Noname manuscript No.
(will be inserted by the editor)

A Python package for multi-stage stochastic programming

Lingquan Ding · Shabbir Ahmed · Alexander Shapiro

Received: date / Accepted: date

Abstract This paper presents a Python package to solve multi-stage stochastic linear programs (MSLP) and multi-stage stochastic integer programs (MS-IP). Algorithms based on an extensive formulation and Stochastic Dual Dynamic (Integer) Programming (SDDP/SDDiP) method are implemented. The package is synthetically friendly and has a number of features which are not available in the competing software packages. In particular, the package deals with some of the restrictions on the underlying data process imposed by the previously available software packages. As an application of the package, three large-scale real-world problems - power system planning, portfolio optimization, airline revenue management, are discussed.

Keywords Python · Stochastic Dual Dynamic Programming · dynamic equations · Markov chain · Sample Average Approximation · risk averse · integer programming

1 Introduction

Since the publication of the pioneering paper by (Pereira & Pinto, 1991) on the Stochastic Dual Dynamic Programming (SDDP) method, considerable efforts have been made to apply/enhance the algorithm in both academia and industry. Especially in recent years, open-source software packages, including SDDP.jl (Dowson & Kapelevich, 2017), StructDualDynProg.jl (Legat, 2018),

H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332
Lingquan Ding
E-mail: lding47@gatech.edu
Shabbir Ahmed
E-mail: shabbir.ahmed@isye.gatech.edu
Alexander Shapiro (Research of this author was partly supported by NSF grant 1633196)
E-mail: ashapiro@isye.gatech.edu

StochDynamicProgramming.jl (Leclère, Gérard, Pacaud, & Rigaut, 2018), FAST (Cambier, 2018), StOpt (Gevret, Langrené, Lelong, Warin, & Maheshwari, 2018), based on the SDDP algorithm, have sprung up and are able to solve a wide variety of problems. A comprehensive comparison of these packages provided in (Dowson & Kapelevich, 2017) shows that SDDP.jl has the least restrictions on the underlying data process. It also could be mentioned that the SDDiP.jl (Kapelevich, 2018), which is an extension of SDDP.jl, implements the integer version of SDDP proposed in (Zou, Ahmed, & Sun, 2018).

All of the above mentioned packages are built for the case when the underlying data process is finite discrete. Their typical user input is a finite list of scenarios or a Markov chain, and all of the implementations and analysis are based on that assumption. However, quite often it is natural to model the data process as having a continuous distribution with an infinite number of possible realizations. (We refer to that as the 'true' problem). This leads to two implications. First, in order to use those packages, users themselves need to discretize the true process. This procedure is quite involved if the modeled process is Markovian. Second, these packages do not provide a solution and a way of analyzing the 'true' problem. This is somewhat disappointing since a poorly discretized problem may have a considerable deviation from the naturally modeled problem. When solving multistage stochastic integer programs (MSIP), binarization is sometimes made to make cuts tight, which further obscures the situation by bringing another layer of approximation. Users of these packages should be extremely cautious about what they are actually solving.

This paper presents the so-called MSPPy package which makes a step forward in removing these concerns. The layout of the paper is the following. In sections 2 to 10, we provide theoretical background for the MSPPy package. We start with mathematical formulation of multi-stage stochastic linear programs (MSLP). Sections 2 and 3 give nested formulation and dynamic equations; readers can be referred to (Shapiro, Dentcheva, & Ruszczyński, 2009) for a further discussion of these concepts. Section 4 introduces three approaches to the discretization. An important distinction between the true problem and its finite discretized problem is made. In section 6, an extensive solver and a SDDP solver are demonstrated. Despite the fact that the extensive solver suffers from exponential complexity and is even not able to yield sufficient information for the true problem, it is an important validation tool. Built upon the extensive solver, a rolling horizon based solver is discussed in section 7. The rolling horizon based solver and SDDP solver are able to produce both feasible solution and more information for the true problem. In section 8, we perform statistical analysis for the true problem. Extensions to risk averse problems and multi-stage stochastic integer programs (MSIP) are demonstrated in sections 9 and 10.

Starting from section 11, we present the MSPPy package. Section 11 to section 15 discuss design patterns of the package and various solvers to solve MSLP/MSIP. In section 16, we discuss three real world applications to demonstrate the capability of the MSPPy package. The hydro-thermal power system planning example in section 16.1 is originated from (Shapiro, Tekaya, da Costa,

& Soares, 2013). This problem has been studied in several publications and has produced many interesting results, for example (Shapiro et al., 2013), (Shapiro, Tekaya, da Costa, & Soares, 2011), (Shapiro, Tekaya, da Costa, & Soares, 2012). Recently (Löhndorf & Shapiro, 2019) made a comparison between the Time Series approach and Markov chain approximation to deal with the stage-wise dependent right hand side inflows of energy. Following that, we use the MSPPy package to implement both approaches and make comparisons on the true problem. A simple integer version of the problem is also discussed at the end of that section. The second example in section 16.2 is a multi-period portfolio optimization problem originated from (Dantzig & Infanger, 1993). In that paper, a three-stage problem with a finite stage-wise independent return process is analyzed. We illustrate by virtue of the MSPPy package, more sophisticated/realistic return process can be incorporated and analyzed. In addition, a risk management consideration can be easily taken into account. The third application in section 16.3 is taken from (Möller, Römisch, & Weber, 2008) in which the true demands are modeled as Markovian processes. In that paper, a scenario tree approximation of the demand processes is constructed and used to solve the problem. In the following publication (Zou et al., 2018), the problem is approximated by making stage-wise independent samplings from the true process and the corresponding Markovian structure is completely ignored. Anyways, neither of these approaches solve the true problem, and the suggested solutions may have a considerable bias. We show that by using our methodology, the problem can be solved and understood better. In section 17, we compare the MSPPy package with other existing open-source software packages. In particular, SDDP.jl is used as the benchmark of performance for the hydro-thermal power system problem. For the sake of performance comparison, we solve a stage-wise independent finite discrete version of the problem.

Readers are directed to the package website at `github.com/lingquant/msppy` for more examples and details of the MSPPy package.

2 Nested Formulation

We start with presenting the mathematical formulation of a multi-stage stochastic linear program. In multistage setting, the underlying data process is revealed sequentially. The data process is modeled as a stochastic process $(\xi_1, \dots, \xi_T)$ so that ξ_1 is deterministic and $\xi_2, \dots, \xi_T$ is to be revealed over time. We denote the history of the stochastic process up to time t by $\xi_{[t]} := (\xi_1, ..., \xi_t)$. In this package, we only consider stage-wise independent stochastic processes and Markovian processes (as it will be discussed later, in some cases stage-wise dependence can be reformulated in a stage-wise independent form). The stochastic process $\{\xi_t\}$ is called stage-wise independent if ξ_t is independent of $\xi_{[t-1]}$ for $t = 2, \dots, T$. The stochastic process $\{\xi_t\}$ is called Markovian if the conditional distribution of ξ_t given $\xi_{[t-1]}$ is the same as that of ξ_t given ξ_{t-1} for $t = 2, \dots, T$. To distinguish between these two types of stochastic processes, we denote stage-wise independent stochastic processes by $\{\xi_t\}$ and denote Markovian processes by $\{\eta_t\}$. The package also allows to model the data process as a combination of a stage-wise independent stochastic process $\{\xi_t\}$ and a Markovian process $\{\eta_t\}$ as long as $\{\xi_t\}$ and $\{\eta_t\}$ are independent.

Decision variables in each stage are categorized into state variables and control variables, denoted by x and y respectively. At the beginning of stage t (giving $\xi_{[t-1]}$), the state variable x_{t-1} is regarded as information that flows from the stage $t-1$ to the current stage. In the decision process, decision variables x_t, y_t $(t = 2, \dots, T)$ are considered to be functions $\mathbf{x}_t(\xi_{[t]}, \eta_{[t]}), \mathbf{y}_t(\xi_{[t]}, \eta_{[t]})$ of the stochastic process. The decision variables x_1, y_1 are deterministic and the initial value of state variable x_0 is supposed to be known. We call the sequence of decision variables $(x_1, y_1, \mathbf{x}_2, \mathbf{y}_2, \dots, \mathbf{x}_T, \mathbf{y}_T)$ as an implementable policy.

For illustration purpose, we only present minimization problem in this paper unless stated otherwise. The considered multistage stochastic linear program (MSLP) then takes the form of,

$$\min_{A_1x_1+B_1x_0+C_1y_1\geq b_1} u_1^\top x_1 + v_1^\top y_1 + \mathbb{E}_{|\xi_1,\eta_1}\Big[\min_{A_2x_2+B_2x_1+C_2y_2\geq b_2} u_2^\top x_2 + v_2^\top y_2$$
$$+ \cdots + \mathbb{E}_{|\xi_{[T-1]},\eta_{[T-1]}}\big[\min_{A_Tx_T+B_Tx_{T-1}+C_Ty_T\geq b_T} u_T^\top x_T + v_T^\top y_T\big]\Big],$$

in which $(u_1, v_1, A_1, B_1, C_1, b_1)$ is assumed to be known; ξ_t and η_t are assembled from $(u_t, v_t, A_t, B_t, C_t, b_t), t = 1, \dots, T$, some (or) all of which (except ξ_1 and η_1) are random and can be stage-wise independent or Markovian; $\mathbb{E}_{|\xi_{[t]},\eta_{[t]}}$ denotes the respective conditional expectation. In the rest of the paper, we use $(u_t, v_t, A_t, B_t, C_t, b_t)$ and (ξ_t, η_t) to denote uncertainties interchangeably. We will use the former one to emphasize the exact form/location of uncertainties while we will use the latter one to emphasize the type of uncertainties. Some of the feasibility constraints $A_tx_t + B_tx_{t-1} + C_ty_t \geq b_t$ can be equality.

For every feasible implementable policy (implementable policy satisfying all the feasibility constraints) and a sample path $(\xi_{[T]}, \eta_{[T]})$, we call $\sum_{t=1}^{T}[u_t^T x_t + v_t^T y_t]$ a (realized) policy value. We call $\mathbb{E}\left\{\sum_{t=1}^{T}[u_t^T x_t + v_t^T y_t]\right\}$ the expected policy value. The above MSLP is then equivalent to minimization of the expected policy value over all feasible implementable policies.

The package introduces additional control variables z_t acting as local copies of the past state variables x_{t-1}. As a consequence, the dynamic constraint $A_tx_t + B_tx_{t-1} + C_ty_t \geq b_t$ is simplified as $z_t = x_{t-1}$ and is added automatically by the software. Hence, users don't need to bother to specify interrelations between stages. As we will see it also simplifies computation of cutting planes in section 3 and it is beneficial for solving multistage stochastic integer programs

(MSIP) in section 10. The MSLP is then reformulated as,

$$\min_{\substack{A_1x_1+B_1z_1+C_1y_1\geq b_1\\ z_1=x_0}} u_1^\top x_1 + v_1^\top y_1 + \mathbb{E}_{|\xi_1,\eta_1}\Big[\min_{\substack{A_2x_2+B_2z_2+C_2y_2\geq b_2\\ z_2=x_1}} u_2^\top x_2 + v_2^\top y_2$$
$$+\cdots+\mathbb{E}_{|\xi_{[T-1]},\eta_{[T-1]}}\big[\min_{\substack{A_Tx_T+B_Tz_T+C_Ty_T\geq b_T\\ z_T=x_{T-1}}} u_T^\top x_T + v_T^\top y_T\big]\Big]$$

The package assumes the following relatively complete recourse condition. This assumption can always be satisfied by adding slack variables and penalty costs.

Assumption 1 The MSLP has relatively complete recourse. That is, for any realizations of random variables ξ_t, η_t, $t = 1, \ldots, T$, and any values of state variables x_t, there always exists control variables $y_1, \ldots, y_T$ such that $A_t x_t + B_t x_{t-1} + C_t y_t \geq b_t, t = 1, \ldots, T$.

The nested formulation provides only a mathematical description of the multistage stochastic program. The package does not solve the nested formulation directly. In later sections, we will see how the MSPPy package leverages dynamic equations and the extensive formulation to solve the problem.

3 Dynamic equations

Because of the assumed structure of the stochastic process, we can write the respective dynamic programming equations in the form as follows.

3.1 Stage-wise independent problems

Stage-wise independent problems assume that the underlying data process is a stage-wise independent stochastic process $\{\xi_t\}$. Then for $t = T, \ldots, 1$, the dynamic programming equations can be written as

$$Q_t(x_{t-1}, \xi_t) = \min_{\substack{A_tx_t+B_tz_t+C_ty_t\geq b_t\\ z_t=x_{t-1}}} u_t^\top x_t + v_t^\top y_t + \mathcal{Q}_{t+1}(x_t), \tag{1}$$

where the expected cost-to-go (recourse) functions,

$$\mathcal{Q}_{t+1}(x_t) := \begin{cases} \mathbb{E}\{Q_{t+1}(x_t, \xi_{t+1})\}, \text{ for } t = T-1, \ldots, 1, \\ 0, \text{ for } t = T, \end{cases} \tag{2}$$

do not depend on the underlying stochastic process. Note that if ξ_{t+1} has a discrete distribution with a finite number of possible realizations, the corresponding expectation in (2) can be written in a form of finite summation.

3.2 Markovian problems

Markovian problems assume that the underlying data process is a Markovian process $\{\eta_t\}$ or a combination of a Markovian process $\{\eta_t\}$ and a stage-wise independent process $\{\xi_t\}$ (independent from $\{\eta_t\}$). For $t = T, \ldots, 1$, the respective dynamic programming equations are

$$Q_t(x_{t-1}, \xi_t, \eta_t) = \min_{\substack{A_t x_t + B_t z_t + C_t y_t \geq b_t \\ z_t = x_{t-1}}} u_t^\top x_t + v_t^\top y_t + \mathcal{Q}_{t+1}(x_t, \eta_t), \tag{3}$$

where the expected cost-to-go functions,

$$\mathcal{Q}_{t+1}(x_t, \eta_t) := \begin{cases} \mathbb{E}_{|\eta_t}\{Q_{t+1}(x_t, \xi_{t+1}, \eta_{t+1})\}, \text{ for } t = T-1, \ldots, 1, \\ 0, \text{ for } t = T, \end{cases} \tag{4}$$

depend on η_t rather than the whole history $\eta_{[t-1]}$ of the Markov process. If the process $\eta_1, \ldots, \eta_T$, follows a (non-homogeneous) Markov chain with Markov state space $\{s_t | s_t \in S_t\}$, $t = 1, 2 \ldots, T$ (S_1 is a singleton) and transition matrix $\left\{\mathbf{P}^t_{s_{t-1}, s_t} | s_{t-1} \in S_{t-1}, s_t \in S_t\right\}$, $t = 2, \ldots, T$, the expected cost-to-go functions in (4) can be written as,

$$\mathcal{Q}_{t+1}(x_t, s_t) = \sum_{s_{t+1} \in S_{t+1}} \mathbf{P}^t_{s_t, s_{t+1}} \mathbb{E}\{Q_{t+1}(x_t, \xi_{t+1}, s_{t+1})\},$$

where $s_t \in S_t, t = T-1, \ldots, 1$. Again, if ξ_{t+1} is discrete with a finite number of possible realizations, the corresponding expectation can be written as a finite sum.

We end this section with a remark that the relatively complete recourse condition guarantees that $Q_t(x, \xi_t)$ ($Q_t(x, \xi_t, \eta_t)$) is finite for any x and realizations of the random data vectors. Moreover, since $Q_t(\cdot, \xi_t)$ ($Q_t(\cdot, \xi_t, \eta_t)$) is convex, it is subdifferentiable everywhere and its subgradients, at a point x, are given by optimal solutions of the dual problem corresponding to equation $z_t = x$.

4 Discretization

Stage-wise independent stochastic processes $\{\xi_t\}$ and Markov processes $\{\eta_t\}$ may have large/infinite number of possible realizations. Consequently, these processes should be discretized in order to make the problem solvable. Before we discuss certain discretization techniques, we introduce the following two terms.

The original problem (1) or (3) is referred to as the *true problem*. The true problem is *continuous* if the underlying stochastic process follows a continuous distribution. The true problem is *discrete* if the underlying stochastic process follows a discrete distribution. It is further categorized as a *finite (infinite) discrete* true problem if the discrete distribution has finite (infinite) possible values. Continuous and infinite discrete true problems are not amenable for a finite representation in a computer and should be discretized. A finite discrete true problem may be computationally intractable when the discrete distribution has a large number of possible values.

A discretization of the true problem is called a *discretized problem.* As we will discuss in this section, stage-wise independent processes can be discretized by the Sample Average Approximation (SAA) and Markov processes can be discretized by the SAA through a Time Series approach (TS) or by Markov chain approximation (MCA). We refer to the discretized problem as an SAA discretized problem or a Markov chain discretized problem correspondingly.

We would like to mention that though the discretized problem could be solvable, our real interest is still in the true problem rather than the discretized problem. Without careful analysis, an optimal policy generated by the discretized problem does not give a guarantee about its performance for the true problem. It may be even not feasible to implement for the true problem.

4.1 Sample Average Approximation (SAA) approach

Sample average approximation (SAA) generates independent identically distributed (i.i.d.) samples from the true distribution and uses sample average to approximate the expectation. For the state-wise independent stochastic process $\{\xi_t\}$, it is natural to discretize marginal distributions of random vectors ξ_t.

4.2 Time Series (TS) approach

Sometimes the Markovian process $\{\eta_t\}$ can be modeled using the following time series equations (higher order time series models can be transformed to first order by adding auxiliary variables),

$$\eta_t = \mu_t + \Phi_t \eta_{t-1} + a_t \text{ (VAR)}, \tag{5}$$

$$\eta_t = (\mu_t + \Phi_t \eta_{t-1}) a_t \text{ (VAR*)}, \tag{6}$$

$$\ln \eta_t = \mu_t + \Phi_t \ln \eta_{t-1} + a_t \text{ (VGAR)}, \tag{7}$$

where μ_t, a_t are m dimensional vectors, Φ_t is an $m \times m$ coefficient matrix and a_t is a white noise vector process with zero mean in (5) and (7) and mean 1 in (6); the multiplication in (6) is understood pointwise.

If η_t appears on the right hand side of constraints and follows VAR/VAR* models, users can add η_t as additional state variables to the true problem. The reformulated true problem, with a_t viewed as the underlying random data process, is stage-wise independent and thus can be discretized by SAA. On the other hand, if η_t follows VGAR model, or it appears in the objective or left hand side of constraints, adding η_t as additional state variables will destroy the linearity of the true problem. Cases like that should resort to Markov chain approximation instead.

4.3 Markov chain approximation (MCA)

Another more general approach to discretize the Markov process $\{\eta_t\}$ is by Markov chain approximation (MCA). Suppose the sample space of the $\{\eta_t\}$ at stage t is Ω_t (η_1 is known and Ω_1 is a singleton). Suppose further that F is the joint distribution of $\eta_1 \ldots, \eta_T$. From stage two on, we are going to partition Ω_t into K_t subsets $\{\omega_{tk}, k = 1, \ldots, K_t\}$ centered at μ_{tk} and let the centers μ_{tk} to be the Markov states. Such partition is known as the *Voronoi cells.* Let U_t denote the partition $\{\omega_{tk}, k = 1, \ldots, K_t\}$ at stage t and U denote the whole partition $\{U_t, t = 1, \ldots, T\}$. Our objective is to search for a partition U that minimizes the Euclidean distance from the centers (Markov states) and the Markov process $\{\eta_t\}$,

$$\min_U \ \mathbb{E}_\eta \left\{ \sum_{t=2}^{T} \min_{k=1,\ldots,K_t} \|\eta_t - \mu_{tk}\|_2^2 \right\}. \tag{8}$$

The MSPPy package implements three methods to solve the problem.

The first approach is by virtue of the *Sample Average Approximation* (SAA). We draw S independent sample paths $\{\hat\eta_1^s, \ldots, \hat\eta_T^s\}, s = 1, \ldots, S$ from F and construct an SAA of (8),

$$\min_U \sum_{s=1}^{S} \sum_{t=2}^{T} \min_{k=1,\ldots,K_t} \|\hat\eta_t^s - \mu_{tk}\|_2^2,$$

which is equivalent to,

$$\min_U \sum_{t=2}^{T} \sum_{s=1}^{S} \min_{k=1,\ldots,K_t} \|\hat\eta_t^s - \mu_{tk}\|_2^2 = \sum_{t=2}^{T} \min_{U_t} \sum_{s=1}^{S} \min_{k=1,\ldots,K_t} \|\hat\eta_t^s - \mu_{tk}\|_2^2.$$

The summands $\min_{U_t} \sum_{s=1}^{S} \min_{k=1,\ldots,K_t} \|\hat\eta_t^s - \mu_{tk}\|_2^2$ are identical to the objective of a K-means problem. We can then use the Lloyd's algorithm (Lloyd, 1982) to find a local optimum.

Another approach is using the *Stochastic Approximation* (SA) algorithm. Initialize μ_{tk} as μ_{tk}^0, $k = 1, \ldots, K_t$. With a sequence of stepsizes $(\beta_s)_{s=1}^S$, the Markov states are updated recursively by stochastic approximation,

$$\mu_{ti}^s = \begin{cases} \mu_{ti}^{s-1} + \beta_s(\hat\eta_t^s - \mu_{ti}^{s-1}), & \text{if } i = \text{argmin}_{k\in\{1,\ldots,K_t\}}\{\|\hat\eta_t^s - \mu_{tk}^{s-1}\|_2^2\} \\ \mu_{ti}^{s-1}, & \text{otherwise.} \end{cases}$$

We can for example, set a diminishing step size sequence $\beta_s = 1/s, s = 1, \ldots, S$, to ensure that it converges to a local minimum (Bally & Pages, 2003).

The third approach is utilizing the *Robust Stochastic Approximation* (RSA) approach, originally developed for convex stochastic problems (Nemirovski, Juditsky, Lan, & Shapiro, 2009). Given a specified number of iterations N, a constant step size $\frac{c}{\sqrt{N}}$ is used (c is a constant) and the final approximating solution is the average of the iterates. This algorithm has $O(N^{-\frac{1}{2}})$ rate of convergence. In our case, the objective function is not convex and thus the convergence is not guaranteed. Nevertheless, it appears to work well in practice. We attach pseudo code in Algorithm 1.

Algorithm 1 Markov chain approximation (MCA) through Robust Stochastic Approximation (RSA)

```
 1: Given a sample path generator $F$, intended number of Markov states $K_t$, $t = 1,\dots,T (K_1 = 1)$, intended number of training sample paths $N$.
 2: Randomly initialize $\mu^0_{tk}, k \in \{1,\dots,K_t\}, t = 2,\dots,T$
 3: for $s = 1,\dots,S$ do
 4:     Draw one sample path $\{\hat{\eta}_1, \hat{\eta}_2, \dots, \hat{\eta}_T\}$ from $F$
 5:     for $t = 2,\dots,T$ do
 6:         $m = \text{argmin}_k \{\|\hat{\eta}_t - \mu^{s-1}_{tk}\|^2_2, k = 1,\dots,K_t\}$
 7:         Update $\mu^s_{ti} = \begin{cases} \mu^{s-1}_{ti} + \frac{1}{\sqrt{N}}(\hat{\eta}_t - \mu^{s-1}_{ti}), & \text{if } i = m \\ \mu^{s-1}_{ti}, & \text{if } i \in \{1,\dots,K_t\} \setminus m \end{cases}$
 8:     end for
 9: end for
10: Set $\mu_{tk} = \frac{1}{S}\sum_{s=1}^S \mu^s_{tk}, k = 1,\dots,K_t, t = 2,\dots,T$.
```

Finally, the transition matrix is approximated by relative frequencies,

$$\mathbf{P}^t_{ij} = \frac{\sum_{s=1}^S 1_{\hat{\eta}^s_{t-1} \in \omega_{t-1,i}} 1_{\hat{\eta}^s_t \in \omega_{tj}}}{\sum_{s=1}^S 1_{\hat{\eta}^s_{t-1} \in \omega_{t-1,i}}}, t = 2,\dots,T.$$

Where $1_{(\cdot)}$ is the indicator function.

There is no clear evidence showing that one approach is superior to the others. We will empirically compare the three methods in section 16.1.

As above, for $t = 2,\dots,T$, the SAA discretizes ξ_t by generating i.i.d. samples $\xi^j_t, j = 1,\dots,N_t$, and MCA discretizes $\{\eta_t\}_{t=1}^T$ by a Markov chain with Markov state spaces $\{s_t | s_t \in S_t\}$, $t = 1,\dots,T$, and transition matrices $\{\mathbf{P}^t_{s_{t-1},s_t} | s_{t-1} \in S_{t-1}, s_t \in S_t\}$, $t = 2,\dots,T$. The SAA discretization of the true problem (1) takes the form of,

$$\begin{aligned} \tilde{Q}_t(x_{t-1}, \xi^j_t) &= \min_{\substack{A_{tj}x_t + B_{tj}z_t + C_{tj}y_t \geq b_{tj} \\ z_t = x_{t-1}}} u_{tj}^\top x_t + v_{tj}^\top y_t + \tilde{\mathcal{Q}}_{t+1}(x_t), \\ \tilde{Q}_1(x_0, \xi_1) &= \min_{\substack{A_1x_1 + B_1z_1 + C_1y_1 \geq b_1 \\ z_1 = x_0}} u_1^\top x_1 + v_1^\top y_1 + \tilde{\mathcal{Q}}_2(x_1), \end{aligned} \tag{9}$$

where $j = 1,\dots,J_t$, $t = T,\dots,2$, and the approximate expected cost-to-go is defined as

$$\tilde{\mathcal{Q}}_{t+1}(x_t) := \begin{cases} \dfrac{1}{J_{t+1}} \displaystyle\sum_{j=1}^{J_{t+1}} \tilde{Q}_{t+1}(x_t, \xi^j_{t+1}), \text{ for } t \neq T, \\ 0, \text{ for } t = T. \end{cases}$$

The Markov chain discretization of the true problem (3) takes the form of

$$\begin{aligned}\tilde{Q}_t(x_{t-1}, \xi_t^j, s_t) &= \min_{\substack{A_{tj}x_t + B_{tj}z_t + C_{tj}y_t \geq b_{tj} \\ z_t = x_{t-1}}} u_{tj}^\top x_t + v_{tj}^\top y_t + \tilde{\mathfrak{Q}}_{t+1}(x_t, s_t), \\ \tilde{Q}_1(x_0, \xi_1, s_1) &= \min_{\substack{A_1x_1 + B_1z_1 + C_1y_1 \geq b_1 \\ z_1 = x_0}} u_1^\top x_1 + v_1^\top y_1 + \tilde{\mathfrak{Q}}_2(x_1, s_1),\end{aligned} \tag{10}$$

where $j = 1, \ldots, J_t, s_t \in S_t$, $t = T, \ldots, 2$, and the approximate expected cost-to-go is defined as

$$\tilde{\mathfrak{Q}}_{t+1}(x_t, s_t) := \begin{cases} \dfrac{1}{J_{t+1}} \displaystyle\sum_{s_{t+1} \in S_{t+1}} \sum_{j=1}^{J_{t+1}} \mathbf{P}_{s_t, s_{t+1}}^t \tilde{Q}_{t+1}(x_t, \xi_{t+1}^j, s_{t+1}), \text{ for } t \neq T, \\ 0, \text{ for } t = T. \end{cases}$$

5 Extensive formulation of the discretized problem

We can formulate the discretized problem equivalently as a deterministic linear program. We refer to the equivalent deterministic linear program as *Extensive Formulation.* Let $\{n | n \in \tau\}$ be the set of nodes in a constructed scenario tree. Each node n at stage t corresponds to a specific realization of data process $(u_n, v_n, A_n, B_n, C_n, b_n)$ at stage t and also corresponds to a sample path up to stage t. We denote the ancestor of node n by $a(n)$. We also denote the probability of sample path going through node n by P_n. For each $n \in \tau$, we associate a single decision variable (x_n, y_n). In this way the nonanticipativity constraints automatically satisfy. The extensive formulation then takes the form of,

$$\begin{aligned}&\min \sum_{n \in \tau} P_n(u_n^\top x_n + v_n^\top y_n) \\ &\text{s.t. } A_n x_n + B_n y_n + C_n x_{a(n)} \geq b_n.\end{aligned}$$

Solving the extensive formulation provides us with an implementable policy only for the considered scenario tree. As it was mentioned before, the computed optimal policy does not provide an implementable policy for the true problem, at least not in a direct way. Moreover the number of required scenarios in a constructed scenario tree grows exponentially with increase of the number of stages. Nevertheless this approach could be applicable when the number of stages is small (say not larger than 3) and could be useful for verification of a considered model.

The scenario tree can be either stage-wise independent or Markovian. The computation of the probability P_n is straightforward as follows. Firstly, suppose the discretized problem is SAA. For each stage t $(t = 1, \ldots, T)$, denote stage-wise independent scenarios by $\{j_t | j_t \in 1, \ldots, N_t\}$ $(J_1 = 1)$ and related probability measures by $\{P_{j_t}^t | j_t \in 1, \ldots, N_t\}$, $t = 1, \ldots, T$ $(P_{j_1}^1 = 1)$. Consider node n at stage t and its corresponding sample path $(j_1, j_2, \dots, j_t)$. Then P_n is given by

$$P_n = P^1_{j_1} P^2_{j_2} \dots P^t_{j_t}.$$

Secondly, suppose the discretized problem is Markov chain type and has only the Markov chain stochastic process. Let the Markov state space at stage t be S_t (initial state space S_1 is a singleton) and the transition matrix between stage $t-1$ and stage t be $\{\mathbf{P}^t_{s_{t-1},s_t} | s_{t-1} \in S_{t-1}, s_t \in S_t\}$, $t = 2, \dots, T$. Consider node n at stage t and its corresponding sample path $(s_1, s_2, \dots, s_t)$. Then P_n is given by

$$P_n = \mathbf{P}^2_{s_1,s_2} \mathbf{P}^3_{s_2,s_3} \dots \mathbf{P}^t_{s_{t-1},s_t}.$$

Finally, suppose that the Markov chain discretized problem also has the stage-wise independent stochastic process (independent of the Markov chain process). Then we can combine the two corresponding scenario trees into one single scenario tree with P_n given by

$$P_n = P^1_{j_1} P^2_{j_2} \dots P^t_{j_t} \mathbf{P}^2_{s_1,s_2} \mathbf{P}^3_{s_2,s_3} \dots \mathbf{P}^t_{s_{t-1},s_t}.$$

6 Solving the discretized problem

The MSPPy package provides two solvers to solve the discretized problem.

6.1 Extensive solver

The extensive solver uses the extensive formulation and solves a single deterministic, although may be large, linear program. As it was already mentioned, the number of variables and constraints in the extensive formulation grows exponentially with increase of the number of stages. So it suffers from exponential complexity and is only able to solve relatively small problems. In addition, as mentioned above, the solution provided by the extensive formulation is only implementable for the discretized problem. It is unclear what such solution means to the true problem. Nevertheless, it could be useful for small scale problems and could be used as a validation tool for the correctness of other solvers.

6.2 SDDP solver

The SDDP algorithm (Pereira & Pinto, 1991) leverages convexity of the value functions and appears to enjoy more or less linear complexity with respect to the number of stages (this is an empirical observation). Furthermore, the SDDP algorithm provides feasible implementable policies not only for the discretized problem, but also for the true problem. This is beneficial since solving the true problem is of our primal interest.

6.2.1 stage-wise independent SDDP

The SAA discretized problem (9) is solved by the classical SDDP algorithm as shown in Algorithm 2.

Algorithm 2 Stage-wise independent SDDP

```
1: Given discretization $\Omega_t = \{u_{tj}, v_{tj}, A_{tj}, B_{tj}, C_{tj}, b_{tj}\}_{1\le j\le N_t}, t = 2,\dots,T$
2: Given initial value $\bar{x}_0$
3: Given initial approximation of value functions : $\mathfrak{Q}_t^0(\cdot) = \{\theta : \theta(\cdot) \ge l_t\}, t = 2,\dots,T$
4: Initialize: i = 1, LB = -∞
5: while no stopping criterion is met do
     (Forward Step)
6:   for $t = 1,\dots,T$ do
7:     if $t > 1$ then draw a sample $(u_t, v_t, A_t, B_t, C_t, b_t)$ from $\Omega_t$
8:     end if
9:     $\bar{x}_t, \bar{y}_t =$ argmin $\{u_t^\top x_t + v_t^\top y_t + \mathfrak{Q}_{t+1}^{i-1}(x_t) : A_t x_t + B_t z_t + C_t y_t \ge b_t, z_t = \bar{x}_{t-1}\}$
10:  end for
     (Backward Step)
11:  for $t = T,\dots,2$ do
12:    for $j = 1,\dots,N_t$ do
13:      Solve min $\{u_{tj}^\top x_t + v_{tj}^\top y_t + \mathfrak{Q}_{t+1}^i(x_t) : A_{tj} x_t + B_{tj} z_t + C_{tj} y_t \ge b_{tj}, z_t = \bar{x}_{t-1}\}$
         and get the optimal value $\mathscr{V}_{tj}$ and a dual solution $\mathscr{G}_{tj}$ corresponding to $z_t = \bar{x}_{t-1}$
14:    end for
15:    $\mathscr{V}_t := \frac{1}{N_t}\sum_{j=1}^{N_t} \mathscr{V}_{tj}, \mathscr{G}_t := \frac{1}{N_t}\sum_{j=1}^{N_t} \mathscr{G}_{tj}$
16:    $\mathfrak{Q}_t^i \leftarrow \{\theta \in \mathfrak{Q}_t^{i-1}, \theta(x) \ge \mathscr{G}_t(x - \bar{x}_{t-1}) + \mathscr{V}_t\}$
17:  end for
18:  Policy value $= \sum_{t=1}^T [u_t^\top \bar{x}_t + v_t^\top \bar{y}_t]$
19:  LB = min $\{u_1^\top x_1 + v_1^\top y_1 + \mathfrak{Q}_2^i(x_1) : A_1 x_1 + B_1 z_1 + C_1 y_1 \ge b_1, z_1 = \bar{x}_0\}$
20:  $i = i + 1$
21: end while
```

The computed approximations $\mathfrak{Q}_2(\cdot),\dots,\mathfrak{Q}_T(\cdot)$ of the value functions and a feasible first stage solution $\bar{x}_1$ generates a feasible implementable policy $x_t(\xi_{[t]})$ for the discretized as well as the true problem as follows. For any realization (sample path) of the process $\{u_t, v_t, A_t, B_t, C_t, b_t\}$, $t = 2,\dots,T$, we associate a decision vector $x_1(= \bar{x}_1), x_2,\dots,x_T$, generated by solving recursively

$$\min \{u_t^\top x_t + v_t^\top y_t + \mathfrak{Q}_{t+1}(x_t) : A_t x_t + B_t z_t + C_t y_t \ge b_t, z_t = x_{t-1}\}, t = 2,\dots,T.$$

By the assumption of relatively complete recourse, it generates a feasible implementable policy for the considered problem. If we restrict sample paths $\xi_{[t]}$ to the discretized process, the implied policy $x_t(\xi_{[t]})$ is feasible and implementable for the discretized problem. On the other hand, if we consider the sample paths $\xi_{[t]}$ generated from the distribution of the true problem, the policy $x_t(\xi_{[t]})$ is feasible and implementable for the true problem. As a result, the expected value $\mathbb{E}\left\{\sum_{t=1}^T [u_t^\top x_t + v_t^\top y_t]\right\}$ gives an upper bound for the optimal value of the true or the discretized problem depending on what sample paths are considered.

6.2.2 Markov chain SDDP

The Markov chain discretized problem (10) can be solved by a Markov chain version of SDDP algorithm. For instance, Algorithm 3 solves a minimization problem in which $\{u_t\}, \{v_t\}, \{A_t\}, \{B_t\}, \{C_t\}$ are stage-wise independent stochastic processes and $\{b_t\}$ is a Markov chain process,

Algorithm 3 Markov chain SDDP

```
 1: Given discretization $\Omega_t = \{u_{tj}, v_{tj}, A_{tj}, B_{tj}, C_{tj}\}_{1\le j\le N_t}$, state space of Markov chain
    $S_t$, transition matrix $\{\mathbf{P}_{s_{t-1},s_t} | s_{t-1} \in S_{t-1}, s_t \in S_t\}$, $t = 2, \ldots, T$, initial state $s_1$
 2: Given initial approximation of the value functions:
 3: $\mathfrak{Q}_t^0(\cdot, s_{t-1}) = \{\theta(\cdot, s_{t-1}) | \theta(\cdot, s_{t-1}) \ge L_0\}, \forall s_{t-1}$
 4: Initialize: i = 1, LB = -$\infty$
 5: while no stopping criterion is met do
      (Forward Step)
 6:   for $t = 1, \ldots, T$ do
 7:     if $t > 1$ then draw a sample $(u_t, v_t, A_t, B_t, C_t)$ from $\Omega_t$; draw a state $s_t$ w.p.
        $\mathbf{P}_{s_{t-1},s_t}$
 8:     end if
 9:     $\bar{x}_t, \bar{y}_t = \text{argmin}\left\{u_t^\top x_t + v_t^\top y_t + \mathfrak{Q}_{t+1}^{i-1}(x_t, s_t) : A_t x_t + B_t z_t + C_t y_t \ge s_t, z_t = \bar{x}_{t-1}\right\}$
10:   end for
      (Backward Step)
11:   for $t = T, \ldots, 2$ do
12:     for $j = 1, \ldots, N_t$ do
13:       for $s_t \in S_t$ do
14:         Solve min $\left\{u_{tj}^\top x_t + v_{tj}^\top y_t + \mathfrak{Q}_{t+1}^i(x_t, s_t) : A_{tj} x_t + B_{tj} z_t + C_{tj} y_t \ge s_t, z_t = \bar{x}_{t-1}\right\}$
    and get the optimal value $\mathscr{V}_{tj,s_t}$ and a dual solution $\mathscr{G}_{tj,s_t}$ corresponding to $z_t = \bar{x}_{t-1}$
15:       end for
16:     end for
17:     for $s_{t-1} \in S_{t-1}$ do
18:       $\mathscr{V}_{t,s_{t-1}} := \frac{1}{N_t}\sum_{j=1}^{N_t}\sum_{s_t \in S_t}[\mathbf{P}_{s_{t-1},s_t}\mathscr{V}_{tj,s_t}]$,
19:       $\mathscr{G}_{t,s_{t-1}} := \frac{1}{N_t}\sum_{j=1}^{N_t}\sum_{s_t \in S_t}[\mathbf{P}_{s_{t-1},s_t}\mathscr{G}_{tj,s_t}]$
20:       $\mathfrak{Q}_t^i(\cdot, s_{t-1}) \leftarrow \left\{\theta \in \mathfrak{Q}_t^{i-1}(\cdot, s_{t-1}), \theta(x, s_{t-1}) \ge \mathscr{G}_{t,s_{t-1}}(x - \bar{x}_{t-1}) + \mathscr{V}_{t,s_{t-1}}\right\}$
21:     end for
22:   end for
23:   Policy value = $\sum_{t=1}^T \left[u_t^\top \bar{x}_t + v_t^\top \bar{y}_t\right]$
24:   LB = min $\left\{u_1^\top x_1 + v_1^\top y_1 + \mathfrak{Q}_2(x_1, s_1) : A_1 x_1 + B_1 z_1 + C_1 y_1 \ge s_1, z_1 = x_0\right\}$
25:   $i = i + 1$
26: end while
```

The computed approximations $\mathfrak{Q}_2(\cdot, s_1), \ldots, \mathfrak{Q}_T(\cdot, s_{T-1})$, $s_1 \in S_1, \ldots, s_{T-1} \in S_{T-1}$ and a feasible first stage solution $\bar{x}_1$ can generate a feasible implementable policy $x_t(\xi_{[t]}, \eta_{[t]})$ for the true problem as follows. For any realization (sample path) of the process $\{u_t, v_t, A_t, B_t, C_t, b_t\}$, $t = 2, \ldots, T$, assume s_t^* is the closest Markov state to b_t, i.e., $s_t^* = \text{argmin}_{s_t}\{\|s_t - b_t\|_2^2, s_t \in S_t\}$. We then associate a decision vector $x_1 (= \bar{x}_1), x_2, \ldots, x_T$, generated by solving recursively min $\{u_t^\top x_t + v_t^\top y_t + \mathfrak{Q}_{t+1}(x_t, s_t^*) : A_t x_t + B_t z_t + C_t y_t \ge s_t^*, z_t = x_{t-1}\}, t = 2, \ldots, T$. By Assumption 1, it is a feasible implementable policy for the true problem. Again, the expected policy value $\mathbb{E}\left\{\sum_{t=1}^T [u_t^\top x_t + v_t^\top y_t]\right\}$

gives an upper bound for the true or discretized problem depending on what sample paths are considered.

6.3 Stopping criteria

The MSPPy package provides three kinds of stopping criteria of the SDDP algorithm. The first criterion is an upper limit of certain characteristics including time, the number of iterations and the number of iterations that the lower bound doesn't change. The algorithm will be stopped if one of the upper limits is reached.

The second criterion is based on estimation of the optimality gap (Shapiro, 2011). The MSPPy package detects this by evaluating value of the constructed policy on the discretized problem employing Markov Carlo simulation. Users can specify the frequency of such evaluations and the number of simulations M. In each simulation, a sample path $(\xi^m_{[t]}, \eta^m_{[t]})$ is generated independently from the discretized problem, leading to an optimal solution $\bar{x}_{tm}(\xi^m_{[t]}, \eta^m_{[t]})$, $\bar{y}_{tm}(\xi^m_{[t]}, \eta^m_{[t]})$ and policy values

$$V_m = \sum_{t=1}^{T} \left[u^T_{tm}\bar{x}_{tm} + v^T_{tm}\bar{y}_{tm} \right],\ m = 1, \dots, M.$$

Define sample mean as $\bar{V}_M := M^{-1}\sum_{m=1}^{M} V_m$ and sample variance $S^2_M := \frac{1}{M-1}\sum_{m=1}^{M}(V_m - \bar{V}_M)^2$. Let v^* be the expected value of the computed policy. By the CLT we have that $\frac{\bar{V}_M - v^*}{S_M/\sqrt{M}}$ converges in distribution to standard normal $\mathcal{N}(0, 1)$. It follows that the approximate $(1 - \alpha)$ one-sided confidence interval for the expected policy values is $\bar{V}_M + z_{1-\alpha}\frac{S_M}{\sqrt{M}}$, where $\mathbb{P}(Z \geq z_{1-\alpha}) = \alpha$ with $Z \sim \mathcal{N}(0, 1)$. The upper end of the CI provides an upper bound of the optimal value of the discretized problem with confidence of approximately $1-\alpha$. Hence, with confidence $1-\alpha$, the optimality gap for the discretized problem is smaller than

$$gap = \frac{\bar{V}_M + z_{1-\alpha}\frac{S_M}{\sqrt{M}} - \text{LB}}{\text{LB}}, \tag{11}$$

where LB is the lower bound obtained from the backward step. Users can specify an intended tolerance ε for the optimality gap to stop the algorithm. If $gap \leq \varepsilon$, then users can be $(1 - \alpha)$ confident that optimality gap for the discretized problem is not larger than ε. Note that the estimate (11) of the optimality gap is somewhat conservative. Its conservativeness depends on how good is the lower bound estimate and how large is the standard deviation $S_M/\sqrt{M}$.

The third criterion is based on stabilization of the expected policy value. If additional iterations provide little improvement of expected policy values, it is reasonable to stop the algorithm. The MSPPy package compares a recent policy $\hat{x}_t$ with a past policy $\bar{x}_t$ by independently generating M sample paths. For each generated sample path $(\xi^m_{[t]}, \eta^m_{[t]})$, we obtain an optimal solution $\bar{x}_t(\xi^m_{[t]}, \eta^m_{[t]}), \bar{y}_t(\xi^m_{[t]}, \eta^m_{[t]})$ and $\hat{x}_t(\xi^m_{[t]}, \eta^m_{[t]}), \hat{y}_t(\xi^m_{[t]}, \eta^m_{[t]})$. Consequently the differences of policy values

$$D_m := \sum_{t=1}^{T} \left[u_t^\top \bar{x}_t + v_t^\top \bar{y}_t - u_t^\top \hat{x}_t - v_t^\top \hat{y}_t \right],\ m = 1, ..., M,$$

are computed.

Note that D_m is an unbiased estimator of the difference D of the two expected policy values. Similarly as above, the approximate $(1-\alpha)$ one-sided confidence interval for the difference D is given by $\bar{D}_M + z_{1-\alpha}\frac{S_M}{\sqrt{M}}$, where $\bar{D}_M$ is the sample average and S_M^2 is the sample variance of $D_1, ..., D_M$. Users can specify an intended threshold of difference ϵ_D to stop the algorithm. If $\bar{D}_M + z_{1-\alpha}\frac{S_M}{\sqrt{M}} \leq \epsilon_D$, then users can be $(1-\alpha)$ confident that improvement of the expected policy value is not larger than ϵ_D.

6.4 Parallelization

In this section, we will see how the SDDP algorithm can be parallelized in the stage-wise independent setting. Parallelizations in other cases are similar. As shown in Algorithm 4, it does M forward steps and backward steps for each iteration. These M jobs are independent and thus can be allocated to multiple processes.

It is worth noting that this approach has a subtle difference from the serial SDDP. In the serial SDDP, cuts are added for every single forward and backward step. While in the parallel SDDP, cuts are added every M forward and backward steps. Thus given the same number of forward and backward steps, the parallel SDDP will be less accurate than the serial SDDP.

7 Rolling horizon based algorithm

In sections 5 and 6.1, we introduce an algorithm based on the extensive formulation and discuss its two shortcomings. That is, it does not provide implementable policies for the true problem and it suffers from exponential complexity with respect to the number of stages. In this section, we develop a variant of this algorithm that deals with these shortcomings.

For any realization (sample path) of a general (stage-wise independent, Markovian, or non-Markovian) process $\hat{\xi}_{[T]}$, i.e., $\{u_t, v_t, A_t, B_t, C_t, b_t\}$, for $t = 2, \ldots, T$ (we omit the tilde for ease of notation), we ought to associate a single decision vector $x_1, \ldots, x_T$, generated by solving recursively,

$$\begin{aligned} \min\ & \mathbb{E}_{|\hat{\xi}_{[t]}} \left\{ \sum_{i=t}^{T} \left[u_i^\top x_i + v_i^\top y_i \right] \right\} \\ \text{s.t.}\ & A_t x_t + B_t z_t + C_t y_t \geq b_t, z_t = x_{t-1} \end{aligned}$$

Algorithm 4 Parallel stage-wise independent SDDP

```
 1: Given discretization $\Omega_t = \{u_{tj}, v_{tj}, A_{tj}, B_{tj}, C_{tj}, b_{tj}\}_{1\le j\le N_t}, t = 2,\dots,T$
 2: Given initial approximation of value functions : $\mathfrak{Q}_t^0(\cdot) = \{\theta : \theta(\cdot) \ge l_t\}, t = 2,\dots,T$
 3: Initialize: i = 1, LB = -∞
 4: while no stopping criterion is met do
 5:   for $m = 1,\dots,M$ do
        (Forward Step)
 6:     for $t = 1,\dots,T$ do
 7:       if $t > 1$ then draw a sample $(u_t, v_t, A_t, B_t, C_t, b_t)$ from $\Omega_t$
 8:       end if
 9:       $\bar{x}_t^m, \bar{y}_t^m = \text{argmin}\ \{u_t^\top x_t + v_t^\top y_t + \mathfrak{Q}_{t+1}^{i-1}(x_t) : A_t x_t + B_t z_t + C_t y_t \ge b_t, z_t = \bar{x}_{t-1}^m\}$
10:     end for
11:     $V_m = \sum_{t=1}^T [u_t^\top x_t^m + v_t^\top y_t^m]$
        (Backward Step)
12:     for $t = T,\dots,2$ do
13:       for $j = 1,\dots,N_t$ do
14:         Solve min $\{u_{tj}^\top x_t + v_{tj}^\top y_t + \mathfrak{Q}_{t+1}^i(x_t) : A_{tj} x_t + B_{tj} z_t + C_{tj} y_t \ge b_{tj}, z_t = \bar{x}_{t-1}\}$
    and get the optimal value $\mathscr{V}_{tj}^m$ and a dual solution $\mathscr{G}_{tj}^m$ corresponding to $z_t = \bar{x}_{t-1}$
15:       end for
16:       $\mathscr{V}_t^m := \frac{1}{N_t}\sum_{j=1}^{N_t} \mathscr{V}_{tj}^m, \mathscr{G}_t^m := \frac{1}{N_t}\sum_{j=1}^{N_t} \mathscr{G}_{tj}^m$
17:     end for
18:   end for
19:   $\mathfrak{Q}_t^i \leftarrow \{\theta \in \mathfrak{Q}_t^{i-1}, \theta(x_{t-1}) \ge \mathscr{G}_t^m(x_{t-1} - \bar{x}_{t-1}^m) + \mathscr{V}_t^m, m = 1, 2\dots, M\}$
20:   LB = $\min\{u_1^\top x_1 + v_1^\top y_1 + \mathfrak{Q}_2^i(x_1) : A_1 x_1 + B_1 z_1 + C_1 y_1 \ge b_1, z_1 = x_0\}$
21:   sample mean $\bar{V} = (\sum_{i=1}^M V_i)/M$, sample standard deviation $\sigma(V) = \sqrt{\frac{\sum_{i=1}^M (V_i - \bar{V})^2}{M-1}}$
22:   CI of the expected policy value: $[\bar{V} - z_{1-\alpha/2}\frac{\sigma(V)}{\sqrt{M}}, \bar{V} + z_{1-\alpha/2}\frac{\sigma(V)}{\sqrt{M}}]$
23:   $i = i + 1$
24: end while
```

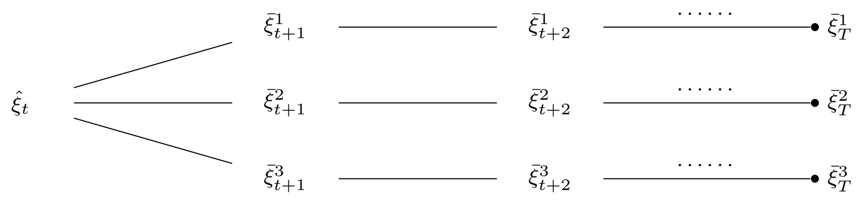
In order to solve the above problem, we need to discretize it. In section 6.2, we approximate the recourse term by iteratively adding cutting planes in backward steps. The rolling horizon based algorithm discretizes the recourse term directly by dynamically constructing scenario tree $\tau(\hat{\xi}_t)$ rooted at $\hat{\xi}_t$ and its extensive formulation,

$$\begin{aligned} \min \sum_{n\in\tau(\hat{\xi}_t)} & P_n[u_n^\top x_n + v_n^\top y_n] \\ \text{s.t. } & A_n x_n + B_n x_{a(n)} + C_n y_n \ge b_n, \forall n \in \tau(\hat{\xi}_t) \\ & x_{a(\hat{\xi}_t)} = x_{t-1} \end{aligned}$$

where we use $a(\cdot)$ to denote the parent node.

To make the discretized problem tractable, the scenario tree $\tau(\hat{\xi}_t)$ should not be too big. Probably the most intuitive way to do that is to branch the next stage only since the next stage is the immediate future that matters the most. In other words, in stage $t + 1$, we generate M samples (nodes) $\bar{\xi}_{t+1}^i, i = 1,\dots,M$ from the conditional probability distribution of ξ_{t+1} given $\xi_{[t]} = \hat{\xi}_{[t]}$; for stages $s = t + 2,\dots,T$, following each of $\bar{\xi}_{s-1}^i$, we generate only one sample (node) $\bar{\xi}_s^i$ from the conditional probability distribution of ξ_s given

Fig. 1: construction of $\tau(\hat{\xi}_t)$



$\xi_{[t]} = \hat{\xi}_{[t]}, \xi_{t+1} = \bar{\xi}^i_{t+1}, \ldots, \xi_{s-1} = \bar{\xi}^i_{s-1}$. In this way, the scenario tree grows linearly with the number of stages. Figure 1 visualizes an example of such construction of $\tau(\hat{\xi}_t)$ where we chose $M = 3$.

There are some existing similar rolling horizon based algorithms in the literature, such as the algorithm used in (Mahmutoğulları, Ahmed, Çavuş, & Aktürk, 2019). But as far as we now, the existing algorithms are all implemented on a preconstructed static scenario tree and hence does not solve the true problem.

8 Evaluating on the true problem

The aforementioned SDDP solver and extensive solver are used to solve the discretized problem while solving the true problem is our real concern. On the other hand, the rolling horizon solver solves the true problem directly. Ideally, solving the problem would, for example, give us a 2% optimality gap with 95% confidence for the true problem. In practice, quite often it is not possible to get such an answer, especially for large scale problems. The next good answer would be that a solution is reasonably good for the true problem. This is less clear and thus we should be more cautious about it. In this section, we will see how we obtain these two kinds of answers.

8.1 Optimality gap for the true problem

As shown in the section 6.2, by solving a discretized problem, the SDDP solver provides a feasible implementable policy for the true problem. Consequently, we can evaluate the value of this policy, of the true problem, by Monte Carlo simulations. Similarly as we did in section 6.3, by generating samples from the *true* data process instead of the discretized problem, we can construct the approximate $(1 - \alpha)$ confidence interval for the expected policy value. The upper end of the CI now provides an upper bound of the optimal value of the true problem with confidence of about $(1 - \alpha)$. By contrast, we can not derive an upper bound for the true problem using the extensive solver.

By randomizing SAA discretization (generating different sets of i.i.d samples), we are also able to compute a statistical lower bound for the true problem, for both the SDDP solver and the extensive solver. It is not difficult to show that the expectation of the optimal value of the SAA discretized problem is not larger than the optimal value of the true problem. Users can thus get a lower bound for the true problem with a chosen confidence level.

Therefore, it is possible to compute the optimality gap for the true problem by randomizing SAA discretization. However, in practice considered problems often are so large that we cannot afford the cost of solving them several times. Also, note that there is no well-established way to get a lower bound for the true problem when using the Markov chain approximation approach.

On the other hand, the rolling horizon based approach solves the true problem directly and provides us with a feasible implementable policy for the true problem. Therefore, it only gives a statistical upper bound.

We summarize this section by table 1.

Table 1: Summary of the solvers

problem	solver	implem-entable	determi-nistic LB	statisti-cal UB	optimality gap
independent	extensive	✗	✓	✗	✗
	SDDP	✓	✓	✓	✓
	rolling	✓	✗	✓	✗
Markovian	extensive	✗	✗	✗	✗
	SDDP	✓	✗	✓	✗
	rolling	✓	✗	✓	✗

8.2 Reasonably good solution to the true problem

The less perfect approach to policy evaluation, discussed below, is to validate the goodness of an discretized problem. This approach is only valid for the SDDP solver.

Consider two implementable feasible policies $\bar{x}_t^{(1)}$ and $\bar{x}_t^{(2)}$, referred to as policy one and policy two respectively. We say policy one is *significantly better* than policy two if the policy value for the true problem of policy one is significantly lower than policy two in terms of the following one-sided hypothesis testing. Similarly to the way we compared two policies in the last section, we generate M samples from the *true* data process and compute the differences of policy values $D_m (m = 1, \dots, M)$ (policy 2 subtracting policy 1). Compute its sample mean $\bar{D}_M$ and sample variance S_M^2. Under the null hypothesis of zero mean of the difference, the statistic $T = \frac{\bar{D}_M}{S_M/\sqrt{M}}$ converges in distribution to $\mathcal{N}(0,1)$. If $T \geq z_\alpha$ we reject the null hypothesis at the confidence level $(1-\alpha)$ and conclude that policy one is significantly better than policy two. Note that when failing to reject the null hypothesis we do not claim that the true expected value of the considered policies is the same, but only that we cannot detect the difference by this t-test. Note also that this procedure depends on the choice of the number M of generated samples and that its accuracy increases with increase of the number M.

We say a policy is "reasonably good" if by reasonable efforts, it is not possible to find a significantly better policy. The reasonable efforts may include increasing the fineness of discretization and changing the method of discretization. For example, consider a stage-wise independent continuous true problem. Suppose we have got a policy by solving SAA problem with 50 samples per stage. One way to verify that the sample size 50 is large enough is to get another policy by solving SAA problem with 100 sample size. If we compare the two policies by the above test and it fails to reject the null hypothesis with a reasonable number M of generated samples, we argue that our discretization is reasonably good and it provides a reasonably good solution to the true problem. For another example, consider a Markovian continuous true problem with a right-hand-side random data process. It is suggested to test both TS approach and MCA to discretize Markovian data process. The two approaches provide us with different discretized problems, possibly giving us significantly different policies (see section 16.1 for an example). These procedures of course are heuristic and should be performed with care.

9 Risk averse true problem

So far we only considered optimizing (minimizing) the *expected* policy value. However, for some realizations of the stochastic process the generated costs could be significantly bigger than their average values. This motivates to introduce a risk averse approach trying to control high quantiles of the random costs. The build-in risk measure provided by the MSPPy package is a convex combination of the expectation and Average Value-at-risk (AVaR) risk measure (also called Conditional Value-at-risk, Expected Shortfall, Expected Tail Loss)

$$\rho_t(\cdot) = (1-\lambda_t)\mathbb{E}(\cdot) + \lambda_t \mathrm{AV@R}_{\alpha_t}(\cdot),\ \alpha_t, \lambda_t \in (0,1), \tag{12}$$

where $\mathrm{AV@R}_{\alpha_t}(Z) := \inf_x\{x + \alpha_t^{-1}\mathbb{E}[Z-x]_+\}$. The nested formulation in risk averse setting as follows is the same as in the risk neutral setting except replacing the conditional expectation with conditional analogues $\varrho_{t|\xi_{[t-1]},\eta_{[t-1]}}$ of ρ_t (recall that the first stage variables are deterministic).

$$\min_{\substack{A_1x_1+B_1z_1+C_1y_1\geq b_1\\ z_1=x_0}} u_1^\top x_1 + v_1^\top y_1 + \rho_{2|\xi_1,\eta_1}\Bigg[\min_{\substack{A_2x_2+B_2z_2+C_2y_2\geq b_2\\ z_2=x_1}} u_2^\top x_2 + v_2^\top y_2$$
$$+\cdots+\rho_{T|\xi_{[T-1]},\eta_{[T-1]}}\Big[\min_{\substack{A_Tx_T+B_Tz_T+C_Ty_T\geq b_T\\ z_T=x_{T-1}}} u_T^\top x_T + v_T^\top y_T\Big]\Bigg].$$

The same applies to the dynamic equations. For example, in the stage-wise independent setting, the dynamic equations with risk measures take the form of,

$$Q_t(x_{t-1},\xi_t) = \min_{\substack{A_tx_t+B_tz_t+C_ty_t\geq b_t\\ z_t=x_{t-1}}} u_t^\top x_t + v_t^\top y_t + \mathcal{Q}_{t+1}(x_t),$$

where the value functions

$$\mathcal{Q}_{t+1}(x_t) := \begin{cases} \rho_{t+1|\xi_t}\left[Q_{t+1}(x_t, \xi_{t+1})\right], \text{ for } t = T-1, \dots, 1, \\ 0, \text{ for } t = T, \end{cases}$$

The risk averse problem after discretization can be solved by directly adjusting the gradient computation in line 15 in Algorithm 2. In particular for the risk measure defined in equation (12), the computation is the following. Sort $\mathscr{V}_{tj}, J = 1, \dots, N_t$ into ascending order $\mathscr{V}_{t,\pi(1)}, \dots, \mathscr{V}_{t,\pi(N_t)}$ and let κ be the smallest integer such that $\pi(\kappa) \geq (1-\alpha)N_t$. Then

$$\mathscr{V}_t = (1-\lambda_t)\frac{1}{N_t}\sum_{j=1}^{N_t}\mathscr{V}_{tj} + \lambda_t\biggl(\mathscr{V}_{t,\kappa} + \frac{1}{\alpha_t N_t}\sum_{j=1}^{N_t}(\mathscr{V}_{tj} - \mathscr{V}_{t,\kappa})_+\biggr),$$

$$\mathscr{G}_t = (1-\lambda_t)\frac{1}{N_t}\sum_{j=1}^{N_t}\mathscr{G}_{tj} + \lambda_t\biggl(\mathscr{G}_{t,\kappa} + \frac{1}{\alpha_t N_t}\sum_{j=1}^{N_t}(\mathscr{G}_{tj} - \mathscr{G}_{t,\kappa})\mathbb{I}_{\mathscr{V}_{tj}-\mathscr{V}_{t,\kappa}\geq 0}\biggr), \quad (13)$$

where $\mathbb{I}(\cdot)$ is the indicator function. In the end, replace line 15 with (13).

Unfortunately there is no longer easy way to compute the risk-adjusted cost,

$$u_1^\top x_1 + v_1^\top y_1 + \rho_{2|\xi_1,\eta_1}\Bigl[u_2^\top x_2 + v_2^\top y_2 + \cdots + \rho_{T|\xi_{[T-1]},\eta_{[T-1]}}[u_T^\top x_T + v_T^\top y_T]\Bigr].$$

Therefore, in the forward steps of the SDDP algorithm, the MSPPy package considers the value of its risk neutral component $\sum_{t=1}^{T}[u_t^\top x_t + v_t^\top y_t]$. As a result, the second stopping criterion, namely evaluation of the optimality gap, is not valid.

In addition, when evaluating the policy generated by risk averse SDDP, the policy value is computed only based on its risk neutral component. In such a way, we can compare policies that generated from solving risk neutral and risk averse problems. One may expect policies generated from risk averse problems have less favourable expected policy value but lower variance of policy values (an example is in section 16.2).

10 Multi-stage stochastic mixed integer programs (MSIP)

Multistage stochastic integer programming (MSIP) asserts integrality restrictions on the feasible region and takes the form of (in the risk neutral setting),

$$\min_{\substack{A_1x_1+B_1z_1+C_1y_1\geq b_1\\ z_1=x_0\\ x_1,y_1\in\mathbb{RZ}}} u_1^\top x_1 + v_1^\top y_1 + \mathbb{E}_{|\xi_1,\eta_1}\Bigl[\min_{\substack{A_2x_2+B_2z_2+C_2y_2\geq b_2\\ z_2=x_1\\ x_2,y_2\in\mathbb{RZ}}} u_2^\top x_2 + v_2^\top y_2$$

$$+ \cdots + \mathbb{E}_{|\xi_{[T-1]},\eta_{[T-1]}}\bigl[\min_{\substack{A_Tx_T+B_Tz_T+C_Ty_T\geq b_T\\ z_T=x_{T-1}\\ x_T,y_T\in\mathbb{RZ}}} u_T^\top x_T + v_T^\top y_T\bigr]\Bigr],$$

where $\mathbb{RZ}$ denotes a generic mixed-integer set.

Most of the procedures in the above sections still work in the MSIP setting by simply adding required integrality conditions. For the extensive solver and the rolling horizon solver introduced in section 6.1 and section 7 respectively, the only difference in the MSIP setting is to construct equivalent deterministic mix-integer programs rather than just linear programs. While for the SDDP solver introduced in section 6.2, it is more involved to generalize. This section will demonstrate how to generalize SDDP to stochastic dual dynamic integer programming (SDDiP) in the stage-wise independent setting. Markovian problems can be done similarly.

Again, we first construct a discretized problem by SAA discretization and writing dynamic programming equations in a way similar to equations (9). We then use the so-called SDDiP procedure to solve the discretized problem as follows. In the forward steps, the SDDiP solver draws samples from the discretized problem and solves the following mixed integer programs[1] recursively for $t = 1, \dots, T$,

$$\min_{x_t, y_t \in \mathbb{RZ}} \{u_t^\top x_t + v_t^\top y_t + \mathfrak{Q}_{t+1}(x_t) : A_t x_t + B_t z_t + C_t y_t \geq b_t, z_t = \bar{x}_{t-1}\},$$

and obtain a feasible solution in each stage $\bar{x}_1, \dots, \bar{x}_T$. Here $\mathfrak{Q}_{t+1}(\cdot)$ is the current approximation of expected recourse function $\tilde{\mathcal{Q}}_{t+1}(\cdot)$.

In backward steps, for $t = T, \dots, 2$, for every scenario $j \in N_t$, SDDiP solver solves

$$G^* := \min_{x_t, y_t \in \mathbb{RZ}} \{u_{tj}^\top x_t + v_{tj}^\top y_t + \mathfrak{Q}_{t+1}(x_t) : A_{tj} x_t + B_{tj} z_t + C_{tj} y_t \geq b_{tj}, z_t = \bar{x}_{t-1}\} \tag{14}$$

The MSPPy package provides different types of cuts as follows to $\mathfrak{Q}_t(\cdot)$. We say a cut (v_t, k_t) is: (i) *valid*, if $\mathfrak{Q}_t(x) \geq v_t + k_t^\top x$, $\forall x$, (ii) *tight*, if $\mathfrak{Q}_t(\bar{x}_{t-1}) = v_t + k_t^\top \bar{x}_{t-1}$.

10.1 Benders' cuts

We make the following LP relaxation of (14)

$$\min_{x_t, y_t} \{u_{tj}^\top x_t + v_{tj}^\top y_t + \mathfrak{Q}_{t+1}(x_t) : A_{tj} x_t + B_{tj} z_t + C_{tj} y_t \geq b_{tj}, z_t = \bar{x}_{t-1}\}. \tag{15}$$

The optimal value $\mathscr{V}_{tj}$ and an optimal dual solution $\mathscr{G}_{tj}$ corresponding to constraints $z_t = \bar{x}_{t-1}$ serve as coefficients of a Benders' cut of $\tilde{\mathcal{Q}}_{t+1}(\cdot)$ at $\bar{x}_{t-1}$ for scenario j. The Benders' cut is then given by,

$$\mathfrak{Q}_t(x) \geq (\frac{1}{N_t} \sum_{j=1}^{N_t} \mathscr{G}_{tj}^\top)(x - \bar{x}_{t-1}) + \frac{1}{N_t} \sum_{j=1}^{N_t} \mathscr{V}_{tj}.$$

The Benders' cut is valid but not necessarily tight.

[1]For simplicity, we omit the superscript i representing the index of iteration.

10.2 Strengthened Benders' cuts

Consider the following Lagrangian dual problem of (14),

$$\begin{aligned} & g^* := \max_{\pi}\{g(\pi)\}, \\ & g(\pi) := \min_{x_t, y_t \in \mathbb{R}\mathbb{Z}} \{u_{tj}^\top x_t + v_{tj}^\top y_t + \mathfrak{Q}_{t+1}(x_t) - \pi^\top (z_t - \bar{x}_{t-1}) \\ & : A_{tj} x_t + B_{tj} z_t + C_{tj} y_t \geq b_{tj}\}. \end{aligned} \tag{16}$$

Note that the outer problem is unconstrained with respect to π, hence every π will provide a valid cut. In particular, we let $\pi = \mathscr{G}_{tj}$ and solve the following program

$$\min_{x_t, y_t \in \mathbb{R}\mathbb{Z}} \{u_{tj}^\top x_t + v_{tj}^\top y_t + \mathfrak{Q}_{t+1}(x_t) - \mathscr{G}_{tj}^\top z_t : A_{tj} x_t + B_{tj} z_t + C_{tj} y_t \geq b_{tj}\}.$$

Suppose its optimal value is $\mathscr{V}_{tj}^{SB}$. The strengthened Benders' cut is then given by

$$\mathfrak{Q}_t(x) \geq (\frac{1}{N_t}\sum_{j=1}^{N_t} \mathscr{G}_{tj})^\top x + \frac{1}{N_t}\sum_{j=1}^{N_t} \mathscr{V}_{tj}^{SB}.$$

Note that the strengthened Benders' cut is paralleled and superior to its corresponding Benders' cut. The strengthened Benders' cut is valid but still not necessarily tight.

10.3 Lagrangian cut

The Lagrangian cut is obtained by solving the dual problem (16). Since $g(\pi)$ is a concave piece-wise linear function, we can use the level method (Lemaréchal, Nemirovskii, & Nesterov, 1995) to solve it, as shown in Algorithm 5.

It is recommended to set the step size as 0.2929 (Lemaréchal et al., 1995). We don't know the optimal value of (16) in general. Thus in Algorithm 5, we keep updating our estimate of g^* by the set of cutting planes. In the next section, we will see binarization of the state space makes $G^* = g^*$ and thus

$$\mathfrak{Q}_t(\bar{x}_{t-1}) = \left(\frac{1}{N_t}\sum_{j=1}^{N_t} \mathscr{G}_{tj}^{LG}\right)^\top \bar{x}_{t-1} + \frac{1}{N_t}\sum_{j=1}^{N_t} \mathscr{V}_{tj}^{LG},$$

assuring that Lagrangian cuts are tight. The stopping criteria of the level method are similar to those in the SDDP solver.

Algorithm 5 Level method to compute Lagrangian cuts at stage t

```
1: Given a fixed step size $\lambda$ and state $\bar{x}_{t-1}$.
2: for $j = 1, \ldots, N_t$ do
3:     Initialize $\pi_1 = \mathscr{G}_{tj}$, $k = 1$.
4:     Initial approximation $\Psi^0 = \{\tau : \tau(\cdot) \le u\}$
5:     while no stopping criterion is met do
6:         Solve the inner problem: $g(\pi_k) := \min_{x_t, y_t \in \mathbb{RZ}} \{u_{tj}^\top x_t + v_{tj}^\top y_t + \mathfrak{Q}_{t+1}(x_t) - \pi_k^\top (z_t - \bar{x}_{t-1}) : A_{tj} x_t + B_{tj} z_t + C_{tj} y_t \ge b_{tj}\}$ and obtain an optimal solution $x_t = x_t^{(\pi_k)}, z_t = z_t^{(\pi_k)}$.
7:         Update cut collection $\Psi^k \leftarrow \{\tau \in \Psi^{k-1} | \tau(\pi_k) \le g(\pi_k) - (\pi - \pi_k)^T (z_t^{(\pi_k)} - \bar{x}_{t-1})\}$
8:         Solve: $g^* = \max_\pi \{\tau(\pi) \in \Psi^k\}$
9:         Update best so far solution $\pi_k^*$ such that $g_* := g(\pi_k^*) = \max_{i=1,\ldots,k}\{g(\pi_i)\}$
10:        Update level $l_k = \lambda g_* + (1-\lambda) g^*$
11:        Update $\pi_{k+1} = \mathrm{argmin}_\pi \{\|\pi - \pi_k^*\|_2^2 | g(\pi_i) - (\pi - \pi_i)^T (z_t^{(\pi_i)} - \bar{x}_{t-1}) \ge l_k, \forall i = 1, 2, \ldots, k\}$
12:        $k = k + 1$
13:    end while
14:    $\mathscr{G}_{tj}^{LG} = \pi^*, \mathscr{V}_{tj}^{LG} = g(\pi^*) - (\frac{1}{N_t} \sum_{j=1}^{N_t} \mathscr{G}_{tj}^{LG})^\top \bar{x}_{t-1}$
15: end for
16: $\mathfrak{Q}_t \leftarrow \{\theta \in \mathfrak{Q}_t, \theta(x) \ge (\frac{1}{N_t} \sum_{j=1}^{N_t} \mathscr{G}_{tj}^{LG})^\top x + \frac{1}{N_t} \sum_{j=1}^{N_t} \mathscr{V}_{tj}^{LG}\}$
```

10.4 Binarization

In order to binarize an MSIP, we make the following additional assumption.

Assumption 2 The polyhedral set $\{x_t = (x_{t,1}, \ldots, x_{t,n_t}) \in \mathbb{RZ}^{n_t} : A_t x_t + B_t x_{t-1} + C_t y_t \ge b_t\}$ is bounded over time, where n_t is the dimension of the state space and $\mathbb{RZ}^{n_t}$ is a mix-integer set of dimension n_t. That is, there exist $U_t = (u_{t,1}, \ldots, u_{t,n_t})$ and $V_t = (v_{t,1}, \ldots, v_{t,n_t})$ such that $U_t \le x_t \le V_t$ in the element-wise sense, i.e., $u_{t,i} \le x_{t,i} \le v_{t,i}$, $i = 1, \ldots, n_t$, for all x_t in the polyhedral set, $t = 1, \ldots, T$.

If Assumption 2 holds, we approximate the decimal variable $x_{t,i}$ by a binary variable vector $\mathbf{x}_{t,i}^{(2)}$ as follows:

$$
\begin{aligned}
&10^{k_{t,i}} (x_{t,i} - u_{t,i}) = H_{t,i}^\top x_{t,i}^{(2)}, \\
&H_{t,i} = (2^0, \ldots, 2^{B_{t,i}-1})^\top, \\
&B_{t,i} = \lfloor \log_2 [10^{k_{t,i}} (v_{t,i} - u_{t,i})] \rfloor + 1, \\
&x_{t,i}^{(2)} \in \{0, 1\}^{B_{t,i}},
\end{aligned}
$$

where $k_{t,i}$ is the number of decimal places of accuracy for each dimension i in stage t and is set to zero if $x_{t,i} \in \mathbb{Z}$. Thus the state variable variable $x_t = (x_{t,1}, \ldots, x_{t,n_t})$ is approximated by binary state variable $x_t^{(2)} = (x_{t,1}^{(2)\top}, \ldots, x_{t,n_t}^{(2)\top})$ of length $B_t = \sum_{i=1}^{n_t} B_{t,i}$, so as the local copy variable z_t.

We then have the *binarized discretized problem*,

$$
\begin{aligned}
\tilde{Q}_t(x_{t-1}^{(2)}, \xi_t) = \min\ & u_t^\top x_t + v_t^\top y_t + \tilde{\mathcal{Q}}_{t+1}^{(2)}(x_t^{(2)}) \\
\text{s.t.}\ & A_t x_t + B_t z_t + C_t y_t \geq b_t, \\
& z_t^{(2)} = x_{t-1}^{(2)}, \\
& 10^{k_{t,i}}(x_{t,i} - u_{t,i}) = H_{t,i}^\top x_{t,i}^{(2)}, \\
& 10^{k_{t,i}}(z_{t,i} - u_{t,i}) = H_{t,i}^\top z_{t,i}^{(2)}, \\
& x_{t,i}^{(2)} \in \{0,1\}^{B_{t,i}},
\end{aligned}
\tag{17}
$$

for $t = T, \ldots, 1$. Here $x_0^{(2)}$ is the binary representation of initial value x_0 and the binarized approximate expected cost-to-go is defined as

$$
\tilde{\mathcal{Q}}_{t+1}^{(2)}(x_t^{(2)}) := \begin{cases} \dfrac{1}{J_{t+1}} \displaystyle\sum_{j=1}^{J_{t+1}} \tilde{Q}_{t+1}^{(2)}(x_t^{(2)}, \xi_{t+1}^{(2)j}), \text{ for } t = T-1, \ldots, 1, \\ 0, \text{ for } t = T. \end{cases}
$$

Similar to (16), we can make Lagrangian dual problem of (17). As shown in (Zou et al., 2018), there is no gap between (17) and its Lagrangian dual problem; the SDDiP algorithm with Lagrangian cuts guarantees to produce an optimal solution to the binarized discretized problem in a finite number of iterations with probability one.

It was given in (Zou et al., 2018) the upper bound of $k_i, \forall i$, to obtain an ϵ optimal solution the discretized problem by solving the binarized discretized problem. In practice such bound is rarely applicable and our real primal interest is solving the true problem. We now illustrate how to evaluate the obtained policy on the true problem. The SDDiP solver produces approximation of $\tilde{\mathcal{Q}}_2^{(2)}(\cdot), \ldots, \tilde{\mathcal{Q}}_T^{(2)}(\cdot)$, denoted by $\mathfrak{Q}_2^{(2)}(\cdot), \ldots, \mathfrak{Q}_T^{(2)}(\cdot)$, and a feasible first stage solution $\bar{x}_1^{(2)}$. This generates a feasible implementable policy $\bar{x}_t(\xi_{[t]})$ for the true problem as follows. For any realization of the process $\{u_t, v_t, A_t, B_t, C_t, b_t\}, t = 2, \ldots, T$, we associate a decision vector $x_1 (= H\bar{x}_1^{(2)}), x_2, \ldots, x_T$ generated by solving recursively,

$$
\begin{aligned}
\min\ & u_t^\top x_t + v_t^\top y_t + \mathfrak{Q}_{t+1}^{(2)}(x_t^{(2)}) \\
\text{s.t.}\ & A_t x_t + B_t z_t + C_t y_t \geq b_t, \\
& z_t = x_{t-1}, \\
& 0 \leq 10^{k_{t,i}}(x_{t,i} - u_{t,i}) - H_{t,i}^\top x_{t,i}^{(2)} < 1, \\
& x_{t,i}^{(2)} \in \{0,1\}^{B_{t,i}},\ i = 1, \ldots, n,\ t = 2, \ldots, T.
\end{aligned}
$$

Again, since Assumption 1 holds, it provides a feasible implementable policy for the true problem.

11 Skeleton of MSPPy

The current version of the MSPPy package uses the Gurobi solver and the Python interface. It requires **Python 3+** and **gurobipy 7+** (L. Gurobi Optimization, 2018). The entire package is based on object oriented programming, making codes readable and further development easy. It has five main modules, *msp*, *sp*, *discretize*, *solver* and *evaluation*.

The *msp*, *sp*, *discretize* modules are used to build and discretize the true problem. The *msp* module contains an MSLP base class, corresponding to a multistage stochastic program. The *sp* module contains a StochasticModel base class representing an individual stage problem. The *discretize* module includes the MCA approach to discretize Markovian data processes.

The *solver* module is used to construct solver objects to solve a discretized problem. An extensive solver object can be used to solve an MSLP/MSIP discretized problem. An SDDP(SDDiP) solver object can be used to solve an MSLP(MSIP) discretized problem. A rolling solver object can be used to solve directly an MSLP/MSIP true problem.

The *evaluation* module includes two classes, *Evaluation* and *EvaluationTrue*, to understand policies obtained by SDDP/SDDiP solver. An *Evaluation* object evaluates a policy on the discretized problem while an *EvaluationTrue* object evaluates a policy on the true problem.

12 Using the SDDP solver to solve MSLP

In this section, we illustrate a standardized procedure to solve MSLP using the SDDP solver. We will see how to use the SDDiP solver to solve MSIP in section 13.

12.1 Step one: build the true problem

To build the true problem, the first step is to create an MSLP object. An MSLP object is composed of a list of StochasticModel objects. Users are then required to fill in each StochasticModel object. The StochasticModel class is a stochastic version of the gurobipy.Model that built in the Gurobi library. It allows users to directly write into a stochastic model rather than treating the deterministic counterpart and uncertainties separately. In order to achieve it while staying close to the gurobipy syntax, the MSPPy package encapsulates the gurobipy.Model and its randomness. Hence, all things that work on gurobipy.Model will work on Stochastic Model. In addition, four routines from gurobipy are overwritten and several new routines are created for modeling convenience. The four overwritten routines as shown in the snippet below, *addVar*, *addVars*, *addConstr*, *addConstrs*, include additional arguments called *uncertainty* and *uncertainty_dependent* to incorporate stage-wise independent data process and Markov process. Uncertainties that appear in the objective, u_i, v_i, can be added along with adding related variables (by *addVar* or *addVars*). Uncertainties that appear in the constraints, A_i, B_i, C_i, b_i, can be added along with adding related constraints (by *addConstr* or *addConstrs*).

```
m = StochasticModel()
m.addVars(...,uncertainty, uncertainty_dependent)
m.addVar(..., uncertainty, uncertainty_dependent)
m.addConstrs(...,uncertainty, uncertainty_dependent)
m.addConstr(...., uncertainty, uncertainty_dependent)
```

Two new routines are added in order to include state variable(s). Local copy variable(s) will be added correspondingly behind the scenes. In the following snippet, now is a reference to the added state variable(s) and past is a reference to the corresponding local copy variable(s).

```
now, past = m.addStateVars(...,uncertainty,uncertainty_dependent)
now, past = m.addStateVar(...,uncertainty,uncertainty_dependent)
```

Using the above routine to add multiple stage-wise independent uncertainties sequentially inherently assume the added uncertainties are independent. The MSPPy package provides another routine *add_continuous_uncertainty* that is able to incorporate an uncertainty that follows a multivariate distribution,

```
m.add_continuous_uncertainty(uncertainty, locations)
```

12.1.1 Stage-wise independent problem

We first indicate procedures and examples to build stage-wise independent MSLP. Stage-wise independent uncertainties should be specified by scenarios or random variable generators.
Step 1.1 (Stage-wise independent discrete/continuous) Create an MSLP instance by specifying number of time periods T. This will create a list of T empty StochasticModel objects.
Step 1.2 (Stage-wise independent discrete/continuous) Run a for loop and fill in the T StochasticModel objects, namely,

$$\min_{A_1x_1+B_1z_1+C_1y_1\geq b_1} u_1^\top x_1 + v_1^\top y_1, \cdots, \min_{A_Tx_T+B_Tz_T+C_Ty_T\geq b_T} u_T^\top x_T + v_T^\top y_T$$

Note that the balance constraints $z_t = x_{t-1}$ will be added behind the scenes.

The following snippet specifies a three stage MSLP with stage-wise independent finite discrete uncertainty. In every stage, a state variable x_t and a local copy z_t are added. The random objective coefficients u_2, u_3 are independent and take 1.5 or 0.5 with equal probability.

```
uncertainty = [[1],[1.5,0.5],[1.5,0.5]]
discrete = MSLP(T = 3)
for t in range(3):
    m = discrete[t]
    now, past = m.addStateVar(uncertainty=uncertainty[t])
```

The following snippet specifies a three stage MSLP with a stage-wise independent continuous uncertainty. The random objective coefficients u_2, u_3 are independent and follow the log-normal distribution $\log\mathcal{N}(0,1)$.

```
def f(random_state):
    return random_state.lognormal(0,1)
continuous = MSLP(T = 3)
for t in range(3):
    m = continuous[t]
    if t == 0:
        now, past = m.addStateVar()
    else:
        now, past = m.addStateVar(uncertainty=f)
```

12.1.2 Markovian MSLP

We now demonstrate procedures and examples to build Markovian MSLP. A nonhomogeneous Markov chain process should be specified by Markov state spaces and transition matrices for each stage. A Markovian continuous process should be specified by a sample path generator. Each dimension of the Markovian uncertainty is then added to the right place (either in the objective or in constraints) to the constructed MSLP.
Step 1.1(Markovian discrete/continuous) Create an MSLP instance by specifying number of time periods T and a sample path generator/Markov chain. This will create a list of T empty StochasticModel objects.
Step 1.2(Markovian discrete/continuous) Run a for loop and fill in the T StochasticModel objects, namely,

$$\min_{A_1x_1+B_1z_1+C_1y_1\geq b_1} u_1^\top x_1 + v_1^\top y_1, \cdots, \min_{A_Tx_T+B_Tz_T+C_Ty_T\geq b_T} u_T^\top x_T + v_T^\top y_T$$

Again, the balance constraints $z_t = x_{t-1}$ will be added behind the scenes.

The following snippet formulates a three stage MSLP with Markov chain objective coefficient u_t. The initial Markov state $u_1 = 1$. The Markov states and transition matrices for stage two and stage three are given by,

$$\begin{array}{cc} & \begin{array}{cc} 2 & 3 \end{array} \\ 1 & \begin{pmatrix} 0.2 & 0.8 \end{pmatrix} \end{array}, \quad \begin{array}{cc} & \begin{array}{cc} 4 & 5 \end{array} \\ \begin{array}{c} 2 \\ 3 \end{array} & \begin{pmatrix} 0.3 & 0.7 \\ 0.4 & 0.6 \end{pmatrix} \end{array}$$

```
mc = MSLP(T = 3)
mc.add_MC_uncertainty(
    Markov_states=[[1],[2,3],[4,5]],
    transition_matrix=[[[1]],[[0.2,0.8]],[[0.3,0.7],[0.4,0.6]]]
)
for t in range(3):
    m = MC[t]
    now, past = m.addStateVar(uncertainty_dependent=0)
```

Note that in this case the Markovian uncertainty is unidimensional, the dimension index *uncertainty_dependent* is simply 0 (it starts with zero in Python). In other words, the first dimension of the Markovian process appears at the objective coefficient of the variable *now*.

The following snippet formulates a three stage MSLP with Markovian objective coefficient u_t where u_t follows an autoregressive time series model: $u_t = 0.5u_{t-1} + \epsilon_t$, $\epsilon_t \overset{i.i.d}{\sim} \mathcal{N}(0,1)$ with initial value $u_0 = 0$.

```
markovian = MSLP(T = 3)
def sample_path_generator(random_state, size):
    u = numpy.zeros([size,T,1])
    for t in range(1,T):
        u[:,t,:] = 0.5 * u[:,t-1,:]
            + random_state.normal(0,1,size=size))
    return a
markovian.add_Markovian_uncertainty(sample_path_generator)
for t in range(3):
    m = Markovian[t]
    now, past = m.addStateVar(uncertainty_dependent=0)
```

Finally, users can specify a risk measure. Also, for a stage-wise independent finite discrete uncertainty, users can specify the probability mass function that it follows.

12.2 Step two: discretize the true problem

Stage-wise independent uncertainties are discretized by SAA. The following snippet makes a discretization of with 100 i.i.d samples for each stage except stage one.

```
continuous.discretize(n_samples=100)
```

Markovian uncertainties can be discretized by MCA. The following snippet uses the SA method and 1000 sample paths to train an approximating Markov chain with 10 Markov states from stage two on.

```
markovian.discrete(n_Markov_states=10, n_sample_paths=1000,
    method='SA')
```

12.3 Step three: solve the discretized problem

We can now use the SDDP solver to solve the discretized problem with certain stopping criterion. First, create an SDDP instance solver,

```
example_SDDP = SDDP(example)
```

Next, call the *solve* method to run SDDP algorithm with the following three types of stopping criterion to choose from.

The first type: an upper limit of certain characteristics among time, the number of iterations and the number of iterations that the deterministic bound does not change.

```
example_SDDP.solve(max_iterations, max_time,
    max_stable_iterations)
```

The second type: evaluate the policy every *freq_evaluations* iterations by running *n_simulations* Monte Carlo simulations. If the gap becomes not larger than *tol*, the algorithm will be stopped.

```
example_SDDP.solve(freq_evaluations, n_simulations, tol)
```

The third type: compare the policy every *freq_comparisons* iterations by computing the CI of difference of the expected policy values. If the upper end of CI becomes not larger than *tol_diff*, the algorithm will be stopped.

```
example_SDDP.solve(freq_comparsions, n_simulations, tol_diff)
```

SDDP solver utilizes the **multiprocessing** library to make parallelization. Parallelization can be switched on by passing to the solve method the number of forward and backward steps to run per iteration and the number of processes to parallelize the jobs.

```
example_SDDP.solve(n_steps, n_processes)
```

The obtained policy by SDDP solver is a first stage state solution and approximations of value functions (cutting planes). Writing all models to disk reveal all the cutting planes in a readable way.

```
example.first_stage_solution
example.write()
```

12.4 Step four: evaluate the SDDP policy on the true problem

The MSPPy package evaluates the SDDP policy by Monte Carlo simulation. As an example, the following snippet queries the confidence interval of the simulated policy values of the true problem and queries the solution of a variable called "purchase".

```
example_result = EvaluationTrue(example)
example_result.run(n_simulations, query=["purchase"])
example_result.CI
example_result.solution["purchase"]
```

13 Using the SDDiP solver to solve MSIP

The five essential steps to solve MSIP by the SDDiP solver are:
Step one: build the true problem.
Step two: discretize the true problem.
Step three: binarize the discretized problem. (This step can be skipped.)
Step four: solve the (binarized) discretized problem.
Step five: evaluate the computed policy.

Compared to solving MSLP, for step two and step five, the usage syntax is exactly the same as shown in section 12; for step one, users only need to change the key word from MSLP to MSIP. Hence, in this section we will only go through steps three and four.

13.1 Step three: binarize the discretized problem

The following snippet binarizes the discretized problem for stages before *bin_stage* (exclusive). Precision parameter k dictates the number of decimal places of accuracy.

```
MSIP().binarize(bin_stage, precision)
```

13.2 Step four: solve the (binarized) discretized problem

The three types of stopping criterion mentioned in section (12.3) are still valid here. If Lagrangian cuts are used, stopping criterion and specifics of the level method can be specified.

```
SDDiP().solve(cuts=['LG'], level_step_size, level_tol)
```

The SDDiP solver should be called with arguments specifying the types of cuts. The following three cut patterns are supported.

For a cyclical cut pattern, cuts are added cyclically. The following snippet dictates for every ten iterations, adding Benders' cut for the first three, adding Strengthened Benders' cut for the next three, and adding Lagrangian cut for the last four.

```
SDDiP().solve(cuts=['B','SB','LG'], pattern={'cycle':[3,3,4]})
```

For a barrier-in cut pattern, cuts are added from certain iteration. The following snippet dictates adding Benders' cut from beginning, adding Strengthened Benders' cut from iteration 10, and adding Lagrangian cut from iteration 20.

```
SDDiP().solve(cuts=['B','SB','LG'], pattern={'in':[0,10,20]})
```

For some problems with exceptionally long horizon, sometimes the integrality conditions in the far future may contribute negligibly to the objective. The following routine does an LP relaxation to the stage models after stage 100.

```
SDDiP().solve(cuts=['B','SB','LG'], relax_stage=100)
```

14 Using the extensive solver to solve MSLP/MSIP

The three essential steps to solve MSLP/MSIP by the extensive solver are:
Step one: build the true problem.
Step two: discretize the true problem.
Step three: solve the discretized problem.

Recall that the extensive solver does not provide an implementable policy for the true problem. Therefore, there is no step four to evaluate on the true problem. Compared to the SDDP/SDDiP solver, for step one and step two, the usage syntax is exactly the same as shown in the previous sections. For step three, the extensive solver just constructs an equivalent deterministic LP/IP and solves it.

```
Extensive().solve()
```

15 Using the rolling solver to solve MSLP/MSIP

The essential step to solve MSLP/MSIP by the rolling solver are:
Step one: build the true problem.
Step two: solve the true problem and evaluate the obtained policy.

Recall that the rolling solver is able to solve the true problem directly. As a result, there is no need to discretize the problem and we can combine the procedure of solving and evaluation. The first step is again, exactly the same as before. In step two, for Markovian MSLP/MSIP, apart from the sample path generator, the number of branches and the conditional probability distribution used to construct dynamic scenario trees should be specified. Evaluation parameters should also be inputted. For example, for the true problem with underlying process $u_t = 0.5u_{t-1} + \epsilon_t$, where $\epsilon_t \overset{i.i.d}{\sim} \mathcal{N}(0,1)$ constructed in section 12.1.2, the following snippet solves it and evaluates the obtained policy,

```
def conditional_dist(random_state, prev, t):
    noise = random_state.normal(0,1)
    return 0.5 * prev + noise
Rolling().solve(n_branches, n_simulations,
    conditional_dist=conditional_dist)
```

16 Applications of the MSPPy package

In this section, we apply the MSPPy package to solve three real world problems. The three problems are chosen specially from three fields, namely, power system, airline and finance, where the stochastic programming algorithms have been widely used. In Table 2, we summarize the characteristics of the three problems. The considerable distinctions of these large-scale test problems manifest the sophistication, capability and robustness of the MSPPy package.

Table 2: Summary of the test problems (numbers in parenthesis are the dimensions)

Problem	Random Data Process		#State Vars		#stages
	Independent	Markovian	Cont.	Int.	
Power system	None	VAR(1) with multiplicative errors (4)	4/8	0	120
Portfolio Opt.	Normal (100)	AR(1) with a Normal GARCH(1,1) innovation (3)	101	0	12
Airline revenue	None	NHPP (#arrivals: Gamma, arrival patterns: Beta (72)	0	144	14

16.1 Hydro-thermal power system planning

The Brazilian interconnected power system (Shapiro et al., 2013) have four regions ($i = 1, 2, 3, 4$). In each region i, demand is satisfied by energy generation q_i (in megawatt, the same below) from an integrated reservoir, energy generation g_k from local thermal plants $k \in \Omega_i$ and energy inflow $\mathrm{ex}_{j\to i}$ from other regions ($j \in \{1, 2, 3, 4\}$) or a transshipment station ($j = 5$). If demand d_i can not be satisfied, system i borrows df_{ij} units of energy from the deficit account in system j and a cost of e_{ij} (in dollars per megawatt, the same below) will be incurred. The deficit accounts are set up in a way that ensures the complete recourse condition is held. There are tiny costs for each unit of energy spillage and energy exchanges, denoted by b_i, $c_{j\to i}$ (energy flowed from region j to region i) respectively. We assume a cost of u_k for each unit of energy thermal plant k produces. The dynamics of the system is that the stored energy v_{it} of each reservoir i in stage t is equal to the previous stored energy $v_{i,t-1}$ plus the current water inflow a_{it} minus the energy generation q_{it} and the water spillage s_{it}. The four stored energy $v_{i,t}, i = 1, \ldots, 4$ represents the state variables and the rest are the control variables. The transshipment station is solely for water exchanges between regions. It is required to make decisions sequentially of how much energy to store after seeing the realization of inflow energy as shown in the below chain relationship. Here v_i and a_i are four dimensional vectors. v_0 is given and a_1 is deterministic. The objective is to obtain a sequential scheme to minimize the total operation cost over a design period of T stages with a discount factor γ while meeting energy requirements and feasibility constraints. We add additional subscripts t to distinguish variables and costs at different stages.

The problem is originally a 60-stage (60-month) multistage problem with a monthly discount factor $\gamma = 0.9906$ (this discount factor corresponds to the annual discount rate of 12%, i.e., $1/\gamma^{12} = 1.12$ for $\gamma = 0.9906$). In order to deal with the so-called end-of-horizon effect, 60 more stages are added to the problem, which results in $T = 120$ stage problem.

$$
\begin{aligned}
\min \sum_{t=1}^{T} \gamma^{t-1} \Big\{ & \sum_{i=1}^{4} \Big[b_{it} s_{it} + \sum_{j=1}^{5} \mathrm{c}_{j\to i,t} \mathrm{ex}_{j\to i,t} + \sum_{k\in\Omega_i} u_k g_{kt} + \sum_{j=1}^{4} e_{ij} \mathrm{df}_{ijt} \Big] \\
& + \sum_{j=1}^{4} c_{j\to 5,t} \mathrm{ex}_{j\to 5,t} \Big\} \\
\text{s.t. } & \forall t = 1, \dots, T \\
& v_{it} + s_{it} + q_{it} - v_{i,t-1} = a_{it}, \forall i, \\
& q_{it} + \sum_{k\in\Omega_i} \mathrm{g}_{kt} + \sum_{j=1}^{4} \mathrm{df}_{ijt} - \sum_{j=1}^{5} \mathrm{ex}_{i\to j,t} + \sum_{j=1}^{5} \mathrm{ex}_{j\to i,t} = d_{it}, \forall i, \\
& \sum_{j=1}^{4} \mathrm{ex}_{j\to 5,t} = \sum_{j=1}^{4} \mathrm{ex}_{5\to j,t}, \\
& 0 \le v_{it} \le \bar{v}, 0 \le q_{it} \le \bar{q}_i, \forall i, \\
& \underline{g}_k \le g_{kt} \le \bar{g}_k, \forall k, \\
& 0 \le df_{ijt} \le \bar{df}_{ij}, 0 \le ex_{i\to j,t} \le \bar{ex}_{i\to j}, \forall i, \forall j, \\
& v_{i,0} \text{ are given}, a_{i,1} \text{ are deterministic}, \forall i.
\end{aligned}
$$

As shown in (Shapiro et al., 2013) and (Löhndorf & Shapiro, 2019), the right hand side energy inflow process $\{a_{it}\}$ can be modeled as VAR*, making the true problem continuous and stage-wise dependent. The problem can then be discretized using either the Time Series or the Markov chain approximation approaches with different fineness. In Table 3, we provide a summary of the construction and solving of the discretized problems. The first column gives the discretization methods and granularity of discretization. Columns 2 and 3 show the number of sample paths and the time cost to train the approximating Markov chain. Columns 4 and 5 report the number of iterations and the time cost for performing SDDP algorithm. Column 6 reports the total time in getting the policies. The last two columns report the statistical gaps and deterministic bounds. It manifests that SA/RSA/SAA policies consume much longer time per iteration to obtain than TS policies. Part of the reasons is due to the high memory consumption of the Markov chain approach. On the other hand, SA/RSA/SAA policies produce much smaller gaps.

In particular, Figure 2 visually compares the evolution of gaps for SA100 and TS100. The approximate confidence intervals for the expected policy value are computed based on the policy values obtained from six forward passes per iteration. The statistical upper bounds are then the upper end of the approximate confidence intervals. To better depict the trend of the statistical upper bounds, the mean value of the policy values obtained from the six forward passes U_i for each iteration $i = 1, \dots, N$ is smoothed by a convex curve $x_i, i = 1, \dots, N$ that solves the following quadratic program proposed in

(Löhndorf & Shapiro, 2019),

$$\min_{x_1,\ldots,x_N} \left\{ \sum_{i=1}^{N} (x_i - U_i)^2 : x_i + x_{i-2} \geq 2x_{i-1}, x_{i-1} \geq x_i \right\}$$

Table 3: SDDP implementation results for discretized problems

model	#iter. MCA(M)	MCA (sec.)	#iter. SDDP	SDDP (sec.)	total (sec.)	gap	bound ($M)
SA50	0.1	300	100	2,163	2,463	11.6%	185.0
SA100	1	4,039	100	4,771	8,810	13.0%	186.7
RSA50	0.1	346	100	2,142	2,488	10.7%	181.3
RSA100	1	4,513	100	4,829	9.342	12.6%	184.2
SAA50	0.1	10,521	100	2,122	12,643	11.0%	177.2
TS50	/	/	293	2,412	2,412	27.0%	186.2
TS100	/	/	400	7,724	7,724	23.3%	188.6

Fig. 2: Comparison of SA100 (on the left) and TS100 (on the right)

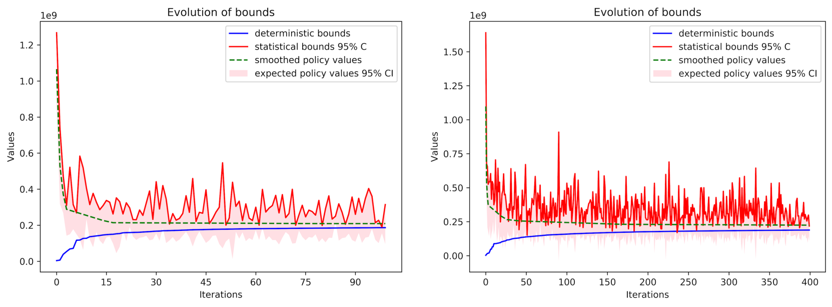


Table 4: One-sided p-values for pairwise comparison of policies. P-values are written in the lower triangular part if the sample mean value of the row policy is lower than that of the column policy (i.e., the row policy is better than the column policy).

	SA50	SA100	RSA50	RSA100	SAA50	TS50	TS100
SA50			<0.001	<0.001	<0.001		
SA100	<0.001		<0.001	<0.001	<0.001		
RSA50					<0.001		
RSA100			<0.001		<0.001		
SAA50							
TS50	0.127	0.249	0.022	0.083	0.002		
TS100	0.025	0.068	0.002	0.013	<0.001	<0.001	

Table 5: Policy evaluation

policy	CI for approx. model ($M)		CI for true problem ($M)	
SA50	201.1	206.5	228.3	236.5
SA100	205.2	211.0	227.0	235.1
RSA50	195.3	200.8	230.8	239.1
RSA100	201.4	207.3	229.1	237.1
SAA50	191.6	196.8	233.6	242.1
TS50	228.8	236.4	225.3	233.0
TS100	225.1	232.5	223.1	230.6

In Table 4, we pairwisely compare the obtained seven policies with 3000 Monte Carlo simulations. In Table 5 we provide 95% confidence intervals for both the discretized and true problems. We make the following observations. First, the granularity of discretization does matter. In this case, 50 samples per stage are not sufficient for both the TS approach and MCA. Second, given the same granularity of discretization, TS policies dominates the MCA policies, although TS policies are obtained by solving the discretized problem with a larger optimality gap. Third, the SA approach for MCA seems to dominate the RSA and SAA approaches.

We now turn to consider an integer version of the problem by adding a thermal security constraint. That is, whenever the stored energy is below certain level, thermal energy must be produced to ensure enough operational maneuver options for hydro plants and for electrical network reliability. This can be achieved by including binary variables as follows,

$$v_{it} \geq z_i v_i^*, \sum_{k \in \Omega_i} g_{kt} \geq (1 - z_i) g_i^*, g_i^* \text{ and } v_i^* \text{ given}, z_i \in \{0, 1\}, \forall i.$$

Table 6 summarizes solutions to the problem when the number of periods (stages) is small. In column 4, both of the extensive solver and the SDDiP solver provide lower bounds for the discretized problems. In column 5 for $T = 2, 3$, both of the solvers provide upper bounds for the discretized problems; more specifically, the extensive solver gives the best-so-far objective found in branch and bound; the SDDiP solver gives the exact expected policy values by exhaustively evaluating the policy under all scenarios (this is doable for tiny problems since the number of total scenarios is small). In column 5 for $T = 6$, the SDDiP solver provides a confidence interval of the expected policy values with 3000 Monte Carlo simulations. All discretized problems are constructed by the TS approach with 100 i.i.d. samples for each stage. Observe that the extensive solver quickly becomes obsolete with an increasing number of stages. But it still validates the correctness of the SDDiP solver when the number of stages is small (the number of total scenarios is 10^2 for $T = 2$ and 100^3 for $T = 3$). Comparing the solutions given by different types of cuts, Strengthened Benders' cut and Lagrangian cut are able to reduce the optimality gap (column 7) dramatically for discretized problems. Surprisingly, as shown in the last column, their benefits to the true problem are not observable in this case.

Table 6: Summary of solving a mix-integer power system problem

solver	cut	T	LB	UB/ CI appr.	#iter. SDDiP	gap	time (sec.)	CI true
Ext.	/	2	1,623,203	1,623,203	/	0.00%	<1	/
SDDiP	B	2	1,623,203	1,623,203	10	0.00%	<1	1,624,384 1,626,363
Ext.	/	3	3,078,162	3,084,143	/	0.19%	1000	/
SDDiP	B	3	2,110,928	3,074,398	303	45.64%	15	3,071,920 3,079,293
SDDiP	SB	3	2,998,418	3,074,398	57	2.53%	15	3,071,920 3,079,293
SDDiP	LG	3	2,999,141	3,074,398	19	2.51%	15	3,071,926 3,079,299
SDDiP	B	6	3,116,866	4,999,105 5,013,830	345	60.86%	60	5,000,001 5,022,540
SDDiP	SB	6	4,946,515	5,002,811 5,056,414	53	2.22%	60	5,004,602 5,081,744
SDDiP	LG	6	4,946,263	5,005,058 5,060,038	33	2.30%	60	5,005,753 5,083,226

16.2 Multi-period portfolio optimization

A portfolio manager oversees multiple assets ($i = 1, \dots, N$) and a bank account ($i = N+1$). For a specified number T of stages, the manager wants to maximize his utility by dynamically rebalancing the portfolio. Let $\{r_{it}\}$ be the return process of asset i. At the end of each period, the position of i^{th} asset x_{it} equals the start position $x_{i,t-1}$, plus the realized return $r_{it}x_{i,t-1}$ during the period, plus the newly long positions b_{it}, minus the newly short positions s_{it}. Transaction costs are f_b, f_s for buying and selling respectively. The capital in the bank account will be adjusted accordingly.

We consider a simple asset pricing model that decomposes the excess return as the return explained by Capital Asset Pricing Model (CAPM), alpha and idiosyncratic risk,

$$r_{it} = r_{ft} + \beta_i(r_{Mt} - r_{ft}) + \epsilon_{it}, \text{where } \epsilon_{it} \overset{i.i.d}{\sim} N(\alpha_i, \sigma_i),$$

where $\alpha_i, \beta_i, \sigma_i$ are assumed to be constant. We refer to $\{r_{ft} + \beta_i(r_{Mt} - r_{ft})\}$ as the market-exposure return process and $\{\epsilon_{it}\}$ as the idiosyncratic return process. The market return process $\{r_{Mt}\}$ is modeled as a first-order autoregressive process (AR) with a normal generalized autoregressive conditional heteroscedastic GARCH(1,1) innovation due to (Akgiray, 1989),

$$\begin{aligned} r_{Mt} &= \mu + \phi r_{M,t-1} + \epsilon_{Mt}, \\ \epsilon_{Mt} &= \sigma_{Mt} e_{Mt}, \\ \sigma^2_{Mt} &= \omega + \alpha_1 \epsilon^2_{M,t-1} + \alpha_2 \sigma^2_{M,t-1}, \\ e_{Mt} &\overset{i.i.d}{\sim} \mathcal{N}(0,1). \end{aligned}$$

Note that the above model can be further improved. For example, the considered asset pricing model fails to incorporate important factors such as SMB (small market capitalization minus big) and HML (high book-to-market ratio minus low). A more realistic asset pricing model such as (Fama & French, 2015) can be considered. Furthermore, empirical evidence such as (Bollerslev et al., 1987) have found that the normal GARCH innovation fails to capture the leptokurtosis and heavy tails in financial time series. Nevertheless, our model is sophisticated enough for the purpose of illustrating the MSPPy package, so we will pretend that the AR(1)-GARCH(1,1) is our "true process". In the end, the multi-period optimization program is the following,

$$
\begin{aligned}
\max \quad & U(r_T^\top x_T) \\
\text{s.t.} \quad & \forall t = 1, \dots, T, \forall i = 1, \dots, N, \\
& x_{it} = x_{i,capm,t} + x_{i,idio,t} + x_{i,t-1} + b_{it} - s_{it}, \\
& x_{N+1,t} = (1 + r_{ft}) x_{N+1,t-1} - (1 + f_b) \sum_{i=1}^{N} b_{it} + (1 - f_s) \sum_{i=1}^{N} s_{it}, \\
& x_{i,capm,t} = \big[r_{ft} + \beta_i (r_{Mt} - r_{ft}) \big] x_{i,t-1} \\
& x_{i,idio,t} = \epsilon_{it} x_{i,t-1}, \\
& \epsilon_{it} \overset{i.i.d}{\sim} N(\alpha_i, \sigma_i), \{r_{Mt}\} \sim \mathrm{AR}(1) - \mathrm{GARCH}(1,1), \\
& x_{i0} = 0, x_{N+1,0} = \$100 \\
& b_{it}, s_{it}, x_{it}, x_{N+1,t} \geq 0.
\end{aligned}
$$

We consider $K = 100$ and $T = 12$. The resulting true problem has mixed types of data processes and should be discretized accordingly. The stage-wise independent idiosyncratic return processes $\{\epsilon_{it}\}$ are discretized by SAA with 100 i.i.d. samplings. The Markovian market-exposure return processes $\{r_{ft} + \beta_i(r_{Mt} - r_{ft})\}$ are non-linear and thus should be discretized by MCA rather than the TS approach. Given the fact that the market-exposure return processes are virtually determined by the market return process, we can first discretize the market return process and then arithmetically compute the discretization of the market-exposure return process.

We set the granularity of Markov chain discretization to be 100 for all stages except the first stage. As shown in Figure 3 and Table 7, the Markov chain process gives fairly good approximation of the true market return process. Most of the statistics, including expected return (sample mean), risk (sample standard deviation), 5% value at risk, skewness are similar. But it fails to fully resemble the kurtosis and tails. This behavior is expected since the Markov states are the centers of partitions of the sample space and are not likely to cover the corner of the space.

We obtain five policies by solving the constructed discretized problem for 50 iterations and 3 steps per each iteration under risk measures of different combinations of expectation and AVaR. In particular, the risk neutral discretized problem is solved with a 2.19% optimality gap. Table 8 and Table 9 gives the evaluation results for these policies with 1000 Monte Carlo simulations, in which λ represents the weight of AVaR and α represents the level

Fig. 3: Fan plots of the true process and its Markov chain approximation through SA

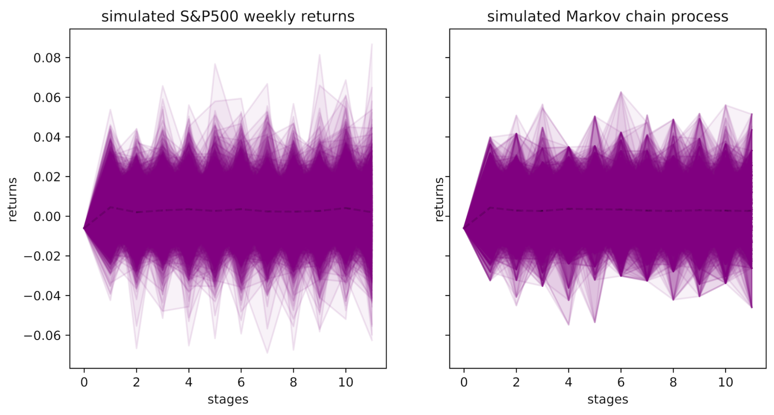


Table 7: Statistics of the cumulative return of the true market return process versus that of its approximating Markov chain process over the whole period (kurtosis is calculated as the fourth standardized moment. For example, kurtosis of the normal distribution is 3).

	expected return (%)	risk (std)	5%VaR ($)	skewness	kurtosis	Sharpe Ratio
True process	2.813	4.791	4.671	0.248	3.707	0.47
Markov chain	2.993	4.916	4.926	0.210	3.311	0.50

of VaR. The introduction of risk measures changes the distribution of the final wealth dramatically. Policy $\lambda(0.75) - \alpha(0.25)$ almost gives a riskless allocation scheme that keeps allocating all the money in the bank account (the final return of a riskless strategy is $(1+0.05\%)^{11} - 1 = 0.551\%$). If we use the return-volatility ratio (the last column) as the performance evaluation metrics, policy $\lambda(0.25) - \alpha(0.25)$ seems to be a promising strategy that beats a passive strategy holding the (simulated) market index. If we also take value-at-risk (column 4) into consideration, policy $\lambda(0.50) - \alpha(0.25)$ is most desirable since it offers both a slightly bigger Sharpe Ratio and a lower value-at-risk than the market index.

Table 8: Policy evaluation for the portfolio optimization discretized problem

policy	expected return (%)	risk (std)	5%VaR ($)	skewness	kurtosis	Sharpe Ratio
Risk neutral	16.538	22.071	17.410	0.538	3.664	0.75
$\lambda(0.25) - \alpha(0.25)$	10.285	6.960	1.068	0.043	2.893	1.48
$\lambda(0.50) - \alpha(0.25)$	4.298	3.392	1.570	-0.205	3.558	1.27
$\lambda(0.75) - \alpha(0.25)$	0.551	<0.001	-0.551	0.101	3.691	/

Table 9: Policy evaluation for the portfolio optimization true problem

policy	expected return (%)	risk (std)	5%VaR ($)	skewness	kurtosis	Sharpe Ratio
Risk neutral	14.425	22.045	19.327	0.407	3.180	0.63
$\lambda(0.25)-\alpha(0.25)$	6.137	7.259	5.756	-0.166	3.595	0.77
$\lambda(0.50)-\alpha(0.25)$	2.640	3.680	3.980	-0.370	3.441	0.57
$\lambda(0.75)-\alpha(0.25)$	0.551	<0.001	-0.551	-0.029	3.816	/

16.3 Airline revenue management

An airline company manages flights between three cities A, B, C. In the middle of the three cities, there is a hub H serves as a transition point. The company wants to determine the seat protection level for all itineraries, fare classes to maximize the revenue over a horizon of T stages. Every stage corresponds to a departure date. Cancellation rates Γ are deterministic. Demands are random.

$$\begin{aligned}
\max\ & \sum_{t=1}^{T}\sum_{i}\sum_{j}[f_{i,j}^{\top}b_{i,j,t}-f_{i,j}^{\top}c_{i,j,t}]\\
\text{s.t. } & \forall t=1,\dots,T\\
& B_{i,j,t}=B_{i,j,t-1}+b_{i,j,t}, C_{i,j,t}=C_{i,j,t-1}+c_{i,j,t}, \forall i,\forall j,\\
& C_{i,j,t}=\lfloor \Gamma_{i,j}B_{i,j,t}+0.5\rfloor, \forall i,\forall j,\\
& \sum_{i,j}A_{k,l}^{i,j}(B_{i,j,t}-C_{i,j,t})\le R_{k,l}, \forall k,\forall l,\\
& b_{i,j,t}\le d_{i,j,t}, \forall i,\forall j,\\
& B_{i,j,0}=0, C_{i,j,0}=0, \forall i,\forall j,\\
& b_{i,j,t}, c_{i,j,t}, B_{i,j,t}, C_{i,j,t}\in\mathbb{Z}^{+}, \forall i,\forall j.
\end{aligned}$$

The index i represents an itinerary from AH, HA, BH, HB, CH, HC, AHB, BHA, AHC, CHA, BHC, CHB. The index j represents a fare class from E1, E2, E3, E4, B1, B2 (Four economy classes, two business classes). The index l represents a flight leg from AH, HA, BH, HB, CH, HC. The index k represents either a economy cabin or a business cabin. In each stage t, the current number of cumulative fulfilled booking (cancellation) requests $B_{i,j,t}(C_{i,j,t})$ is equal to its value in the previous stage $B_{i,j,t-1}(C_{i,j,t-1})$ plus the number of new fulfilled booking(cancellation) requests $b_{i,j,t}(c_{i,j,t})$. $\Gamma_{i,j}$ represents cancellation rate. The number of new fulfilled booking requests $b_{i,j,t}$ can not be larger than the demand $d_{i,j,t}$. The capacity for each cabin and each flight leg is given by $R_{k,l}$. $A_{k,l}^{i,j}$ is one if and only if itinerary i includes flight leg l and fare class j is in cabin k; otherwise it is zero. The revenue of a fulfilled booking request is $f_{i,j}$. We assume full refund for cancellations, thus the cost of cancellations is also $f_{i,j}$.

The demand process $\{d_{i,j,t}\}$ is modeled as a non-homogeneous Poisson Process, due to (Chen & Homem-de Mello, 2010) and (de Boer, Freling, & Piersma, 2002). The number of arrivals (booking requests) $G_{i,j}$ for itinerary i and fare class j over the whole time periods T is assumed to independently follow a Gamma distribution. Gamma distribution is chosen due to empirical evidence and analytical convenience that Gamma distribution is the conjugate prior of the Poisson distribution. The arrival pattern of the booking requests $\beta_{i,j}(t)$ is assumed to follow a Beta distribution to allow flexible shape. The arrival intensity function of booking requests is then given by $\lambda_{i,j}(t) = \beta_{i,j}(t)G_{i,j}$. It follows that the arrival intensity of booking request $d_{i,j}(t)$ over a booking interval $[t_n, t_m]$ is $G_{i,j}[F_{\beta_{i,j}}(t_m) - F_{\beta_{i,j}}(t_n)]$. In the end, demand process is modeled by number of booking requests subtracting cancellation. This is achieved by scaling $G_{i,j}$ with a cancelling rate.

The demand process $d_{i,j}(t)$ is thus 72-dimensional and is Markovian if augmented with $G_{i,j}$. To get an idea of what the true process looks like, we show fan plots of 1000 sample paths of simulated demands from the true process for itinerary AH and all fare classes (E1,E2,E3,E4,B1,B2) in the left column of Figure 4. The bell pattern as shown in the plots suggests different granularity of Markov chain discretization for different stages. We construct three discretized problems by MCA through the SA approach. The parameters of the Markov chains are rounded to the closet integers after training (since demands are integers). In Table 10, we provide summary of the three discretized problems. Columns 2 and 3 show the number of sample paths and time cost to train the Markov chain; the last few columns give the granularity of the Markov chain for each stage.

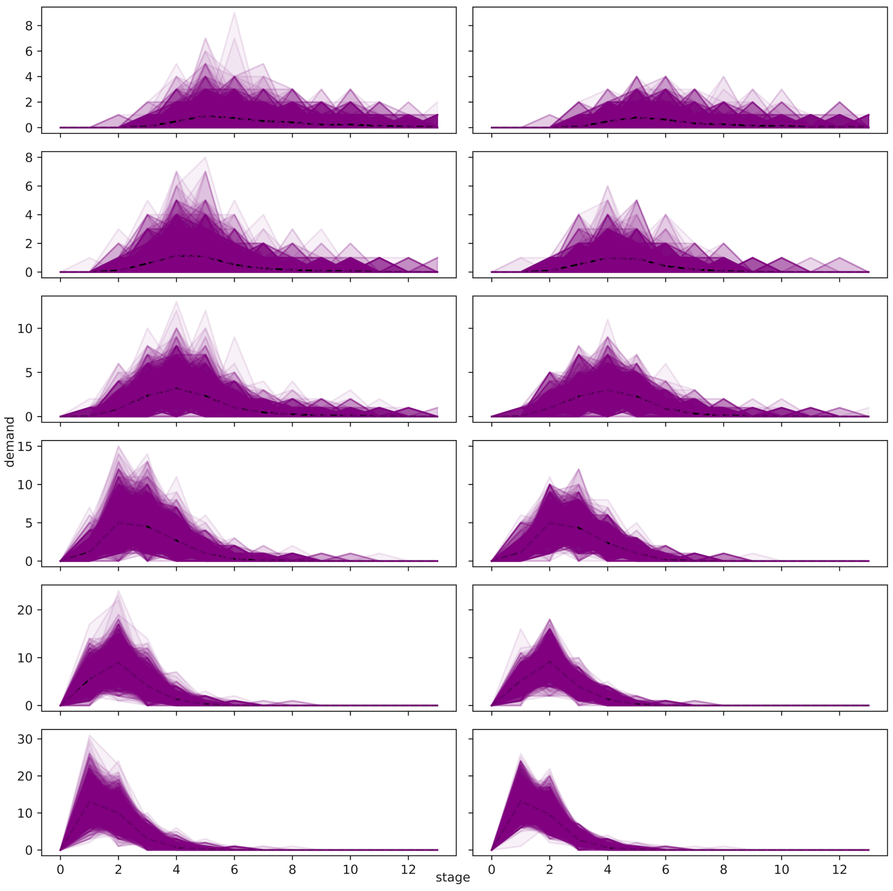
Table 10: Three airline revenue management discretized problems

model	#iter. MCA	time	granularity of discretization for each stage						
			1	2...9	10	11	12	13	14
1	150,000	190	1	100...100	100	100	79	59	40
2	300,000	766	1	200...200	199	197	122	95	50
3	600,000	3,247	1	400...400	396	387	237	168	79

In the right column of Figure 4, we show fan plots of 1000 sample paths of demands simulated from discretized problem 1. The true processes and the corresponding Markov chain processes look similar.

We solve the three discretized problems using the SDDiP solver. We implement both Benders' cut and Strengthened Benders' cut for 100 iterations with a single step for each iteration. The MIP tolerance is the default value of 10^{-4}. Information about the construction and evaluation of the six policies (with 3000 Monte Carlo simulations) are summarized in table 11. Compared to the Benders' cut, the Strengthened Benders' cut reduces the optimality gap (column 4) for discretized problems by around 1% and it increases the expected revenue marginally (the last column). In addition, given the same type of cut, all three discretized problems provide similar expected revenues; hence, discretized problem 3 seems to already provide us with a reasonable

Fig. 4: Fan plots of the true demand process (on the left) and its Markov chain approximations (on the right)

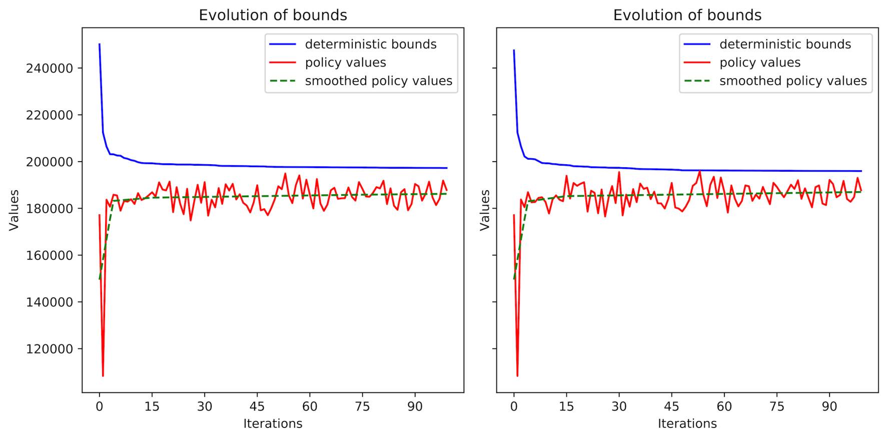


good policy. Figure 5 visualizes the evolution of the bounds for discretized problem 3.

Table 11: Solving the three airline revenue management discretized problems and evaluation results

model	cut	time (sec.)	gap	bound ($K)	95% CI approx. ($K)		95% CI true ($K)	
1	B	807	5.74%	200.37	188.86	189.16	189.55	189.94
1	SB	5,086	4.79%	199.19	189.64	189.94	190.24	190.63
2	B	1,573	5.90%	196.48	184.89	185.23	189.50	189.88
2	SB	9,913	4.95%	195.26	185.60	185.95	190.23	190.62
3	B	3,181	5.83%	197.24	185.74	186.08	189.56	189.95
3	SB	18,232	4.80%	195.94	186.53	186.87	190.44	190.83

Fig. 5: Evolution of the bounds for Bender's cuts (on the left) and Strengthed Bender's cuts (on the right)



The above power system planning with integers, portfolio optimization and airline revenue management problems were implemented using Python 3.7.1 and Gurobi 8.1.0 on a Mac Pro (macOS Sierra 10.12.6) with a 2.7GHz Intel Core i5-5257U processor. The power system planning problem without integers was implemented in two separate steps: the Markov chain approximation part was implemented using Python 3.7.1 on a Mac Pro (macOS Sierra 10.12.6) with a 2.7GHz Intel Core i5-5257U processor; the SDDP algorithm part was implemented using Python 3.5.6 and Gurobi 7.0.2 on a Linux (Red Hat 7.6) with a 2.3GHz Intel Xeon E5-2630 processor.

17 Innovation and performance of the MSPPy package

Throughout previous sections, we have seen that a crucial step is to evaluate the policy of the true problem. As far as we know, none of the existing software packages provides good answer to that. All of them starts with formulation of a finite discrete problem as (9) or (10). In other words, they make the assumption that the number of scenarios is finite. In real world applications, finiteness barely holds. In other cases where the number of scenarios is finite but huge, true problem is intractable and also needed to be discretized. Consequently, the existing software packages will only be able to solve discretized problems. However, solving the discretized problem, even to optimality, guarantees nothing about its performance for the true problem. It also raises another important question and makes it more problematic. What is a proper way of discretization to get a reasonable discretized problem in the first place? A proper way may include guidance on how granular should the discretization be and how to chose among different discretization techniques. Unfortunately, the existing software packages will not help you with that. Therefore, to understand the whole picture of a multistage stochastic program, users of existing software packages should create their own interface to make discretization and to evaluate policy on their true problem. As a result, they have to switch interfaces back and forth, which is highly inconvenient.

The main innovation of the MSPPy package is that it offers the all-in-one functionality to build and discretize the true problem, solve the discretized problem and evaluate the computed policy on the true problem in the same interface. Users are able to clearly understand how policies perform on the true problem and choose the best among them through statistical analysis as we did in the previous sections. The MSPPy package is also the first one to implement a rolling solver (based on rolling horizon algorithm)

The MSPPy package chooses Python as its main interface since Python is one of the sky-rocketing programming languages. The abundant analytic tools and libraries make optimization, statistical analysis, and visualization very easy. The MSPPy package leverages the NumPy package to make fast numerical computation and the Pandas package to make visualization and output readable result. Gurobi (L. Gurobi Optimization, 2018) is chosen as the external solver since some empirical evidence have shown, Gurobi solver performs better than other solvers in various cases, e.g., (I. Gurobi Optimization, 2018). While the binding with Gurobi also brings in certain limitations on the types of subproblems the MSPPy package can solve. Given the fact that up to now Gurobi solver is not able to solve generic convex programs, the MSPPy package only supports LP, MIP, QP subproblems. On the other hand, SDDP.jl leverages the JuMP package (Dunning, Huchette, & Lubin, 2017), which provides a generic interface of various solvers. As a result, SDDP.jl and its extension SDDiP.jl can also solve generic convex subproblems. Nevertheless, it is very rare to have a non-quadratic convex subproblems.

The MSPPy package largely inherits the syntax in the Gurobi python interface. Hence the high-level modeling capability of Python is retained. SDDP.jl, on the other hand, leverages the macro-programming provided by JuMP, has a different modeling benefit. We compare the performance of SDDP.jl and MSPPy by implementing the simplified version of the Brazilian hydro-thermal power system example introduced in section 16.1.

For comparison convenience, here we simply model energy inflows as stage-wise independent discrete stochastic processes. The sample space is composed of historical monthly inflows (around 100). For simplicity, The tiny costs for water spillage and water exchanges are ignored. The discount factor is set to be 1.

Table 12 gives implementation result of the problem after running both the packages for 200 seconds for repetitively ten times. In column 2, we report the number of processes the solver use; column 3 shows the average lower bounds. As shown, the MSPPy package is more efficient than SDDP.jl in this case. Part of the reasons is because the cost of additional abstraction layer JuMP includes to adapt to various solvers. This implementation was done using Python 3.5.0, Julia 0.6 and Gurobi 7.0.2 on a Mac Pro (macOS Sierra 10.12.6) with a 2.7 GHz Intel Core i5-5257U processor.

Table 12: Comparison of MSPPy and SDDP.jl

solver	# processes	lower bound ($M)
SDDP.jl	1	291.8
SDDP.jl	3	292.6
MSPPy	1	293.1
MSPPy	3	294.4

References

Akgiray, V. (1989). Conditional heteroscedasticity in time series of stock returns: Evidence and forecasts. *Journal of business*, 55–80.

Bally, V., & Pages, G. (2003). A quantization algorithm for solving multi-dimensional discrete-time optimal stopping problems. *Bernoulli*, *9*(6), 1003–1049.

Bollerslev, T., et al. (1987). A conditionally heteroskedastic time series model for speculative prices and rates of return. *Review of economics and statistics*, *69*(3), 542–547.

Cambier, L. (2018). Fast. *URL https://github.com/leopoldcambier/FAST*.

Chen, L., & Homem-de Mello, T. (2010). Re-solving stochastic programming models for airline revenue management. *Annals of Operations Research*, *177*(1), 91–114.

Dantzig, G. B., & Infanger, G. (1993). Multi-stage stochastic linear programs for portfolio optimization. *Annals of Operations Research*, *45*(1), 59–76.

de Boer, S. V., Freling, R., & Piersma, N. (2002). Mathematical programming for network revenue management revisited. *European Journal of Operational Research*, *137*(1), 72–92.

Dowson, O., & Kapelevich, L. (2017). Sddp. jl: A julia package for stochastic dual dynamic programming. *Optimization Online*.

Dunning, I., Huchette, J., & Lubin, M. (2017). Jump: A modeling language for mathematical optimization. *SIAM Review*, *59*(2), 295-320. doi: 10.1137/15M1020575

Fama, E. F., & French, K. R. (2015). A five-factor asset pricing model. *Journal of financial economics*, *116*(1), 1–22.

Gevret, H., Langrené, N., Lelong, J., Warin, X., & Maheshwari, A. (2018). *Stochastic optimization library in c++* (Unpublished doctoral dissertation). EDF Lab.

Gurobi Optimization, I. (2018). Gurobi 7.5 performance benchmarks.

Gurobi Optimization, L. (2018). Gurobi optimizer reference manual, url: http://www.gurobi.com.

Kapelevich, L. (2018). Sddip. *URL https://github.com/lkapelevich-/SDDiP.jl*.

Leclère, V., Gérard, H., Pacaud, F., & Rigaut, T. (2018). Stochdynamicprogramming. jl a julia library for multistage stochastic optimization.

Legat, B. (2018). Structdualdynprog. jl (2018). *URL https://github.com/blegat/StructDualDynProg. jl*.

Lemaréchal, C., Nemirovskii, A., & Nesterov, Y. (1995). New variants of bundle methods. *Mathematical programming*, *69*(1-3), 111–147.

Lloyd, S. (1982). Least squares quantization in pcm. *IEEE transactions on information theory*, *28*(2), 129–137.

Löhndorf, N., & Shapiro, A. (2019). Modeling time-dependent randomness in stochastic dual dynamic programming. *European Journal of Operational Research*, *273*(2), 650–661.

Mahmutoğulları, A. I., Ahmed, S., Çavuş, Ö., & Aktürk, M. S. (2019). The value of multi-stage stochastic programming in risk-averse unit commitment under uncertainty. *IEEE Transactions on Power Systems*, *34*(5), 3667–3676.

Möller, A., Römisch, W., & Weber, K. (2008). Airline network revenue management by multistage stochastic programming. *Computational Management Science*, *5*(4), 355–377.

Nemirovski, A., Juditsky, A., Lan, G., & Shapiro, A. (2009). Robust stochastic approximation approach to stochastic programming. *SIAM Journal on optimization*, *19*(4), 1574–1609.

Pereira, M. V., & Pinto, L. M. (1991). Multi-stage stochastic optimization applied to energy planning. *Mathematical programming*, *52*(1-3), 359–375.

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

Shapiro, A., Dentcheva, D., & Ruszczyński, A. (2009). *Lectures on stochastic programming: modeling and theory.* SIAM.

Shapiro, A., Tekaya, W., da Costa, J. P., & Soares, M. P. (2011). *Report for technical cooperation between georgia institute of technology and ons-operador nacional do sistema elétrico.* Citeseer.

Shapiro, A., Tekaya, W., da Costa, J. P., & Soares, M. P. (2012). Final report for technical cooperation between georgia institute of technology and ons-operador nacional do sistema eletrico. *Georgia Tech ISyE Report*.

Shapiro, A., Tekaya, W., da Costa, J. P., & Soares, M. P. (2013). Risk neutral and risk averse stochastic dual dynamic programming method. *European journal of operational research*, *224*(2), 375–391.

Zou, J., Ahmed, S., & Sun, X. A. (2018). Stochastic dual dynamic integer programming. *Mathematical Programming*, 1–42.