the post-decision value function or simply value function. The expectation is taken with respect to the random variable $\tilde{\omega}_t$ conditional on the given x_t. Since we assume stagewise independence (as in SDDP), this conditional expectation is equivalent to the expectation with respect to $\tilde{\omega}_t$. Note that, if we take the expectation with the successive stage random variable, the function $\mathbb{E}\left[h_{t+}\left(\tilde{x}_{t+}\right)\right]$ is a function of x_t and u_t. The objective function in the optimization problem is often referred to as the pre-decision value function [6]:

$$f_t\left(x_t, u_t\right) := \left\langle c_t, x_t \right\rangle + \left\langle d_t, u_t \right\rangle + \mathbb{E}\left[h_{t+}\left(\tilde{x}_{t+}\right)\right] \tag{5}$$

We have a relationship in which the post-decision value function equals the minimum of the pre-decision value function:

$$h_t(x_t) = \min_{u_t \in \mathcal{U}_t(x_t)} f_t\left(x_t, u_t\right) \tag{6}$$

In most MSLP problems, the size of the random space can be enormous (possibly infinite), in which case it is computationally intractable to enumerate all samples. One can choose a small number of the samples for each ω_t and create a so-called sample average approximation (SAA) problem [9] by replacing the expectation with the average over the selected samples. Thus, the corresponding sampled value function (or SAA function) is expressed as:

$$\check{f}_t\left(x_t, u_t\right) = \left\langle c_t, x_t \right\rangle + \left\langle d_t, u_t \right\rangle + \sum_{\omega_t \in \hat{\Omega}_t} p\left(\omega_t\right) \check{h}_{t+}\left(x_{t+}\right) \tag{7}$$

where the summation is over the sampled scenarios at t stage, denoted as $\hat{\Omega}_t$, and $\check{h}_{t+}\left(x_{t+}\right)$ is the sampled post-decision value function at $t+1$ stage. The estimate $p\left(\omega_t\right)$ is the empirical probability that scenario ω_t occurs. In an SAA problem with N_t Monte Carlo samples, the true random space is replaced by a simulated sample set $\{\omega_t^1, \omega_t^2, \ldots, \omega_t^{N_t}\}$, where each sample vector ω_t^n has the empirical probability $p(\omega_t^n) = 1/N_t, \forall n \in [\![1, N_t]\!]$. We refer to one SAA problem as one replication.

For consistency, we first summarize the assumptions that will be used in this paper:

- (A1) The feasible set of root-stage decision $\mathcal{U}_0$ is compact.

- (A2) The recourse is complete at all non-root stages, that is, the feasible set $\mathcal{U}_t(x_t)$ is non-empty for all states x_t for all $t \in \mathcal{T}\backslash\{0\}$.
- (A3) The recourse matrix is fixed, i.e., the matrices D_t are deterministic. Matrices D_t are assumed to have full row rank.
- (A4) Zero is a valid lower bound for all post-decision value functions.
- (A5) The exogenous information is stagewise independent, and its support is finite. This is a common assumption for SP models while using SDDP.
- (A6) The expected stagewise cost for $T+1$ stage is 0

These assumptions are reasonably common for SP models [13, 19, 21]. The first two assumptions (A1-A2) imply that the objective value is less than $+\infty$. (A2) ensures that the feasible set is always non-empty, and thus, Benders-type feasibility cuts are not necessary. (A3) is a generalization of the fixed recourse assumption, common in two-stage problems. The assumption (A4) is valid in many engineering problems, where the objective represents cost, which is generally non-negative. Notice if for some situations, the lower bound of the value function is lower than 0, we can raise the objective function value by adding a constant whose absolute value is equal to the absolute value of the lower bound. This will not change the optimal decision, and we can get the true optimal objective value by subtracting the constant after the optimization. With (A4), the optimal value is not negative infinity, and thus each stage has a dual feasible subproblem. The finite support assumption (A5) ensures that the set Ω_t is finite, and the stagewise independence assumption enables cut sharing in the algorithm. Finally, since we focus on the finite horizon problems, (A6) is required.

3 Compromise Policy: An Ensemble Method

In most MSLP problems, the number of random variables is enormous. Due to the independence assumption, the size of the random space increases exponentially as the number of stages increases. As a result, optimizing the 'true' value function and finding the optimal policy is intractable. Moreover, in many real-world problems, we usually do not know the true distribution of all the random variables. Thus, the SAA method is used to approximate the expectation with an empirical estimate (of the SAA function) which converges uniformly to the 'true' function [21]. Note that the computational demands with increasing sample sizes also increase significantly [9]. We will use distributed computation to overcome the increases in computational complexity, and this strategy will result in creating a multi-stage compromise policy which will serve as a generalization of the compromise decision introduced in [14].

We first set up the concepts for solving MSLP problems, then construct the compromise policy and illustrate its effectiveness with an example. For this example, we solve the problem with multiple replications, where each replication is solved by the SDDP algorithm. The details of the SDDP algorithm and the calculation of bounds are summarized in Appendix A.

3.1 Multi-replication Approach for Compromise Function and Compromise Policy

For the purpose of analysis, we differentiate among three different types of pre-decision value functions at each stage of an MSLP problem. We begin with the *'true' value function*, denoted as $f_t(x_t, u_t)$, which embodies the finite-sum 'true' expectation, e.g., (5). This function might be too large to enumerate completely, leading to the approximation functions. Sample average approximation (SAA) is the most common method, which replaces the expectation with a finite sample set and formulates an *SAA function*, denoted by $\check{f}_t(x_t, u_t)$. One example of the SAA function is in (7). Finally, we define the *algorithmic approximation function*, denoted as $\hat{f}_t(x_t, u_t)$, which is the approximation function generated by a specific algorithm. Take SDDP as an example. During optimization, SDDP applies linear programming duality to build an outer approximation of the SAA function. This outer approximation is the algorithmic approximation function used in SDDP.

As shown in [21], under some regularity conditions, we have that $\check{f}_t(x_t, u_t)$ converges pointwise to the 'true' function $f_t(x_t, u_t)$ w.p.1 as the sample size $N \to \infty$. Under the assumptions (A1)-(A6), the 'true' function is finite valued, and continuous. We have that $\check{f}_t(x_t, u_t)$ converges uniformly to $f_t(x_t, u_t)$. Philpott and Guan [19] further show that, under certain assumptions, the SDDP algorithmic approximation function $\hat{f}_t(x_t, u_t)$ converges finitely to the SAA function $\check{f}_t(x_t, u_t)$.

Suppose now that we solve an MSLP problem with M replications in parallel, where, in each replication, we generate an iid sample set with N paths and formulate the corresponding SAA function, denoted as $\{\check{f}_t^m(x_t, u_t)\}_{m=1}^M$. Hence the average SAA function is given by

$$\check{F}_t(x_t, u_t) := \frac{1}{M} \sum_{m=1}^{M} \check{f}_t^m(x_t, u_t), \quad t \in \mathcal{T} \backslash \{T\}. \tag{8}$$

We use an SDDP-type algorithm for each replication, which generates an approximation of the SAA function. The algorithm iteratively improves the approximation and terminates according to a pre-defined stopping criterion. We represent the algorithmic approximation function in the m-th replication as $\hat{f}_t^m(x_t, u_t)$. Next, define the average pre-decision value function across the replications as

$$\hat{F}_t(x_t, u_t) := \frac{1}{M} \sum_{m=1}^{M} \hat{f}_t^m(x_t, u_t), \quad t \in \mathcal{T} \backslash \{T\} \tag{9}$$

which is referred to as the **compromise function**. The **compromise policy** is then defined as

$$\hat{\mu}_t^c(x_t) := \underset{u_t \in \mathcal{U}_t(x_t)}{\arg\min}\ \hat{F}_t(x_t, u_t), \quad t \in \mathcal{T}\backslash\{T\}. \tag{10}$$

This policy is defined for all non-terminal stages. At the terminal stage T, it is a linear programming (LP) problem, and we can apply an LP solver directly to find the decision u_T for a given state x_T.

Since the average SAA function $\check{F}_t(x_t, u_t)$ considers all the data in the M replications (each replication with N paths), it is equivalent to an SAA function with $M \times N$ sample paths. Applying the same arguments as in [21], we have that as $M \times N \to \infty$, the average SAA function $\check{F}_t(x_t, u_t)$ converges uniformly to the 'true' function $f_t(x_t, u_t)$. Together with the convergence of the SDDP algorithm [19], we see that the compromise function $\hat{F}_t(x_t, u_t)$ converges to the 'true' function, that is,

$$\hat{F}_t(x_t, u_t) \xrightarrow{[19]} \check{F}_t(x_t, u_t) \xrightarrow{[21]} f_t(x_t, u_t). \tag{11}$$

While (11) is the conceptual basis for the SDDP algorithmic process, the size of most scenario trees for practical MSLP instances may be so large that the assumptions underlying (11) (see [19]) may not be satisfied within reasonable computational effort (time). *This paper suggests a multi-replication approach so that alternative sample paths can be explored due to alternative paths created in parallel, thereby increasing the likelihood that these alternative paths can collectively ensure* (11)*, thus increasing the likelihood that the convergence embodied in* (11) *may hold for the ensemble.* We will discuss the calculation of the upper and lower bounds of the optimal value in Section 4. If the gap between the upper and lower bounds is larger than the acceptable tolerance, the algorithmic approximation functions are not accurate enough. In this case, we increase either the sample size N or the number of replications M.

3.2 An Example: Hydro-Thermal Scheduling

In order to give the reader a concrete sense of the issues connected with the convergence of SDDP for MSLP models, we now consider a four-stage hydro-thermal scheduling problem, which is an extension of the hydro-thermal problem in the *sddp.jl* tutorial [15]. Our objective is to operate one hydro generator and one thermal generator in order to meet the demand in the face of inflow uncertainty while the cost is minimized. The detailed formulation is provided in Appendix B. This is a relatively simple MSLP problem, which has one state variable (volume of the hydro generator), three control variables (thermal generation, hydro generation, and hydro spill), and one random variable (inflow) at each stage. We consider a case in which the random variable has low variance. We choose 30 different sample sets for the random variables independently and construct 30 SAA problems, where each one includes 10^8

possible scenario paths. We solve the problem with the SDDP algorithm for each replication, which terminates when the lower bound estimate is stable. Specifically, the algorithm stops when the change of the lower bound estimate is less than 10^{-4} for 20 consecutive iterations.

Fig. 1 shows the lower bound estimate and the 95% percent confidence interval of the in-sample upper bound. The upper bound is based on 2,000 in-sample scenario paths ($N' = 2000$), and calculated with Alg. 4 in Appendix A. The first 30 policies are the output policies from the 30 replications, where each replication is solved using SDDP. We refer to each such policy as an individual policy. As we can see, all replications' lower and upper bound estimates are very close. However, the estimated optimal objectives in replications are very different from each other. In other words, individual policies' estimated objectives have significant variance. If we only solve one replication, the estimated objective can be any of them. Thus, this estimate is unreliable, and the output policy might not work well for the 'true' problem.

The last policy is the compromise policy constructed based on Eq. (10), where the compromise function is an average of the algorithmic approximation function in the 30 replications. We calculate the upper bound and the compromise lower bound with variance (details will be discussed in the next section). As we can see in Fig. 1, its lower and upper bounds are very close.

Fig. 1 In-Sample Upper Bound CIs and Lower Bounds

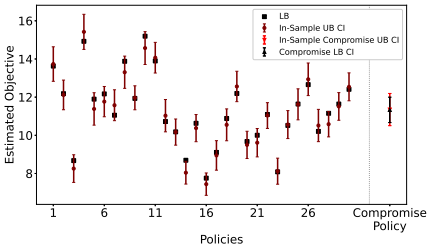
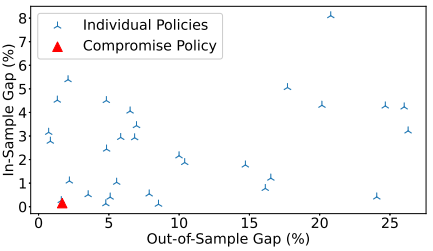


Fig. 2 Out-of-Sample Gap v.s. In-Sample Gap



To evaluate the performance of these policies in the 'true' problem, we need to choose out-of-sample scenarios paths and calculate the confidence interval of the out-of-sample upper bounds. We calculate the out-of-sample upper bounds with 20,000 scenario paths based on Alg. 4 (in Appendix). Fig. 2 compares the out-of-sample optimality gap and in-sample optimality gap of all the policies, where the optimality gap is calculated by

$$\text{optimality gap} = \frac{|\text{upper bound estimate - lower bound estimate}|}{\text{upper bound estimate}}. \quad (12)$$

Note that although the in-sample optimality gaps in all replications are low, in some replications, the out-of-sample gaps can be greater than 25%!

The performances of the individual policies on the 'true' problem vary considerably. For one individual policy, the in-sample gap is less than 4%. However, the out-of-sample gap is around 27%. Thus, the in-sample estimated objective is unreliable, and the output policy might not be optimal. Note that in this example, we already consider 10^8 possible scenarios, and the variance of the random variable is low. In real-world problems, the performance of *one-replication* SDDP algorithm can be far worse.

We construct the compromise policy to reduce the variance and find a stable policy. In Fig. 2, we can see that the performance of the compromise policy is much more stable than the individual policies. The compromise policy has a lower variance estimate. Based on the compromise policy, we can also calculate a confidence interval of the lower bound estimate. We will show that this compromise lower bound is better than the mean of the lower bounds from replications. Combining the upper and lower bound CI, we can estimate the optimality gap of the compromise policy.

4 Variance and Bias Reduction

In this section, we show that the compromise policy not only reduces the variance of the function approximation but also reduces the bias, which provides a tight lower bound for the optimal value. In general, the performance of the compromise policy is better than individual policies. To construct the compromise policy, we run each replication in parallel and entirely independently, and thus, the extra computation time is negligible. As a result, the compromise policy is an *"almost free lunch"*, given that one uses M replications anyway.

4.1 Variance Reduction

Suppose we solve an MSLP problem with M replications, where each replication has N samples in each stage. We first analyze the following case: in each replication, the algorithm (i.e., SDDP) solves the problem until optimality, where the algorithmic approximation function equals the SAA function, that is $\hat{f}_t(x_t, u_t) \approx \check{f}_t(x_t, u_t)$ for all (x_t, u_t).

For any stage t with state x_t, we further define

$$v_t^*(x_t) := \min_{u_t \in \mathcal{U}_t(x_t)} f(x_t, u_t) \tag{13}$$

as the optimal value of the true problem; $\hat{v}_t^c(x_t)$ as the **compromise lower bound**,

$$\hat{v}_t^c(x_t) := \min_{u_t \in \mathcal{U}_t(x_t)} \hat{F}_t(x_t, u_t); \tag{14}$$

and $\hat{v}_t^m(x_t)$ as the optimal value of the m-th replication,

$$\hat{v}_t^m(x_t) := \min_{u_t \in \mathcal{U}_t(x_t)} \hat{f}_t^m(x_t, u_t), \quad \forall m \in [\![1, M]\!]. \tag{15}$$

Since the initial state x_0 is fixed, we will use $v_0^*, v_0^c, \hat{v}_0^m$ as the corresponding optimal values for the first stage.

Theorem 1 *Suppose Assumption (A1-A6) hold, and we solve an MSLP problem with M replications, where each replication is solved with the SDDP algorithm. We further assume that the sampling scheme in all replications is independent and identically distributed (iid). Suppose the M individual policies from the SDDP algorithm are optimal policies for their corresponding SAA problems.*

i) At a given point (x_t, u_t), suppose that the variance of the SAA function $\check{f}_t(x_t, u_t)$ is $\sigma^2_{x_t,u_t}$. Then, we have $|\hat{F}_t(x_t, u_t) - f_t(x_t, u_t)| \leq O_p(\sigma_{x_t,u_t}/\sqrt{M})$.

ii) For any stage t with state x_t, $\hat{v}_t^m(x_t), \hat{v}_t^c(x_t)$ and $v_t^*(x_t)$ are the objective value from the individual policy, compromise policy and 'true' optimal policy, respectively. We denote $(\hat{\sigma}_t(x_t))^2$ as the variance of $|\hat{v}_t^m(x_t) - v_t^*(x_t)|$. Then, we have that the compromise policy has reduced variance compared with the individual policies. Specifically, $|\hat{v}_t^c(x_t) - v_t^*(x_t)| \leq O_p(\hat{\sigma}_t(x_t)/\sqrt{M})$.

Proof See Appendix C.1. □

In the M replications, the SAA function value $\{\check{f}_t^m(x_t, u_t)\}_{m=1}^M$ are different realizations of SAA function. The realizations depend on their sample set $\{\hat{\Omega}^m\}_{m=1}^M$, where each sample set contains N iid samples. At a fixed point (x_t, u_t), $\check{f}_t(x_t, u_t)$ follows a normal distribution $\mathcal{N}(\mu_{x_t,u_t}, \sigma^2_{x_t,u_t})$, where the mean value μ_{x_t,u_t} equals the true function value $f_t(x_t, u_t)$ and the value of $\sigma^2_{x_t,u_t}$ depends on the problem structure and the sample size. Note that the average SAA function $\check{F}_t(x_t, u_t)$ (defined in (8)) is an average of the functions in $\{\check{f}_t^m(x_t, u_t)\}_{m=1}^M$, which is equivalent to an SAA function with $M \times N$ samples. Therefore, $|\check{F}_t(x_t, u_t) - f_t(x_t, u_t)|$ is bounded by $O_p(\sigma_{x_t,u_t}/\sqrt{M})$, while the individual function approximations is bounded by $|\check{f}_t^m(x_t, u_t) - f_t(x_t, u_t)| \leq O_p(\sigma_{x_t,u_t}), m \in [\![1, M]\!]$.

The compromise policy is obtained by optimizing the compromise function in Eq. (9). Since each $\hat{f}_t^m(x_t, u_t)$ is approximately equal to $\check{f}_t^m(x_t, u_t)$, $\hat{F}_t(x_t, u_t)$ is approximately equal to the average SAA function $\check{F}_t(x_t, u_t)$. Due to the finite support assumption (A5) and the known probability distributions, $\{\check{f}_t^m(x_t, u_t)\}_{m=1}^M$ and $\check{F}_t(x_t, u_t)$ have only finitely many pieces. By applying the functional CLT, we can show that the compromise policy has reduced variance.

The algorithmic approximation function is generally less than or equal to the SAA function. In practice, these two functions may not be equal at all points. However, they are close in the neighborhood of the optimal solutions. In the SDDP algorithm, as the iteration number increases, the algorithmic approximation function keeps approaching the SAA function. As a result, the algorithm identifies the decision near the optimal one, further improving the approximation function near the optimal solution. Hence, $\hat{F}_t(x_t, u_t)$ enjoys the same variance reduction property of $\check{F}_t(x_t, u_t)$ near the optimal solutions.

In [22], the authors show the variance reduction property of the compromise decision in two-stage stochastic linear programming, which only requires the

equality of the algorithmic approximation function and SAA function at their own optimal solution locally. Consider a given state $\hat{x}_t$. Let $\hat{U} = \{\hat{u}_t^1, \ldots, \hat{u}_t^M\}$ denote the set of the optimal solutions in all the replications, where $\hat{u}_t^m = \arg\min_{u_t} \hat{f}_t^m(\hat{x}_t, u_t)$. We define

$$\bar{u}_t = \frac{1}{M} \sum_{m=1}^{M} \hat{u}_t^m, \tag{16a}$$

$$\text{and } \hat{u}_t^c = \arg\min_{u_t} \hat{F}_t(\hat{x}_t, u_t). \tag{16b}$$

The authors in [22] prove that when $\hat{u}_t^c$ and $\bar{u}_t$ are equal, the estimate $\hat{F}_t(x_t, \hat{u}_t^c)$ has reduced variance. Here, we extend this concept and show the variance reduction of $\hat{F}_t(x_t, \hat{u}_t^c)$ if $\hat{u}_t^c$ is in a neighborhood of $\bar{u}_t$.

Theorem 2 *Suppose Assumption (A1-A6) hold. Consider a given state $\hat{x}_t$. Suppose we solve the problem with M replications, where each replication has N samples and is optimized by SDDP. Suppose the samples in all replications are iid. For $m \in [\![1, M]\!]$, suppose that $\hat{f}_t^m(\hat{x}_t, \hat{u}_t^m)$ is close to the true function such that $|f_t(\hat{x}_t, \hat{u}_t^m) - \hat{f}_t^m(\hat{x}_t, \hat{u}_t^m)|$ is no greater than $O_p(N^{-1/2})$. $\bar{u}_t$ is defined in Eq. (16a) and $\hat{u}_t^c$ is defined in Eq. (16b). There exists a neighborhood of $\bar{u}_t$, denoted as $NE(\bar{u}_t)$, such that $\hat{u}_t^c \in NE(\bar{u}_t)$ implies*

$$|f_t(\hat{x}_t, \hat{u}_t^c) - \hat{F}_t(\hat{x}_t, \hat{u}_t^c)| \leq O_p((N \times M)^{-1/2}).$$

Proof See Appendix C.2. □

Theorem 1 and Theorem 2 show that our method produces a policy that reduces the variance of the objective function by implementing multiple replications, where the variance reduction is inversely proportional to the number of replications. In two-stage problems, we only identify the static first-stage *decisions*, while in multi-stage problems, we need to consider the *policies* for all stages. Since the variability propagates exponentially across stages, the variance reduction in multi-stage is much more demanding than for two-stage problems. Moreover, unlike other multiple replication procedures [23] which create multiple policies, our method recommends only one policy, the compromise policy. This compromise policy can be better than any individual policy.

4.2 Bias Reduction

Next, we show that the compromise policy has reduced bias as well. Many algorithms in MSLP construct an algorithmic approximation function for the corresponding SAA function. These algorithmic approximation functions are lower bounds of their related SAA functions. Moreover, the average of these SAA functions' optimal values is negatively biased by the true optimal value

[24, 25]. In Theorem 3, we show that the compromise lower bound reduces this negative bias.

Theorem 3 *Suppose Assumption (A1-A6) hold. Suppose the samples in all replications are iid. The initial state x_0 is fixed. $\hat{v}_0^c$ is defined in Eq. (14); $\bar{v} := \frac{1}{M}\sum_{m=1}^{M} \hat{v}_0^m$. Then, $\hat{v}_0^c$ and $\bar{v}$ are valid lower bound estimates for the optimal value of the true problem. And we have*

$$\bar{v} \leq \hat{v}_0^c.$$

Proof See Appendix C.3. □

Theorem 3 shows that our method provides a tighter lower bound estimate for the optimal value. The bias reduction of our lower bound value origins from the fact that we first average the function approximations and formulate the compromise function, then minimize the compromise function to obtain the value. On the contrary, the commonly used multi-replication method [21] first conducts minimization and then takes the average. In their method, $\bar{v}$ is a lower bound estimate of the 'true' optimal value for two reasons: 1) the algorithm constructs a lower bound approximation of the SAA function, i.e., $\hat{f}_0^m(x_0, u_0) \leq \check{f}_0^m(x_0, u_0)$ for $m \in [\![1, M]\!]$; 2) the mean of the optimal values of multiple SAA functions is negatively biased estimate of the 'true' optimal value, i.e., $\mathbb{E}[\min_{u_0} \check{f}_0(x_0, u_0)] \leq v_0^*$. In contrast, our $\hat{v}_0^c$ is a valid lower bound estimate only because of the first reason. Hence, our compromise policy eliminates this negative bias, since we have $\min_{u_0} \check{F}_0(x_0, u_0) \approx \min_{u_0} \mathbb{E}[\check{f}_0(x_0, u_0)] = v_0^*$. As a result, the compromise policy provides a tighter lower bound estimate.

For the given state x_0, we have the decision $\hat{u}_0^c = \hat{\mu}_0^c(x_0)$. At $(x_0, \hat{u}_0^c)$, the function value $\hat{F}_0(x_0, \hat{u}_0^c)$ is an average of M estimates. These estimates vary because of their corresponding sample sets, which provide an estimated variance for the compromise lower bound:

$$(\hat{\sigma}^c)^2 := \frac{1}{M} \frac{(\hat{f}_0^m(x_0, \hat{u}_0^c) - \hat{v}_0^c)^2}{M-1} \tag{17}$$

This variance is similar to the parametric variance in Markov decision processes [26].

5 A Meta-Algorithm for MSLP

Combined with the compromise policy, we propose a meta-algorithm for the MSLP problems. Our meta-algorithm takes advantage of distributed/parallel computing, providing a more accurate function approximation within a limited computational time. It can incorporate most of the value function-based algorithms for MSLP, such as the SDDP and SDLP algorithms. By formulating the compromise policy, the meta-algorithm reduces the variance and bias

Algorithm 1 A Meta-Algorithm for MSLP

Initialization: True dataset Ω, total number of replications M, number of validation sample paths N'.

Distributed Optimization:

1: **for** $m = 1, \ldots, M$ replication (in parallel) **do**
2: Formulate the SAA problem with a sampled dataset $\hat{\Omega}^m$.
3: **while** *In-sample stopping rule* is not satisfied **do**
4: Iteration count $k \leftarrow k + 1$
5: **Forward pass:** Find the state/decision path along a sample path: $(x_t^{m,k}, u_t^{m,k}), t \in \mathcal{T}\backslash\{T\}$.
6: **Backward pass:** Update value function approximation: $\hat{f}_t^{m,k}(x_t, u_t), t \in \mathcal{T}\backslash\{T\}$.
7: **end while**
8: **end for**

Aggregated Validation:

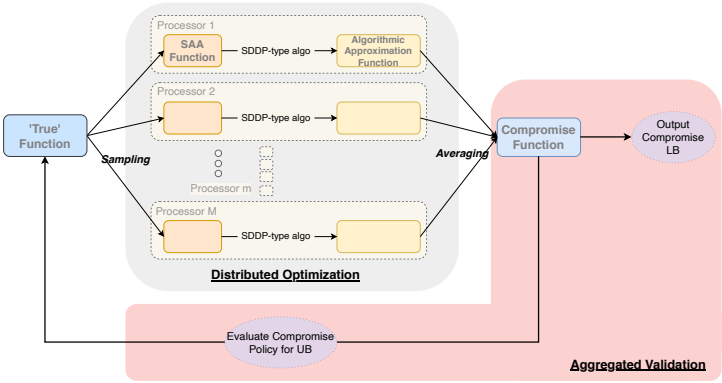
9: Formulate the average pre-decision value function $\hat{F}_t(x_t, u_t)$ defined in Eq. (9). Construct the compromise policy $\hat{\mu}_t^c(x_t)$ with Eq. (10).
10: Calculate the confidence interval of the compromise lower bound according to Eq. (14) and Eq. (17).
11: Simulate the compromise policy with N' paths from the true dataset Ω. Calculate the out-of-sample upper bound value with Alg. 4.
12: **if** *Out-of-sample stopping rule* is not satisfied **then**
13: Choose larger sample sets $\{\hat{\Omega}^m\}_{m=1}^M$ or enlarge the number of replication M. Go to Line 1.
14: **else**
15: Stop. Output the compromise policy $\hat{\mu}_t^c(x_t)$ for $t \in \mathcal{T}\backslash\{T\}$, and the upper and lower bound confidence intervals.
16: **end if**

of the function approximation and thus significantly improves the quality of our policies.

Our meta-algorithm consists of two parts: distributed optimization and aggregated validation. We provide a diagram in Fig. 3 to illustrate the workflow. In the distributed optimization procedure, we exploit distributed computing, where we construct multiple function approximations in parallel. Within each replication, we check the *in-sample stopping rule*, which tests whether each algorithmic approximation function $\{\hat{f}^m(\cdot,\cdot)\}_{m=1}^M$ converges to the corresponding SAA function $\{\check{f}^m(\cdot,\cdot)\}_{m=1}^M$. In the aggregated validation procedure, we formulate a compromise policy that considers the function approximations in all replications. We check the optimality of the compromise policy by computing the out-of-sample upper bound and lower bound, which is referred to as the *out-of-sample stopping rule*. This step tests whether $\hat{F}(\cdot,\cdot)$

provides an optimal policy of the true problem. Our meta-algorithm sequentially enlarges the sample size or the number of replications to construct a more accurate function approximation, reducing the optimality gap to zero. The details are summarized in Alg. 1, where the forward/backward pass can be those in any value function-based algorithms, such as SDDP. In our meta-algorithm, we first choose a sample size and solve multiple SAA problems in parallel. We then formulate the compromise policy and check its optimality. If the out-of-sample optimality gap is less than the tolerance, we stop; otherwise, we choose larger sample sets and solve them again.

Fig. 3 Diagram of the meta-algorithm



5.1 Distributed Optimization

The distributed optimization step solves multiple replications of the SAA problems independently. By exploiting distributed/parallel computing, we can accelerate our algorithm's optimization process. In an SDDP-type algorithm, we can compute the following steps in parallel:

1. Multiprocessor parallelization: Each processor works on an independent replication. The whole optimization process is conducted independently in each replication, from scenario data generation to value function update. As a result, we can obtain the function approximations and policy based on the samples from parallel replications.
2. Scenario path parallelization: in the forward pass, We can run multiple scenario path simulations in parallel, which will provide various state paths and decision paths.
3. Cutting plane parallelization: We calculate the new cutting planes for each sample using the new state and decision during the backward pass. The cutting plane for each sample can be found in parallel. A disadvantage of this parallelization is that we will lose the warm starts when solving linear programs with the dual simplex algorithm.

Our computation only focuses on the first parallelization method: the multiprocessor parallelization. Since each processor applies the optimization algorithm independently and end-to-end, there is no communication cost among processors. When each processor pauses, it takes barely any time to construct the compromise policy by averaging the algorithmic function approximations in all replications. Our meta-algorithm can be extended to incorporate other parallelization tools, further reducing the computational time.

Modern programming languages have made the process of parallel computing easy to implement. Take `Julia`, one of the more popular languages in the Operation Research community, as an example. We only need to modify a few lines of code to implement multiprocessor parallelization. For multi-threaded computing, we need to simply add the macro `Threads.@threads` before the iterations, such as

```
Threads.@threads for m = 1:M
          solve_Replication_m
      end
```

For distributed computing, the `Distributed.jl` package can be easily applied. These minor changes significantly reduce the computational time. The computational experiments were carried out on a workstation with two Intel(R) Xeon(R) Platinum 8270 CPU @2.70GHz processors and 64.0 GB RAM. Each processor has 26 cores and 52 logical processors. The computations exploit the state-of-the-art implementation of the SDDP solver, the *sddp.jl* package [15]. We use Julia 1.7.1 with Gurobi 9.5 as the LP solver. In Julia, we set the maximum number of threads to be 104.

For one replication of the distributed optimization in our meta-algorithm, we implement the SDDP algorithm to replace the generic forward-backward pass in `Line 5-6` in Alg. 1, where the details of the SDDP algorithm is detailed in Alg. 3. Although we illustrate the meta-algorithm with SDDP, our algorithm can incorporate most of the value-function-based algorithms in MSLP, such as those in [12, 13, 16, 27], or even a combination of them. Different replications of the algorithm run in parallel.

5.2 Aggregated Validation

The aggregated validation step is to examine the optimality of the output policy. If the stopping rule is satisfied, then we stop. Otherwise, we solve a more accurate SAA problem. To do so, traditionally, one enlarges the number of samples N in each replication, and hence, more accurate SAA problems are constructed. Then, one optimizes each replication and formulates the compromise policy. The compromise policy performs better since it considers a more extensive sample set. With our meta-algorithm, we recommend a new method, which is to solve more replications of SAA problems with the same sample size N. We then average all the value function approximations to formulate the compromise function. Since we consider more replications, the new compromise function has lower variance and higher accuracy. Due to parallel settings,

we can increase the replication number M to fully utilize all the computational resources (threads or processors). The computational time is much less than the case with increased N. Moreover, when enlarging M, the previously generated function approximations can still be averaged in the compromise function. On the contrary, when increasing N, one must solve all new replications from scratch. The previous calculations are discarded. In both cases, when the optimality is obtained, our compromise policy is equivalent to the optimal policy of an SAA problem with $N \times M$ samples.

To check the out-of-sample stopping rule, we first build the compromise function and formulate the compromise policy defined in (10). In Section 4, we show that the compromise policy provides a tight lower bound (LB) estimate of the optimal value. The out-of-sample upper bound (UB) estimate usually requires Monte Carlo Simulation. That is, we generate N' iid out-of-sample scenario paths from the first to the last stage. For these N' paths, we implement the compromise policy and calculate the sum of stagewise objectives, which give us N' estimates of objective value. Their mean and variance are used to estimate the upper bound value of the optimal value. We provide the details of the upper bound calculation in Algorithm 4 (see Appendix). Combining the upper and lower bound estimates, we verify the optimality of the compromise policy.

Since we wish to compare our meta-algorithm with the original single-replication SDDP algorithm, we apply the stopping rule that checks whether the optimality gap (defined in (12)) contains zero or not. This test follows the same idea as in [12, 15]. Notice that the meta-algorithm is a general framework, which can also incorporate the stopping criteria of [17, 18]. Because the compromise policy contributes a tight LB estimate and a low-variance UB estimate, it improves the efficiency of all these stopping rules.

6 Online Dual Dynamic Programming

In our meta-algorithm with SDDP, we first need to choose a sample set and solve the fixed SAA problem in each replication. The selection of the sample size is a balance between efficiency and accuracy. If the sample size is large, the SAA problem is more accurate while the computational cost is high. If the optimality gap is larger than the tolerance during the out-of-sample test, the SAA problems are not accurate enough. We have to choose larger sample sets and solve the new SAA problem again from scratch. For SDDP, it appears that sample size selection is a matter which depends on prior experience with an application.

In order to automatically find the sample size and avoid starting over, we propose a sequential sampling algorithm for MSLP, referred to as the online dual dynamic programming (ODDP) algorithm. With ODDP, users do not need to provide the 'true' distribution. Instead, ODDP only requires a simulator of the stochastic phenomena. The mechanism described in this algorithm automatically assimilates the approximate distribution during the execution

Algorithm 2 Online Dual Dynamic Programming Algorithm (One iteration)

Initialization: Count k. Sample Oracle. Initial state x_0. Set of cuts: $\mathcal{J}_t$.

Forward pass: Simulate the state/decision path along a sample path:

Set $x_0^k = x_0$.

Generate a sample path $\omega_{[T]}^k$ from the oracle.

Update the observed sample set $\hat{\Omega}_t \leftarrow \hat{\Omega}_t \bigcup \{\omega_t^k\}, \forall t \in \mathcal{T}$, and empirical probability $p(\omega_t) \leftarrow \frac{k-1}{k} p(\omega_t) + \frac{1}{k} \mathbb{I}(\omega_t = \omega_t^k), \forall \omega_t \in \hat{\Omega}_t, \forall t \in \mathcal{T}$.

for $t = 0, \ldots, T$ **do**

  Solve the optimization problem $u_t^k = \arg\min_{u_t} \hat{f}_t^{k-1}(x_t^k, u_t)$ to obtain the decision.

  Find the next state based on system dynamics $x_{t+}^k = \mathcal{D}_t\left(x_t^k, u_t^k, \omega_t^k\right)$.

end for

Backward pass: Update pre-decision value function approximation:

for $t = T-1, \ldots, 0$ **do**

  for $\omega_t \in \hat{\Omega}_t$ **do**

    Find the state $x_{t+} = \mathcal{D}_t\left(x_t^k, u_t^k, \omega_t\right)$

    Solve the subproblem with x_{t+} as input:

$$\min_{u_{t+}} \hat{f}_{t+}^k(x_{t+}, u_{t+}),$$

    and obtain optimal dual solution π_{t+}.

    Compute lower bounding affine function $l_{t+}(x_{t+}) := \alpha_{t+} + \langle \beta_{t+}, x_{t+} \rangle$, where $\alpha_{t+} = \langle b_{t+}, \pi_{t+} \rangle$; $\beta_{t+} = c_{t+} - \langle C_{t+}, \pi_{t+} \rangle$.

    Update the set of cuts as: $\mathcal{J}_{t+} \leftarrow \mathcal{J}_{t+} \bigcup \{(\alpha_{t+}, \beta_{t+}, k)\}$

  end for

  Obtain the updated stage cost-to-go value function approximation using

$$\hat{h}_{t+}^k(x_{t+}) = \max_{(\alpha_{t+}, \beta_{t+}, j) \in \mathcal{J}_{t+}} \{\frac{j}{k} (\alpha_{t+} + \langle \beta_{t+}, x_{t+} \rangle)\}$$

  to obtain

$$\hat{f}_t^k(x_t, u_t) = \langle c_t, x_t \rangle + \langle d_t, u_t \rangle + \sum_{\omega_t \in \hat{\Omega}_t} p(\omega_t) \hat{h}_{t+}^k(x_{t+}) \tag{18}$$

end for

and asymptotically provides an optimal policy under the assumption that the simulators provide the distribution which is consistent with the underlying stochastic process. In other words, our algorithm does not require an offline sample set before the optimization step, and it collects samples on-the-fly. This algorithm is the most suitable for the online or streaming-data setting.

Unlike the SDLP algorithm, which solves quadratic programming problems in the forward pass, ODDP solves only linear problems. In fairness, however, the proximal iterates of SDLP are much more stable and are able to maintain a finite approximation while providing a unique accumulation point as in Stochastic Decomposition (SD) [22, 28]. The ODDP algorithm can deal with random variables with evolving distributions as with the aforementioned methods. We exploit distributed computing and run each replication in parallel. A compromise policy is formulated for a more stable performance. We only need the out-of-sample stopping rule, which checks the optimality gap between the out-of-sample upper bound and the compromise lower bound. In ODDP, we pause the algorithm for every L iterations to check the out-of-sample stopping rule. If it is not satisfied, we resume the algorithm and generate more samples to update the function approximation instead of solving a new SAA problem from scratch. The extensive computational results show the effectiveness and efficiency of our algorithm.

The details of the ODDP algorithm are shown in Alg. 2, where we highlight the difference with SDDP in blue color. In `Line 2` of Alg. 2, ODDP generates a sample path from the 'true' oracle instead of a sampled dataset. Next, in `Line 3`, the ODDP updates the empirical distribution of the observed sample set. Because of the stagewise independence and finite support assumption (A5), we will ultimately observe all possible samples at any stage. As a result, there exists a finite number K, such that, when the iteration number $k > K$, the set $\hat{\Omega}_t$ will be fixed. Hence, the loop over the observed sample set (`Line 9-14`) will be finite. We collect the set of cuts with the corresponding iteration number in `Line 13` and decrease the previously generated cuts in `Line 15`. The decrease of the cuts makes them valid outer approximation of the corresponding SAA function, provided that the expected recourse functions are all non-negative. This update follows the same pattern as in two-stage stochastic decomposition [22, 28].

7 Further Computational Results

This section conducts further experiments with our meta-algorithm and compares the difference with the one replication SDDP algorithm. We consider the hydro-valley thermal scheduling problem with four state variables in each stage. The objective is to operate one thermal generator and four hydro-dams in a valley chain over τ stages, considering rainfall uncertainty. The aim is to minimize the generation cost (including hydro generation and thermal generation) while the power demand is met. The details of the formulation are in Appendix B. All codes and datasets are available on the cORe platform [29].

7.1 Results for *Meta + ODDP*

This subsection focuses on the performance of the meta-algorithm with ODDP. We show that our meta-algorithm with ODDP iteratively increases the lower

bound estimate and decreases the upper bound estimate. As a result, it provides a tighter optimality gap as the number of iterations increases. The sequential sampling property enables the algorithm to pause and resume, where we pause the algorithm for an optimality test and resume it (as necessary) for continuously updating function approximations. The algorithm automatically finds the number of samples required based on the specified tolerance. In addition, we demonstrate the power of the compromise policy by noting that the compromise policy outperforms any individual policy in all iterations.

Consider a ten-stage hydro-valley thermal scheduling problem with four continuous state variables and 13 continuous decision variables in each stage. We solve this problem with ODDP under the meta-algorithm framework, setting $L = 100$. We run $M = 30$ replications in parallel. Every L inner iterations, we pause the algorithm and check the out-of-sample stopping rule. If the stopping rule is not satisfied, we resume the algorithm, run L more inner iterations, and continue. The process of running L inner iteration and checking the out-of-sample stopping rule for the compromise policy composes one outer iteration.

Fig. 4 Optimality Gap v.s. Outer Iteration; Out-of-Sample Upper Bound and Lower Bound v.s. Outer Iteration

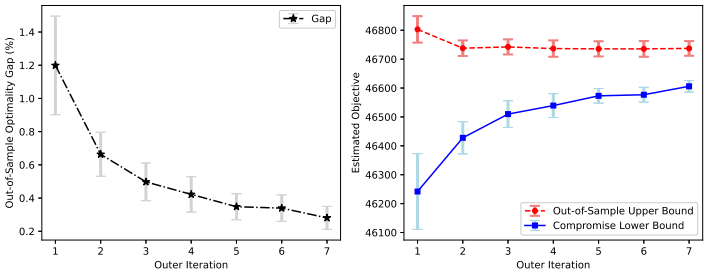


Fig. 4 shows that the upper and lower bound estimates get close as the iteration number increases, where their optimality gap reduces to zero asymptotically. The error bars of the optimality gap and upper/lower bounds represent their 95% confidence intervals. In the figure on the right, the red line is the out-of-sample upper bound estimate based on 20,000 scenario paths. We calculate the mean and variance based on Alg. 4. The blue line is the compromise lower bound, whose value and variance are calculated according to Eq. (14) and Eq. (17), respectively. In the figure on the left, the optimality gap is calculated based on Eq. (12). Since the upper and lower bound estimates are normally distributed and independent, the optimality gap variance is the sum of upper and lower bound variances. As the number of outer iterations increases, the compromise function approaches the 'true' value function, enhancing the compromise policy's performance. As a result, the upper bound decreases, and the lower bound increases. As shown, the optimality gap decreases quickly and, as a percentage of the upper bound, it is reasonably

close to zero. When the algorithm terminates, the optimality gap is less than 0.3%.

Fig. 5 Out-of-Samples upper bound and lower bound of the individual policies and the compromise policy in different outer iteration

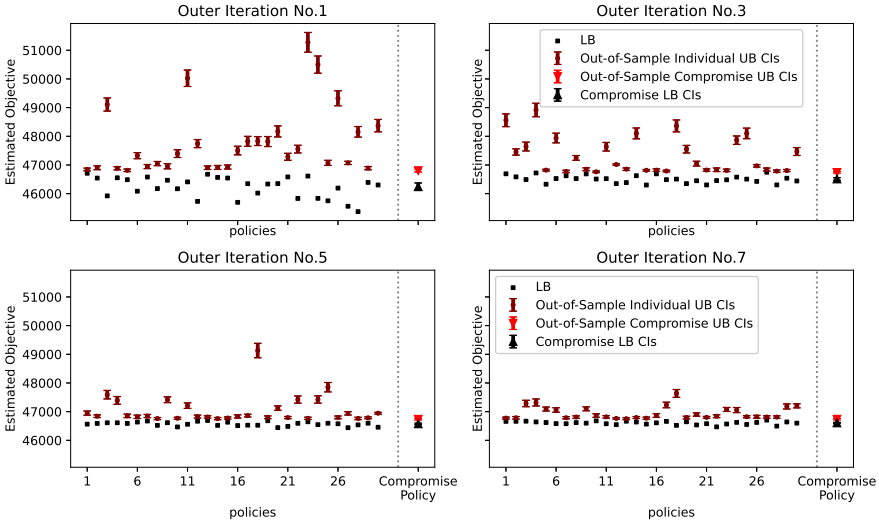
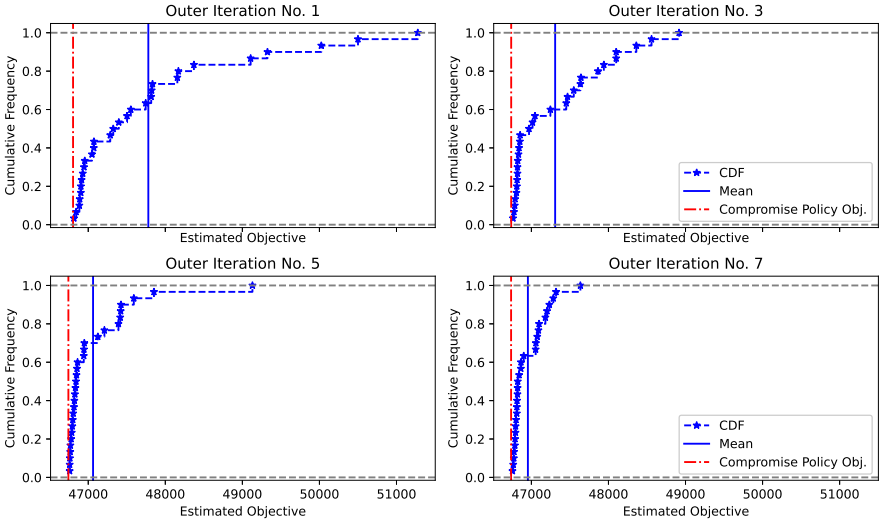


Fig. 5 compares the upper and lower bounds of all the individual policies, as well as the compromise policy. It shows the performance in the 1/3/5/7 outer iterations. As we can see, the optimality gap of each individual policy decreases as the algorithm runs, while the compromise policy always performs well. As the iteration number increases, more samples are observed in all replications. As a result, the upper bound estimates keep decreasing while the lower bound estimates keep increasing. This is illustrated by the descent of the red lines and the ascent of the black squares in Fig. 5. This figure also shows the stability of the compromise policy. If we run only one replication, the output policy can be any of the first 30 individual policies whose optimality gap can be enormous. In contrast, the compromise policy always has an insignificant optimality gap.

We also compare the policies with respect to the performance of the 'true' problem. In Fig. 6, we show the estimated objective of all the policies based on the simulation of 20,000 out-of-sample Monte Carlo scenario paths in different iterations. The blue line plot with stars illustrates the cumulative frequency of costs of an individual policy's performance. The blue vertical line is the mean value corresponding to the objective values of individual policies, which can be thought of as the expected policy performance if we only run one replication. The red vertical line is the objective value of the compromise policy. As we can see, the expected individual policy performance is improved as more samples are observed in the ODDP algorithm. Meanwhile, the compromise

Fig. 6 Cumulative distribution function of individual policy evaluation in different outer iteration



policy always has the lowest estimated objective compared with all individual policies. In general, the compromise policy outperforms the individual policies, which illustrates the bias reduction property of the compromise policy. We find that the compromise policy outperforms all the individual policies in every outer iteration. This property demonstrates the consistency of the compromise policy's advantage.

7.2 Comparison among Algorithms

With more computations, we compare the performance of the *SDDP* algorithm, our meta-algorithm with SDDP (labeled as *meta+SDDP*), and our meta-algorithm with ODDP (labeled as *meta+ODDP*). We consider five hydro-valley thermal scheduling problems whose total stages (τ) are 24, 48, 72, 96, and 120, respectively. They all consider four hydro generators and one thermal generator. In the *SDDP* algorithm, we choose ten samples in each stage. Thus, the total number of possible scenarios is 10^{τ}. We calculate the in-sample optimality gap according to Eq. (12), with the lower bound estimate and the in-sample upper bound confidence intervals (based on 1,000 scenario paths). The algorithm stops when zero is within 95% confidence intervals (CIs) of the in-sample optimality gap. This stopping rule is commonly used in the SDDP-type algorithms [15, 30]. For the *meta+SDDP* algorithm, we run $M = 30$ replications in parallel, where each replication has 10 samples per stage. The in-sample stopping rule is the same as the SDDP algorithm. We formulate the compromise policy based on the function approximations in

all replications. The out-of-sample optimality gap is calculated with the compromise lower bound CIs and the out-of-sample upper bound CIs (based on 2,000 out-of-sample scenario paths). The out-of-sample stopping rule is satisfied if zero is within 95% CIs of the out-of-sample optimality gap or the time of distributed optimization is greater than 1,500 seconds. If the out-of-sample stopping rule is not satisfied, we add ten more samples per stage in each replication and solve the new SAA problems. For the *meta+ODDP* algorithm, we compute $M = 30$ replications in parallel. Every $L = 500$ inner iterations, we pause the optimization procedure, formulate the compromise policy and check the out-of-sample optimality test, which is the same one as in *meta+SDDP*. If the optimality test is not satisfied, we resume the distributed optimization and refine the function approximations.

For all the algorithms, we summarize the upper bound estimate (based on 2,000 out-of-sample scenario paths), lower bound estimate, and out-of-sample optimality gap with 95% CIs in Table 1, which are visualized in Fig. 7. The distributed optimization time (or training time) in seconds is also included. As we can see, the performance of the one-replication SDDP algorithm is unreliable. Although zero is within 95% of its in-sample optimality gap, its out-of-sample optimality gap is significantly greater than zero. In the problem with $\tau = 96$ stages, its out-of-sample optimality gap is greater than 40%! Thus, we should implement a multiple-replication algorithm to obtain a better policy for the 'true' problem. As shown, in all instances, the *meta+SDDP* algorithm has the best performance regarding the upper bound estimate, as well as the optimality gap. This demonstrates the advantage of the compromise policy, which not only provides a tighter lower bound estimate (with confidence intervals) but also constructs a better policy. Our meta-algorithm considerably improves the performance of SDDP, while the extra computational effort is not significant. The training time for *meta+SDDP* approximately equals the longest training time out of all the replications. As for the *meta+ODDP* algorithm, the optimality gaps in all instances are less than 5%. The sequential sampling property makes the *meta+SDDP* algorithm suitable for online settings and evolving environments. Finally, the computational results in Table 1 validate the variance reduction property of our meta-algorithm. For all the instances, the standard deviations of the upper bound estimate in the meta-algorithm are much lower than those in the standard SDDP algorithm. For example, considering the most difficult instance we solved (i.e., $\tau = 120$), the standard error of the upper bound is only 17.38 with *meta+SDDP*, while the value of the standard error with the *SDDP* algorithm is as high as 1734.68. Thus, the performance of our compromise policy is much more stable.

Notice that when we try to get an accurate evaluation for the policy with Algorithm 4, the time for evaluation is significant. For instance, for the problem with $\tau = 24$, the evaluation time for the policy from *meta+SDDP* with $N' = 2,000$ scenario paths is 96.5 seconds. With our meta-algorithm, we only need to evaluate one policy: the compromise policy. However, in the commonly used multi-replication SAA method [21], when researchers solve M replications,

they need to evaluate all the individual policies from the replications and then choose the best one as the final policy. As a result, their evaluation time can be M times as much as that in our method. Moreover, we have already illustrated that the compromise policy often performs better than any individual policy.

Table 1 Comparison among different algorithms: the upper/lower bounds and optimality gap (with 95% confidence intervals).

Algorithms	**Properties**	τ**=24**	τ**=48**	τ**=72**	τ**=96**	τ**=120**
SDDP	UB $\bar{Q}$	16954.46 ($\pm$ 220.29)	38429.00 ($\pm$ 805.07)	50599.06 ($\pm$ 589.62)	107103.85 ($\pm$ 3114.61)	90657.98 ($\pm$ 1734.68)
	LB $\hat{v}_0$	15961.66	31393.26	47175.33	62234.40	81681.85
	Gap (%)	5.86 ($\pm$ 1.30)	18.31 ($\pm$ 2.09)	6.77 ($\pm$ 1.17)	41.89 ($\pm$ 2.91)	9.90 ($\pm$ 1.91)
	time (s)	33	20.1	39.1	30.4	99.5
meta +SDDP	UB $\bar{Q}$	15965.32 ($\pm$ 7.19)	31971.32 ($\pm$ 25.43)	48010.64 ($\pm$ 12.90)	64039.92 ($\pm$ 15.18)	80114.97 ($\pm$ 17.38)
	LB $\hat{v}_0^c$	15911.25 ($\pm$ 86.95)	31864.24 ($\pm$ 204.38)	47964.18 ($\pm$ 274.61)	64042.43 ($\pm$ 282.84)	79926.52 ($\pm$ 521.84)
	Gap (%)	0.34 ($\pm$ 0.55)	0.33 ($\pm$ 0.64)	0.10 ($\pm$ 0.57)	0.00 ($\pm$ 0.44)	0.24 ($\pm$ 0.65)
	time (s)	41.4	35.8	52.9	99	230
meta +ODDP	UB $\bar{Q}$	15949.12 ($\pm$ 24.45)	32147.86 ($\pm$ 135.12)	48257.73 ($\pm$ 169.21)	64514.94 ($\pm$ 384.31)	80927.77 ($\pm$ 493.61)
	LB $\hat{v}_0^c$	15741.94 ($\pm$ 11.57)	31367.00 ($\pm$ 32.14)	46843.21 ($\pm$ 68.22)	62237.11 ($\pm$ 71.97)	77555.76 ($\pm$ 114.79)
	Gap (%)	1.30 ($\pm$ 0.17)	2.43 ($\pm$ 0.43)	2.93 ($\pm$ 0.38)	3.53 ($\pm$ 0.61)	4.17 ($\pm$ 0.63)
	time (s)	1632	3028	4528	7219	8855

In our hydro-valley thermal scheduling problem, the *meta+ODDP* algorithm is much more computationally intensive compared with others. It stops because the training time is longer than the limit in all instances. Since *meta+ODDP* considers more samples per stage, the computational time is much longer. At the 500^{th} inner iteration, each replication may apply up to 500 samples to construct one cut during the backward pass. On the other hand, the *meta+SDDP* algorithm exploits ten samples to generate one cut. In hindsight, since the optimality gap in *meta+SDDP* is small enough, we may conclude that ten samples per stage with 30 replications may be sufficiently accurate to capture the random structure of this problem. However, for instances with greater complications in the stochastic process (e.g., SSN problems in [22]), we need to explore many more samples per stage to get an accurate approximation.

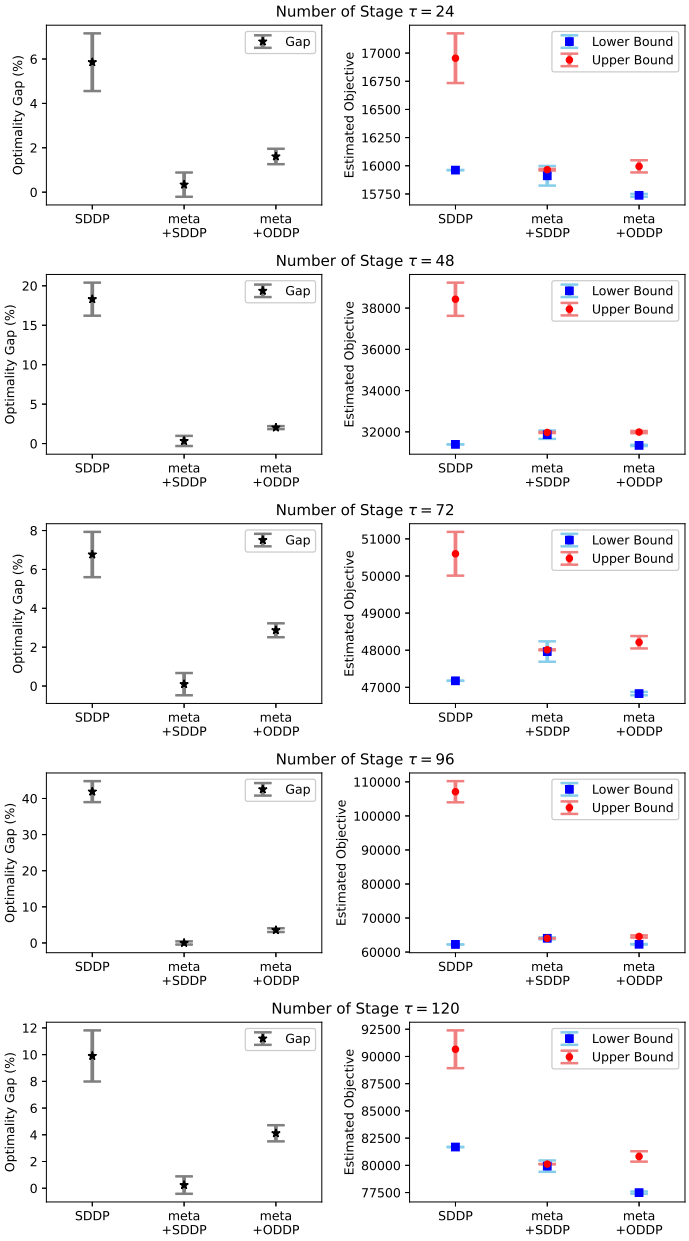
8 Conclusions

We propose an ensemble model called the compromise policy for MSLP problems, the only (almost) free lunch to build a more stable and effective policy. The compromise policy exploits distributed/parallel computing and build a more accurate function approximation with minimal extra computational time. We show that the compromise policy not only reduces the variance of the function approximation but also reduces the bias. It provides a tighter lower bound estimate (with confidence interval) for the optimal objective value. With regard to the compromise policy, we provide a general meta-algorithm based on in-sample and out-of-sample stopping rules for MSLP problems. Our meta-algorithm can incorporate SDDP-type algorithms, improving their performance and stability. We also introduce an online dual dynamic programming (ODDP) algorithm that iteratively improves algorithmic approximation and SAA function with sequential sampling. The ODDP algorithm automatically decides the sample size of the SAA function and is capable of pausing/resuming during the optimality test.

Declarations

- Acknowledgements: We sincerely thank Dr. Oscar Dowson for his open-source SDDP solver in Julia, the *sddp.jl* package. This state-of-the-art implementation enhanced our ability to upgrade and realize our algorithms. We are also grateful to Professor Andy Philpott for sharing his observations on SDDP.
- This research was funded by AFOSR grant FA9550-20-1-0006. We are grateful for this support.
- Conflict of interest/Competing interests: None
- Availability of data and materials: the data and materials can be found on the cORe platform https://core.isrd.isi.edu/chaise/record/#1/Core:Instance/RID=WCZE. The data is generated within the model file.

Fig. 7 Comparison among different algorithms: the upper/lower bounds and optimality gap (with 95% confidence intervals).



- Code availability: the code is available on the cORe platform https://core.isrd.isi.edu/chaise/record/#1/Core:Instance/RID=WCZE.

- The first author was responsible for all computational experiments, and both authors are jointly responsible for the overall architecture, experimental design, and conclusions presented in this paper.

Appendix A SDDP Algorithm

In order to highlight our contributions, it is best to summarize the nature of the SDDP algorithm [12]. One iteration of the SDDP algorithm consists of a forward pass and a backward pass, summarized in Algorithm 3. The forward pass simulates the decision path along a selected sample path, while the backward pass updates the cost function approximation. The approximation approaches the SAA function as the number of iterations increases. This kind of forward-backward pass algorithm [13] is common to tackle the MSLP problems. This cutting planes method is able to accommodate the non-smoothness and stochasticity of the models.

We describe one iteration of the optimization step in detail as follows. The forward pass starts from the root node to the end of the horizon. The initial state is given at the root node, which is x_0. Then, we iterate over all the time stage to identify the state and decision paths: $\{x_t\}_{t=0}^{T}$ and $\{u_t\}_{t=0}^{T}$, for the given sample-path $\omega_{[T]}$. At each stage, we iteratively optimize the function approximation to obtain the decision (`Line 4` in Alg. 3) and apply the system dynamics to obtain the state (`Line 5`). The backward pass starts from the last stage to the first, where we iteratively update the pre-decision value function approximation. At each stage, we loop through all realizations of the random variables to generate new cuts (`Line 8-13` in Alg. 3). We collect all these new cuts and construct a better piecewise affine function for the pre-decision value function (`Line 14` in Alg. 3).

A.1 Calculation of Bounds in SDDP

As mentioned earlier, MSLP problems are usually complicated and challenging. A standard procedure to solve an MSLP problem is first to choose a sample set $\hat{\Omega}$, and construct an SAA problem. Next, we apply an algorithm (e.g., SDDP) to solve the SAA problem. We run the algorithm until a stopping rule is satisfied. This procedure is also referred to as the SAA method (or single replication SAA method) for MSLP.

The stopping rule often involves the calculation of upper bound (UB) and lower bound (LB) estimates of the optimal objective value. Since the SDDP algorithm constructs an outer approximation of the SAA function, we can obtain a lower bound estimate with:

$$\hat{v}_0 = \min_{u_0 \in \mathcal{U}_0} \hat{f}_0(x_0, u_0).$$

This value is a valid LB for the SAA problem, which is one point estimate of the LB for the 'true' problem. In [31], the authors provide an approach

Algorithm 3 Stochastic Dual Dynamic Programming Algorithm (One iteration)

Initialization: Iteration number k. Fixed sample set $\hat{\Omega}$. Initial state x_0. Set of cuts $\mathcal{J}_t$.

Forward pass: Simulate the state/decision path along a sample path:

Set $x_0^k = x_0$.

Generate a sample path $\omega_{[T]}^k$ from $\hat{\Omega}$.

for $t = 0, \ldots, T$ **do**

  Solve the optimization problem $u_t^k = \arg\min_{u_t} \hat{f}_t^{k-1}(x_t^k, u_t)$ to obtain the decision.

  Find the next state based on system dynamic $x_{t+}^k = \mathcal{D}_t\left(x_t^k, u_t^k, \omega_t^k\right)$.

end for

Backward pass: Update pre-decision value function approximation:

for $t = T-1, \ldots, 0$ **do**

  for $\omega_t \in \hat{\Omega}_t$ **do**

    Find the state $x_{t+} = \mathcal{D}_t\left(x_t^k, u_t^k, \omega_t\right)$

    Solve the subproblem with x_{t+} as input:

$$\min_{u_{t+}} \hat{f}_{t+}^k(x_{t+}, u_{t+}),$$

    and obtain optimal dual solution π_{t+}.

    Compute lower bounding affine function $l_{t+}(x_{t+}) := \alpha_{t+} + \langle \beta_{t+}, x_{t+} \rangle$, where $\alpha_{t+} = \langle b_{t+}, \pi_{t+} \rangle$; $\beta_{t+} = c_{t+} - \langle C_{t+}, \pi_{t+} \rangle$.

    Update the set of cuts as: $\mathcal{J}_{t+} \leftarrow \mathcal{J}_{t+} \bigcup \{(\alpha_{t+}, \beta_{t+})\}$

  end for

  Obtain the updated stage cost-to-go value function approximation using

$$\hat{h}_{t+}^k(x_{t+}) = \max_{(\alpha_{t+}, \beta_{t+}) \in \mathcal{J}_{t+}} \{\alpha_{t+} + \langle \beta_{t+}, x_{t+} \rangle\}$$

  to obtain

$$\hat{f}_t^k(x_t, u_t) = \langle c_t, x_t \rangle + \langle d_t, u_t \rangle + \sum_{\omega_t \in \hat{\Omega}_t} p(\omega_t) \hat{h}_{t+}^k(x_{t+}) \qquad \text{(A1)}$$

end for

with theoretical justification to obtain an LB with a confidence interval. They choose the minimum of the in-sample LB and in-sample UB as their LB, whose uncertainty is estimated by the uncertainty level of the in-sample UB.

The SDDP algorithm does not provide an upper bound directly. However, optimizing the algorithmic approximation function, the SDDP algorithm constructs an admissible policy. We can obtain an unbiased estimate for the upper

Algorithm 4 Upper Bound Calculation

Initialization: Fixed sample set $\hat{\Omega}$, initial state x_0, and sample size N'.
Forward pass: Simulate the state/decision paths along N' sample paths.

1: **for** $n = 1, \ldots, N'$ **do**
2: Set $x_0^n = x_0$.
3: Generate a sample path $\omega_{[T]}^n$ from $\hat{\Omega}$.
4: **for** $t = 0, \ldots, T$ **do**
5: Solve the optimization problem $u_t^n = \arg\min_{u_t} \hat{f}_t(x_t^n, u_t)$ to obtain the decision.
6: Find the next state based on system dynamics $x_{t+}^n = \mathcal{D}_t\left(x_t^n, u_t^n, \omega_t^n\right)$.
7: **end for**
8: $Q^n = \sum_{t=0}^{T} [\langle c_t, x_t^n \rangle + \langle d_t, u_t^n \rangle]$
9: **end for**

Output: The upper bound sample mean: $\bar{Q} = \frac{1}{N'} \sum_{n=1}^{N'} Q^n$; and the standard deviation of $\bar{Q}$: $\frac{1}{N'} \sqrt{\frac{\sum_{n=1}^{N'} (Q^n - \bar{Q})^2}{N'-1}}$.

bound value by simulating it with Monte Carlo sampling with this admissible policy. We apply this policy to a sampled scenario from the first stage to the end of the horizon, which will provide one estimate of the objective. This estimate is the upper bound of the optimal objective since it applies an implementable policy. Multiple scenarios are usually sampled to estimate the upper bound value accurately. The detailed calculation for the upper bound estimate is shown in Algorithm 4. Since the sample set is the same as the set to obtain the policy, this is also referred to as the in-sample upper bound. In Algorithm 4, if we replace the in-sample set $\hat{\Omega}$ with the 'true' dataset, the output is the out-of-sample upper bound or an upper bound estimate for the 'true' problem.

The commonly used stopping rule checks whether the lower and in-sample upper bounds are close enough. If their difference is less than the predefined tolerance, the algorithm terminates and outputs the policy. However, their difference only indicates the optimality of policy for the SAA problem, not the 'true' problem.

Appendix B The Hydro-Thermal Scheduling Problem

The hydro-thermal scheduling problem we consider is a variety of the one in the sddp.jl tutorial. The goal is to operate one thermal generator and N hydro-dams in a valley chain over τ stages, considering the rainfall uncertainty. The objective is to minimize the generation cost (including the cost of hydro

and thermal generation) while the power demand is satisfied. The problem has a hazard-decision scheme, where, at each stage, the decision is made after observing a realization of the random data.

At stage $t = 1, \ldots, \tau$, we have the N state variables:

- r_t^i: represents the volume in reservoir (hydro generator) $i = 1, \ldots, N$

where the initial states $r_1^i, \forall i = 1, \ldots, N$ are given.

The decision variables are

- g_t: the power generated by the thermal generator;
- o_t^i: the water from reservoir $i = 1, \ldots, N$ used for power generation;
- s_t^i: the water spilling out of reservoir $i = 1, \ldots, N$;
- l_t^i: the water flowing in reservoir $i = 1, \ldots, N$;

All the state/decision variables have the unit MWh. In general, water is measured in m^3. We measure it in the energy-equivalent MWh unit for simplicity. In practice, we can build a function to convert water in m^3 to the value in MWh.

The system dynamics of our problem are as follows:

$$\begin{aligned} r_{t+}^1 &= r_t^1 + l_t^1 - o_t^1 - s_t^1 \\ r_{t+}^i &= r_t^i + l_t^i - o_t^i - s_t^i + o_t^{i-1} + s_t^{i-1}, \forall i = 2, \ldots, N \end{aligned} \tag{B2}$$

And the extra constraints are

- ω_t: rainfall, the stagewise independent random data;
- c_t^0: cost per unit of power generated by the thermal generator;
- c_t^i: cost per unit power generated by the reservoir $i = 1, \ldots, N$;
- a^i: the coefficient for reservoir $i = 1, \ldots, N$ that transform the rainfall ω_t to the inflow l_t^i.
- d_t: the amount of demand.
- b^i: the capacity for reservoir $i = 1, \ldots, N$.

We have the following constraints,

$$\begin{aligned} \sum_{i=1}^{N} o_t^i + g_t &= d_t \\ l_t^i &= a^i \omega_t, \forall i = 1, \ldots, N \end{aligned} \tag{B3}$$

where the first one means that electricity demand is met, and the second is the relation between the rainfall and inflow for all the hydro generators. In addition, all the state variables and decision variables are greater than 0. The volume in each reservoir is less than its capacity.

The stagewise cost is the cost for power generation which is the sum of thermal generator cost and hydro generators cost:

$$c_t^0 g_t + \sum_{i=1}^{N} c_t^i o_t^i \tag{B4}$$

To summarize, the value function at stage $t = 1, \ldots, \tau$, with state r_t and the random data ω_t can be obtained solving the following problem

$$\begin{aligned} h_t(r_t, \omega_t) = \min_{g_t, o_t, s_t, l_t, r_{t+}} \quad & c_t^0 g_t + \sum_{i=1}^{N} c_t^i o_t^i + \mathbb{E}[h_{t+}(r_{t+}, \omega_{t+})] \\ & r_{t+}^1 = r_t^1 + l_t^1 - o_t^1 - s_t^1 \\ & r_{t+}^i = r_t^i + l_t^i - o_t^i - s_t^i + o_t^{i-1} + s_t^{i-1}, \forall i = 2, \ldots, N \\ & \sum_{i=1}^{N} o_t^i + g_t = d_t \\ & l_t^i = a^i \omega_t, \forall i = 1, \ldots, N \\ & g_t, s_t^i, o_t^i, l_t^i \geq 0, \forall i = 1, \ldots, N \\ & 0 \leq r_{t+}^i \leq b^i, \forall i = 1, \ldots, N \end{aligned} \tag{B5}$$

In Section 3.2, the example considers a $\tau = 4$ stage problem, where we have only $N = 1$ hydro generator. In Section 7, we have $N = 4$ hydro generators. In Section 7.1, we consider a $\tau = 10$ stage problem, while in Section 7.2, we consider the problems where $\tau = 24/48/72/96/120$.

Appendix C Proofs

C.1 Proof of Theorem 1

Proof i) First, note that due to the finite support assumption (A5), the dual vectors of the last stage form a finite set. Hence, the "true" value function for replication m (denoted $f_T^m(x_T, u_T)$) is a piecewise linear convex function with finitely many pieces. Since the probability distribution in any replication is fixed, the approximation of the SAA function $\{\check{f}_T^m(x_T, u_T)\}_{m=1}^M$ must have finitely many pieces and is, therefore, piecewise affine. As a result, the average of finitely many value functions, i.e., the average SAA function $\check{F}_T(x_T, u_T)$, is also a function with finitely many pieces.

For any arbitrary feasible point (x_T, u_T), we have

$$\check{F}_T(x_T, u_T) = \frac{1}{M} \sum_{m=1}^{M} \check{f}_T^m(x_T, u_T).$$

Because $\check{f}_T^m(x_T, u_T), \forall m \in [\![1, M]\!]$ are different realizations of SAA function, their variances are approximately the same, equal to σ_{x_T, u_T}^2. By applying the central limit

theorem (CLT), we have,

$$M^{1/2}(\check{F}_T(x_T,u_T)-f_T(x_T,u_T)) \xrightarrow{\mathrm{d}} \mathcal{N}(0,\sigma^2_{x_T,u_T}),$$

where the notation $\xrightarrow{\mathrm{d}}$ denotes convergence in distribution. Since for $m=1,\ldots,M$, the expectation of $\check{f}^m_T(x_T,u_T)$ equals $f_T(x_T,u_T)$, the expectation of $\check{F}_T(x_T,u_T)$ is also $f_T(x_T,u_T)$. We have that $|\check{F}_T(x_T,u_T)-f_T(x_T,u_T)|$ is bounded by $O_p(\sigma_{x_T,u_T}/\sqrt{M})$. Since $\hat{F}_T(x_T,u_T)=\check{F}_T(x_T,u_T)$, $|\hat{F}_T(x_T,u_T)-f_T(x_T,u_T)|$ is bounded by $O_p(\sigma_{x_T,u_T}/\sqrt{M})$.

The above is the ground step of the backward induction, which uses only the finite-dimensional CLT. Having established the property for the last stage (stage T), we now move to stage $T-1$, which also uses a value function with finitely many pieces. At stage $T-1$, because of the finite support assumption (A5) and the known probability distributions, the 'true' function and the SAA functions $\{\check{f}^m_{T-1}(x_{T-1},u_{T-1})\}^M_{m=1}$ have finitely many pieces. Hence, the LP approximation of the value functions used in stage $T-1$ results in a finite-dimensional LP, implying that there are only finitely many dual extreme points in stage $T-1$. By applying the CLT, we have that,

$$M^{1/2}(\check{F}_{T-1}(x_{T-1},u_{T-1})-f_{T-1}(x_{T-1},u_{T-1})) \xrightarrow{\mathrm{d}} \mathcal{N}(0,\sigma^2_{x_{T-1},u_{T-1}}).$$

Because of the equality of $\check{F}_{T-1}(x_{T-1},u_{T-1})$ and $\hat{F}_{T-1}(x_{T-1},u_{T-1})$, we have that $|\hat{F}_{T-1}(x_{T-1},u_{T-1})-f_{T-1}(x_{T-1},u_{T-1})|$ is bounded by $O_p(\sigma_{x_{T-1},u_{T-1}}/\sqrt{M})$. Proceeding in this manner, we conclude that the value functions in all stages are piecewise linear functions. Moreover, in all stages, we apply the CLT to assert that, $|\hat{F}_t(x_t,u_t)-f_T(x_t,u_t)|\le O_p(\sigma_{x_t,u_t}/\sqrt{M})$.

ii) Next, we show that the compromise policy has reduced variance. Recall that we formulate the compromise policy at the end of running SDDP. We have already shown that the compromise function at any particular point has reduced variance compared with the variance associated with individual value functions of any replication. That is, $|\hat{F}_t(x_t,u_t)-f_T(x_t,u_t)|\le O_p(\sigma_{x_t,u_t}/\sqrt{M})$, while $|\hat{f}^m_t(x_t,u_t)-f_T(x_t,u_t)|\le O_p(\sigma_{x_t,u_t})$. At any stage, the compromise policy is averaged over finitely many replications. Since $\{\hat{f}^m_t(x_t,u_t)\}^M_{m=1}$ consist of finitely many pieces, their average function $\hat{F}_t(x_t,u_t)$ also consists of finitely many pieces. Moreover, with the finite support assumption (A5), the number of paths going forward is finite. In the following, we will prove the variance reduction of the compromise policy from the root node to the terminal node.

At the root node, the initial stage x_0 is given. $\mathcal{S}_0$ denotes the set of optimal decisions. $Y(u_0)$ is defined as a random variable following a normal distribution with mean 0 and variance $\sigma^2_{x_0,u_0}$, written $Y(u_0)\sim\mathcal{N}(0,\sigma^2_{x_0,u_0})$. With the assumptions (A1) and (A2), the objective value is finite valued. Together with the assumption (A4), the expected value function has a finite value at any point. Since the value functions at all stages are piecewise linear functions, the variance is also finite. By applying the Theorem 5.7 in [21], we have

$$\begin{aligned}&\hat{v}^c_0=\min_{u_0\in\mathcal{S}_0}\hat{F}_0(x_0,u_0)+O_p((N\times M)^{-1/2}),\\&M^{1/2}(\hat{v}^c_0-v^*_0)\xrightarrow{\mathrm{d}}\min_{u_0\in\mathcal{S}_0}Y(u_0).\end{aligned}\tag{C6}$$

If, moreover, $\mathcal{S}_0=\{\hat{u}_0\}$ is a singleton, then

$$M^{1/2}(\hat{v}^c_0-v^*_0)\xrightarrow{\mathrm{d}}\mathcal{N}(0,\sigma^2_{x_0,\hat{u}_0}).\tag{C7}$$

Similarly, applying the same result (Theorem 5.7 in [21]) to the individual policies, we have

$$\begin{aligned} &\hat{v}_0^m = \min_{u_0 \in \mathcal{S}_0} \hat{f}_0^m(x_0, u_0) + O_p(N^{-1/2}), \forall m \in [\![1, M]\!], \\ &\hat{v}_0^m - v_0^* \xrightarrow{\mathrm{d}} \min_{u_0 \in \mathcal{S}_0} Y(u_0). \end{aligned} \tag{C8}$$

When $\mathcal{S}_0 = \{\hat{u}_0\}$ is a singleton,

$$(\hat{v}_0^m - v_0^*) \xrightarrow{\mathrm{d}} \mathcal{N}(0, \sigma^2_{x_0, \hat{u}_0}), \forall m \in [\![1, M]\!]. \tag{C9}$$

Thus, when we denote $\hat{\sigma}_0^2$ as the variance of $|\hat{v}_0^m - v_0^*|$, we have that $|\hat{v}_0^c - v_0^*|$ is bounded by $O_p(\hat{\sigma}_0/\sqrt{M})$. As a result, we have that the variance of the compromise policy at the root node defined as

$$\hat{\mu}_0^c(x_0) := \arg\min_{u_0 \in \mathcal{U}_0} \hat{F}_0(x_0, u_0) = \arg\min_{u_0 \in \mathcal{U}_0} \sum_{m=1}^{M} \hat{f}_0^m(x_0, u_0),$$

is $1/M$ of the variance of the individual policy, defined as

$$\hat{\mu}_0^m(x_0) := \arg\min_{u_0 \in \mathcal{U}_0} \hat{f}_0^m(x_0, u_0).$$

We implement the compromise policy $\hat{\mu}_0^c(x_0)$, and apply the system dynamics $\mathcal{D}_t(x_t, u_t, \omega_t)$ to achieve the state x_1 at the second stage. First, note that the multi-stage model assumes finite support, and the SDDP algorithm visits a finite number of nodes of the first stage. Hence SDDP estimates the first-stage value using optimal LP values from a subset of first-stage values of the sub-sampled scenario tree. If we replicate SDDP runs M times, then using Theorem 5.7 in [21] provides a variance reduction property for the second-stage optimal values in the form $(\hat{\sigma}_1(x_1))^2$ as the variance of $|\hat{v}_1^m(x_1) - v_1^*(x_1)|$. Then it follows that $|\hat{v}_1^c(x_1) - v_1^*(x_1)| \leq O_p(\hat{\sigma}_1(x_1)/\sqrt{M})$.

Moving forward recursively, we conclude that the variance of the compromise policy is $1/M$ of the variance of the individual policies. At stage t with state x_t, suppose we denote $(\hat{\sigma}_t(x_t))^2$ as the variance of $|\hat{v}_t^m(x_t) - v_t^*(x_t)|$. Then it follows that $|\hat{v}_t^c(x_t) - v_t^*(x_t)| \leq O_p(\hat{\sigma}_t(x_t)/\sqrt{M})$.

□

C.2 Proof of Theorem 2

Proof Based on the assumptions, $f_t(\hat{x}_t, \cdot)$ is a convex Lipschitz continuous function. Since $\bar{u}_t$ satisfies (16a), there exists a neighborhood of $\bar{u}_t$, denoted as $NE(\bar{u}_t)$, such that $u_t \in NE(\bar{u}_t)$ implies $f_t(\hat{x}_t, u_t) \leq \frac{1}{M} \sum_{m=1}^{M} f_t(\hat{x}_t, \hat{u}_t^m)$. If $\hat{u}_t^c \in NE(\bar{u}_t)$, then,

$$\begin{aligned} f_t(\hat{x}_t, \hat{u}_t^c) &\leq \frac{1}{M} \sum_{m=1}^{M} f_t(\hat{x}_t, \hat{u}_t^m) \\ &= \frac{1}{M} \sum_{m=1}^{M} \hat{f}_t^m(\hat{x}_t, \hat{u}_t^m) + \frac{1}{M} \sum_{m=1}^{M} \left(f_t(\hat{x}_t, \hat{u}_t^m) - \hat{f}_t^m(\hat{x}_t, \hat{u}_t^m) \right) \\ &\leq \frac{1}{M} \sum_{m=1}^{M} \hat{f}_t^m(\hat{x}_t, \hat{u}_t^c) + \frac{1}{M} \sum_{m=1}^{M} \left(f_t(\hat{x}_t, \hat{u}_t^m) - \hat{f}_t^m(\hat{x}_t, \hat{u}_t^m) \right) \\ &= \hat{F}_t(\hat{x}_t, \hat{u}_t^c) + \frac{1}{M} \sum_{m=1}^{M} \left(f_t(\hat{x}_t, \hat{u}_t^m) - \hat{f}_t^m(\hat{x}_t, \hat{u}_t^m) \right) \end{aligned} \tag{C10}$$

Since $|f_t(\hat{x}_t, \hat{u}_t^m) - \hat{f}_t^m(\hat{x}_t, \hat{u}_t^m)|$ is bounded by $O_p(N^{-1/2})$, we have $\frac{1}{M}\sum_{m=1}^{M} |f_t(\hat{x}_t, \hat{u}_t^m) - \hat{f}_t^m(\hat{x}_t, \hat{u}_t^m)|$ is bounded by $O_p((N \times M)^{-1/2})$. □

C.3 Proof of Theorem 3

Proof Due to linear programming duality, we have $\hat{f}_0^m(x_0, u_0) \leq \check{f}_t^m(x_0, u_0)$ for $m \in [\![1, M]\!]$. Then,

$$\hat{v}_0^m = \min_{u_0} \hat{f}_0^m(x_0, u_0) \leq \min_{u_0} \check{f}_0^m(x_0, u_0).$$

Note that $\check{f}_0^m(x_0, u_0)$ is one realization of the SAA function. Moreover, $\mathbb{E}[\min_{u_0} \check{f}_0(x_0, u_0)] \leq \min_{u_0} \mathbb{E}[\check{f}_0(x_0, u_0)] = v_0^*$. Thus, $\hat{v}_0^m$ is a valid lower bound estimate of v_0^*. Also, $\bar{v}$ is a lower bound estimate of the 'true' optimal value. Since $\hat{F}_0^m(x_0, u_0) \leq \check{F}_0^m(x_0, u_0)$, we have $\hat{v}_0^c = \min_{u_0} \hat{F}_0^m(x_0, u_0) \leq \min_{u_0} \check{F}_0^m(x_0, u_0) \approx v_0^*$. Thus, $\hat{v}_0^c$ is a valid lower bound estimate. In addition,

$$\hat{v}_0^c = \min_{u_0} \hat{F}_0(x_0, u_0) = \min_{u_0} \frac{1}{M} \sum_{m=1}^{M} \hat{f}_0^m(x_0, u_0) \geq \frac{1}{M} \sum_{m=1}^{M} \min_{u_0} \hat{f}_0^m(x_0, u_0) = \bar{v}.$$

□

References

[1] Carino, D.R., Myers, D.H., Ziemba, W.T.: Concepts, technical issues, and uses of the Russell-Yasuda-Kasai financial planning model. Operations Research **46**(4), 450–462 (1998)

[2] Golari, M., Fan, N., Jin, T.: Multistage stochastic optimization for production-inventory planning with intermittent renewable energy. Production and Operations Management **26**(3), 409–425 (2017)

[3] Shiina, T., Birge, J.R.: Multistage stochastic programming model for electric power capacity expansion problem. Japan Journal of Industrial and Applied Mathematics **20**(3), 379–397 (2003)

[4] Gangammanavar, H., Sen, S.: Two-scale stochastic optimization for controlling distributed storage devices. IEEE Transactions on Smart Grid **9**(4), 2691–2702 (2016)

[5] Bertsekas, D.: Dynamic Programming and Optimal Control: Volume I. Athena scientific (2012)

[6] Powell, W.B.: Approximate Dynamic Programming: Solving the Curses of Dimensionality. John Wiley & Sons (2007)

[7] Dyer, M., Stougie, L.: Computational complexity of stochastic programming problems. Mathematical Programming **106**(3), 423–432 (2006)

[8] Hanasusanto, G.A., Kuhn, D., Wiesemann, W.: A comment on "Computational Complexity of Stochastic Programming Problems". Mathematical Programming **159**(1), 557–569 (2016)

[9] Shapiro, A.: On complexity of multistage stochastic programs. Operations Research Letters **34**(1), 1–8 (2006)

[10] Dowson, O.: Applying stochastic optimisation to the new zealand dairy industry. PhD thesis, University of Auckland (2018)

[11] Philpott, A.: Applications of sddp in electricity markets with hydroelectricity. In: SESO Workshop (2017). http://cermics.enpc.fr/~delara/SESO/SESO2017/SESO2017_Thursday_Philpott.pdf

[12] Pereira, M.V., Pinto, L.M.: Multi-stage stochastic optimization applied to energy planning. Mathematical programming **52**(1), 359–375 (1991)

[13] Gangammanavar, H., Sen, S.: Stochastic dynamic linear programming: A sequential sampling algorithm for multistage stochastic linear programming. SIAM Journal on Optimization **31**(3), 2111–2140 (2021)

[14] Sen, S., Liu, Y.: Mitigating uncertainty via compromise decisions in two-stage stochastic linear programming: Variance reduction. Operations Research **64**(6), 1422–1437 (2016)

[15] Dowson, O., Kapelevich, L.: SDDP.jl: A Julia package for stochastic dual dynamic programming. INFORMS Journal on Computing **33**, 27–33 (2021). https://doi.org/10.1287/ijoc.2020.0987

[16] Donohue, C.J., Birge, J.R.: The abridged nested decomposition method for multistage stochastic linear programs with relatively complete recourse. Algorithmic Operations Research **1**(1) (2006)

[17] Shapiro, A.: Analysis of stochastic dual dynamic programming method. European Journal of Operational Research **209**(1), 63–72 (2011)

[18] Homem-de-Mello, T., De Matos, V.L., Finardi, E.C.: Sampling strategies and stopping criteria for stochastic dual dynamic programming: a case study in long-term hydrothermal scheduling. Energy Systems **2**(1), 1–31 (2011)

[19] Philpott, A.B., Guan, Z.: On the convergence of stochastic dual dynamic programming and related methods. Operations Research Letters **36**(4), 450–455 (2008)

[20] Dowson, O.: The policy graph decomposition of multistage stochastic

programming problems. Networks **76**(1), 3–23 (2020)

[21] Shapiro, A., Dentcheva, D., Ruszczyński, A.: Lectures on Stochastic Programming: Modeling and Theory. SIAM (2014)

[22] Sen, S., Liu, Y.: Mitigating uncertainty via compromise decisions in two-stage stochastic linear programming: Variance reduction. Operations Research **64**(6), 1422–1437 (2016)

[23] Bayraksan, G., Morton, D.P.: Assessing solution quality in stochastic programs. Mathematical Programming **108**(2), 495–514 (2006)

[24] Smith, J.E., Winkler, R.L.: The optimizer's curse: Skepticism and postdecision surprise in decision analysis. Management Science **52**(3), 311–322 (2006)

[25] Peer, O., Tessler, C., Merlis, N., Meir, R.: Ensemble bootstrapping for q-learning. In: Proceedings of the 38th International Conference on Machine Learning, ICML 2021. Proceedings of Machine Learning Research, vol. 139, pp. 8454–8463 (2021). PMLR

[26] Mannor, S., Simester, D., Sun, P., Tsitsiklis, J.N.: Bias and variance approximation in value function estimates. Management Science **53**(2), 308–322 (2007)

[27] Baucke, R., Downward, A., Zakeri, G.: A deterministic algorithm for solving multistage stochastic programming problems. Optimization Online, 1–25 (2017)

[28] Higle, J.L., Sen, S.: Finite master programs in regularized stochastic decomposition. Mathematical Programming **67**(1), 143–168 (1994)

[29] Deng, Y., Kesselman, C., Sen, S., Xu, J.: Computational operations research exchange (core): A cyber-infrastructure for analytics. In: 2019 Winter Simulation Conference (WSC), pp. 3447–3456 (2019). IEEE

[30] Rougé, C., Tilmant, A.: Using stochastic dual dynamic programming in problems with multiple near-optimal solutions. Water Resources Research **52**(5), 4151–4163 (2016)

[31] De Matos, V.L., Morton, D.P., Finardi, E.C.: Assessing policy quality in a multistage stochastic program for long-term hydrothermal scheduling. Annals of Operations Research **253**(2), 713–731 (2017)