Computational Optimization and Applications manuscript No.
(will be inserted by the editor)

Adaptive Partition-based SDDP Algorithms for Multistage Stochastic Linear Programming with Fixed Recourse

Murwan Siddig · **Yongjia Song**

Received: date / Accepted: date

Abstract In this paper, we extend the adaptive partition-based approach for solving two-stage stochastic programs with fixed recourse matrix and fixed cost vector to the multistage stochastic programming setting where the stochastic process is assumed to be stage-wise independent. The proposed algorithms integrate the adaptive partition-based strategy with a popular approach for solving multistage stochastic programs, the stochastic dual dynamic programming (SDDP) algorithm, according to two main strategies. These two strategies are distinct from each other in the manner by which they refine the partitions during the solution process. In particular, we propose a refinement outside SDDP strategy whereby we iteratively solve a coarse scenario tree induced by the partitions, and refine the partitions in a separate step outside of SDDP, only when necessary. We also propose a refinement within SDDP strategy where the partitions are refined in conjunction with the machinery of the SDDP algorithm. We then use, within the two different refinement schemes, different tree-traversal strategies which allow us to have some control over the size of the partitions. We performed numerical experiments on a hydro-thermal power generation planning problem. Numerical results show the effectiveness of the proposed algorithms that use the refinement outside SDDP strategy in comparison to the standard SDDP algorithm and algorithms that use the refinement within SDDP strategy.

Keywords Stochastic optimization · Multistage stochastic linear programs · Partition-based approach · SDDP algorithm

1 Introduction

Multistage stochastic programming is a well-recognized mathematical optimization model for problems that require optimization under uncertainty over time. These problem arise in

M. Siddig
Clemson University, Clemson, SC, USA
E-mail: msiddig@clemson.edu

Y. Song
Clemson University, Clemson, SC, USA
E-mail: yongjis@clemson.edu

a variety of applications, such as energy [17, 25, 26, 30, 33, 38], finance [4, 11, 14, 15], transportation [5, 16, 20] and sports [29], among others. In particular, in this paper, we consider a class of multistage stochastic programming models which have applications in long-term hydro-thermal power generation planning, in which one aims to construct an optimal operational strategy under the uncertainty of rainfall volume in order to minimize power generation cost to meet deterministic demand. The sequential nature in the decision-making structure and the uncertainty in the problem data such as, future (water) inflows, demand, fuel cost, etc., make the hydro-thermal power generation planning problem a classical problem in applications of multistage stochastic programming.

In the stochastic programming literature, there is a good deal of work on how to tackle such problems in a computationally tractable fashion. One typical approach is to approximate the underlying stochastic process governing the uncertainty in the problem data using scenario trees. The result of doing this is that, as the number of decision stages in the planning horizon increases, the increasing scenario tree size requires an exponential growth of computational resources for solving the corresponding multistage stochastic program [7]. As such, it becomes necessary to use specialized algorithms which employ decomposition techniques to solve the resulting large-scale mathematical programs.

In this paper we are concerned with the computational efficiency for solving multistage stochastic programming problems based on enhancements to one of the most successful decomposition algorithms for these problems, the *Stochastic Dual Dynamic Programming* (SDDP) algorithm [31]. The proposed enhancements are based on the idea of employing adaptive partition-based formulation [2, 40], which gives a relaxation of the original mathematical program obtained by aggregating variables and constraints according to a partition over the set of scenarios. Using the dual information associated with each scenario, the partition can be refined during the solution process until it yields an optimal solution to the problem. Partition-based strategies have shown to be very effective in the two-stage stochastic programming setting due to the reduced computational effort in generating cutting planes that are adaptively refined in the solution process. The task of integrating adaptive partition-based strategies to the SDDP algorithm poses the following questions:

1. How to choose the way for the SDDP algorithm to traverse the scenario tree with aggregated variables and constraints according to the scenario partition in each stage?
2. For a given tree-traversal strategy, how accurate should the solution be during different phases of the solution process?
3. How should we refine the scenario partition in each stage?

Using different strategies to traverse the scenario tree, refining the partition in an *adaptive partition-based SDDP* algorithm can be done in many different ways. We investigate two different strategies, namely *refinement outside of the SDDP algorithm*, and *refinement within the SDDP algorithm*. Moreover, we develop a method which exploits the nature in problems of optimal policies with special structures. This is done by incorporating partition-based strategies only to a selected subset of stages in the planning horizon, while the standard approach in SDDP is applied to other stages.

The rest of this paper is organized as follows: in Section 2 we introduce a general multistage stochastic programming formulation and provide an overview of the theoretical background in adaptive partition-based strategies and the SDDP algorithm. In Section 3, we demonstrate how the adaptive partition-based approach can be used in the multistage setting, provide the ingredients of our proposed algorithms and show its finite convergence. In Section 4 we describe in details three types of adaptive partition-based SDDP algorithms: *Refinement outside SDDP*, *Refinement within SDDP* and *Adaptive partition-based SDDP*

with structured policies cut generation. In Section 5 an extensive computational analysis is presented, and the proposed approach is compared with alternative approaches in terms of their computational performance. Finally, in Section 6 we conclude with some final remarks.

2 Preliminaries on Multistage Stochastic Linear Programs and Decomposition Schemes

In stochastic programming (SP) models, the underlying data uncertainty involved in the problem is characterized by a random vector ξ with known probability distribution. In its most simple form, two-stage SP, two kinds of decisions are involved: *first-stage* decisions x_1 that are made prior to the realization of random vector ξ, and *second-stage* recourse decisions $x_2 := x_2(\xi)$ that are made after observing the realization of random vector ξ. A two-stage stochastic linear program (2SLP) can be formulated as follows

$$\min_{\substack{A_1x_1=b_1\\ x_1\in\mathbb{R}^{n_1}_+}} c_1^\top x_1 + \mathbb{E}_\xi \left[\min_{\substack{B_2x_1+A_2x_2=b_2\\ x_2\in\mathbb{R}^{n_2}_+}} c_2^\top x_2 \right], \tag{1}$$

where $\xi = (c_2, B_2, A_2, b_2)$, and the expectation $\mathbb{E}_\xi[\cdot]$ is taken with respect to the probability measure of random vector $\xi \in \Xi$.

Multistage stochastic linear programs (MSLPs) provide an explicit framework which generalizes the 2SLP for multiple stages of sequential decision making under uncertainty. In a planning horizon of T stages, the dynamic realization of uncertainty is typically modeled as a stochastic process $(\xi_1, \xi_2, \dots \xi_T)$ such that, ξ_1 is deterministic, and for each $t = 2,3,\dots,T$, $\xi_t \in \Xi_t$ is random vector that will be realized in stage t. The history of this stochastic process up to time t is denoted by $\xi_{[t]} := (\xi_1, \dots \xi_t)$. The nested form of an MSLP can be expressed as:

$$\min_{\substack{A_1x_1=b_1\\ x_1\in\mathbb{R}^{n_1}_+}} c_1^\top x_1 + \mathbb{E}_{|\xi_{[1]}} \left[\min_{\substack{B_2x_1+A_2x_2=b_2\\ x_2\in\mathbb{R}^{n_2}_+}} c_2^\top x_2 + \mathbb{E}_{|\xi_{[2]}} \left[\cdots + \mathbb{E}_{|\xi_{[T-1]}} \left[\min_{\substack{B_Tx_{T-1}+A_Tx_T=b_T\\ x_T\in\mathbb{R}^{n_T}_+}} c_T^\top x_T \right] \right] \right], \tag{2}$$

where some (or all) data $\xi_t = (c_t, B_t, A_t, b_t)$ can be subject to uncertainty for $t = 2,\dots,T$. The expectation $\mathbb{E}_{|\xi_{[t]}}[\cdot]$ on each stage t is taken with respect to the probability measure of the future, conditional on the past. The sequence of decisions $(x_1, x_2, \dots, x_t)$ where $x_t = x_t(\xi_{[t]}), \forall\, t = 1,2,\dots,T$ is referred to as a decision policy for problem (2). Such policy provides a decision rule at every stage t based on the realization of the data process up to time t. The aim of an MSLP is to find an optimal policy to (2).

The question of how to construct scenarios to induce a decision policy and measure its quality is beyond the scope of this paper. In this paper, we assume that a scenario tree $\mathscr{T}$ is given, where a finite number of realizations is available for each ξ_t, $\forall\, t = 2,3,\dots,T$. As such, (1) and (2) can be written as large-scale linear programs, known as the *deterministic equivalent programs* (DEP). This is, for instance, the case in which (2) is a sample average approximation (SAA) of an original MSLP where the underlying random vector ξ_t in each stage t follows a continuous probability distribution. We refer the reader to [36] for a discussion on the relationship between an SAA problem and the original MSLP with a continuous distribution.

For the remainder of the paper, considering the complex nature of notation in MSLP, we shall reserve all the subscript notations to the standard notation for stages in MSLP, and all the superscript notations for scenarios, sample paths and everything invoked by the usage of partition-based strategies.

2.1 Preliminaries on Two-stage Stochastic Linear Programs and the Adaptive Partition-based Approach

Consider solving the following DEP for the 2SLP (1) with a set of scenarios indexed by $N = \{1, 2, \dots |\Xi|\}$ and assume that each scenario k happens with probability p^k, $\forall\, k \in N$:

$$\begin{aligned} z^* = \min\ & c_1^\top x_1 + \sum_{k\in N} c_2^{k\top} x_2^k \times p^k \\ \text{s.t.}\ & A_1 x_1 = b_1 \\ & B_2^k x_1 + A_2^k x_2^k = b_2^k, \quad \forall\, k \in N \\ & x_1 \in \mathbb{R}_+^{n_1}, x_2^k \in \mathbb{R}_+^{n_2},\ \forall\, k \in N. \end{aligned} \tag{3}$$

It can be easily seen that, as the number of scenarios grows, the DEP (3) becomes computationally challenging to solve. Nevertheless, the DEP has a special structure that lends itself to decomposition techniques developed to solve large-scale LPs. These include variants of *Benders decomposition* (also called the L-shaped method [39]) for the two-stage setting, which generalizes to *nested Benders decomposition* for MSLPs.

2.1.1 Benders decomposition and inexact oracles

Standard Benders decomposition consists of iteratively solving the so-called *master problem* (4)

$$\begin{aligned} \min_{x_1 \in \mathbb{R}_+^{n_1}}\ & c_1^\top x_1 + \check{\mathfrak{Q}}(x_1) \\ \text{s.t.}\ & A_1 x_1 = b_1, \end{aligned} \tag{4}$$

where $\check{\mathfrak{Q}}(x_1)$ is a cutting-plane outer approximation to the second-stage expected cost function $\mathfrak{Q}(x_1) := \mathbb{E}_\xi[Q(x_1,\xi)] = \sum_{k\in N} Q(x_1,\xi^k) \times p^k$, and for each $k \in N$:

$$Q(x_1,\xi^k) = \min_{x_2^k \in \mathbb{R}_+^{n_2}} \left\{ c_2^{k\top} x_2^k \mid B_2^k x_1 + A_2^k x_2^k = b_2^k \right\}. \tag{5}$$

An optimal solution to the master problem, denoted by x_1, will be sent to the second stage and evaluated by solving each of the $k \in N$ scenarios subproblem (5). The cutting plane outer approximation $\check{\mathfrak{Q}}(\cdot)$ is then improved by adding *optimality cuts* generated using information obtained from solving the subproblems (*feasibility cuts* are added when any subproblem is infeasible). We refer the reader to [9] and [22] for a detailed discussion on the topic.

From an abstract viewpoint coming from nonsmooth optimization, such decomposition schemes can be seen as variants of Kelley's cutting plane method for solving convex programs [23]. In the terminology of convex programming, an *oracle* is referred to as a routine that returns the function value information and the subgradient information at a given point x_1. If the information provided by the routine is accurate, we call the oracle an *exact oracle*. Otherwise, the oracle is called *inexact oracle*. In the context of the 2SLP, oracles constructed by solving the scenario subproblem (5) for each $k \in N$ correspond to exact oracles, yielding exact function value $Q(x_1,\xi^k)$ and subgradient information from the optimal dual solutions.

It is intuitively clear that when the number of scenarios $|\Xi|$ is large, the exact oracle could be computationally expensive. As such, it may be beneficial to use a relatively coarse oracle at the beginning of the algorithm, which is just used to get the process "warm-started"; the exact oracle is used afterwards whenever it is *necessary*. Oliveira and Sagastizabal [28] formalize this idea, and introduce the concept of inexact oracle with on-demand accuracy for a generic (nonsmooth) convex optimization problem. Inexact oracles can be constructed in many different ways, however, in the context of 2SLPs, van Ackooij, Oliveira, and Song [2] study an inexact oracle defined by scenario partitions, which we explain in detail next.

2.1.2 Adaptive partition-based inexact oracles

Definition 1 *A partition $\mathscr{N} = \{\mathscr{P}^1, \mathscr{P}^2, \ldots, \mathscr{P}^L\}$ of the scenario set N is a collection of nonempty subsets of scenarios such that, $\mathscr{P}^1 \cup \mathscr{P}^2 \cup \ldots \mathscr{P}^L = N$, and $\mathscr{P}^\ell \cap \mathscr{P}^{\ell'} = \emptyset$, $\forall\, \ell, \ell' \in \{1,2,\ldots,L\}, \ell \neq \ell'$. Each of these subsets is called a scenario cluster.*

For a given partition $\mathscr{N}$, the second-stage expected cost function $\mathfrak{Q}(x_1)$ can be alternatively written as:

$$\mathfrak{Q}(x_1) = \sum_{k \in N} Q(x_1, \xi^k) \times p^k = \sum_{\ell=1}^{L} \sum_{k \in \mathscr{P}^\ell} Q(x_1, \xi^k) \times p^k. \tag{6}$$

As a mean to solve the problem faster, a very natural idea is to partition the scenario set N into L clusters, such that $N = \cup_{\ell=1}^{L} \mathscr{P}^\ell$, and each cluster $\mathscr{P}^\ell$ is represented by a single scenario $\bar{\xi}^\ell$ which "aggregates" the information from all of the scenarios $k \in \mathscr{P}^\ell$. These representative scenarios are then used to approximate $\mathfrak{Q}(\cdot)$. The idea of using aggregations has been well studied in the literature, see, e.g., [6, 8, 19, 21, 34, 35, 41, 43, 44, 45]. Nevertheless, the aggregation method presented in [40] and further improved in [2] is distinct from those existing in the literature in that, instead of applying the scenario clustering and reduction technique to the original scenario set N in a static manner, the scenario partitions are updated dynamically during the iterative solution process and these updates (refinements) of the partitions are guided by the intermediate solutions. Such adaptability makes computational effort spent on constructing the outer approximation of $\mathfrak{Q}(\cdot)$ adaptive to the quality of the intermediate candidate solutions.

There are many ways in which one can choose the representative realization $\bar{\xi}^\ell$. However, it is not clear whether this resulting partition-based approximation of the recourse function, which we denote by $\bar{\mathfrak{Q}}(\cdot)$, would overestimate or underestimate $\mathfrak{Q}(\cdot)$, and more importantly, what is more desired. From an algorithmic point of view, since our goal is to use the second-stage subproblems to construct a cutting-plane *outer* approximation for $\mathfrak{Q}(\cdot)$, it is more reasonable to choose $\bar{\xi}^\ell$ in such a way so that the resulting approximate recourse function $\bar{\mathfrak{Q}}(\cdot)$ underestimates the true function $\mathfrak{Q}(\cdot)$. One possible way to proceed is to define $\bar{\xi}^\ell := \mathbb{E}[\xi | \xi \in \mathscr{P}^\ell] = \sum_{k \in \mathscr{P}^\ell} \frac{p^k}{\bar{p}^\ell} \times \xi^k$, for $\ell = 1,2,\ldots,L$ where $\bar{p}^\ell = \sum_{k \in \mathscr{P}^\ell} p^k$ is the weight associated with every cluster $\mathscr{P}^\ell$, for $\ell = 1,\ldots,L$. Nonetheless, this definition does not suffice for $\bar{\mathfrak{Q}}(\cdot)$ to be an underestimator of $\mathfrak{Q}(\cdot)$ and it rather hinges on the following assumption which we maintain for the remainder of this work. Note that this assumption was also presented in [40].

Assumption 1 *Fixed recourse. We assume that the recourse matrix A_t and the cost vector c_t for $t = 2,\ldots,T$, are the same for all realizations of ξ_t. Hence, the uncertainty in each stage is characterized only by $\{\xi_t^k = (B_t^k, b_t^k)\}_{k=1}^{|\Xi_t|}$.*

Given a partition $\mathscr{N}$, let us define $\bar{\xi}^\ell = (\bar{B}_2^\ell, \bar{b}_2^\ell)$ as the realization of the random vector ξ chosen to represent cluster $\mathscr{P}^\ell$, where $\bar{B}_2^\ell = \sum_{k\in\mathscr{P}^\ell} \frac{p^k}{\bar{p}^\ell} \times B_2^k$ and $\bar{b}_2^\ell = \sum_{k\in\mathscr{P}^\ell} \frac{p^k}{\bar{p}^\ell} \times b_2^k$. Starting with an initial partition $\mathscr{N}$ (e.g., we typically start with a partition $\mathscr{N}$ where all scenarios are aggregated in a single cluster, i.e., $\mathscr{N} = \{\{1,2,\ldots,|\Xi|\}\}$), the method proposed in [2] proceeds by solving the master problem (4) where the recourse function $\mathfrak{Q}(\cdot)$ is approximated by:

$$\bar{\mathfrak{Q}}(x_1) = \sum_{\ell=1}^{L} Q(x_1, \bar{\xi}^\ell) \times \bar{p}^\ell \tag{7}$$

and

$$Q(x_1, \bar{\xi}^\ell) := \min_{x_2\in\mathbb{R}_+^{n_2}} \left\{ c_2^\top x_2 \mid \bar{B}_2^\ell x_1 + A_2 x_2 = \bar{b}_2^\ell \right\}. \tag{8}$$

We make the distinction between (5) and (8) by referring to (5) as the *scenario-based* subproblem and (8) as the *partition-based* subproblem. To that end, since $Q(x_1,\cdot)$ is convex (a consequence of Assumption 1), it follows from Jensen's inequality that $\mathbb{E}[Q(x_1,\xi)|\xi \in \mathscr{P}^\ell] \geq Q(x_1, \mathbb{E}[\xi|\xi\in\mathscr{P}^\ell])$. Hence, by multiplying both sides of this inequality by p^ℓ and summing over ℓ we obtain that $\mathfrak{Q}(x_1) \geq \bar{\mathfrak{Q}}(x_1)$.

Definition 2 *Let $z_{\mathscr{N}} = \min_{x_1\in\mathbb{R}_+^{n_1}} \left\{ c_1^\top x_1 + \bar{\mathfrak{Q}}(x_1) \mid A_1 x_1 = b_1 \right\}$ be the optimal objective of the 2SLP (4) when the recourse function $\mathfrak{Q}(x_1)$ is approximated by $\bar{\mathfrak{Q}}(x_1)$ using the partition $\mathscr{N}$. A partition $\mathscr{N}$ is ε-sufficient if $z_{\mathscr{N}} \geq z^* - \varepsilon$, where $\varepsilon > 0$ is a chosen parameter. In particular, we say that $\mathscr{N}$ is completely sufficient when $z_{\mathscr{N}} = z^*$, i.e., $\mathscr{N}$ is 0-sufficient [40].*

Partition refinement. To demonstrate how the procedure of refining the partition works, it is important to note that as a result of Assumption 1, the dual feasible region for both the second-stage *scenario-based* subproblem (5) and *partition-based* subproblem (8) are the same for every scenario $k \in N$ and every cluster $\mathscr{P}^\ell$ for $\ell = 1,\ldots,L$, which is given by $\Pi := \{\pi \in \mathbb{R}^{m_2} \mid A_2^\top \pi \leq c_2\}$. This means that any dual solution that is feasible to (5) is also feasible to (8) and vice versa. Moreover, by strong duality we have that

$$Q(x_1, \xi^k) := \max_{\pi\in\mathbb{R}^{m_2}} \left\{ (b_2^k - B_2^k x_1)^\top \pi \mid \ \pi \in \Pi \right\}. \tag{9}$$

As such, the inexactness of a partition-based oracle $\bar{\mathfrak{Q}}(x_1)$ obtained by solving (9) for every cluster $\mathscr{P}^\ell \in \mathscr{N}$ is given by the gap between $\mathfrak{Q}(x_1)$ and $\bar{\mathfrak{Q}}(x_1)$. This can be reduced by the so-called *partition refinement* step. We define that $\mathscr{N}'$ is a refinement of $\mathscr{N}$, if $L' > L$ where L is the number of clusters in $\mathscr{N}$ and L' is the number of clusters in $\mathscr{N}'$, and $\forall\ \mathscr{P}' \in \mathscr{N}'$, $\exists\ \mathscr{P} \in \mathscr{N}$ such that $\mathscr{P}' \subseteq \mathscr{P}$, i.e., $\mathscr{P}'$ is obtained by subdividing some clusters (if any) of $\mathscr{N}$. Song and Luedtke [40] propose a refinement strategy driven by the optimal dual solutions where, whenever the refinement step is done, the gap between the resulting (refined) partition-based oracle $\bar{\mathfrak{Q}}'(x_1)$ and $\bar{\mathfrak{Q}}(x_1)$ shrinks. In other words, after the refinement step we have that $\mathfrak{Q}(x_1) \geq \bar{\mathfrak{Q}}'(x_1) > \bar{\mathfrak{Q}}(x_1)$. This refinement strategy and the resulting conclusion is motivated by the following lemma:

Lemma 1 *[40, Lemma 2.4] Let $\mathscr{N} = \{\mathscr{P}^1, \mathscr{P}^2, \ldots \mathscr{P}^L\}$ be a partition of the scenario set N, and define $\Pi^*(x_1,\xi^k) := \arg\max_{\pi^k\in\Pi}\{(b_2^k - B_2^k x_1)^\top \pi^k\}$ as the set of dual optimal solutions to the subproblem (9) given $x_1 \in \chi_1$. If for every cluster $\mathscr{P}^\ell \in \mathscr{N}$, there exists an optimal dual solution $\bar{\pi}^\ell \in \Pi^*(x_1,\bar{\xi}^\ell)$, such that $\bar{\pi}^\ell \in \cap_{k\in\mathscr{P}^\ell}\Pi^*(x_1,\xi^k)$, then $\mathfrak{Q}(x_1) = \bar{\mathfrak{Q}}(x_1)$. Otherwise, $\mathfrak{Q}(x_1) > \bar{\mathfrak{Q}}(x_1)$.*

The intuitive explanation to Lemma 1 is that, if by solving the dual problem (9) with $\bar{\xi}^\ell$ for every cluster $\mathscr{P}^\ell \in \mathscr{N}$ one can get an optimal dual solution $\bar{\pi}^\ell$ that is also optimal for (9) with ξ^k for $k \in \mathscr{P}^\ell$, then $\mathfrak{Q}(x_1) = \bar{\mathfrak{Q}}(x_1)$. However, when such dual optimal solution $\bar{\pi}^\ell$ does not exist, a gap between $\mathfrak{Q}(x_1)$ and $\bar{\mathfrak{Q}}(x_1)$ must be present. Therefore, given a candidate solution $x_1 \in \chi_1$, whenever $\mathfrak{Q}(x_1) > \bar{\mathfrak{Q}}(x_1)$ a natural way to do the refinement in any cluster $\mathscr{P}^\ell \in \mathscr{N}$, $\forall\, \ell = 1,\ldots,L$ is to compare the dual optimal solution of every scenario-based subproblem (9) $k \in \mathscr{P}^\ell$ so that after the refinement is done, scenarios within the same cluster in the original partition which share the same optimal dual solution are kept together in the same cluster in the refined partition, and those that do not share the same optimal dual solution are split apart in different clusters in the refined partition. Please refer to [2, 40] for the complete proof of Lemma 1 and several refinement strategies that can be derived from it.

Our refinement strategy and its computational complexity. In this paper we will adopt a very simple refinement strategy by which every pair of scenarios within a cluster $\mathscr{P}^\ell$, $\forall\, \ell = 1,2,\ldots,L$ are separated whenever the Euclidean distance between the corresponding optimal dual vectors is sufficiently large. This refinement strategy is referred to as the absolute rule in [40]. To do this, first, a scenario-based subproblem (9) must be solved for every scenario $k \in \mathscr{P}^\ell$ to obtain an optimal dual vector $\pi^k \in \Pi^*(x_1, \xi^k)$. Then, if there exists a scenario $k' \in \mathscr{P}^\ell$ with an optimal dual vector $\pi^{k'}$ such that $\|\pi^k - \pi^{k'}\| > \varepsilon^\pi$ where $\varepsilon^\pi > 0$ is a user-specified tolerance parameter, then scenario k' should be separated from k, i.e., they will not be in the same cluster in the refined partition. Alternatively, to address the potential degeneracy in (9) (i.e., the situation where there exist multiple dual optimal solutions associated with a scenario k), we ~~also~~ compare the objective values given by $Q(x_1, \xi^k)$ and $Q(x_1, \xi^{k'})$. Given any user-specified parameter $\varepsilon^Q > 0$, if $|Q(x_1, \xi^k) - Q(x_1, \xi^{k'})| \leq \varepsilon^Q$, then scenario k and k' will stay in the same cluster after the refinement regardless of the Euclidean distance between their optimal dual vectors.

After the *partition refinement* step is finished, ideally, one would hope to end up with a refined cluster $\mathscr{N}'$ which its size L' is as small as possible. However, this might be computationally inefficient since this problem can in fact be formulated as the so-called *maximum clique* problem [10] which is NP-hard. Instead, we consider a simple fast heuristic approach. Specifically, for any arbitrary cluster $\mathscr{P} \in \mathscr{N}$, let k be the first element in $\mathscr{P}$, we first create an initial cluster $\mathscr{P}'^1 = \{k\}$ and let scenario $k \in \mathscr{P}$ represent this cluster with $\pi^k \in \Pi^*(x_1, \xi^k)$ being its corresponding optimal dual vector. Then we compare the associated optimal dual vector $\pi^{k'}$ of every scenario $k' \in \mathscr{P}$ with π^k, and following the absolute refinement rule, if $\|\pi^k - \pi^{k'}\| \leq \varepsilon^\pi$, then scenario k' is added to cluster $\mathscr{P}'^1$; otherwise, a new cluster $\mathscr{P}'^2 = \{k'\}$ is created with scenario k' being the representative scenario and $\pi^{k'}$ being the representative optimal dual vector. The process is then repeated with other scenarios in $\mathscr{P}$, with the scenarios now being compared to both $\mathscr{P}'^1$ and $\mathscr{P}'^2$. Repeat this process until all of the scenarios in $\mathscr{P}$ are examined, and all the newly created clusters will be added to the refined partition $\mathscr{N}'$ to replace $\mathscr{P}$. See Algorithm 1 for a precise description of the procedure, from which one clearly sees that the computational complexity of our refinement strategy for refining a cluster $\mathscr{P} \in \mathscr{N}$ is $\mathscr{O}(|\mathscr{P}|^2)$. The complexity of refining the entire partition $\mathscr{N}$ is thus $\mathscr{O}(\sum_{\mathscr{P}\in\mathscr{N}} |\mathscr{P}|^2)$.

2.2 Preliminaries on Multi-stage Stochastic Linear Programs and the SDDP Algorithm

Algorithm 1 The refinement step.

Input:
- a partition $\mathscr{N}$ and a cluster $\mathscr{P} \in \mathscr{N}$ to be refined
- a set of optimal dual vectors $\{\pi^k\}_{k\in\mathscr{P}}$
- a set of objective values $\{Q(x_1,\xi^k)\}_{k\in\mathscr{P}}$

Initialize: Initialize
- cluster $\mathscr{P}'^1 = \{k_1\}$ with k_1 being the first element of $\mathscr{P}$, and π^{k_1} being the dual vector to represent $\mathscr{P}'^1$
- a collection of refined clusters $\mathscr{C} = \{\mathscr{P}'^1\}$

Refinement procedure:
for $k \in \mathscr{P} \setminus \{k_1\}$ **do**
  - set $flag = 0$
  for $\mathscr{P}' \in \mathscr{C}$ **do**
    - let k' and π' be the representing scenario and the dual vector associated with $\mathscr{P}'$, respectively
    if $\|\pi^k - \pi'\| \le \varepsilon^\pi$ or $|Q(x_1,\xi^k) - Q(x_1,\xi^{k'})| \le \varepsilon^Q$ **then**
      - append k to $\mathscr{P}'$, set $flag = 1$, and break out of the inner loop
    end if
  end for
  if $flag = 0$
    - initialize a new cluster $\mathscr{P}'' = \{k\}$ with π^k being the dual vector to represent $\mathscr{P}''$
    - append $\mathscr{P}''$ to $\mathscr{C}$
end for

Output: return the refined partition $\mathscr{N}' \leftarrow \mathscr{N} \setminus \mathscr{P} \cup \mathscr{C}$

The multistage stochastic programming (MSP) problem presented in (2) has a nested form. As such, to facilitate a computationally tractable formulation, we make the following *stage-wise independence* assumption:

Assumption 2 *Stage-wise independence. We assume that the stochastic process $\{\xi_t\}$ is stage-wise independent, i.e., ξ_t is independent of the history of the stochastic process up to time $t-1$, for $t = 1,2,\dots T$, which is given by $\xi_{[t-1]}$.*

This *stage-wise independence* allows for problem (2) to be formulated using the following dynamic programming equations:

$$Q_t(x_{t-1},\xi_t) := \begin{cases} \min\limits_{x_t\in\mathbb{R}^{n_t}_+} & c_t^\top x_t + \mathfrak{Q}_{t+1}(x_t) \\ \text{s.t.} & A_t x_t = b_t - B_t x_{t-1}\,, \end{cases} \tag{10}$$

where $\mathfrak{Q}_{t+1}(x_t) := \mathbb{E}[Q_{t+1}(x_t,\xi_{t+1})]$ for $t = T-1,\dots 1$, and $\mathfrak{Q}_{T+1}(x_T) := 0$. The first-stage problem becomes

$$\begin{cases} \min\limits_{x_1\in\mathbb{R}^{n_1}_+} & c_1^\top x_1 + \mathfrak{Q}_2(x_1) \\ \text{s.t.} & A_1 x_1 = b_1\,. \end{cases} \tag{11}$$

and $\mathfrak{Q}_{t+1}(\cdot)$ in (10) is referred to as the *cost-to-go* function. Note that, if the number of scenarios per stage is finite, the cost-to-go functions are convex piecewise linear functions [37, Chap. 3]. Before we discuss the solution approaches developed to solve problem (11), it is important to address non-degeneracy and feasibility assumptions. Let $\chi_t := \{x_t \in \mathbb{R}^{n_t}_+ \mid B_t x_{t-1} + A_t x_t = b_t\}$, $\forall t = 2,\dots,T$.

Non-degeneracy: The cost-to-go functions $\mathfrak{Q}_{t+1}(\cdot)$'s are defined as the optimal future cost at stage t, for $t = 1,2\ldots,T-1$. It may happen that for some feasible $x_t \in \chi_t$ and a scenario $\xi_t \in \Xi_t$, the *stage*-$(t+1)$ subproblem is unbounded from below, i.e., $Q_{t+1}(x_t,\xi_{t+1}) = -\infty$. This is a somewhat pathological and unrealistic situation meaning that for such feasible x_t, there exist a positive probability, by which one can reduce the future cost indefinitely. One should make sure at the modeling phase that this does not happen.

Assumption 3 *Non-degeneracy. We assume that the cost-to-go function $\mathfrak{Q}_{t+1}(\cdot)$ at every stage is finite valued. i.e., $Q_{t+1}(x_t,\xi_{t+1}) > -\infty$, $\forall \xi_{t+1} \in \Xi_{t+1}$, $\forall t = 1,2\ldots,T-1$.*

Feasibility: If for some $x_t \in \chi_t$ and scenario $\xi_t \in \Xi_t$ problem (10) is infeasible, the standard practice is to set $Q_{t+1}(x_t,\xi_t) = +\infty$ such that x_t cannot be an optimal solution of the *stage*-$(t+1)$ subproblem. It is said that the problem has relatively complete recourse if such infeasibility does not happen.

Assumption 4 *Relatively complete recourse. We assume that $\forall\, x_t \in \chi_t$ and $\xi_t \in \Xi_t$, $\exists\, x_{t+1} \in \mathbb{R}^{n_{t+1}}_+$ such that, problem (10) is feasible.*

2.2.1 Stochastic Dual Dynamic Programming (SDDP)

Perhaps one of the most popular algorithms for solving MSLPs, the SDDP algorithm [31] draws influence from the backward recursion techniques developed in dynamic programming [7]. More importantly, under Assumption 2, it provides an implementable policy not only for the approximation problem, but also for the true problem through decision rules induced by the approximate cost-to-go functions.

Leveraging the dynamic equations (10), the SDDP algorithm alternates between two main steps: a forward simulation (*forward pass*) which evaluates the current policy obtained by sequentially solving problem (10) in each stage t with the current approximation for the cost-to-go functions $\check{\mathfrak{Q}}_{t+1}(\cdot)$ to provide a sequence of decisions $\check{x}_t = \check{x}_t(\xi_t)$, for $t = 1,2,\ldots,T$, and a backward recursion (*backward pass*) to improve the approximate cost-to-go functions $\check{\mathfrak{Q}}_{t+1}(\cdot)$, for $t = 1,\ldots T$. After the forward step, a statistical upper bound for the optimal value of (11) can be computed, and after the backward step an improved lower bound for the optimal value of (11) is obtained. We summarize the two main steps below and refer the reader to [31] for a more detailed discussion on the topic.

Forward step. Consider taking a sample of $\mathscr{S}$ scenarios of the stochastic process, which is denoted by $\{\xi^s\}_{s\in\mathscr{S}}$, with $|\mathscr{S}| \ll |\Xi_1| \times |\Xi_2| \times \cdots \times |\Xi_T|$ and $\xi^s = (\xi_2^{s_2},\ldots,\xi_T^{s_T})$. Let $\check{\mathfrak{Q}}_{t+1}(\cdot)$ be the current approximation of the cost-to-go function $\mathfrak{Q}_{t+1}(\cdot)$ at stage t. In order to evaluate the quality of the decision policy induced by $\check{\mathfrak{Q}}_{t+1}(\cdot)$ for $t = 1,2,\ldots,T$, trial decisions $\check{x}_t = \check{x}_t(\xi_t)$, $t = 1,\ldots,T$, are computed recursively going forward with $\check{x}_1$ being an optimal solution of (11) with $\check{\mathfrak{Q}}_2(\cdot)$, and $\check{x}_t$ being an optimal solution of

$$\underline{Q}_t(\check{x}_{t-1},\xi_t^{s_t}) := \begin{cases} \min\limits_{x_t\in\mathbb{R}^{n_t}_+} & c_t^\top x_t + \check{\mathfrak{Q}}_{t+1}(x_t) \\ \text{s.t.} & A_t x_t = b_t - B_t\check{x}_{t-1} \end{cases} \tag{12}$$

where $\check{\mathfrak{Q}}_{T+1}(\cdot) := 0$ and $\underline{Q}_t(\cdot)$ is a lower approximation for $Q_t(\cdot)$. The value

$$z(\xi^s) = \sum_{t=1}^{T} c_t^\top \check{x}_t(\xi_t^{s_t}),\ \forall\, s \in \mathscr{S}, \tag{13}$$

as well as

$$\begin{aligned} \bar{z} &= \tfrac{1}{|\mathscr{S}|} \sum_{s\in\mathscr{S}} z(\xi^s),\ \forall\, s \in \mathscr{S}, \\ \sigma^2 &= \tfrac{1}{|\mathscr{S}|} \sum_{s\in\mathscr{S}} \left[z(\xi^s) - \bar{z}\right]^2,\ \forall\, s \in \mathscr{S}, \end{aligned} \tag{14}$$

are computed, with $\bar{z}$ and σ^2 being the sample average and sample variance of $z(\cdot)$, respectively. Note that $\check{x}_t(\xi_t^{s_t})$ is a feasible and implementable policy for problem (2).
The sample average $\bar{z}$ provides an unbiased estimator for an upper bound of the optimal value of (2), which is given by

$$\hat{z} = \mathbb{E}\left[\sum_{t=1}^{T} c_t^\top \check{x}_t(\xi_t).\right] \tag{15}$$

Additionally, $\bar{z} + 1.96\sigma/\sqrt{|\mathscr{S}|}$ provides a statistical upper bound for the optimal value of (2) with 95% confidence level. These bounds can be used as a possible stopping criterion whenever $\bar{z} + 1.96\sigma/\sqrt{|\mathscr{S}|} - \underline{z} \le \varepsilon$, for a given tolerance $\varepsilon > 0$. We refer to [36, Sec.3] for a discussion on this subject.
Backward step. Given the trial decisions $\check{x}_t = \check{x}_t(\xi_t)$ obtained in the *forward step* and an approximation of the cost-to-go function $\check{\mathfrak{Q}}_{t+1}(\cdot)$, for $t = 1, 2, \ldots, T$; exploiting the fact that $\mathfrak{Q}_{T+1}(\cdot) := 0$ at stage $t = T$, the following problem is solved for each $\xi_T \in \{(B_T^k, b_T^k)\}_{k=1}^{|\Xi_T|}$

$$\underline{Q}_T(\check{x}_{T-1}, \xi_T) = \begin{cases} \min\limits_{x_T \in \mathbb{R}_+^{n_T}} & c_T^\top x_T \\ \text{s.t.} & A_T x_T = b_T - B_T \check{x}_{T-1}. \end{cases} \tag{16}$$

Let $\check{\pi}_T = \check{\pi}_T(\xi_T)$ be an optimal dual solution of problem (16). Then define $\alpha_T := \mathbb{E}[b_T^\top \check{\pi}_T]$ and $\beta_T := -\mathbb{E}[B_T^\top \check{\pi}_T] \in \partial \mathfrak{Q}_T(\check{x}_{T-1})$ such that

$$q_T(x_{T-1}) := \beta_T^\top x_{T-1} + \alpha_T \quad = \quad \mathfrak{Q}_T(\check{x}_{T-1}) + \langle \beta_T, x_{T-1} - \check{x}_{T-1} \rangle,$$

is a lower cutting-plane approximation for $\mathfrak{Q}_T(x_{T-1})$ satisfying

$$\mathfrak{Q}_T(x_{T-1}) \ge q_T(x_{T-1}) \quad \forall\, x_{T-1},$$

with $q_T(\cdot)$ being a supporting hyperplane for $\mathfrak{Q}_T(\cdot)$, i.e., $\mathfrak{Q}_T(\check{x}_{T-1}) = q_T(\check{x}_{T-1})$. This linear cutting-plane approximation is added to the collection of supporting hyperplanes of $\mathfrak{Q}_T(\cdot)$ by letting the *new* approximation be $\max\{\check{\mathfrak{Q}}_T(x_{T-1}), q_T(x_{T-1})\}$. That is, the cutting-plane approximation for $\check{\mathfrak{Q}}_T(\cdot)$ is constructed from the maximum of a collection J_T of cutting-plane approximation:

$$\check{\mathfrak{Q}}_T(x_{T-1}) = \max_{j \in J_T} \left\{ \beta_{T,j}^\top x_{T-1} + \alpha_{T,j} \right\}.$$

For $t = T-1, T-2, \ldots, 2$, we update $\check{\mathfrak{Q}}_t(\cdot)$ in the same spirit as $\check{\mathfrak{Q}}_T(\cdot)$, and the following problems are solved $\forall\ \xi_t \in \{(B_t^k, b_t^k)\}_{k=1}^{|\Xi_t|}$

$$\underline{Q}_t(\check{x}_{t-1}, \xi_t) = \begin{cases} \min\limits_{x_t \in \mathbb{R}_+^{n_t}} & c_t^\top x_t + \check{\mathfrak{Q}}_{t+1}(x_t) \\ \text{s.t.} & A_t x_t = b_t - B_t \check{x}_{t-1} \end{cases} \equiv \begin{cases} \min\limits_{(x_t, r_{t+1}) \in \mathbb{R}_+^{n_t} \times \mathbb{R}} & c_t^\top x_t + r_{t+1} \\ \text{s.t.} & A_t x_t = b_t - B_t \check{x}_{t-1} \\ & \beta_{t+1,j}^\top x_t + \alpha_{t+1,j} \le r_{t+1}, \quad j \in J_{t+1}. \end{cases} \tag{17}$$

Let $\check{\pi}_t = \check{\pi}_t(\xi_t)$ be the optimal dual vector associated with constraint $A_t x_t = b_t - B_t \check{x}_{t-1}$. Then the cutting-plane approximation

$$q_t(x_{t-1}) := \mathbb{E}[\underline{Q}_t(\check{x}_{t-1}, \xi_t)] + \langle \beta_t, x_{t-1} - \check{x}_{t-1} \rangle \tag{18}$$

of $\mathfrak{Q}_t(\cdot)$ is constructed with

$$\beta_t := -\mathbb{E}[B_t^\top \check{\pi}_t] \in \mathbb{E}[\partial \underline{Q}_t(\check{x}_{t-1}, \xi_t)]$$

such that $\mathfrak{Q}_t(x_{t-1}) \geq q_t(x_{t-1}) \ \ \forall\, x_{t-1}$. Note that the above inequality holds when $t = T-1$ since $\check{\mathfrak{Q}}_T(\cdot)$ approximates $\mathfrak{Q}_T(\cdot)$ from below; then $\underline{Q}_{T-1}(\check{x}_{T-2}, \xi_{T-1}) \leq Q_{T-1}(\check{x}_{T-2}, \xi_{T-1})$ implying that $q_{T-1}(\cdot)$ underestimates $\mathfrak{Q}_{T-1}(\cdot)$. The result for stage t follows by using a backward induction argument from $T, T-1, \ldots, t$. Once the cutting-plane $q_{T-1}(x_{t-1})$ is computed, then the current approximation $\check{\mathfrak{Q}}_t(x_{t-1})$ is updated at stage t by: $\check{\mathfrak{Q}}_t(x_{t-1}) = \max\{\check{\mathfrak{Q}}_t(x_{t-1}), q_t(x_{t-1})\}$. It is worth noting that, unlike in stage T where the value of the cost-to-go function is precisely known ($\mathfrak{Q}_{T+1} = 0$), the linear approximation given (in early iterations) by $q_t(\cdot)$, $\forall t = T-1, \ldots, 1$, might be a strict under-estimator of $\mathfrak{Q}_t(\cdot)$ for all feasible x_{t-1}. In other words, $q_t(\cdot)$ might only be a cutting plane but not necessarily a supporting hyperplane. Finally, at $t = 1$, the following LP is solved

$$\underline{z} = \left\{ \begin{array}{ll} \min\limits_{x_1 \in \mathbb{R}^{n_1}_+} & c_1^\top x_1 + \check{\mathfrak{Q}}_2(x_1) \\ \text{s.t.} & A_1 x_1 = b_1 \end{array} \right. \equiv \left\{ \begin{array}{ll} \min\limits_{(x_1, r_2) \in \mathbb{R}^{n_1}_+ \times \mathbb{R}} & c_1^\top x_1 + r_2 \\ \text{s.t.} & A_1 x_1 = b_1 \\ & \beta_{2,j}^\top x_1 + \alpha_{2,j} \leq r_2, \quad j \in J_2\,. \end{array} \right. \tag{19}$$

The value $\underline{z}$ provides a lower bound for the optimal value of (2). The updated $\check{\mathfrak{Q}}_t(\cdot)$ for $t = 2, \ldots, T$, can be used to induce an implementable policy. The convergence analysis of the method can be found in [12, 24, 32, 36].

3 Adaptive Partition-based SDDP for Multistage Stochastic Linear Programs

In this section we discuss how the adaptive partition-based strategies discussed in Section 2.1.2 can be extended to the multistage setting. We do this by first, showing the validity of the cutting planes generated using the adaptive partition-based strategies to approximate the cost-to-go functions $\mathfrak{Q}_{t+1}(\cdot)$ in (10), present the ingredients of the different variants of adaptive partition-based SDDP algorithm to be discussed in Section 4, and finally show the finite convergence of the proposed method.

3.1 Adaptive Partition-based Cutting Planes for Multistage Stochastic Linear Programs

Let $\check{x}_t, t = 1, 2, \ldots, T-1$ be the trial points (e.g., collected along a sample path during the forward pass of the SDDP algorithm), and let $\{(B_t^k, b_t^k)\}_{k=1}^{|\Xi_t|}$ be the set of realizations corresponding to the random vectors in each stage $t = 2, \ldots, T$. In the same way as defined in Section 2.1.2, at every stage t we partition $\{(B_t^k, b_t^k)\}_{k=1}^{|\Xi_t|}$ into L_t scenario clusters, such that the stage-t *partition* is given by $\mathscr{N}_t = \{\mathscr{P}_t^\ell\}_{\ell=1}^{L_t}$ and $\bar{p}_t^\ell$ is the weight associated with each cluster $\mathscr{P}_t^\ell$ for $\ell = 1, \ldots, L_t$. At stage $t = T$, a *scenario-based subproblem* can be defined for each realization $\xi_t^k = (B_T^k, b_T^k) \in \Xi_T$ as follows:

$$\underline{Q}_T(\check{x}_{T-1}, \xi_T^k) := \min_{x_T} \left\{ c_T^\top x_T \mid A_T x_T = b_T^k - B_T^k \check{x}_{T-1} \right\}, \tag{20}$$

where a *partition-based subproblem* can be defined similarly to (8) for each cluster $\mathscr{P}_T^\ell \in \mathscr{N}_T$ as follows:

$$\underline{Q}_T(\check{x}_{T-1}, \bar{\xi}_T^\ell) := \min_{x_T} \left\{ c_T^\top x_T \mid A_T x_T = \bar{b}_T^\ell - \bar{B}_T^\ell \check{x}_{T-1} \right\}, \tag{21}$$

with $\bar{b}_T^\ell := \sum_{k \in \mathscr{P}_T^\ell} b_T^k \times \frac{p_T^k}{\bar{p}_T^\ell}$ and $\bar{B}_T^\ell := \sum_{k \in \mathscr{P}_T^\ell} B_T^k \times \frac{p_T^k}{\bar{p}_T^\ell}$ and

$$\check{\underline{\Omega}}_T(\check{x}_{T-1}) := \sum_{\ell=1}^{L_T} \bar{p}_T^\ell \times \underline{Q}_T(\check{x}_{T-1}, \bar{\xi}_T^\ell). \tag{22}$$

Due to the fact that $\check{\underline{\Omega}}_{T+1}(\cdot) := 0$ for any feasible trial point $\check{x}_T$, it can be easily seen that (21), which is obtained by aggregating the constraints and variables of (20) for all scenarios $k \in \mathscr{P}_T^\ell$, has a similar structure to that of the two-stage setting given by (5) and (8), respectively. Therefore,

$$\underline{Q}_T(\check{x}_{T-1}, \bar{\xi}_T^\ell) \leq \sum_{k \in \mathscr{P}_T^\ell} \frac{p_T^k}{\bar{p}_T^\ell} \times \underline{Q}_T(\check{x}_{T-1}, \xi_T^k) = \sum_{k \in \mathscr{P}_T^\ell} \frac{p_T^k}{\bar{p}_T^\ell} \times Q_T(\check{x}_{T-1}, \xi_T^k).$$

Treating each cluster $\mathscr{P}_T^\ell$ as a scenario, a coarse cut $\bar{\beta}_{T,j}^\top x_{T-1} + \bar{\alpha}_{T,j} \leq r_T$ can also be generated in the same way that a *standard Benders cut* is generated (see the derivations after equation (16)), using the corresponding optimal dual solutions of (21) for each cluster $\mathscr{P}_T^\ell \in \mathscr{N}_T$. When the coarse cuts do not improve the representation of $\Omega_T(\cdot)$ at the trial point $\check{x}_{T-1}$ with respect to the current relaxation for the cost-to-go function, i.e., $\check{\underline{\Omega}}_T(\check{x}_{T-1}) \geq \bar{\beta}_{T,j}^\top \check{x}_{T-1} + \bar{\alpha}_{T,j}$, the partition $\mathscr{N}_T$ can be refined by solving the subproblem (20) for each $k = 1, 2, \ldots, |\Xi_T|$, where the corresponding optimal dual vectors will guide the partition refinements (see Algorithm 1).

Now consider any other stage $t \in \{2, 3, \ldots, T-1\}$, the scenario-based problem now involves the cutting plane approximation for $\Omega_{t+1}(x_t)$ which is given by

$$\check{\Omega}_{t+1}(x_t) = \max_{j \in J_{t+1}} \{\alpha_{t+1,j} + \beta_{t+1,j} x_t\} \text{ (see (17) – (18))}.$$

Then the scenario-based subproblem for each realization $\xi_t^k = (B_t^k, b_t^k) \in \Xi_t$ is given by:

$$\underline{Q}_t(\check{x}_{t-1}, \xi_t^k) := \begin{cases} \min & c_t^\top x_t + r_{t+1} \\ \text{s.t.} & A_t x_t = b_t^k - B_t^k \check{x}_{t-1} \qquad (\check{\pi}_t^k) \\ & r_{t+1} - \beta_{t+1,j}^\top x_t \geq \alpha_{t+1,j}, j \in J_{t+1} \quad , \end{cases} \tag{23}$$

whereas the partition-based subproblem for each cluster $\mathscr{P}_t^\ell$ of the partition $\mathscr{N}_t$, for $\ell = 1, 2, \ldots, L_t$, can be defined as follows:

$$\underline{Q}_t(\check{x}_{t-1}, \bar{\xi}_t^\ell) = \begin{cases} \min & c_t^\top x_t + r_{t+1} \\ \text{s.t.} & A_t x_t = \bar{b}_{t,j}^\ell - \bar{B}_t^\ell \check{x}_{t-1} \qquad (\check{\pi}_t^\ell) \\ & r_{t+1} - \bar{\beta}_{t+1,j}^\top x_t \geq \bar{\alpha}_{t+1,j}, j \in J_{t+1} \quad , \end{cases} \tag{24}$$

which again can be obtained by aggregating variables and constraints of scenario-based subproblems (23) for all $k \in \mathscr{P}_t^\ell$ such that

$$\check{\underline{\Omega}}_t(\check{x}_{t-1}) := \sum_{\ell=1}^{L_t} \bar{p}_t^\ell \times \underline{Q}_t(\check{x}_{t-1}, \bar{\xi}_t^\ell), \quad \forall t = 2, \ldots, T-1. \tag{25}$$

Similar to the case when $t = T$, the partition refinement is guided by the optimal dual multiplies $\{\check{\pi}_t^k\}_{k\in\mathscr{P}_t^\ell}$, i.e., scenarios are put together in the same cluster if the corresponding $\check{\pi}_t^k$ are identical (or close enough in Euclidean distance). It can be seen from [40, Theorem 2.5] that this partition refinement rule guarantees that, after refining $\mathscr{N}_t$ into $\mathscr{N}_t' = \{\mathscr{P}_t^\ell\}_{\ell=1}^{L_t'}$,

$$\check{\mathfrak{Q}}_t'(\check{x}_{t-1}) = \sum_{\ell=1}^{L_t'} \bar{p}_t^\ell \times \underline{Q}_t(\check{x}_{t-1}, \bar{\xi}_t^\ell) = \sum_{k=1}^{|\Xi_t|} p_t^k \times \underline{Q}_t(\check{x}_{t-1}, \xi_t^k) = \check{\mathfrak{Q}}_t(\check{x}_{t-1}).$$

Applying the idea of scenario partitions is likely to speed up the process of approximating the cost-to-go functions, particularly in the early stages, where exact information may not be important for generating initial cutting-plane relaxations. This has been validated in our experiment results shown in Section 5.

3.2 Ingredients of Adaptive Partition-based SDDP Algorithms

As thoroughly discussed in [2, 40], the key reason for the efficiency of the adaptive partition-based strategy in the context of the 2SLP is that, it makes the cut generation effort during the solution procedure adaptive to the solution progress. To demonstrate this further, since most decomposition algorithms used for solving SPs usually rely on sequences of candidate solutions (trial points in the sequel) $\check{x}_1$'s for generating cutting plane approximations to $\mathfrak{Q}_2(\cdot)$, it is intuitively clear that generating such approximations to $\mathfrak{Q}_2(\cdot)$ with high precision for early iterates, likely far from an optimal solution, is surely a wasteful use of computing power — accuracy will need to be integrated adaptively as those candidate solutions $\check{x}_1$'s get close to being optimal. Ultimately, this procedure not only builds an approximation for $\mathfrak{Q}_2(\cdot)$ in a computationally efficient manner but also yields a *sufficient* partition $\mathscr{N}$ whose size is (often) smaller than the original number of realizations $|\Xi|$.

Therefore, it is very tempting to conclude that this reduction in the size of the problem will naturally mitigate the computational burden of solving MSLP due to the large number of stages in the planning horizon T, and the large number of realizations per stage $|\Xi_t|$. However, incorporating the idea of scenario partition for the 2SLP to the MSLP is not necessarily straightforward. To see this, let $\bar{\mathscr{T}}(\mathscr{N})$ be a coarse tree induced by the sequence of partitions $\mathscr{N} = (\mathscr{N}_2, \ldots, \mathscr{N}_T)$. Suppose we have a procedure by which we sample different sample paths $\xi^s = (\xi_2^{s_2}, \ldots, \xi_T^{s_T})$ from the scenario tree $\mathscr{T}$, and in the backward pass we generate a valid cutting plane approximation for the stage t problem defined over ξ^s, $\forall\, t = 2, \ldots T$ and $s \in \mathscr{S}$ and refine partition $\mathscr{N}_t$ if necessary. For a fixed sequence of trial points $\check{x}(\xi^s)$ along sample path $\xi^s = (\xi_2^{s_2}, \ldots, \xi_T^{s_T})$, one can obtain a sequence of partitions $\mathscr{N}^s = (\mathscr{N}_2^{s_2}, \ldots, \mathscr{N}_T^{s_T})$ which is *locally sufficient* with respect to ξ^s and trial points $\check{x}(\xi^s)$.

It is not difficult to see that a locally sufficient partition to a given sample path s and its associated trial points $\check{x}(\xi^s)$ is not necessarily sufficient for other sample paths (this is why we call them "locally sufficient"). When we consider a new sample path $\xi^{s'}$, it is very likely that we observe a new sequence of trial points in which case $\mathscr{N}^s$ is no longer sufficient for $\xi^{s'}$ and by extension, we would like to consider a new partition that is sufficient for both ξ^s and $\xi^{s'}$. To do so, we perform partition refinements on top of $\mathscr{N}^s$ to obtain the new partition. We denote $\overrightarrow{\mathscr{N}}^p$ as the partition obtained via sequential partition refinements over the sequence of p sample paths, and we call this partition $\overrightarrow{\mathscr{N}}^p$ a *locally sufficient* partition if it is locally sufficient for every sample path $\xi \in \{\xi^s\}_{s=1}^p$ and their associated trial points. We use the $\overrightarrow{\mathscr{N}}$ notation to denote the sequential nature of refining on top of the existing partitions as we sample new sample paths.

Hence, if one wishes to construct a *globally sufficient* partition, which means that it is locally sufficient for all possible sample paths, the corresponding coarse tree is likely to be identical to the full scenario tree, i.e., $\bar{\mathscr{T}}(\overrightarrow{\mathscr{N}}^p) \to \mathscr{T}$ as $p \to \mathscr{S}$. Additionally, in order to proclaim the sufficiency of $\mathscr{N}^s$ to ξ^s we need to decouple problem (2) into a sequence of $(T-1)$ consecutive 2SLPs. This decoupling scheme, which we discuss in more details in Subsection 3.2.2, was referred to in [1] and [42] for deterministic multistage linear programs as *cautious* and *shuffle*, when the scenario tree is traversed forward and backward, respectively. Consequently, and as we shall see, this raises not only the question of how we can incorporate the adaptive partition-based strategy to the SDDP algorithm, but also the question of what the best strategy is to traverse scenario tree.

Moreover, as we intend to employ the adaptive partition-based framework presented in [2] which relies on different types of cutting-plane approximations to the cost-to-go functions $\mathfrak{Q}_{t+1}(\cdot)$ in their level of inexactness; another aspect to consider in our analysis is, for a given tree traversal strategy, how coarse should the cutting-plane approximation be, such that the computational savings acquired by adaptive partition-based strategies via efficient cut generation effort, offsets the inaccuracy inherited in these inexact cuts during solution process.

Finally, as a byproduct of this work we attempt to construct an algorithm which exploits the structural nature of the underlying problem instance. This can be achieved by integrating this accuracy (inaccuracy) in the quality of cutting planes generated by the adaptive partition-based framework to the SDDP algorithm, relative to how much additional information we can gain from being accurate (as compared to being inaccurate) in different stages of the planning horizon. Next, we characterize in details different ingredients of our proposed algorithms.

3.2.1 Types of cutting planes

In accordance with the definitions introduced in [2], our analysis relies on three types of cutting-plane approximations to the cost-to-go function $\mathfrak{Q}_{t+1}(\cdot)$.

1. *Fine cuts*. Given an iterate $\check{x}_t = \check{x}_t(\xi_t)$, a cutting-plane approximation for $\mathfrak{Q}_{t+1}(\cdot)$ is generated by solving the scenario-based subproblem (23) for each scenario $\xi_{t+1} \in \Xi_{t+1}$.
2. *Coarse cuts*. Given an iterate $\check{x}_t = \check{x}_t(\xi_t)$, a cutting-plane approximation for $\mathfrak{Q}_{t+1}(\cdot)$ is generated by solving the partition-based subproblem (24) for each cluster $\mathscr{P}^{\ell}_{t+1} \in \mathscr{N}_{t+1}$, $\ell = 1, 2, \ldots, L_{t+1}$.
3. *Semi-coarse cuts*. We define a *semi-coarse* cut as any hybrid between the *Coarse* and *Fine* cuts. To demonstrate this further, let $\mathscr{L}'_{t+1} \neq \emptyset$ and $\mathscr{L}'_{t+1} \subset \mathscr{L}_{t+1} := \{1, \ldots, L_{t+1}\}$. Then given an iterate $\check{x}_t = \check{x}_t(\xi_t)$, a semi-coarse cutting-plane approximation for $\mathfrak{Q}_{t+1}(\cdot)$ is generated by solving a scenario-based subproblem (23) for each scenario $\xi_{t+1} \in \mathscr{P}^{\ell}_{t+1}$ for $\ell \in \mathscr{L}'_{t+1}$ and a partition-based subproblem (24) for each cluster $\mathscr{P}_{t+1} \in \mathscr{N}_{t+1} \setminus \bigcup_{\ell \in \mathscr{L}'_{t+1}} \{\mathscr{P}^{\ell}_{t+1}\}$. This in turn puts forth the question of how the subset of clusters $\bigcup_{\ell \in \mathscr{L}'_{t+1}} \{\mathscr{P}^{\ell}_{t+1}\}$ is chosen. In our implementation, we start by solving a scenario-based subproblem for every scenario k in the cluster $\mathscr{P}^{\ell'}_{t+1}$ which has the largest cardinality, i.e., $|\mathscr{P}^{\ell'}_{t+1}| \geq |\mathscr{P}^{\ell}_{t+1}|$, $\forall \ell \neq \ell'$, and a partition-based subproblem for all remaining clusters. If this does not successfully yield a violated valid inequality, we move on to the cluster which has the second largest cardinality and so on.

   We refer the reader to [2] for a more thorough discussion on these concepts.

3.2.2 Tree traversal strategies

We rely on two tree traversal strategies in different variants of the proposed algorithm. Those stratigies were formally introduced in [1] and [42] for deterministic multistage linear programs, and further developed in [27] for MSLPs.

Quick pass (QP). Under this scheme, the policy (which is induced by the current approximation for the cost-to-go functions $\mathfrak{Q}_{t+1}(\cdot)$) evaluation process iteratively passes candidate solution $\check{x}_t = \check{x}_t(\xi_t)$ down the scenario tree stage by stage ($t = 1 \to t = 2 \to \ldots t = T$) and cuts (if any exists) are passed directly back up the tree ($t = T \to t = T-1 \to \ldots t = 1$) with no intermediate change of direction between any two consecutive stages t and $t-1$ (see Figure 1(a)). We care to mention that *quick pass* is in fact, the most commonly used strategy in the SDDP algorithm. For example, in an MSP problem with $T = 5$ an iteration of QP would be

- Forward pass: $1 \to 2 \to 3 \to 4 \to 5$.
- Backward pass: $5 \to 4 \to 3 \to 2 \to 1$.

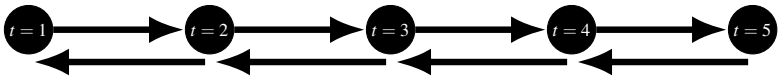
Cautious pass (CP). Under this scheme, the iterative policy evaluation process never goes back up the tree unless all cuts that would be passed back from stage $t+1$ to stage t, are redundant, i.e., the evaluation process will maintain intermediate changes of directions between consecutive stages. Note that, an intermediate change of direction going forward entails updating the candidate solution $\check{x}_t = \check{x}_t(\xi_t)$ (if possible) based on the updated cutting-plane approximation $\check{\mathfrak{Q}}'_{t+1}(\cdot)$. We refer to such intermediate changes of direction as, *inner forward step* and *inner backward step* when the change of direction is forward and backward, respectively (see Figure 1(b)). For example, in an MSP problem with $T = 5$ an iteration of CP would be

- Forward pass: $1 \to 2 \to 3 \to 4 \to 5$.
- Backward pass:
  - $5 \to 4 \to 5 \to 4 \to \cdots \to 5 \to 4$.
  - $4 \to 3 \to 4 \to 3 \to \cdots \to 4 \to 3$.
  - $3 \to 2 \to 3 \to 2 \to \cdots \to 3 \to 2$.
  - $2 \to 1 \to 2 \to 1 \to \cdots \to 2 \to 1$.

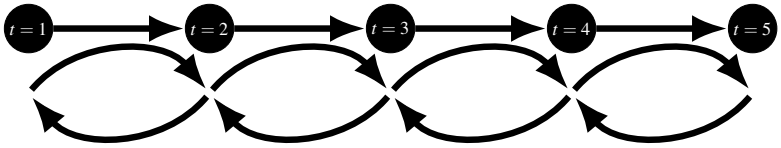
3.2.3 Refinements and sampling strategies

We propose several adaptive partition-based SDDP algorithms which depend on how the refinement step is integrated with the standard SDDP algorithm and the realization of ξ_t sampled in the forward pass. Mainly, we propose the following two schemes:

1. *Refinement within SDDP*. Given the original scenario tree and a sequence of partitions $\mathscr{N} = (\mathscr{N}_2, \ldots, \mathscr{N}_T)$, sample from the original scenario tree $\mathscr{T}$ during the forward pass and refine partitions $\mathscr{N}_t$, $\forall\, t = 2, \ldots T$ while generating cuts during the backward pass of the SDDP algorithm.
2. *Refinement outside SDDP*. Given a coarse tree $\bar{\mathscr{T}}(\mathscr{N})$ induced by a sequence of partitions $\mathscr{N} = (\mathscr{N}_2, \ldots, \mathscr{N}_T)$, perform the SDDP algorithm on $\bar{\mathscr{T}}(\mathscr{N})$, i.e., both sampling and cut generation are from the coarse tree only, without any attempt to refine any partition $\mathscr{N}_t$, $\forall\, t = 2, \ldots T$. Then refine $\mathscr{N}_t$ in a separate refinement step where the refinement is performed locally on sample path(s) $\xi^s = (\xi_2^{s_2}, \ldots, \xi_T^{s_T})$ that is sampled from the original scenario tree $\mathscr{T}$.



(a) *Quick pass.*



(b) *Cautious pass.*

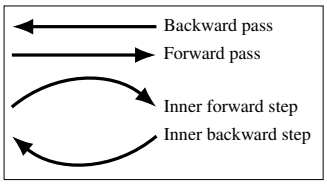


Fig. 1 Tree traversal strategies.

The main feature of the *Refinement within SDDP* strategy is its flexibility to add cuts from both the coarse tree $\bar{\mathscr{T}}(\mathscr{N})$ and original tree $\mathscr{T}$ at each step of the SDDP algorithm during the solution process. However, one concern with this approach is that, at any stage t whenever the coarse cut $\bar{q}_t(\check{x}_{t-1})$ (generated by aggregating the variables and constraints according to $\mathscr{N}_t$) does not yield any violation to the current iterate, the process will always attempt to refine the partition $\mathscr{N}_t$, $\forall\, t = T,\dots 2$. Hence, the partition size can grow very quickly as we step into different sample paths – which makes it difficult to effectively exploit the reduction in the size of the problem.

On the other hand, while the *Refinement outside SDDP* strategy is more restricted in the sense that it only generates cuts from the coarse tree during the execution of the SDDP algorithm, and it capitalizes on the merits of the adaptive partition-based framework, since the SDDP algorithm is now performed on a smaller scenario tree. Nonetheless, the main concern with the *Refinement outside SDDP* strategy is the difficulty of identifying a criterion by which we can claim the insufficiency of the current sequence of partitions $\mathscr{N}$ to be used in generating cuts, which would suggest its refinement.

We finish this section by showing the finite convergence of a generic version of the partition-based SDDP algorithm.

Proposition 1 *Convergence of the partition-based SDDP algorithms. Suppose that Assumptions 1–4 hold, sampling is done with replacement in the SDDP algorithm, and the employed partition refinement rule ensures that the number of clusters L_t of the refined partition strictly increases whenever $\check{\underline{\Omega}}_t(\check{x}_{t-1}) \neq \mathbb{E}[\underline{Q}_t(\check{x}_{t-1},\xi_t)]$ (see definitions in equations (16), (17), (22) and (25)) for any stage t unless $L_t = N_t$. Then w.p.1 after a finite number of forward and backward steps, the algorithm yields an optimal policy for (2).*

Proof See Appendix B.

4 Implementation Details

In this section we describe in more detail the proposed integrated adaptive partition-based SDDP algorithms. As previously noted, it is almost impossible to construct a globally sufficient non-trivial partition (a trivial partition is the one which contains clusters with all singletons, i.e. $\mathscr{N}_t = \{\{k\}_{k\in\mathscr{N}_t}\}$) that accommodates all possible sample paths in the scenario tree. However, depending on the refinement rule, one can easily have some control over the size of the coarse tree $\bar{\mathscr{T}}(\overrightarrow{\mathscr{N}}^p)$ obtained after p sample paths. This refinement rule can be driven by the manner in which the scenario tree is traversed based on different strategies introduced in Subsection 3.2.2. On one hand, employing a CP (recall that CP means "cautious-pass") is clearly a more rigorous strategy in refining the sequence of partitions $\overrightarrow{\mathscr{N}}^p$, since it attempts to create a sufficient sequence of partitions $\overrightarrow{\mathscr{N}}^{p*}$ by solving a sequence of $T-1$ 2SLP problems (defined by 1^{st}-stage $= t$ and 2^{nd}-stage $= t+1$, $\forall\, t = 1,2,\ldots T-1$) to optimality. On the other hand, a QP (recall that QP means "quick-pass") is a more of a lenient strategy to refine $\overrightarrow{\mathscr{N}}^p$ since it is only attempting to refine the partition $\overrightarrow{\mathscr{N}}^p$ without any intention to achieve local sufficiency.

While it is instructive to construct locally sufficient sequence partitions $\overrightarrow{\mathscr{N}}^p$ from an optimality point of view; as we shall see in Section 5, solving a sequence of $T-1$ 2SLP problems up to optimality could be expensive, and perhaps not worth doing, especially at early iterates when the candidate solution $\check{x}_t = \check{x}_t(\xi_t)$ is likely to be far from optimal. To that end, as mentioned in Subsection 3.2.3 we consider two refinement schemes for integrating the adaptive partition-based approach to the SDDP algorithm, namely Refinement within SDDP, and Refinement outside SDDP, which will be described in details in the following two subsections, respectively.

4.1 Refinement outside SDDP

The "high-level" idea of the refinement outside SDDP strategies is to implement standard SDDP algorithm on a coarse tree induced by a partition $\mathscr{N}$. Then, whenever implementing the standard SDDP on this coarse tree does not give much progress to the lower bound, i.e., the problem induced by the coarse tree is "sufficiently solved", we refine $\mathscr{N}$ to update the scenario tree, and resolve the new (coarse) scenario tree induced by the refined partition $\mathscr{N}'$. In other words, the refinement outside SDDP strategies can be thought of as looping between the following two main steps:

1. Implement the standard SDDP on a coarse tree induced by partition $\mathscr{N}$.
2. Refine $\mathscr{N}$ to update the (coarse) scenario tree and go to the first step.

This entire process (looping between steps 1 and 2) is continued until a certain termination criterion, such as until the size of the coarse tree is considered to be "large enough" (i.e., close in size to that of the original scenario tree), or the lower bound does not improve over a number of iterations. Specifically, we present two strategies of performing partition refinement outside SDDP:

- *Adaptive partition-enabled preprocessing for SDDP (APEP-SDDP)*: in this strategy, the adaptive partition-based approach is used only as a preprocessing step for the SDDP algorithm. Once this preprocessing step is finished the standard SDDP algorithm is applied on the original scenario tree. A coarse scenario tree is iteratively refined during the preprocessing step. Once the size of the coarse scenario tree is sufficiently large, the preprocessing step is terminated and the SDDP algorithm will be used for the original scenario tree for the rest of the procedure.
- *SDDP on iteratively refined coarse scenario trees (ITER-SDDP)*: apply the SDDP algorithm for each coarse scenario tree, which is iteratively refined throughout the solution process.

In both strategies, we determine whether the sequence of partitions $\mathscr{N}$ needs to be refined after applying j iterations of the standard SDDP algorithm on the coarse tree $\bar{\mathscr{T}}(\mathscr{N})$, by keeping track of the lower bound progress for the last n consecutive iterations (starting from iteration $n+1$). If $\underline{inner}^j - \underline{inner}^{j-n} \leq \varepsilon$, where $\varepsilon > 0$ is a given tolerance, it indicates that the lower bound does not improve by more than ε over the last n iterations. Both APEP-SDDP and ITER-SDDP will then refine the sequence of partitions $\mathscr{N}$ to $\mathscr{N}'$, and perform the SDDP algorithm on the refined tree $\bar{\mathscr{T}}(\mathscr{N}')$, except that if the coarse scenario tree after the refinement is large enough, the APEP-SDDP algorithm will revert back to the original scenario tree and perform the standard SDDP algorithm on it afterwards. Figure 2 illustrates these two algorithms for *Refinement outside SDDP*.

4.1.1 The APEP-SDDP algorithm

The primary concern with the APEP-SDDP algorithm is the difficulty to identify an adequate criterion for terminating the preprocessing step without jeopardizing the efficacy of the algorithm. Our implementations adopt a heuristic criterion for terminating the preprocessing step based on the size of the coarse tree $\bar{\mathscr{T}}(\mathscr{N})$. More specifically, we define the relative size of the coarse tree as $|\bar{\mathscr{T}}(\mathscr{N})| = \frac{\sum_{t=2}^{T} L_t/|\Xi_t|}{T-1}$ and let $\nu \in (0,1)$ be a user-specified parameter for terminating the preprocessing step. When $|\bar{\mathscr{T}}(\overrightarrow{\mathscr{N}}^p)| > \nu$, we terminate the preprocessing and revert to solving the original scenario tree $\mathscr{T}$ using the standard SDDP algorithm afterwards. Alternatively, one might choose to define the termination criterion as a fixed fraction of the computational budget (time-limit) and set ν accordingly. Finally, we use CP as the tree traversal strategy when refining $\mathscr{N}$. The motivation of this choice is that, using CP to traverse $\mathscr{T}$ is more likely to result in a larger $|\bar{\mathscr{T}}(\mathscr{N})|$, making the preprocessing step to be terminated more quickly. We have found that this choice along with setting $\nu \leq 0.5$ yielded improved performance in our numerical experiments.

4.1.2 The ITER-SDDP algorithm

As previously noted, the ITER-SDDP algorithm extends the APEP-SDDP algorithm in the sense that, here, there is no threshold by which we stop generating cutting planes from the coarse tree $\bar{\mathscr{T}}(\overrightarrow{\mathscr{N}}^p)$ and revert back to using the standard SDDP algorithm on the original scenario tree $\mathscr{T}$. In other words, the ITER-SDDP algorithm will continuously attempt to refine $\bar{\mathscr{T}}(\overrightarrow{\mathscr{N}}^p)$, for every iteration p whenever the lower bound $\underline{z}$ does not make significant progress. In this case, the cut generation from the coarse tree is no longer a "preprocessing" step, but integrated into the entire solution procedure.

The two aforementioned refinement outside SDDP algorithms are summarized in Algorithm 2.

Algorithm 2 The refinement outside SDDP algorithms.

STEP 0: Initialization.

1. Let $p = 0$, $\underline{z}^p := -\infty$, $\check{\Omega}_t^p(\cdot) := -\infty$, $\forall\, t = 2,\ldots,T$.
2. Define an initial sequence of partitions $\overrightarrow{\mathscr{N}}^p = (\overrightarrow{\mathscr{N}}_2^p,\ldots,\overrightarrow{\mathscr{N}}_T^p)$ and the corresponding coarse tree $\bar{\mathscr{T}}(\overrightarrow{\mathscr{N}}^p)$.
3. Define a stability-test parameter $n \in \mathbb{Z}^+$ and choose a tolerance $\varepsilon > 0$
4. Choose a preprocessing termination parameter ν and traversal strategy: $\nu \in (0,1)$ and CP for algorithm APEP-SDDP; $\nu = 1$ and QP for algorithm ITER-SDDP.

STEP 1:

1. Increment $p \leftarrow p+1$.
2. If $|\bar{\mathscr{T}}(\overrightarrow{\mathscr{N}}^p)| > \nu$, go to STEP 3. Otherwise, define an MSLP on $\bar{\mathscr{T}}(\overrightarrow{\mathscr{N}}^p)$ and the cutting plane approximations $\check{\Omega}_t^p(\cdot)\ \forall\, t = 2,\ldots,T$ as follows:
   (i) Initialize $j = 0$ and set $\underline{inner}^j = \underline{z}^p$ and $\check{\Omega}_t^j(\cdot) = \check{\Omega}_t^p(\cdot)$, $\forall\, t = 2,\ldots T$.
   (ii) **while** *true* **do**
      a. Increment $j \leftarrow j+1$.
      b. Apply the forward and backward step of the standard SDDP algorithm to improve the cutting plane approximations and update $\check{\Omega}_t^j(\cdot)$, $\forall\, t = 2,\ldots,T$ and $\underline{inner}^j$.
      c. **if** $j > n$
         if $\underline{inner}^j - \underline{inner}^{j-n} \le \varepsilon$
            set $\underline{z}^p = \underline{inner}^j$, and $\check{\Omega}_t^p(\cdot) = \check{\Omega}_t^j(\cdot)$, $\forall\, t = 2,\ldots T$; BREAK.
         end
      end
   end

STEP 2:

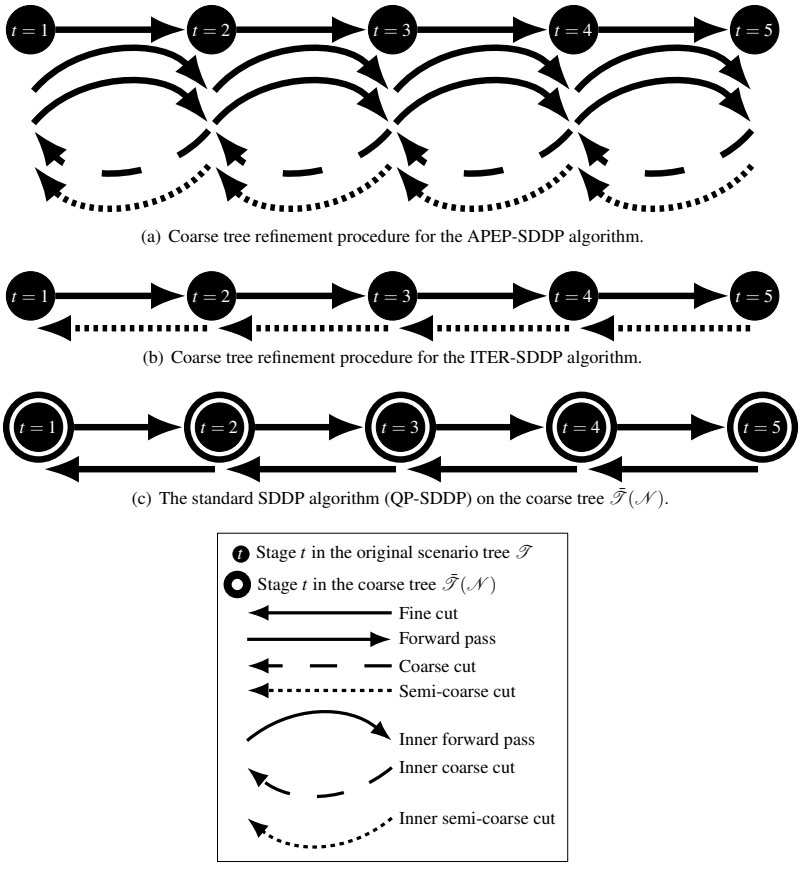
1. Choose a sample path $\xi^p = (\xi_2^{p_2},\ldots,\xi_T^{p_T})$.
2. Refine $\overrightarrow{\mathscr{N}}^p$ by the traversal strategy TS and the adaptive partition-based approach [2, 40] over the sample path ξ^p to construct $\bar{\mathscr{T}}(\overrightarrow{\mathscr{N}}^{p+1})$.
3. Update $\underline{z}^p$ and $\check{\Omega}_t^p(\cdot)$.
4. Go to STEP 1.

STEP 3:

1. Define an MSLP on $\mathscr{T}$ and initialize the approximate cost-to-go functions using $\check{\Omega}_t^p(\cdot)$.
2. Solve it using the standard SDDP algorithm and stop upon a termination criterion (such as a time limit).

4.2 Refinement within SDDP

Unlike the *Refinement outside SDDP* strategy introduced in Section 4.1, where the partition refinement is performed separately from the SDDP algorithm, whenever the lower bound progress is not significant during the SDDP algorithm, the *Refinement within SDDP* strategy is one in which the partition refinement is attempted at every *backward pass* of the SDDP algorithm if necessary, irrespective of the progress in the lower bound. Here, both the partition refinement and the SDDP algorithm are performed based on sample paths generated from the original scenario tree $\mathscr{T}$. In our implementations we consider two variants of the Refinement within SDDP strategy, which will be described in more details next.



(a) Coarse tree refinement procedure for the APEP-SDDP algorithm.

(b) Coarse tree refinement procedure for the ITER-SDDP algorithm.

(c) The standard SDDP algorithm (QP-SDDP) on the coarse tree $\tilde{\mathcal{T}}(\mathcal{N})$.

Fig. 2 Illustration of tree traversal strategies and various types of cutting planes used in the *Refinement outside SDDP* algorithms.

4.2.1 The adaptive partition-based SDDP algorithm with quick cut generation (AP-QP-SDDP)

This algorithm can be seen as the most natural extension of integrating the adaptive partition-based strategies to the backward pass of the standard SDDP algorithm. In the backward pass, at every stage $t = T,\ldots,2$, the AP-QP-SDDP algorithm starts by attempting to generate a *coarse cut*, and if it succeeds in doing so, the process immediately moves on to stage $t-1$; otherwise, it attempts to generate a *semi-coarse cut* by sequentially going through different clusters $\mathscr{P}_t^\ell \in \overrightarrow{\mathcal{N}}_t^p$, and as soon as this process succeeds in generating a violated cut, it updates $\check{\Omega}_t^p(\cdot)$, refines $\overrightarrow{\mathcal{N}}_t^p$ and moves on to stage $t-1$. In other words, any violated

cut, regardless of its quality, is a sufficient criterion for moving back to stage $t-1$. See Figure 3(a).

4.2.2 The adaptive partition-based SDDP algorithm with cautious cut generation (AP-CP-SDDP)

Instead of using the existence of any cut violation regardless of its quality as the criterion for moving back to the previous stage, as what is used in AP-QP-SDDP, the AP-CP-SDDP algorithm attempts to generate all possible cuts of various types by employing inner forward and backward steps just as CP-SDDP. See Figure 3(b).
The two aforementioned refinement within SDDP algorithms are summarized in Algorithm 3.

Remark 1 *As previously noted, in the coarse tree refinement step of the APEP algorithm, we use a CP as the tree traversal strategy. This explains why Figures 2(a) and 3(b) are the same. In other words, the coarse tree refinement step of the APEP algorithm is equivalent to a single iteration of the AP-CP-SDDP.*

Algorithm 3 The refinement within SDDP algorithms.

STEP 0: Initialization.
1. Let $p=0$, $\check{\Omega}_t^p(\cdot):=-\infty$, $\varepsilon\geq 0$.
2. Define an initial sequence of partitions $\overrightarrow{\mathscr{N}}^p=(\overrightarrow{\mathscr{N}}_2^p,\ldots,\overrightarrow{\mathscr{N}}_T^p)$, the corresponding coarse tree $\bar{\mathscr{T}}(\overrightarrow{\mathscr{N}}^p)$.
3. Define a tree-traversal strategy QP for AP-QP-SDDP and CP for AP-CP-SDDP
4. Initialize the corresponding necessary parameters of the SDDP algorithm.

STEP 1: Increment $p\leftarrow p+1$ and choose a sample path $\xi^p=(\xi_2^{p_2},\ldots,\xi_T^{p_T})$.
STEP 2: Implement the SDDP *forward step* over ξ^p to obtain $\check{x}_t=\check{x}_t(\xi_t^{p_t})$, $\forall t=1,\ldots T-1$.
STEP 3: Implement the adaptive partition-based SDDP *Backward-step* as follows:
For $t=T-1,\ldots,1$
- (a) Solve the partition-based subproblem (24) for each cluster $\mathscr{P}_{t+1}^{\ell}\in\overrightarrow{\mathscr{N}}_{t+1}^p$, $\ell=1,2,\ldots,L_{t+1}$ to improve the cutting plane approximation for $\check{\Omega}_{t+1}(\check{x}_t)$.
- (b) **If** $\check{\Omega}_{t+1}(\check{x}_t)-\check{\Omega}_{t+1}^p(\check{x}_t)\leq\varepsilon$, go to (c). **Else**, update $\check{\Omega}_{t+1}(x_t)$ and
  - If the tree-travesal strategy is QP, go to $t-1$.
  - If the tree-travesal strategy is CP, solve the scenario-based subproblem (23) to update $\check{x}_t=\check{x}_t(\xi_t^p)$ and go to (a)
- (c) Compute a semi-coarse cut as described in 3.2.1 and refine $\overrightarrow{\mathscr{N}}_{t+1}^p$
- (d) **If** $\check{\Omega}_{t+1}(\check{x}_t)-\check{\Omega}_{t+1}^p(\check{x}_t)\leq\varepsilon$ go to $t-1$. **Else**
  - If TS = QP, go to $t-1$.
  - If TS = CP, solve the scenario-based subproblem (23) to update $\check{x}_t=\check{x}_t(\xi_t^p)$ and go to (a)

STEP 4: If termination criterion is achieved, STOP. Else, go to STEP 1.

4.3 Adaptive partition-based SDDP with structured policies cut generation

In the previous two subsections, 4.1 and 4.2, we provide algorithms in which the same type of cut generation strategies are implemented uniformly across all stages $t=2,\ldots,T$. In this subsection, we investigate a framework which incorporates different types of cut generation strategies in different stages. To that end, we classify different stages in the planning horizon into two different categories:

1. Stages of "less-importance", where we use *coarse* and *semi-coarse* cuts.
2. Stages of "more-importance", where we use *fine* cuts only.

The motivation for associating *coarse* and *semi-coarse* cuts with stages considered to be less important and *fine* cuts with stages considered to be more important is as follows. In the MSLP setting, even if we simply attempt to generate *fine cuts* without any aggregation, the cutting hyperplane $q_t(x_{t-1}) := \beta_t^\top x_{t-1} + \alpha_t$ generated in stages $t = T-1,\dots,2$ still might not be a supporting hyperplane to $\mathfrak{Q}_t(\cdot)$, since the approximation error propagates as $t = T-1 \to t = 2$. Hence, using aggregated information with respect to a partition $\mathscr{N}_t^p$ might hinder the performance of the algorithm by adding another layer of inaccuracy. In certain situations, we may not be able to afford this additional layer of inaccuracy in stages where we need to be more accurate, in which case we have to use *fine cuts*. On the other hand, in stages where we believe there is little extra information to be gained from *fine* cuts compared to *coarse* cuts, we can save some computational effort by utilizing cutting-plane approximations of any quality (*coarse* and *semi-coarse* cuts). We refer to this algorithm as the SPAP-SDDP algorithm, summarize it in Algorithm 4 and illustrate it in Figure 3(c).

Algorithm 4 The adaptive partition-based SDDP algorithm with structured policies cut generation (SPAP-SDDP).

STEP 0: Initialization.
1. Let $p = 0$, $\check{\mathfrak{Q}}_t^p(\cdot) := -\infty$, $\varepsilon \geq 0$, $\mathscr{Z} \in \mathbb{R}$
2. Classify every stage $t = 2,\dots,T$ such that t is "more-important" if $t \in \text{MI} := \{t \in \mathbb{R}_+ | \bar{z}_t \leq \mathscr{Z}\}$ and t is "less-important" if $t \in \text{LI} := \{t \in \mathbb{R}_+ | \bar{z}_t > \mathscr{Z}\}$.
3. Define an initial sequence of partitions $\overrightarrow{\mathscr{N}}^p = (\overrightarrow{\mathscr{N}}_1^p,\dots,\overrightarrow{\mathscr{N}}_T^p)$, the corresponding coarse tree $\bar{\mathscr{T}}(\overrightarrow{\mathscr{N}}^p)$, $\forall\, t \in \text{LI}$
4. Initialize the corresponding necessary parameters of the SDDP algorithm.

STEP 1: Increment $p \leftarrow p+1$ and choose a sample path $\xi^p = (\xi_2^{p_2},\dots,\xi_T^{p_T})$.

STEP 2: Implement the SDDP *forward step* over ξ^p to obtain $\check{x}_t = \check{x}_t(\xi_t^{p_t})$, $\forall t = 1,\dots T-1$.

STEP 3: Implement the *backward step* as follows:
1. For $t = T-1,\dots,1$
   (a) If $t \in \text{MI}$ compute a *fine cut* by solving the scenario-based subproblem (23) for each scenario $\xi_t \in \Xi_t$. Else, go to (b).
   (b) solve the partition-based subproblem (24) for each cluster $\mathscr{P}_{t+1}^\ell \in \overrightarrow{\mathscr{N}}_{t+1}^p$, $\ell = 1,2,\dots,L_{t+1}$ to compute a coarse cut for $\check{\mathfrak{Q}}_{t+1}(\check{x}_t)$.
   (c) If $\check{\mathfrak{Q}}_{t+1}(\check{x}_t) - \check{\mathfrak{Q}}_{t+1}^p(\check{x}_t) > \varepsilon$, update $\check{\mathfrak{Q}}_{t+1}(x_t)$ and go to $t-1$. Else, go to (d).
   (d) compute a semi-coarse cut as described in (3.2.1) and $\check{\mathfrak{Q}}_{t+1}(x_t)$.

STEP 4: If termination criterion is achieved, STOP. Else, go to STEP 1.

Perhaps the most crucial question to this framework is how we can classify different stages into "more-important" and "less-important". The answer to this question is clearly non-trivial, problem specific, and perhaps one that deserves a separate in-depth study by itself. Nonetheless, in our numerical experiments (see Section 5) we consider a heuristic approach of classification that we justify below.

We preface this by revisiting the hydro-thermal power generation planning problem that is considered in our numerical experiments. Readers are referred to, e.g., [31, 25], for a more thorough discussion on the problem. In this problem, the decision maker aims to minimize the expected total cost which consists of: the power generation expenses and the penalty for the shortage in satisfying the demand. The stochasticity aspect of the problem arises due to the uncertainty about the amount of rainfall in the future – which the decision maker can use

to generate power via the interconnected network of hydro plants. As such, from a stochastic programming point of view, the amount of rain available at every stage $t = 1, \ldots, T$ is what defines the valuable information that the decision maker will utilize in order to construct an optimal policy. In our implementation, we classify the stages as following:

1. "Dry season" stage, which we label as "less important".
2. "Wet season" stage, which we label as "more important".

The motivation for considering stages in the wet seasons to be of more importance and stages in the dry seasons to be of less importance is that, from an optimization point of view, the decision policy plays a more important role when it has more valuable resources at its disposal compared to when it does not. To put this into perspective, given the nature of the problem being a resource allocation/planning problem, an optimal policy is characterized by how balanced the availabilities of various types of resources are at times of abundance with their amounts at times of deficit. In dry seasons there is less of a decision to be made about resource allocation and more of a cost to be paid as a recourse action; whereas during the wet seasons the decision maker has to achieve a balance by allocating some of the available resources for generating power to meet the immediate demand while reserving some for hedging against the potential deficit in the future.

In our implementation, we classify the dry and wet season stages by doing the following for every stage $t = 2, \ldots, T$:

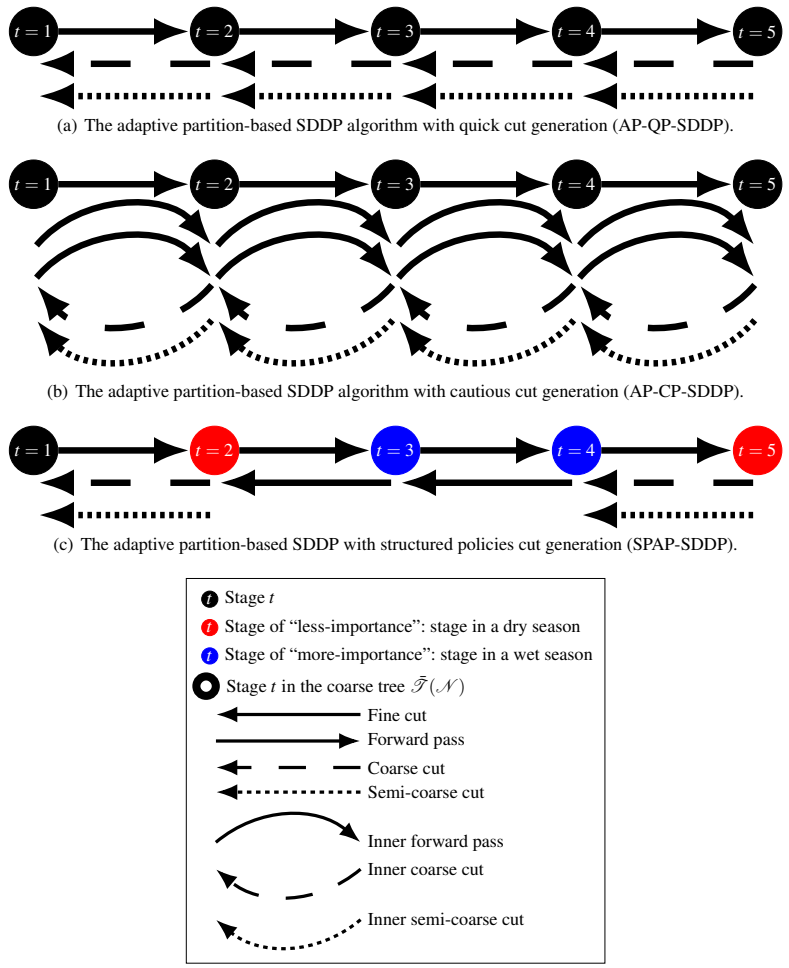
1. First, in order to differentiate between the different stages solely based on the inflow of rain, we equalize for everything by setting the water level carried over from the previous stage at every reservoir to be zero. That is, we assume that every reservoir is empty. This step is done by setting the state variable $x_{t-1} = 0$.
2. Next, we optimize myopically with respect to stage t by solving the stage t subproblem for every realization of random vector $\xi_t \in \Xi_t$.
3. Let $z^*(t, \xi_t)$ be the optimal objective value corresponding to every respective problem solved in the previous step.
4. Define $\bar{z}_t = \sum_{k=1}^{|\Xi_t|} z^*(t, \xi_t^k)$ be the "worst-case" immediate cost at stage t, and let $\mathscr{Z}$ be a user pre-specified parameter.
5. If $\bar{z}_t \leq \mathscr{Z}$, then stage t is a "wet-stage". Otherwise, t is a "dry-stage".

Below is a summary of all the variants of the adaptive partition-based SDDP algorithm being considered in our implementations. All the acronyms are also provided in Table 1.

1. **APEP-SDDP:** perform the SDDP algorithm iteratively on a coarse tree and refine the tree using a cautious pass whenever the tree is not providing "significant" progress. This is used as a preprocessing step for the standard SDDP.
2. **ITER-SDDP:** perform the SDDP algorithm iteratively on a coarse tree and refine the tree using a quick pass whenever the tree is not providing significant progress. This process is repeated until the termination criterion (e.g., time) is met.
3. **AP-SDDP-QP:** sample from the original scenario tree going forward and attempt to generate cuts (and refine) going backward using a quick pass.
4. **AP-SDDP-CP:** sample from the original scenario tree going forward and attempt to generate cuts (and refine) going backward using a cautious pass.
5. **SPAP-SDDP:** sample from the original scenario tree going forward and attempt to generate, fine cuts in "more-important" stages and (semi) coarse cuts in "less-important" stages when going backward.

Acronyms	Description
1. APEP-SDDP	Adaptive partition-enabled preprocessing for SDDP
2. ITER-SDDP	SDDP on iteratively refined coarse scenario trees
3. AP-SDDP-QP	Adaptive partition-based SDDP with quick cut generation
4. AP-SDDP-CP	Adaptive partition-based SDDP with cautious cut generation
5. SPAP-SDDP	Adaptive partition-based SDDP with structured policies cut generation

Table 1 Acronyms of different variants of the adaptive-partition based SDDP aelgorithms.



(a) The adaptive partition-based SDDP algorithm with quick cut generation (AP-QP-SDDP).

(b) The adaptive partition-based SDDP algorithm with cautious cut generation (AP-CP-SDDP).

(c) The adaptive partition-based SDDP with structured policies cut generation (SPAP-SDDP).

Fig. 3 Illustration of tree traversal strategies and various types of cutting planes used in the *Refinement within SDDP* algorithms and the SPAP-SDDP.

5 Numerical Results

In this section we report and analyze our numerical experiment results to show the empirical performances of the proposed algorithms. We first present the test instances and give an overview of different algorithms tested in the numerical experiments. Specifically, in Subsection 5.2 we compare the *Refinement outside SDDP* algorithms with the standard SDDP algorithm. In Subsection 5.3 we compare the *Refinement within SDDP* algorithms with the corresponding SDDP algorithms with different tree traversal strategies. Finally, in Subsection 5.4 we compare the *adaptive partition-based SDDP with structured policies cut generation* with the standard SDDP algorithm and different adaptive partition-based strategies.

We implemented all algorithms in *Julia* 0.6.2, using package *JuMP* 0.18.4 [13], with commercial solver *Gurobi*, version 8.1.1 [18]. All the tests are conducted on Clemson University's primary high-performance computing cluster, the *Palmetto* cluster, where we used an *R830 Dell Intel Xeon* compute node with 2.60GHz and 1.0 TB memory. The number of cores is set to be 24.

5.1 Test Instances and Algorithms

As previously noted, we consider the multistage hydro-thermal power generation planning problem described in [25, 3]. We also use the same problem instance provided by E. Finardi and F. Beltrán, which models the Brazilian hydro-thermal power system. From the original data set, in order to create a variety of instances, we consider different planning horizons $T \in \{24, 60, 96, 120\}$ and sample sizes $|\Xi_t| \in \{50, 200, 1000\}$ of the random vector ξ_t, $\forall\, t = 2, \dots T$. In particular, the original data set contains 200 scenarios per stage, which are directly used in our instances; for instances with $|\Xi_t| = 50$, we simply pick the first 50 scenarios in the data set; for instances with $|\Xi_t| = 1000$, we first fit a Gamma distribution for every stage and for each hydro plant in the network around the 200 scenarios of that stage, and then use this Gamma distribution to generate a sample of 1000 scenarios for the instance. We let the number of realizations to be the same at every stage. For example, when $T = 120$ and $|\Xi_t| = 1000$, we will have $|\Xi_2| \times |\Xi_3| \times \dots |\Xi_{120}| = 1000^{119}$ scenarios (sample paths).

Additionally, following on from the descriptions of different tree traversal strategies in Subsection 3.2.2 we implement two different variants of the SDDP algorithm:

1. *SDDP with quick pass (QP-SDDP)*: adopt a quick pass strategy in traversing the scenario tree. We emphasize that QP-SDDP is the most commonly used variant of the *SDDP* algorithm and we also refer to it as the *standard SDDP* algorithm.

2. *SDDP with cautious pass (CP-SDDP)*: adopt a cautious pass strategy in traversing the scenario tree.

To get an initial lower bound for the cost-to-go function $\check{\bar{\Omega}}_t(\cdot)$, $\forall\, t = 1, 2, \dots T-1$, we solve the mean value problem with respect to the $(t+1)$-th stage problem by taking the expectation of the random vector ξ_t and treating $\bar{x}_t$ in (11) as decision variables. To measure the performances of different algorithms we mainly report two statistics: (i) the *lower* bound $\underline{z}$ value (which is an important metric when solving MSLP problems); and (ii) the *upper* bound (Monte Carlo) estimates $\tilde{z}$ after one, three and six hours (3600, 10800 and 21600 seconds, respectively) of processing. To construct these upper bound estimates, we generate a random sample with sample size $|\mathscr{S}| = 10000$ at each evaluation points. One important

thing to note here is that, since constructing the upper bound estimates requires a significant number of scenarios ($|\mathscr{S}| = 10000$ in this case), a common approach in practice is to use a different number of sample paths which is smaller, in every forward-backward pass iteration of the SDDP algorithm to construct the cuts. In our experiments, we considered varying the values of this different (smaller) number of sample paths and we have found that using a *single* sample path per forward-backward step (one scenario per iteration) to work best. This was the case in all algorithms that we tested.

To that end, we analyze the performance results by focusing on the following three factors:

1. The total number of stages in the planning horizon T.
2. The number of realizations per stage $|\Xi_t|$.
3. The processing time limit.

5.2 Numerical Results for the Refinement outside SDDP Strategy

We report and analyze the results of the two variants of the *Refinement outside SDDP* strategy presented in Subsection 4.1, namely APEP-SDDP and ITER-SDDP, and compare them to those obtained by the standard SDDP algorithm, i.e., QP-SDDP. We report the numerical results in Tables 2, 3 and a few selected instances in Figure 4, from which we observe the following:

1. Performance with respect to T:
   - The overall performances of both APEP-SDDP and ITER-SDDP compared to that of the standard SDDP algorithm is consistently improving as the number of stages T increases. Specifically, except for the case when $T = 24$, both APEP-SDDP and ITER-SDDP outperform QP-SDDP for all processing time limits. Specifically, averaged across all instances with different $|\Xi_t|$ and different time limits, for instances where $T = 24$, QP-SDDP outperformed APEP-SDDP and ITER-SDDP by 3% and 6%, respectively. Whereas, for instances where $T = 60, T = 96$, and $T = 120$, APEP-SDDP outperformed QP-SDDP by 13%, 35%, and 45%, respectively; and ITER-SDDP outperformed QP-SDDP by 14%, 35%, and 46%, respectively.
2. Performance with respect to $|\Xi_t|$:
   - The overall performances of both APEP-SDDP and ITER-SDDP (averaged across different values of T) compared to that of the standard SDDP algorithm is consistently improving as $|\Xi_t|$ increases.
   - At its peak, when $|\Xi_t| = 1000$ and $T = 120$, the improvements over QP-SDDP reach to about 125% and 126% after one hour, and 73% and 77% after six hours, for APEP-SDDP and ITER-SDDP, respectively, and 107% for both after three hours.
3. Performance with respect to the processing time limit:
   - The overall performances of the two algorithms APEP-SDDP and ITER-SDDP have the following trend under different processing time limits: as the processing time limit increases, the relative gap between the lower bounds of *Refinement outside SDDP* algorithms and the SDDP algorithm, either deteriorates if it was superior or improves if it was inferior.
   - For number of stages T and realizations per stage $|\Xi_t|$ at the smaller end of our instances set, the *Refinement outside SDDP* algorithms yield inferior performance compared to the SDDP algorithm and start to improve as processing time limit increases; whereas for larger instances, during earlier periods of processing time, *Re-*

finement outside SDDP algorithms inherit a significant lead over the SDDP algorithm in its lower bound progress, but this lead steadily shrinks as the processing time increases.

- The advantage of the *Refinement outside SDDP* algorithms over the SDDP algorithm is most apparent after one hour of processing time limit, yielding a better performance in 9 out of 12 instances for both the APEP-SDDP and ITER-SDDP algorithms.
- Convergence behaviour: we can see that
  - for smaller instances e.g., $T = 24$ and $|\Xi_t| = 50$: all different algorithms show significant progress in closing the optimality gap after the first hour.
  - for the the medium instances e.g., $T \in \{60, 96\}$ and $|\Xi_t| = 200$: all different algorithms show some progress in closing the optimality gap at the end of the sixth hour.
  - for the the large instances e.g., $T = 120$ and $|\Xi_t| = 1000$: none of the algorithms shows much progress in closing the optimality gap even after six hours.
- In the smaller set of instance e.g., $T \in \{24, 60\}$ and $|\Xi_t| \in \{50, 200\}$ we can see that the lower bound value in both the refinement outside SDDP strategies and the QP-SDDP algorithm grows immediately. However, for the larger set of instance e.g., $T \in \{96, 120\}$ and $|\Xi_t| \in \{200, 1000\}$ we can see that the lower bound value obtained by the QP-SDDP algorithm (especially when $T = 120$ and $|\Xi_t| = 1000$) takes very long to grow above zero. This is not necessarily the case for the APEP-SDDP and ITER-SDDP algorithms, although relatively slower compare to smaller instances. This is not difficult to postulate since in early iterations of such large instances when using the QP-SDDP algorithm one would have to solve a scenario subproblem for every realization in every stage during the backward pass. Whereas, the refinement outside SDDP strategies could benefit tremendously from the coarse cuts added using aggregated scenarios. This time saving during the backward pass allows for more iterations and hence more sample paths to be visited, which in turn allows for more exploration of the state space.

We attribute the aforementioned observations to the following:

- It should come as no surprise that, the larger the instance in terms of T and $|\Xi_t|$, the better the performance of *Refinement outside SDDP* algorithms should be. This advantage for the APEP-SDDP and ITER-SDDP algorithms over the standard SDDP algorithm, is a natural consequence due to the merits of adaptive partition-based strategies in making the cut generation effort adaptive to the solution progress.
- The aforementioned advantage does not hold for smaller instances, such as when $T = 24$ and $|\Xi_t| = 50$. This is because the computational savings provided by this framework via faster cut generation effort from coarse scenario trees do not offset the significant inaccuracy inherited in these coarse cuts on these instances. This incompetence, however, steadily vanishes as the processing time limit increases when the coarse tree gets more and more refined.
- This explains the aforementioned observations regarding the performance with respect to the processing time limit:
  - In smaller instances, the larger the processing time limit is, the more the algorithm is able to compensate for the inaccuracy compromised during the early phase of the solution process.

  - In larger instances, the larger the processing time limit is, the bigger that the size of the coarse tree gets, making the cut generation effort in the *Refinement outside SDDP* algorithms similar to that of the standard SDDP algorithm.
- Finally, comparing between the two *Refinement outside SDDP* algorithms, we see that their performances are comparable, with *ITER-SDDP* prevailing in all instances except for a few instances, e.g., ($T = 24, |\Xi_t| = 50$), and ($T = 24, |\Xi_t| = 1000$). We attribute these two cases to the heuristically chosen parameters n used in Algorithm 3 as a criterion to perform partition refinement on $\overrightarrow{\mathcal{N}}^p$.

We conclude this subsection by emphasizing that, our work is an attempt to provide a framework which mitigates the computational burden of solving MSLPs brought by the large number of stages T and large number of realizations per stage $|\Xi_t|$. This, if anything, can only testify to the competitive nature of *adaptive partition-based* strategies in solving large-scale problems.

5.3 Numerical Results for the Refinement within SDDP Strategy

In this subsection, we compare different *Refinement within SDDP* algorithms described in Subsection 4.2 and the corresponding SDDP algorithms (where no adaptive partition is employed) under different tree traversal strategies. We report the numerical results in Tables 4 and 5 as well as a few selected instances in Figure 5, from which we observe the following.

1. Performance with respect to T:
   - Similar to the observations made in Subsection 5.2, the performance (averaged across different values of $|\Xi_t|$ and different time-limits) of the AP-QP-SDDP algorithm compared to the QP-SDDP algorithm improves as T increases. Whereas the performance of the AP-CP-SDDP algorithm compared with that of CP-SDDP gets worse as T increases.
   - Except for a few instances the *Refinement within SDDP* algorithms are consistently outperformed by the SDDP algorithm with the corresponding tree traversal strategy.
2. Performance with respect to $|\Xi_t|$:
   - Averaged across different values of T, the performance for the AP-QP-SDDP compared to QP-SDDP improves as $|\Xi_t|$ increases. Whereas the performance for the AP-CP-SDDP compared to CP-SDDP deteriorates as $|\Xi_t|$ increases. This behaviour is consistent for different time limits.
3. Performance with respect to the processing time limit:
   - Unlike the observation made in Subsection 5.2, the overall performances of the two *Refinement within SDDP* algorithms do not seem to have a consistent pattern for different time limits.

We attribute the aforementioned observations to the following:

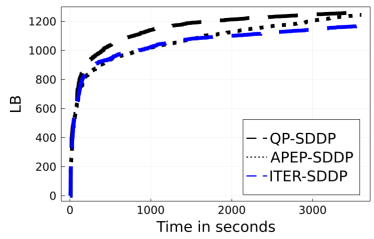
- As one might expect, allocating a significant cut generation effort to a small number of sample paths, is surely a waste of computational budget. This perhaps justifies the overall disappointing results of any algorithm with a tree traversal strategy that has some cautiousness aspects to it, since a cautious tree traversal strategy usually results in a decision policy that is overfitting to the subset of sample path(s) visited so far in the solution process.
- This overfitting in the decision policy could also explain the absence for any clear pattern in the performance of AP-CP-SDDP compared to that of CP-SDDP in different

Instances		time limit (hrs)	$\%\underline{z}$	$\underline{z}$	$\tilde{z}$	$\pm 1.96\tilde{\sigma}/\lvert\mathscr{S}\rvert$
T	$\lvert\Xi_t\rvert$					
24	50	1	0%	1268.8	1578.2	260.0
		3	1%	1351.8	1650.4	255.4
		6	0%	1382.7	1689.7	264.9
	200	1	4%	558.1	1121.6	224.2
		3	-1%	658.9	1033.9	195.7
		6	-3%	695.9	889.5	179.9
	1000	1	-21%	293.4	939.3	201.7
		3	-7%	429.7	1044.2	196.9
		6	-2%	507.6	1031.3	199.0
60	50	1	-1%	11542.1	20243.9	1316.6
		3	-1%	13184.6	18809.5	1224.9
		6	1%	13992.7	17936.8	1208.9
	200	1	22%	6915.5	14057.2	1054.5
		3	4%	8155.6	15258.6	1106.1
		6	0%	8832.1	13790.7	1066.3
	1000	1	39%	3425.1	13303.9	1057.3
		3	29%	5209.3	12240.7	1002.3
		6	21%	6128.0	12708.9	1063.5
96	50	1	-4%	20440.0	43158.4	2171.7
		3	-4%	24748.6	43057.4	2142.9
		6	-4%	26423.4	40410.1	2047.7
	200	1	35%	8802.5	29835.2	1801.4
		3	16%	12272.4	29523.0	1781.1
		6	3%	13842.2	27967.9	1692.6
	1000	1	160%	5135.1	29857.2	1855.8
		3	65%	7743.8	23569.6	1538.3
		6	45%	9201.8	24856.5	1577.7
120	50	1	3%	24675.2	56631.6	2543.5
		3	-1%	29347.3	53209.1	2482.7
		6	-3%	31181.5	54567.1	2580.4
	200	1	48%	10766.7	42268.2	2286.6
		3	32%	14555.9	39501.1	2180.5
		6	24%	17008.6	39200.8	2170.3
	1000	1	125%	4573.2	37001.7	2129.7
		3	107%	7919.8	35109.2	2053.8
		6	73%	9597.2	35442.0	2108.0

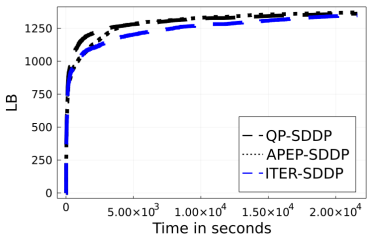
Table 2 Lower bound $\underline{z}$ progress and the upper bound estimates $\tilde{z}$ obtained by the APEP-SDDP algorithm (see Section 4.1.1) compared to the QP-SDDP algorithm. $\%\underline{z} = 100 \times \frac{(\underline{z} - \underline{z}_{QP-SDDP})}{(\underline{z}_{QP-SDDP})}$ and $|\mathscr{S}| = 10000$.

Instances		time limit (hrs)	$\%\underline{z}$	$\underline{z}$	$\tilde{z}$	$\pm 1.96\tilde{\sigma}/\lvert\mathscr{S}\rvert$
T	$\lvert\Xi_t\rvert$					
24	50	1	-6%	1195.0	1659.4	251.4
		3	-2%	1307.6	1504.2	256.9
		6	-2%	1354.2	1735.2	288.9
	200	1	4%	558.3	1112.8	221.6
		3	-2%	649.3	1223.6	226.3
		6	-1%	711.9	1015.2	204.9
	1000	1	-21%	293.2	938.5	202.2
		3	-13%	398.9	987.7	196.9
		6	-8%	479.1	1117.6	216.0
60	50	1	4%	12149.9	19291.0	1302.8
		3	2%	13594.8	18755.8	1253.2
		6	2%	14145.5	18845.2	1217.9
	200	1	22%	6910.3	14166.8	1064.9
		3	4%	8163.0	15104.6	1109.9
		6	2%	9005.7	14854.4	1124.2
	1000	1	39%	3425.1	13303.9	1057.3
		3	29%	5208.3	12126.5	994.5
		6	21%	6122.8	12707.4	1044.9
96	50	1	-2%	20866.1	43234.5	2141.9
		3	-1%	25449.3	43107.0	2164.4
		6	-2%	27071.4	41903.3	2074.4
	200	1	34%	8795.7	29902.0	1796.2
		3	17%	12348.1	31100.0	1876.1
		6	4%	14049.0	27391.5	1654.3
	1000	1	160%	5129.8	29203.4	1840.2
		3	62%	7618.8	25347.5	1643.1
		6	40%	8885.5	26294.7	1606.5
120	50	1	5%	25104.6	56957.2	2640.5
		3	1%	29921.9	52880.2	2479.2
		6	-1%	31806.5	51540.2	2472.3
	200	1	48%	10792.1	42730.3	2298.7
		3	33%	14616.4	40973.7	2251.8
		6	22%	16757.0	38618.4	2113.7
	1000	1	126%	4598.0	36480.5	2101.1
		3	107%	7950.9	36795.5	2166.3
		6	77%	9816.3	35866.8	2112.6

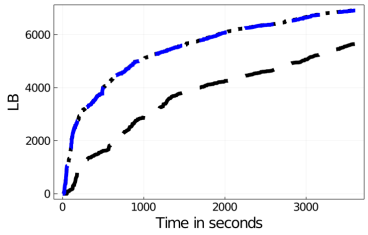
Table 3 Lower bound $\underline{z}$ progress and the upper bound estimates $\tilde{z}$ obtained by the ITER-SDDP algorithm (see Section 4.1.2) compared to the QP-SDDP algorithm. $\%\underline{z} = 100 \times \frac{(\underline{z} - \underline{z}_{QP-SDDP})}{(\underline{z}_{QP-SDDP})}$ and $|\mathscr{S}| = 10000$.



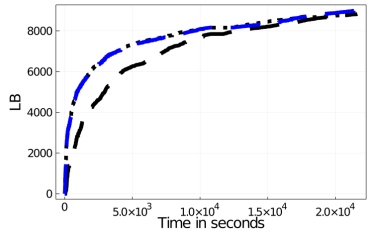
(a) $T = 24, |\Xi_t| = 50$. One hour of processing.



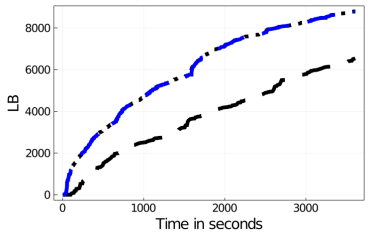
(b) $T = 24, |\Xi_t| = 50$. Six hours of processing.



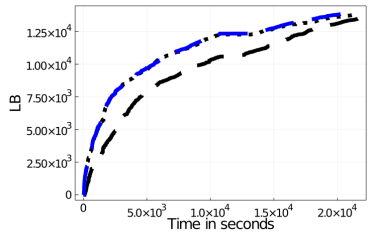
(c) $T = 60, |\Xi_t| = 200$. One hour of processing.



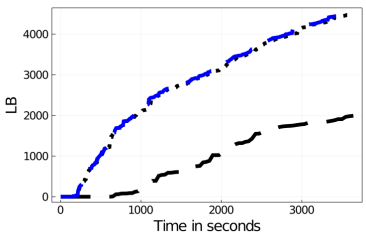
(d) $T = 60, |\Xi_t| = 200$. Six hours of processing.



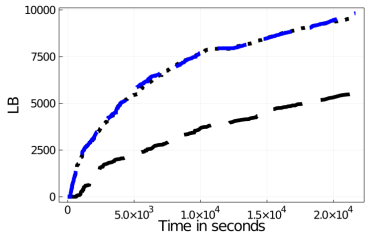
(e) $T = 96, |\Xi_t| = 200$. One hour of processing.



(f) $T = 96, |\Xi_t| = 200$. Six hours of processing.



(g) $T = 120, |\Xi_t| = 1000$. One hour of processing.



(h) $T = 120, |\Xi_t| = 1000$. Six hours of processing.

Fig. 4 Solution progress using *Refinement outside SDDP* compared to QP-SDDP after one and six hours of processing on different instances.

instances, since the performances depend on the sample paths visited by the algorithms, which are randomly generated.

- When analyzing the incompetence in the performance of *Refinement within SDDP* algorithms, and in particular the AP-QP-SDDP algorithm compared to *Refinement outside SDDP* algorithms, it is important to note the following:
  - In the *Refinement outside SDDP* algorithms, where we only generate coarse cuts by implementing the standard SDDP algorithm on the coarse tree $\bar{\mathscr{T}}(\overrightarrow{\mathscr{N}}^p)$, there is a criterion by which we measure the added value of these coarse cuts to the current cost-to-go function approximations $\check{\check{\Omega}}_t^p(\cdot)$. This is done by keeping track of the lower bound progress when generating cuts from the coarse tree at every iteration (see Algorithm 3). Hence, we restrict the algorithm to allocate some of the computational budget to generating coarse cuts using $\overrightarrow{\mathscr{N}}^p$, only if these coarse cuts have significant added value to them. Otherwise, $\overrightarrow{\mathscr{N}}^p$ will be refined.
  - In the *Refinement within SDDP* algorithms, the process is continuously attempting to generate coarse cuts, irrespective of their added value to the decision policy. This might hinder the performance of the algorithm, especially in the case of AP-QP-SDDP algorithm, where the existence of any violated cut, regardless of its quality, is the criterion for going back from stage t to $t-1$. This makes refining the coarse tree (in order to obtain some accuracy) less frequent.

We conclude this subsection by mentioning that, while the coarse cuts generated by the *Refinement within SDDP* algorithms do not seem to serve its desired purpose, as far as the previous analysis goes, this may not necessarily be the case under different time limits or different integration scheme such as the SPAP-SDDP algorithm that we shall discuss next.

5.4 Computational Experiments on the Adaptive Partition-based SDDP with structured cut generation policies

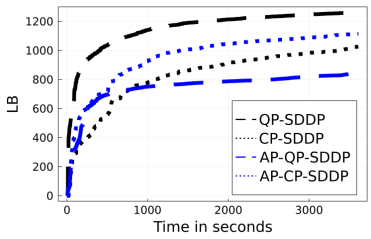
In this subsection we report and analyze the results for the performance of the SPAP-SDDP algorithm presented in Subsection 4.3. In Table 6 we report its performance and compare it to the standard SDDP algorithm. Additionally, we illustrate a few selected instances in Figure 6 which compare the performance of the SPAP-SDDP algorithm with that of different *Refinement outside SDDP* algorithms, as well as the AP-QP-SDDP and QP-SDDP algorithms.

The overall pattern in the behavior of the SPAP-SDDP algorithm for different number of stages T, realizations per stage $|\Xi_t|$ and processing time limits, is very similar to the observations made regarding the *Refinement outside SDDP* algorithms. We summarize these observations as follows:

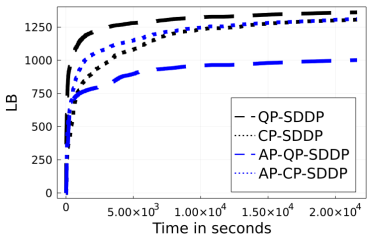
- Except for the first hour of the instance where $(T = 120, |\Xi_t| = 1000)$, the performance of the SPAP-SDDP algorithm compared to that of the standard SDDP algorithm is consistently improving as T and/or $|\Xi_t|$ increases.
- Similar to the observations made in Subsection 5.2 and except for when $(T = 24, |\Xi_t| = 200)$ and $(T = 120, |\Xi_t| = 1000)$, the overall performances of the SPAP-SDDP algorithm have the following trend under different processing time limits: as the processing time limit increases, the relative gap between the lower bounds of the SPAP-SDDP algorithm and the SDDP algorithm, either deteriorates if it was superior or improves if it was inferior.

Instances		time limit (hrs)	$\%\underline{z}$	$\underline{z}$	$\tilde{z}$	$\pm 1.96\tilde{\sigma}/\lvert\mathscr{S}\rvert$
T	$\lvert\Xi_t\rvert$					
24	50	1	-31%	874.4	1655.6	245.0
		3	-25%	1000.9	1623.4	256.9
		6	-24%	1041.1	1858.7	306.9
	200	1	-33%	359.3	874.7	205.3
		3	-30%	467.2	1258.2	237.7
		6	-30%	502.9	1088.1	213.1
	1000	1	-32%	252.6	933.3	190.8
		3	-27%	334.4	974.8	203.5
		6	-25%	389.1	1060.4	212.3
60	50	1	-33%	7843.9	19615.6	1285.2
		3	-28%	9643.6	20420.9	1324.1
		6	-26%	10300.5	19652.6	1288.4
	200	1	-12%	4946.5	15811.3	1155.7
		3	-18%	6424.8	13697.9	1049.0
		6	-20%	7021.9	14782.3	1100.5
	1000	1	19%	2910.0	15085.2	1122.5
		3	7%	4329.2	12978.3	1027.5
		6	0%	5049.0	13046.6	1009.6
96	50	1	-19%	17222.6	41401.7	2102.9
		3	-20%	20665.7	42076.4	2108.9
		6	-19%	22275.2	40545.2	2053.2
	200	1	-13%	5666.4	31427.5	1809.0
		3	-20%	8499.6	28384.1	1721.1
		6	-23%	10422.0	30921.5	1886.0
	1000	1	-8%	1814.7	28645.6	1774.6
		3	-2%	4603.0	25738.3	1657.9
		6	13%	7153.4	26009.3	1657.9
120	50	1	-14%	20403.0	54505.4	2533.5
		3	-10%	26882.5	54056.0	2512.3
		6	-12%	28206.4	55706.3	2534.5
	200	1	-4%	6941.5	43669.3	2357.0
		3	-1%	10913.8	40205.1	2173.5
		6	-8%	12562.8	38633.8	2114.2
	1000	1	-4%	1907.4	38862.3	2148.3
		3	39%	5312.6	35381.5	2021.7
		6	46%	8058.3	34850.4	2018.4

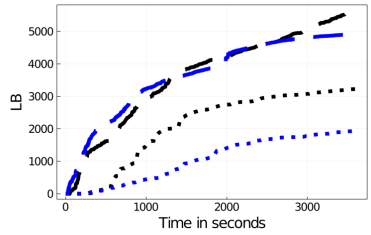
Table 4 Lower bound $\underline{z}$ progress and the upper bound estimates $\tilde{z}$ obtained by the AP-QP-SDDP algorithm (see Section 4.2.1) compared to the QP-SDDP algorithm. $\%\underline{z} = 100 \times \frac{(\underline{z} - \underline{z}_{QP-SDDP})}{(\underline{z}_{QP-SDDP})}$ and $\lvert\mathscr{S}\rvert = 10000$.



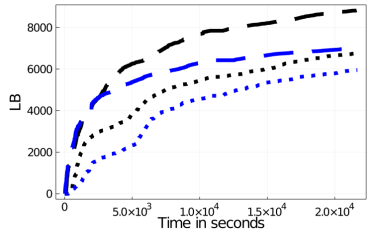
(a) $T = 24, |\Xi_t| = 50$. One hour of processing.



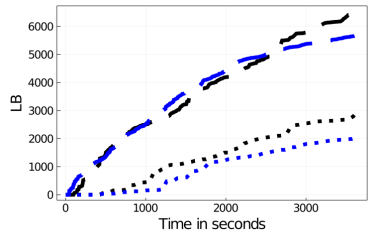
(b) $T = 24, |\Xi_t| = 50$. Six hours of processing.



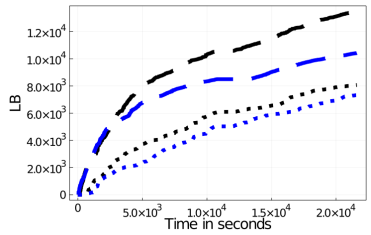
(c) $T = 60, |\Xi_t| = 200$. One hour of processing.



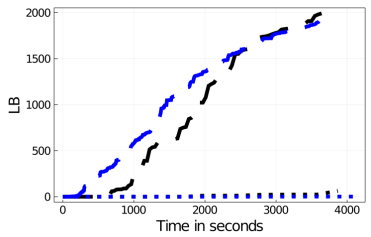
(d) $T = 60, |\Xi_t| = 200$. Six hours of processing.



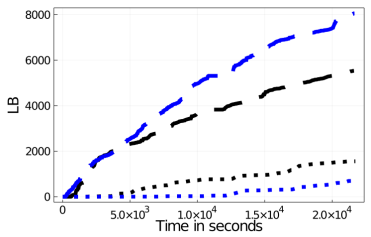
(e) $T = 96, |\Xi_t| = 200$. One hour of processing.



(f) $T = 96, |\Xi_t| = 200$. Six hours of processing.



(g) $T = 120, |\Xi_t| = 1000$. One hour of processing.



(h) $T = 120, |\Xi_t| = 1000$. Six hours of processing.

Fig. 5 Solution progress using *Refinement within SDDP* algorithms compared to the corresponding SDDP algorithm under different tree traversal strategies for one and six hours of processing on different instances.

Instances T	Instances $\lvert\Xi_t\rvert$	time limit (hrs)	$\%\underline{z}$	$\underline{z}$	$\tilde{z}$	$\pm 1.96\tilde{\sigma}/\lvert\mathscr{S}\rvert$
24	50	1	-11%	1140.5	1735.2	276.0
		3	-5%	1269.0	1857.0	293.9
		6	-5%	1311.2	1599.0	259.1
	200	1	-48%	280.8	1081.6	233.8
		3	-35%	435.2	1174.6	215.5
		6	-21%	568.6	1023.2	199.1
	1000	1	-72%	103.0	1129.2	226.2
		3	-56%	201.1	1300.6	233.9
		6	-44%	291.4	934.3	201.8
60	50	1	-22%	9044.5	19487.6	1236.2
		3	-15%	11370.9	19209.6	1241.4
		6	-10%	12438.2	18276.1	1261.4
	200	1	-66%	1938.7	14873.0	1091.0
		3	-40%	4714.3	15367.9	1125.1
		6	-33%	5951.7	15317.5	1154.4
	1000	1	-97%	82.0	27024.0	1355.5
		3	-82%	731.6	13743.6	1071.8
		6	-68%	1608.5	14197.0	1163.0
96	50	1	-33%	14248.8	44633.0	2178.3
		3	-23%	19767.4	42015.9	2086.7
		6	-17%	22779.3	42574.8	2157.2
	200	1	-69%	2022.0	34056.0	1956.6
		3	-53%	5028.1	34264.9	2097.7
		6	-46%	7327.6	34779.9	2169.4
	1000	1	-96%	72.2	129691.0	3046.6
		3	-91%	424.0	31538.5	1957.0
		6	-88%	789.6	32030.4	2017.7
120	50	1	-42%	13862.7	63115.0	2958.8
		3	-28%	21327.9	60063.0	2806.7
		6	-22%	24935.1	56700.9	2563.5
	200	1	-87%	958.9	53364.3	2765.8
		3	-60%	4451.9	49244.0	2593.4
		6	-47%	7190.1	42010.5	2374.2
	1000	1	-100%	1.6	462666.6	4382.6
		3	-99%	48.0	140768.3	3341.4
		6	-87%	697.9	38660.0	2189.2

Table 5 Lower bound $\underline{z}$ progress and the upper bound estimates $\tilde{z}$ obtained by the AP-CP-SDDP algorithm (see Section 4.2.2) compared to the CP-SDDP algorithm. $\%\underline{z} = 100 \times \frac{(\underline{z} - \underline{z}_{CP-SDDP})}{(\underline{z}_{CP-SDDP})}$ and $|\mathscr{S}| = 10000$.

We attribute the aforementioned observations to the following:

- Most of the reasoning made in Subsection 5.2 regarding the performance of the *Refinement outside SDDP* algorithms can also be made to the SPAP-SDDP algorithm. This reasoning being, in large instances, where accuracy is more computationally expensive to obtain, the computational savings come from the fact that the algorithm makes the cut generation effort adaptive to the solution progress. Except here, this adaptability is

not integrated by the value which a coarse cut adds immediately to the lower bound progress, but instead, by the added value of a coarse cut from a particular stage to the decision policy at giving point in the processing time, which affects the lower bound progress implicitly.

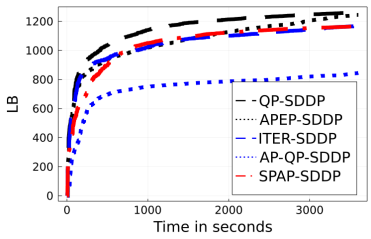
- The most notable difference between the performance the SPAP-SDDP algorithm and the *Refinement outside SDDP* algorithms is that, the performance of the SPAP-SDDP algorithm is very stable in outperforming QP-SDDP compared to that of the *Refinement outside SDDP* algorithms. That is to say, the SPAP-SDDP algorithm is outperformed by QP-SDDP in only two out of the twelve instances, unlike the refinement-outside SDDP strategies which are being outperformed by QP-SDDP in four out of the twelve instances. However, in instances where the refinement-outside SDDP strategy outperform QP-SDDP, it is by a large margin compared to that of SPAP-SDDP.
- Overall, it is not difficult to see that the SPAP-SDDP algorithm bridges the gap between the computational ease of generating excess of inaccurate coarse cuts using *AP-QP-SDDP* and the computational burden of generating a few, but accurate fine cuts using the standard SDDP algorithm.

Finally, we also solve one of the smallest instances in our experiments ($T = 24$ and $|\Xi_t| = 50$) with a large time limit (for 48 hours of processing time) to see the convergence behaviors of different algorithms in terms of the optimality gaps. The refinement strategy coupled with Assumptions 1 implies that coarser trees have lower expected costs. Hence, evaluating the policies on coarse trees could yield inconsistent values. To ensure fairness in comparison, we generate the set $\mathscr{S}$, used to calculate the upper bound estimates, from the original (fine) scenario tree. In particular, we generate $\mathscr{S}$ before running the different algorithms and use it at the various evaluation points. In Table 7, we report the optimality gaps obtained by $100 \times \frac{\tilde{z}-\underline{z}}{\tilde{z}}\%$ at different time limits and in Figure 7, we show the lower bound progress for 48 hours of processing time. From Table 7, we can see that the optimality gaps get smaller as the processing time increases. We can also see that after 48 hours of processing time, the APEP-SDDP algorithm has the smallest $\tilde{z}$ value, followed by QP-SDDP, SPAP-SDDP, ITER-SDDP, CP-SDDP, AP-QP-SDDP and finally AP-CP-SDDP. From Figure 7, we can see that the lower bound value starts to stabilize after, approximately, six hours of processing time. This, however, is the case for all different algorithms except for the AP-QP-SDDP algorithm, which makes its progress gradually throughout the entire processing time.

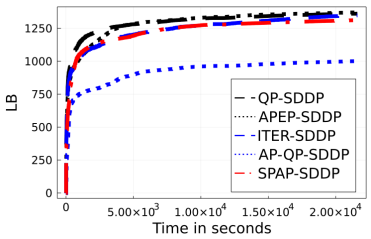
Summary of numerical experiment results. In sum, based on the aforementioned analysis on the numerical results, the *Refinement outside SDDP* algorithms and the SPAP-SDDP algorithm yield the most significant improvements over the standard SDDP algorithm in terms of the lower bound progress. We thereby suggest that as a rule of thumb, integrating the adaptive partition-based strategies into the SDDP algorithm should be done via the *Refinement outside SDDP* approach, and structured policies cut generation should be pursued (like the SPAP-SDDP algorithm) if any structure can be exploited from the underlying problem instance. The collections of computational tools that we propose and develop in this paper can be potentially useful for use in any SDDP related problem instances to make different computational trade-offs, which may enable further exploitation of specific problem structures for better computational performances. One potential drawback of the proposed approach is that, when a nontrivial sufficient partition or even locally sufficient partition is hard to obtain due to high variance of the random vectors in the problem or the high dimensionality/complexity of the dual polyhedron, the adaptive partition-based SDDP becomes a

Instances		time limit (hrs)	$\%\underline{z}$	$\underline{z}$	$\tilde{z}$	$\pm 1.96\tilde{\sigma}/\lvert\mathscr{S}\rvert$
T	$\lvert\Xi_t\rvert$					
24	50	1	-7%	1180.9	1649.6	272.4
		3	-4%	1287.8	1504.2	246.6
		6	-4%	1324.8	1812.8	281.5
	200	1	-3%	522.2	1179.9	242.5
		3	-3%	642.6	1183.3	225.4
		6	-4%	690.4	923.2	191.5
	1000	1	18%	440.1	844.4	202.5
		3	13%	519.7	1134.3	214.7
		6	11%	572.4	995.6	216.6
60	50	1	1%	11757.3	20139.4	1308.5
		3	1%	13522.6	19116.8	1251.3
		6	2%	14085.0	18412.3	1252.6
	200	1	10%	6237.5	15287.1	1104.7
		3	6%	8347.9	14402.7	1087.7
		6	2%	8967.3	14749.8	1108.9
	1000	1	16%	2846.7	14412.9	1092.0
		3	19%	4816.3	12469.7	1014.8
		6	8%	5456.1	14169.9	1131.1
96	50	1	4%	22111.0	42287.9	2162.1
		3	1%	26061.5	41698.4	2102.4
		6	0%	27677.1	40104.2	2047.0
	200	1	23%	8054.0	33350.5	1960.7
		3	12%	11875.4	28794.3	1747.2
		6	4%	13973.7	27844.1	1732.4
	1000	1	27%	2498.1	30230.6	1752.4
		3	22%	5706.1	26405.8	1652.9
		6	24%	7845.1	28220.9	1692.9
120	50	1	5%	25145.1	56949.2	2647.2
		3	5%	31202.0	54784.6	2525.1
		6	3%	33063.0	54943.4	2577.5
	200	1	19%	8642.8	42736.5	2333.2
		3	15%	12648.4	42767.0	2304.3
		6	14%	15589.7	39099.1	2195.9
	1000	1	-3%	1938.2	43857.0	2444.0
		3	31%	5027.5	36974.5	2099.9
		6	23%	6812.7	36372.2	2148.6

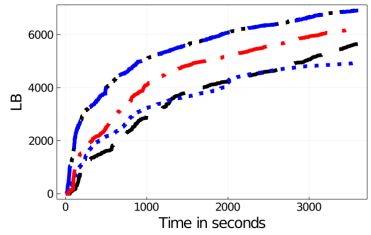
Table 6 Lower bound $\underline{z}$ progress and the upper bound estimates $\tilde{z}$ obtained by the SPAP-SDDP algorithm (see Section 4.3) compared to the QP-SDDP algorithm. $\%\underline{z} = 100 \times \frac{(\underline{z} - \underline{z}_{QP-SDDP})}{(\underline{z}_{QP-SDDP})}$ and $|\mathscr{S}| = 10000$.



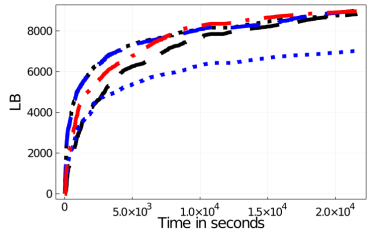
(a) $T = 24, |\Xi_t| = 50$. One hour of processing.



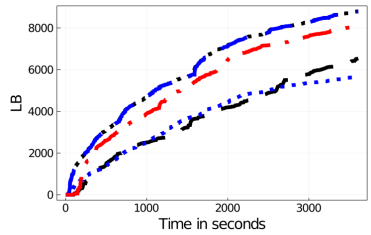
(b) $T = 24, |\Xi_t| = 50$. Six hours of processing.



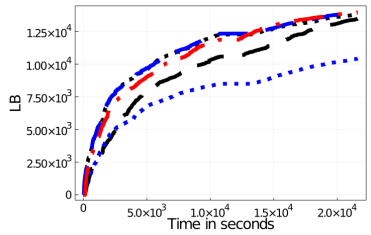
(c) $T = 60, |\Xi_t| = 200$. One hour of processing.



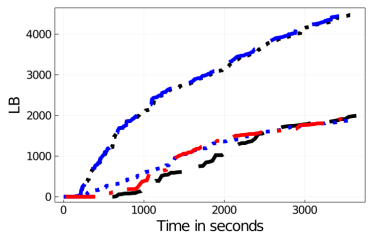
(d) $T = 60, |\Xi_t| = 200$. Six hours of processing.



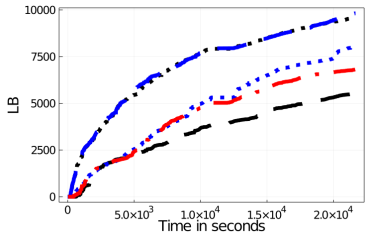
(e) $T = 96, |\Xi_t| = 200$. One hour of processing.



(f) $T = 96, |\Xi_t| = 200$. Six hours of processing.



(g) $T = 120, |\Xi_t| = 1000$. One hour of processing.



(h) $T = 120, |\Xi_t| = 1000$. Six hours of processing.

Fig. 6 Solution progress using SPAP-SDDP algorithm compared to *Refinement outside SDDP* algorithms, AP-QP-SDDP and QP-SDDP after one and the six hours of processing on different instances.

	1hr	3hrs	6hrs	12hrs	24hrs	48hrs
QP-SDDP	19.8%	11.7%	9.7%	4.5%	1.2%	-1.6%
CP-SDDP	81.6%	28.2%	17.7%	11.1%	7.2%	2.9%
APEP-SDDP	27.4%	9.8%	6.9%	2.6%	0.8%	-2.0%
ITER-SDDP	29.7%	15.1%	9.3%	5.3%	2.3%	-1.1%
AP-QP-SDDP	89.7%	61.2%	52.2%	37.9%	22.2%	17.8%
AP-CP-SDDP	42.2%	30.0%	19.6%	10.0%	5.8%	3.2%
SPAP-SDDP	29.4%	20.8%	14.3%	8.7%	3.2%	-1.5%

Table 7 The optimality gaps obtained by $100 \times \frac{\bar{z}-z}{\bar{z}}$ and $|\mathscr{S}| = 10000$ at different time limits on the instance with $T = 24$ and $|\Xi_t| = 50$.

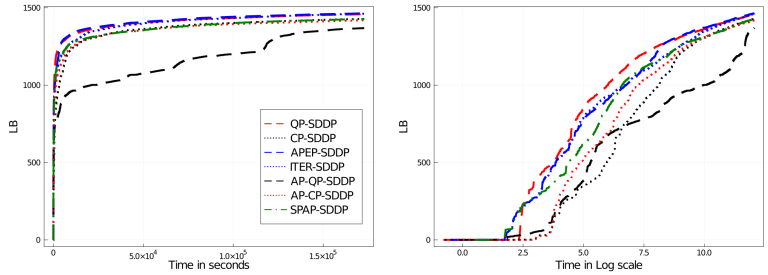


Fig. 7 Solution progress of different algorithms for 48 hours of processing time on the instance with $T = 24$ and $|\Xi_t| = 50$.

standard SDDP rather quickly, in which case it may not worth the extra work generating all different variants of partition-based cuts.

6 Conclusions

In this study, we have investigated various ways to enhance the performance of the SDDP algorithm in terms of its lower bound progress by employing various inexact cut generations and scenario tree traversal strategies. Specifically, we have integrated the adaptive partition-based approaches, which have been shown to be effective in two-stage stochastic programs, into the SDDP algorithm for multi-stage stochastic programs in two different manners: performing partition refinement within the SDDP and outside the SDDP algorithm. In addition, we have proposed a structured cut generation strategy across all stages, which takes advantage of the underlying seasonal uncertainty structure in the class of problems that we use as the test instances.

We have conducted extensive numerical experiments to empirically validate the effectiveness of the proposed algorithms and compare them to the standard SDDP algorithm. From the results, we can see that the refinement *outside* SDDP algorithms outperform the refinement *within* SDDP algorithms. This is likely to be due to the fact that, the refinement *outside* SDDP algorithms generate the coarse cuts (using the partitions) while taking into consideration how much each cut can improve the decision policy. Unlike the refinement

within SDDP algorithms which continuously attempt to generate coarse cuts – irrespective of their added value to the decision policy. Moreover, we have found that the effectiveness of different adaptive partition-based SDDP algorithms were more evident in instances with a large number of stages and scenarios. This, indeed, testified to the competitive nature of the adaptive partition-based approach in solving large-scale problems. Nevertheless, we note that none of the proposed algorithms or the standard SDDP did converge in the allotted time frame within a reasonable tolerance, especially for the large-scale instances.

We have identified several directions to pursue for future research. First, it is of interest to investigate how the proposed algorithms can be applied to address more challenging problem classes such as distributionally robust multistage stochastic programs and multistage stochastic integer programs. Second, from an algorithmic perspective it is worth investigating novel cut generation strategies that are adaptive to the underlying problem structure such as the decision policy structures and/or uncertainty structures during the solution process, as opposed to imposing these structures a priori as we did in the SPAP-SDDP algorithm.

Acknowledgements We would like to acknowledge the coordinating editor and two anonymous referees for their constructive suggestions that considerably improved the original version of this article. Clemson University is acknowledged for generous allotment of compute time on Palmetto cluster. The authors acknowledge partial support by the National Science Foundation [Grant CMMI 1854960]. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Data and code availability The data and code that support the findings of this study are available from the corresponding author upon request.

References

1. Abrahamson, P.G.: Nested-decomposition approach for solving staircase linear programs. Tech. rep., Stanford Univ., CA (USA). Dept. of Operations Research (1983)
2. van Ackooij, W., de Oliveira, W., Song, Y.: Adaptive partition-based level decomposition methods for solving two-stage stochastic programs with fixed recourse. INFORMS Journal on Computing **30**(1), 57–70 (2017)
3. van Ackooij, W., de Oliveira, W., Song, Y.: On level regularization with normal solutions in decomposition methods for multistage stochastic programming problems. Computational Optimization and Applications **74**(1), 1–42 (2019)
4. Ahmed, S., King, A.J., Parija, G.: A multi-stage stochastic integer programming approach for capacity expansion under uncertainty. Journal of Global Optimization **26**(1), 3–24 (2003)
5. Alonso, A., Escudero, L.F., Ortuno, M.T.: A stochastic 0–1 program based approach for the air traffic flow management problem. European Journal of Operational Research **120**(1), 47–62 (2000)
6. Bean, J.C., Birge, J.R., Smith, R.L.: Aggregation in dynamic programming. Operations Research **35**(2), 215–220 (1987)
7. Bellman, R.: Dynamic programming. Princeton University Press (1957)
8. Birge, J.R.: Aggregation bounds in stochastic linear programming. Mathematical Programming **31**(1), 25–41 (1985)
9. Birge, J.R., Louveaux, F.: Introduction to stochastic programming. Springer Science & Business Media (2011)
10. Bomze, I.M., Budinich, M., Pardalos, P.M., Pelillo, M.: The maximum clique problem. In: Handbook of combinatorial optimization, pp. 1–74. Springer (1999)
11. Ch, P.G., Werner, R.: Modeling, measuring and managing risk. World Scientific (2007)
12. Chen, Z.L., Powell, W.B.: Convergent cutting-plane and partial-sampling algorithm for multistage stochastic linear programs with recourse. Journal of Optimization Theory and Applications **102**(3), 497–524 (1999)
13. Dunning, I., Huchette, J., Lubin, M.: Jump: A modeling language for mathematical optimization. SIAM Review **59**(2), 295–320 (2017)

14. Dupačová, J.: Portfolio Optimization and Risk Management via Stochastic Programming. Osaka University Press (2009)
15. Dupačová, J., Polívka, J.: Asset-liability management for czech pension funds using stochastic programming. Annals of Operations Research **165**(1), 5–28 (2009)
16. Fhoula, B., Hajji, A., Rekik, M.: Stochastic dual dynamic programming for transportation planning under demand uncertainty. In: 2013 International Conference on Advanced Logistics and Transport, pp. 550–555. IEEE (2013)
17. Goel, V., Grossmann, I.E.: A stochastic programming approach to planning of offshore gas field developments under uncertainty in reserves. Computers & Chemical Engineering **28**(8), 1409–1429 (2004)
18. Gurobi Optimization, L.: Gurobi optimizer reference manual (2019). URL http://www.gurobi.com
19. Hallefjord, Å., Storøy, S.: Aggregation and disaggregation in integer programming problems. Operations Research **38**(4), 619–623 (1990)
20. Herer, Y.T., Tzur, M., Yücesan, E.: The multilocation transshipment problem. IIE transactions **38**(3), 185–200 (2006)
21. Jardim, D., Maceira, M., Falcao, D.: Stochastic streamflow model for hydroelectric systems using clustering techniques. In: 2001 IEEE Porto Power Tech Proceedings (Cat. No. 01EX502), vol. 3, pp. 6–pp. IEEE (2001)
22. Kall, P., Wallace, S.W., Kall, P.: Stochastic programming. Springer (1994)
23. Kelley Jr, J.E.: The cutting-plane method for solving convex programs. Journal of the society for Industrial and Applied Mathematics **8**(4), 703–712 (1960)
24. Linowsky, K., Philpott, A.B.: On the convergence of sampling-based decomposition algorithms for multistage stochastic programs. Journal of Optimization Theory and Applications **125**(2), 349–366 (2005)
25. 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)
26. 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)
27. Morton, D.P.: An enhanced decomposition algorithm for multistage stochastic hydroelectric scheduling. Annals of Operations Research **64**(1), 211–235 (1996)
28. de Oliveira, W., Sagastizábal, C.: Level bundle methods for oracles with on-demand accuracy. Optimization Methods and Software **29**(6), 1180–1209 (2014)
29. Pantuso, G.: The football team composition problem: a stochastic programming approach. Journal of Quantitative Analysis in Sports **13**(3), 113–129 (2017)
30. Pereira, M.V., Granville, S., Fampa, M.H., Dix, R., Barroso, L.A.: Strategic bidding under uncertainty: a binary expansion approach. IEEE Transactions on Power Systems **20**(1), 180–188 (2005)
31. Pereira, M.V., Pinto, L.M.: Multi-stage stochastic optimization applied to energy planning. Mathematical programming **52**(1-3), 359–375 (1991)
32. Philpott, A.B., Guan, Z.: On the convergence of stochastic dual dynamic programming and related methods. Operations Research Letters **36**(4), 450–455 (2008)
33. Rebennack, S.: Combining sampling-based and scenario-based nested benders decomposition methods: application to stochastic dual dynamic programming. Mathematical Programming **156**(1-2), 343–389 (2016)
34. Rogers, D.F., Plante, R.D., Wong, R.T., Evans, J.R.: Aggregation and disaggregation techniques and methodology in optimization. Operations Research **39**(4), 553–582 (1991)
35. Rosa, C.H., Takriti, S.: Improving aggregation bounds for two-stage stochastic programs. Operations research letters **24**(3), 127–137 (1999)
36. Shapiro, A.: Analysis of stochastic dual dynamic programming method. European Journal of Operational Research **209**(1), 63–72 (2011)
37. Shapiro, A., Dentcheva, D., Ruszczyński, A.: Lectures on Stochastic Programming: Modeling and Theory, Second Edition. Society for Industrial and Applied Mathematics, Philadelphia, PA (2014)
38. Shapiro, A., Tekaya, W., da Costa, J.P., Soares, M.P.: Risk neutral and risk averse stochastic dual dynamic programming method. European Journal of Operational Research **224**(2), 375–391 (2013)
39. van Slyke, R.M., Wets, R.: L-shaped linear programs with applications to optimal control and stochastic programming. SIAM Journal on Applied Mathematics **17**(4), 638–663 (1969)
40. Song, Y., Luedtke, J.: An adaptive partition-based approach for solving two-stage stochastic programs with fixed recourse. SIAM Journal on Optimization **25**(3), 1344–1367 (2015)
41. Trukhanov, S., Ntaimo, L., Schaefer, A.: Adaptive multicut aggregation for two-stage stochastic linear programs with recourse. European Journal of Operational Research **206**(2), 395–406 (2010)
42. Wittrock, R.J.: Advances in a nested decomposition algorithm for solving staircase linear programs. Tech. rep., Stanford Univ., CA (USA). Dept. of Operations Research (1984)

43. Wright, S.: Primal-dual aggregation and disaggregation for stochastic linear programs. Mathematics of Operations Research **19**(4), 893–908 (1994)
44. Zipkin, P.H.: Bounds for row-aggregation in linear programming. Operations Research **28**(4), 903–916 (1980)
45. Zipkin, P.H.: Bounds on the effect of aggregating variables in linear programs. Operations Research **28**(2), 403–418 (1980)

A Summary of Notation

Notation	Description
T	terminal stage in the planning horizon
t	a stage in the planning horizon and $t \in \{1,2,\ldots,T\}$
ξ_t	a random vector in stage $t \in \{1,2,\ldots,T\}$
$\xi_{[t]}$	the history of the random vector ξ_t up to stage t
Ξ_t	the support of the random vector ξ_t in stage t
N_t	set of the scenario indexes in stage t
p_t^k	probability of scenario $k \in \{1,2,\ldots,N_t\}$ in stage t
$\mathscr{N}_t$	a *partition* of the scenario set N_t in stage t
$\mathscr{N}$	a sequence of partitions for every stage $t \in 1,\ldots,T$
$\overrightarrow{\mathscr{N}}^p$	a partition constructed after taking p sample paths
L_t	the number of clusters in $\mathscr{N}_t$
$\mathscr{P}_t^\ell$	a scenario cluster $\mathscr{P}_t^\ell \subseteq \mathscr{N}_t, \forall\ j \in \{1,2,\ldots,L_t\}$ in stage t
ξ_t^ℓ	the realization of ξ_t chosen to represent cluster $\mathscr{P}_t^\ell$ in stage t
$\bar{p}_t^k$	the probability weight associated with the cluster $\mathscr{P}_t^\ell$ in stage t
$\mathscr{N}_t'$	refined partition of $\mathscr{N}_t$ in stage t
$\mathscr{P}_t'$	refined cluster of $\mathscr{P}_t$ in stage t
(c_t, B_t, A_t, b_t)	cost vector and coefficients parameters in stage t
$Q_t(\cdot)$	value function in stage t
$\underline{Q}_t(\cdot)$	a lower approximation for the value function $Q_t(\cdot)$ in stage t
$\overline{\mathfrak{Q}}_t(\cdot), \bar{\mathfrak{Q}}_t(\cdot)$	Expected cost-to-go function obtained by the scenario/partition-based formulation
$\check{\mathfrak{Q}}_t(\cdot), \check{\bar{\mathfrak{Q}}}_t(\cdot)$	Approximation of the expected cost-to-go function obtained by the scenario/partition-based formulation
$\check{\mathfrak{Q}}_t^p(\cdot), \check{\bar{\mathfrak{Q}}}_t^p(\cdot)$	Approximation of the expected cost-to-go function obtained by the scenario/partition-based formulation after p iterations
$\check{x}_t$	a candidate solution at stage t
Π	the dual feasible region of the subproblem in each stage
$\Pi_t^*(x_t, \xi_t)$	the set of optimal dual solutions to the stage t subproblem
π_t	optimal dual vector associated with the constraint $A_t x_t = b_t - B_t x_{t-1}$ at stage t
$q_t(\cdot), \bar{q}_t(\cdot)$	cutting/supporting hyperplane for $\mathfrak{Q}_t(\cdot), \bar{\mathfrak{Q}}_t(\cdot)$
J_t	collection of cutting/supporting hyperplanes for $\mathfrak{Q}_t(\cdot)$
r_t	the maximizer of the cutting/supporting hyperplanes in J_t at stage t
β_{t+1}, α_t	coefficients of the cutting/supporting $q_t(\cdot)$
$\bar{\beta}_{t+1}, \bar{\alpha}_t$	coefficients of the cutting/supporting $\bar{q}_t(\cdot)$
$\underline{z}$	lower bound for the optimal value of the optimal value $z(\cdot)$
$\hat{z}$	unbiased estimator for the upper bound of the optimal value of the optimal value $z(\cdot)$
$\bar{z}$	the sample average of the optimal value $z(\cdot)$
σ^2	the sample variance of the optimal value $z(\cdot)$
$\mathscr{S}$	set of sample paths (used for estimating upper bounds)
$\mathscr{T}$	a scenario tree with $\lvert\Xi_t\rvert$ realization of ξ_t at every stage t
$\bar{\mathscr{T}}(\mathscr{N})$	a coarse scenario tree induced by the sequence of partitions $\mathscr{N}$
ε	user-specified tolerance parameter
n	user-specified stability test parameter
ν	user-specified termination parameter used in refinement outside SDDP algorithms

Table 8 Summary of notations.

B Proof of Proposition 1

Proposition 2 *Convergence of the partition-based SDDP algorithms. Suppose that Assumptions 1–4 hold, sampling is done with replacement in the SDDP algorithm, and the employed partition refinement rule ensures*

that the number of clusters L_t of the refined partition strictly increases whenever $\check{\Omega}_t(\check{x}_{t-1}) \neq \mathbb{E}[\underline{Q}_t(\check{x}_{t-1}, \tilde{\xi}_t)]$ (see definitions in equations (16), (17), (22) and (25)) for any stage t unless $L_t = N_t$. Then w.p.1 after a finite number of forward and backward steps, the algorithm yields an optimal policy for (2).

Proof Let us first consider the trivial but sufficient partition with all singletons $\mathscr{N}_t^{\max} = \{\{k\}_{k\in N_t}\}$ and $\check{\Omega}_t^{\max}(\cdot)$ be its respective approximation of $\Omega_t(\cdot)$. By definition, we have that $\check{\Omega}_t^{\max}(\cdot) = \check{\Omega}_t(\cdot)$. Hence, if $\mathscr{N}_t = \mathscr{N}_t^{\max}$ for every stage $t = 2,\ldots,T$ then the *partition-based* variant of the SDDP algorithm is equivalent to the standard SDDP algorithm and the classical convergence proofs of SDDP apply (see, e.g., [36, Proposition 3.1]).

Let us now consider the situation where $\mathscr{N}_t \neq \mathscr{N}_t^{\max}$. Our goal is to show that for any arbitrary sample path $\xi^s = (\xi_2^{s_2}, \ldots, \xi_T^{s_T})$, the forward/backward pass of the adaptive partition-based SDDP algorithms will either yield a partition $\mathscr{N}_t = \mathscr{N}_t^{\max}$, or a sufficient partition such that $\check{\Omega}_t(\cdot) = \check{\Omega}_t(\cdot),\ \forall t = 2,\ldots,T$ along sample path ξ^s. In both cases, employing the adaptive partition-based SDDP algorithms over the sample path ξ^s is equivalent to the standard SDDP algorithm. We show this by using a backward induction argument as follows.

- Let $\check{x}_t = \check{x}_t(\xi_t^{s_t}), \forall t = 1,\ldots,T-1$ be the candidate solution induced by the approximate cost-to-go function $\check{\Omega}_t(\cdot)$ along ξ^s. At stage $t = T$ the candidate solution $\check{x}_{T-1}$ is evaluated at every *partition-based* subproblem (21) and by Lemma 1 we know that, if there exists an optimal $\check{\pi}_T^{\ell} \in \cap_{k\in\mathscr{P}_T^{\ell}} \Pi^*(\check{x}_{T-1}, \tilde{\xi}_T^k)$ then $\mathscr{N}_T$ is sufficient and $\check{\Omega}_T(\check{x}_{T-1}) = \check{\Omega}_T(\check{x}_{T-1})$. Otherwise, $\check{\Omega}_T(\check{x}_{T-1}) < \check{\Omega}_T(\check{x}_{T-1})$ and $\mathscr{N}_T$ needs to be refined. The case where $\check{\Omega}_T(\check{x}_{T-1}) = \check{\Omega}_T(\check{x}_{T-1})$ is degenerate and hence, we focus on the case where $\check{\Omega}_T(\check{x}_{T-1}) < \check{\Omega}_T(\check{x}_{T-1})$. After refining $\mathscr{N}_T$ using the absolute rule, a cut can be generated to the subproblem at stage $t = T-1$ using the refined partition $\mathscr{N}_T'$ whose size $L_t' > L_t$. Then an updated candidate solution $\check{x}_{T-1}'$ can be obtained by solving $\underline{Q}_{T-1}(\check{x}_{T-2}, \xi_{T-1}^s)$. If there still exists a positive gap between $\check{\Omega}_T'(\cdot)$ and $\check{\Omega}_T(\cdot)$ when evaluated at the updated candidate solution $\check{x}_{T-1}'$, the same process can be repeated again until $\check{\Omega}_T'(\cdot) = \check{\Omega}_T(\cdot)$ for all the candidate solutions encountered or $\mathscr{N}_T' = \mathscr{N}_T^{\max}$.
- Suppose now that $\check{\Omega}_t(\check{x}_{t-1}) = \check{\Omega}_t(\check{x}_{t-1})$ or $\mathscr{N}_t = \mathscr{N}_t^{\max}$ for all $t = T-1,\ldots,\tau+1$. At time $t = \tau$, similarly to the case when $t = T$, the candidate solution $\check{x}_{\tau-1}$ is evaluated at every *partition-based* subproblem (24) and if $\mathscr{N}_\tau$ is not sufficient, the absolute refinement procedure is followed until $\check{\Omega}_\tau'(\check{x}_{\tau-1}) = \check{\Omega}_\tau(\check{x}_{\tau-1})$ or $\mathscr{N}_\tau' = \mathscr{N}_\tau^{\max}$.

Since (by assumption) the sampling is done with replacement, we are guaranteed that each sample path ξ^s is sampled for infinitely many times with probability 1. Therefore, with probability 1, after a finite number of iterations, for any arbitrary sample path ξ^s, $\check{\Omega}_t(\cdot) = \check{\Omega}_t(\cdot)$ or $\mathscr{N}_t = \mathscr{N}_t^{\max}, \forall t = 2,\ldots,T$. At that point, employing the adaptive partition-based SDDP algorithms over sample path ξ^s is equivalent to employing the standard SDDP algorithm, and the convergence of the adaptive partition-based SDDP algorithms can be subsequently proved by applying the convergence proof of the standard SDDP algorithm. □