



Bi-objective multistage stochastic linear programming

O. Dowson¹ · D. P. Morton¹ · A. Downward²

Received: 31 March 2020 / Accepted: 22 July 2022 / Published online: 4 August 2022
© Springer-Verlag GmbH Germany, part of Springer Nature and Mathematical Optimization Society 2022

Abstract

We propose an algorithm for solving a class of bi-objective multistage stochastic linear programs. We show that the cost-to-go functions are saddle functions, and we exploit this structure, developing a new variant of the stochastic dual dynamic programming algorithm. Our algorithm is implemented in the open-source stochastic programming solver `SDDP.jl`. We apply our algorithm to a hydro-thermal scheduling problem using data from the Brazilian Interconnected Power System. We also propose and implement a computationally tractable heuristic for bi-objective stochastic convex programs.

Keywords Bi-objective · Non-convex · Stochastic dual dynamic programming · Stochastic programming

Mathematics Subject Classification 90C15 (Stochastic programming) · 90C29 (Multi-objective and goal programming)

1 Introduction

Multi-objective linear programs are well-studied in the literature; see, e.g., [16] and the references therein. A standard formulation of a multi-objective linear program is:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & Cx \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0, \end{aligned} \tag{1}$$

✉ O. Dowson
o.dowson@gmail.com

D. P. Morton
david.morton@northwestern.edu

A. Downward
a.downward@auckland.ac.nz

¹ Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL, USA

² Department of Engineering Science, University of Auckland, Auckland, New Zealand

where C is an $o \times n$ matrix, A is an $m \times n$ matrix, b is an m -dimensional vector, and x is an n -dimensional decision vector. Here and throughout, objective vectors are ranked according to a partial ordering such that $Cx_1 \leq Cx_2 \iff Cx_2 - Cx_1 \in \mathbb{R}_+^o$. Unlike a single-objective program, where a solution is reported via a single vector for x , a solution to a multi-objective program is a set of *Pareto-optimal* vectors, whose corresponding objective vectors are indistinguishable under the partial ordering.

Definition 1 Let \hat{x} be a feasible solution of (1). Then, \hat{x} is called *Pareto-optimal* if there is no feasible x such that $Cx \neq C\hat{x}$ and $Cx \leq C\hat{x}$; $C\hat{x}$ is called *non-dominated*. The set of non-dominated objective vectors is called the *Pareto frontier*. A non-dominated objective vector is called *supported* if it cannot be represented by a strict convex combination of other non-dominated objective vectors.

Typically, multi-objective optimization is applied to deterministic problems, using tools such as the epsilon-constraint and weighted-sum methods to find the Pareto frontier, or an approximation thereof. Both of these approaches convert the multi-objective problem into a family of single-objective problems: the epsilon-constraint method by shifting all-but-one of the objectives to constraints, and the weighted-sum by replacing the objective by a convex combination of each of the scalar objectives; see Ehrgott [16] for details. Less often, multi-objective optimization is applied to stochastic problems, e.g., [21]. This paper focuses on the special case of multistage stochastic linear programs with two objectives.

Multistage stochastic programming is a way of modeling a class of sequential decision problems under uncertainty. One application of multistage stochastic programming that has received much attention in the literature involves hydro-thermal scheduling, in which we seek a policy for controlling the generation of electricity from hydro and thermal generators under inflow uncertainty; e.g., [22, 25, 27]. In general, hydro generation is preferred because it has a lower short-run marginal cost (and environmental impact) than thermal generation technologies such as coal. However, if too much water is used for generation, then there may be insufficient water in the reservoirs to meet demand in future periods, requiring load to be shed at high cost. This results in a trade-off for the decision maker between the short-run thermal cost and the shortage cost of load shedding. These competing objectives must be reconciled in some manner.

In the past ten years, considerable research has focused on using so-called *coherent risk measures* [1] as a way to balance these competing objectives. Instead of minimizing the expected cost over all outcomes, coherent risk measures allow the practitioner to optimize some function that emphasizes the tail of the distribution, e.g., involving 10% of the outcomes with the highest costs. While such an approach has desirable mathematical properties and is computationally tractable, it poses challenges in choosing a risk measure and in interpreting the solution for a decision maker. (See [26] and the references therein for more on this topic.)

Our paper takes a different approach. Instead of using coherent risk measures to control undesirable outcomes, we propose solving a bi-objective multistage stochastic optimization problem in which we directly derive the Pareto frontier. As a result, instead of finding a single “optimal” policy, we are able to provide a range of poli-

cies, each with different characteristics, in order for the decision maker to make an interpretable decision about how to trade-off the competing objectives.

We are not the first to advocate for a multi-objective approach to solving stochastic programs. Readers are directed to [21] for a thorough survey on the topic. Examples of practical applications include [7, 30, 31]. Notably, all of these papers use the epsilon-constraint method to deal with multiple objectives, instead of the weighted-sums approach that we take. We note that we do not address multi-objective integer programs; see, e.g., [6] and references therein.

In recent work, the authors of [23] solve a bi-objective problem using Benders decomposition. However, they transform the problem into single-objective model with a fixed weight, so they do not find the full Pareto frontier. We also highlight the work of [24], which derives a bi-objective algorithm in the context of Dantzig–Wolfe decomposition. Their work is interesting given the similarities between Dantzig–Wolfe decomposition (column generation) and Benders decomposition (row generation), but the results are not directly comparable since they provide no proof of convergence.

Our main contributions in this paper are three-fold:

- (i) an extension of Benders decomposition to bi-objective linear programs;
- (ii) the formalization of bi-objective multistage stochastic programming, along with notions such as Pareto-optimal policies; and
- (iii) an algorithm to efficiently find the Pareto frontier of a bi-objective multistage stochastic linear program, for which we provide an open-source implementation in `SDDP.jl` [13].

The rest of this paper is laid out as follows. In Sect. 2, we begin with some preliminary discussion of the bi-objective simplex method and stochastic dual dynamic programming algorithm [25, 29, 32]. Then, in Sect. 3, we describe a hybrid algorithm that is able to solve bi-objective linear programs using Benders decomposition. In that section we also provide finite convergence results. Next, in Sect. 4, we extend the results from our deterministic Benders setting to bi-objective multistage stochastic programs. In particular, the algorithm we propose is an extension of the stochastic dual dynamic programming algorithm. We conclude with numerical examples in Sect. 5.

2 Preliminaries

2.1 The bi-objective simplex method

In this section we introduce the bi-objective simplex method. For a thorough treatment of the subject, readers are directed to [16, Ch. 6].

Consider a linear program as in problem (1) with two objectives defined by appropriately sized vectors of objective coefficients c_1 and c_2 such that $C = [c_1 \ c_2]^T$. One solution approach to find the Pareto frontier is the weighted-sums method. This method reformulates problem (1) into a single-objective problem by taking a convex combination of the two objectives:

$$\begin{aligned}
 LP(\lambda) = \min_x & \lambda c_1^\top x + (1 - \lambda)c_2^\top x \\
 \text{s.t. } & Ax = b \\
 & x \geq 0,
 \end{aligned} \tag{2}$$

for $\lambda \in [0, 1]$. Since λ appears as an objective coefficient in a minimization linear program, $LP(\lambda)$ is piecewise-linear and concave with respect to λ . Moreover, since (2) is a finite-dimensional linear program, the solution of (2) over $\lambda \in [0, 1]$ can be characterized by a finite number of basic feasible solutions.

The formulation of problem (2) naturally extends to more objectives using a convex combination of weights. We focus on the bi-objective case for simplicity, and because we only establish convergence of our solution algorithm in the bi-objective case.

Before we proceed, it is useful to state a well-known result from the literature.

Theorem 1 *A feasible solution x of problem (1) is Pareto-optimal if and only if it is an optimal solution of (2) for some value of $\lambda \in (0, 1)$.*

Proof See, e.g., [17, Theorem 1]. □

Because problem (1) is a finite-dimensional multi-objective linear program, there are a finite number of supported non-dominated objective vectors. Moreover, the Pareto frontier can be characterized by the convex hull of the supported non-dominated objective vectors. Therefore, to solve problem (1), it is sufficient to find the finite number of supported non-dominated objective vectors and a set of corresponding solution vectors x .

We call the set $\{(\lambda, LP(\lambda)) \in [0, 1] \times \mathbb{R}\}$ *weight-space*, and extreme points in weight-space are values of λ that are kink points of the piecewise-linear function $LP(\lambda)$. By construction, extreme points in weight-space correspond to facets of the Pareto frontier, and so at every extreme point in weight-space there are multiple optimal primal solutions. To overcome numerical issues associated with multiple optimal solutions, we make the following two assumptions throughout the remainder of this paper.

Assumption 1 We have a solver that, given a linear program:

$$\min_{Ax=b, x \geq 0} c^\top x,$$

always returns the same optimal basis for any fixed choice of c , A , and b , regardless of the preceding order of operations.

Assumption 2 Given the weighted-sum linear program (2), assume we have a solver that: when solving at $\lambda = 0$, always returns a solution that first minimizes $c_2^\top x$ and then minimizes $c_1^\top x$ with $c_2^\top x$ fixed to its optimal value; and when solving at $\lambda \in (0, 1]$, always returns a solution that first minimizes $\lambda c_1^\top x + (1 - \lambda)c_2^\top x$ and then minimizes $c_2^\top x$ with $\lambda c_1^\top x + (1 - \lambda)c_2^\top x$ fixed to its optimal value.

In most linear programming solvers, Assumption 1 can be ensured—to the detriment of performance—by choosing a deterministic solution method (e.g., primal simplex)

and always clearing cached solutions to prevent the solver from warm-starting from a previous optimal basis or partial solution. The first part of Assumption 2 is necessary to ensure that when we compute $LP(0)$, we obtain a Pareto-optimal solution. The second part of Assumption 2 gives a similar result for all other values of λ . Both cases can be achieved in a linear programming solver by first solving with respect to problem (2)'s objective for the given λ , fixing that objective as a constraint, and then solving with respect to the modified objective.

The bi-objective simplex method begins by solving (2) at $\lambda = 1$ to find the solution that minimizes the $c_1^\top x$ objective. As a result, we obtain an optimal basis, and by Assumption 2, the basis corresponds to a non-dominated objective vector. Since λ only appears in the objective, this basis remains feasible for all values of λ . However, as we decrease λ , there will, in general, come a value at which a new basis becomes optimal. The new value can be computed in the spirit of a standard simplex pivot as follows.

For a given a value of $\lambda \in (0, 1]$, we have a corresponding optimal basis (see, e.g., [9]); we denote the set of basic variables by \mathcal{B} , and the corresponding columns of the A matrix by B . Moreover, we denote the set of non-basic variables by \mathcal{N} with corresponding columns N . Then, assuming a single objective with objective coefficients c , the reduced costs associated with the non-basic variables can be calculated as follows:

$$\bar{c} = c_{\mathcal{N}} - c_{\mathcal{B}}^\top B^{-1}N,$$

where $c_{\mathcal{X}}$ indicates the components of the vector c associated with the set \mathcal{X} .

Using the reduced costs for each objective, we can compute the minimum value that λ can take in order for \mathcal{B} to remain an optimal basis using the following rule:

$$\lambda^- = \max_{i \in \mathcal{I}} \frac{-e_i^\top \bar{c}_2}{e_i^\top \bar{c}_1 - e_i^\top \bar{c}_2}, \tag{3}$$

where e_i is the i -th unit vector, and \mathcal{I} is the set of non-basic columns with a non-negative reduced cost for the first objective and a negative reduced cost for the second objective, i.e., $\mathcal{I} = \{i \in \mathcal{N} : e_i^\top \bar{c}_1 \geq 0, e_i^\top \bar{c}_2 < 0\}$. If $\mathcal{I} = \emptyset$, then $\lambda^- = 0$.

If $\lambda^- = \lambda$, then there is another optimal solution at the vertex. In this case, we must perform a sequence of simplex pivots until we find a basis for which $\lambda^- < \lambda$ using standard techniques for dealing with degeneracy in the simplex method, e.g., [5, 35]. As a short-hand, we denote the procedure carried out by (3) (with potential additional pivots to find a smaller value of λ^-) to update the value of λ by $\lambda \leftarrow \Lambda(\lambda, \mathcal{B})$.

Once a new value for λ is obtained that represents the next—in decreasing order—possible value of λ at which a new basis may become optimal, problem (2), i.e., $LP(\lambda)$, can be re-solved to obtain the new optimal basis. This process is repeated until $\lambda = 0$, and the bi-objective simplex algorithm terminates yielding a necessarily finite sequence of λ values and a corresponding sequence of basic feasible solutions. The basic feasible solutions can be used to construct a set of non-dominated objective vectors to define the Pareto frontier. Because of the presence of multiple optimal solutions we will not, in general, have found *all* Pareto-optimal solutions, x ; however, we will have found *all* supported non-dominated objective vectors, Cx . By taking the

convex combination of adjacent supported non-dominated objective vectors, we can recover the entire Pareto frontier.

2.2 Stochastic dual dynamic programming

The main result of this paper is an extension of the stochastic dual dynamic programming algorithm [25] to the bi-objective case. In this section, we introduce the algorithm and associated notation. We follow the terminology and assumptions of Philpott and Guan [29], restricting our attention to multistage stochastic programs with the following properties.

Assumption 3 The set of random noise outcomes Ω_t in each stage $t = 2, 3, \dots, T$ is discrete and finite, with a known probability p_t^ω for each outcome $\omega \in \Omega_t$.

Assumption 4 The random vectors of noise in each stage are mutually independent.

Assumption 5 The optimization problem in each stage is feasible and has a finite optimal solution for all achievable incoming state x_{t-1} and all noise realizations $\omega \in \Omega_t$.

In other works, the stochastic dual dynamic programming algorithm has been extended to general convex programs [18], risk-averse programs [20, 28, 33], and more general stage-structures [12]. However, to focus on ideas, in this paper we present an algorithm for the T -stage, linear, risk-neutral version of the problem.

Consider a T -stage multistage stochastic linear program. As is typical in the literature [29, 32], we can express the problem in a recursive form:

$$\begin{aligned}
 V_1 &= \min_{x_1} c_1^\top x_1 + \mathbb{E}_{\omega \in \Omega_2} [V_2(x_1, \omega)] \\
 \text{s.t. } & A_1 x_1 = b_1 \\
 & x_1 \geq 0,
 \end{aligned} \tag{4}$$

where the cost-to-go from stage $t = 2, 3, \dots, T$ is a function $V_t(x_{t-1}, \omega_t)$ that depends on the incoming state variable x_{t-1} and a realization of a stagewise-independent random noise ω_t :

$$\begin{aligned}
 V_t(x_{t-1}, \omega_t) &= \min_{\bar{x}, x_t} c_t(\omega_t)^\top x_t + \mathbb{E}_{\omega \in \Omega_{t+1}} [V_{t+1}(x_t, \omega)] \\
 \text{s.t. } & \bar{x} = x_{t-1} \\
 & T_t(\omega_t)\bar{x} + W_t(\omega_t)x_t = g_t(\omega_t) \\
 & x_t \geq 0,
 \end{aligned} \tag{5}$$

where $V_{T+1}(\cdot, \cdot) = 0$. Here, c_t and g_t are appropriately dimensioned vectors that depend on ω_t , and T_t and W_t are appropriately dimensioned matrices that depend on ω_t .

For notational simplicity and without loss of generality, we have followed [29] and merged the state and control vectors into a single vector, x . The goal of the agent is to solve model (4), minimizing expected cost.

Stochastic dual dynamic programming is a form of nested Benders decomposition, and so we use the fact that $V_t(\cdot, \omega_t)$ is convex with respect to x_{t-1} to form the approximation:

$$\begin{aligned}
 V_1^K &= \min_{x_1, \theta_1} c_1^\top x_1 + \theta_1 \\
 \text{s.t. } &A_1 x_1 = b_1 \\
 &x_1 \geq 0 \\
 &\theta_1 \geq \alpha_{1,k} + \beta_{1,k}^\top x_1, \quad k = 1, \dots, K \\
 &\theta_1 \geq -M_1,
 \end{aligned} \tag{6}$$

and V_t :

$$\begin{aligned}
 V_t^K(x_{t-1}, \omega_t) &= \min_{\bar{x}, x_t, \theta_t} c_t(\omega_t)^\top x_t + \theta_t \\
 \text{s.t. } &\bar{x} = x_{t-1} \quad [v_t] \\
 &T_t(\omega_t)\bar{x} + W_t(\omega_t)x_t = g_t(\omega_t) \\
 &x_t \geq 0 \\
 &\theta_t \geq \alpha_{t,k} + \beta_{t,k}^\top x_t, \quad k = 1, \dots, K \\
 &\theta_t \geq -M_t.
 \end{aligned} \tag{7}$$

Stochastic dual dynamic programming is an algorithm that iteratively constructs the set of cuts in a manner analogous to Benders decomposition: a *forward pass* computes a sequence of decision vectors x_1, x_2, \dots, x_{T-1} , and a *backward pass* refines the approximation of the cost-to-go function, V_t , at the decision vectors computed on the forward pass. Pseudo-code is given in Algorithm 1.

Algorithm 1: The stochastic dual dynamic programming algorithm.

```

Set  $K \leftarrow 1$ 
while not converged do
  /* Forward pass
  Solve  $V_1^{K-1}$ , i.e., (6), and obtain optimal extreme point primal solution  $\hat{x}_1$ 
  for  $t = 2, 3, \dots, T - 1$  do
    Sample  $\omega_t$  from  $\Omega_t$ 
    Solve  $V_t^{K-1}(\hat{x}_{t-1}, \omega_t)$ , i.e., (7), and obtain optimal extreme point primal solution  $\hat{x}_t$ 
  end
  /* Backward pass
  for  $t = T - 1, T - 2, \dots, 1$  do
    for  $\omega \in \Omega_{t+1}$  do
      Solve  $V_{t+1}^{K-1}(\hat{x}_t, \omega)$  and obtain optimal extreme point dual solution  $\hat{v}_{t+1, \omega}$  and objective value  $\hat{V}_{t+1, \omega}$ 
    end
    Set  $\alpha_{t,K} \leftarrow \mathbb{E}_\omega [\hat{V}_{t+1, \omega} - \hat{v}_{t+1, \omega}^\top \hat{x}_t]$ 
    Set  $\beta_{t,K} \leftarrow \mathbb{E}_\omega [\hat{v}_{t+1, \omega}]$ 
    Add the cut  $\theta_t \geq \alpha_{t,K} + \beta_{t,K}^\top x_t$  to  $V_t^{K-1}$ , creating  $V_t^K$ 
  end
  Set  $K \leftarrow K + 1$ 
end
  
```

Unlike deterministic optimization, a feasible or optimal solution is not a single vector of values for x . Instead, a feasible solution is a *policy*, $\pi = \{\pi_t\}_{t=1}^T$, comprising a set of *decision rules*, with one decision rule, π_t , for each stage t . Each decision rule is a mapping, $\pi_t(x_{t-1}, \omega_t) \rightarrow x_t$, that prescribes an action, x_t , given the incoming state, x_{t-1} , and realization of the uncertainty, ω_t . We restrict attention to policies whose decision rules are feasible with respect to the linear programming dynamics in the constraints of (4)–(5). Under Assumptions 1 and 3–5, a decision rule to construct a policy, π^K , can be formed for stage t by taking the arg min of the approximated subproblem corresponding to V_t^K . With a slight abuse of notation, we denote the expected cost of a policy, π , by $V_1(\pi)$. The optimal value of problem (4) is then $V_1 = V_1(\pi^*)$, where π^* is an optimal policy, i.e., a policy satisfying $V_1(\pi^*) \leq V_1(\pi)$ for all π . A lower bound on $V_1(\pi^*)$ is available after each iteration, K , by solving the approximated first-stage problem (6), so that $V_1(\pi^K) \geq V_1(\pi^*) \geq V_1^K$.

3 Bi-objective Benders decomposition

In this section, we extend Benders decomposition [3, 34] to the bi-objective setting. Benders decomposition is not the main focus of our work; instead, it provides a simplified setting in which to prove necessary building blocks for multistage stochastic programming.

Consider a bi-objective linear program with a diagonal block structure amenable to Benders decomposition:

$$\begin{aligned} \min_{x,y} \quad & Cx + Dy \\ \text{s.t.} \quad & Ax = b \\ & Tx + Wy = g \\ & x, y \geq 0, \end{aligned} \tag{8}$$

where A, T , and W are appropriately sized matrices of finite dimension; $C = [c_1 \ c_2]^\top$, $D = [d_1 \ d_2]^\top$, and b, c_1, c_2, d_1, d_2 , and g are appropriately sized vectors of finite dimension; and x and y are commensurate decision vectors.

We propose to solve this problem via the weighted-sums method, and thus in the following, $\lambda \in [0, 1]$. Combining the weighted-sum problem (2) with the typical first- and second-stage subproblems of Benders decomposition, we obtain:

$$\begin{aligned} V_1(\lambda) = \min_x \quad & \lambda c_1^\top x + (1 - \lambda)c_2^\top x + V_2(x, \lambda) \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0, \end{aligned} \tag{9}$$

where a second-stage subproblem defines $V_2(x, \lambda)$, which depends on the first-stage decision x and the scalarizing weight λ :

$$\begin{aligned}
 V_2(x, \lambda) = \min_{\bar{x}, y} & \lambda d_1^\top y + (1 - \lambda) d_2^\top y \\
 \text{s.t.} & \bar{x} = x \quad [v] \\
 & T\bar{x} + Wy = g \\
 & y \geq 0.
 \end{aligned} \tag{10}$$

Similar to Assumption 5, to ensure that the master and subproblem in our Benders decomposition algorithm are feasible and have finite optimal solutions, we assume the second-stage subproblem (10) has *relatively complete recourse* and is *dual feasible*, and the first-stage feasible region is *nonempty* and *compact*:

Assumption 6 For all $\lambda \in [0, 1]$ and any feasible first-stage decision, x , to problem (9), the second-stage subproblem (10) has nonempty primal and dual feasible regions. In addition, the first-stage feasible region, $\{x : Ax = b, x \geq 0\}$, is compact and nonempty.

We make the first part of this assumption so that we do not need to consider Benders infeasibility cuts; this is required for our subsequent extension to multistage stochastic programs. The second part ensures $V_2(x, \lambda)$ is finite, and holds provided subproblem (10) is dual feasible for $\lambda = 0$ and $\lambda = 1$. Coupled with the third part, we are ensured that the overall problem is feasible, and the forthcoming master problem has a finite optimal solution. The next lemma characterizes properties of the cost-to-go function, $V_2(x, \lambda)$.

Lemma 1 Let $V_2(x, \lambda)$ be defined by subproblem (10), and let Assumption 6 hold. Then, $V_2(x, \lambda)$ is finite, piecewise-linear convex with respect to x on $\{x : Ax = b, x \geq 0\}$, piecewise-linear concave with respect to λ on $[0, 1]$, and each piecewise-linear function comprises a finite number of pieces.

Proof For fixed λ , V_2 is the optimal value of a linear program in which x appears only on the right-hand side. Thus $V_2(\cdot, \lambda)$ is piecewise-linear convex. Moreover, given fixed x , V_2 is again a linear program in which λ appears only in the objective coefficients. Thus, $V_2(x, \cdot)$ is piecewise-linear concave. Finiteness of the number of pieces follows immediately from the finite number of basic solutions to subproblem (10). \square

By Lemma 1, the cost-to-go function is a saddle function. Models with this structure have been the subject of a series of recent papers [2, 11, 14], each of which relies on a dualization property to convert the saddle function into a computationally tractable form. The first step involves constructing the master problem approximation of (9), which involves forming a lower-bounding approximation of the saddle function, $V_2(x, \lambda)$, as follows:

$$\begin{aligned}
 V_1^K(\lambda) = \min_{x, \theta} \max_{\gamma} & \lambda c_1^\top x + (1 - \lambda)c_2^\top x + \sum_{k=1}^K \gamma_k \theta_k - L(\gamma^+ + \gamma^-) \\
 \text{s.t. } & Ax = b \\
 & x \geq 0 \\
 & \theta_k \geq \alpha_k + \beta_k^\top x, \quad k = 1, \dots, K \\
 & \theta_k \geq -M, \quad k = 1, \dots, K \\
 & \sum_{k=1}^K \gamma_k \lambda_k + \gamma^+ - \gamma^- = \lambda, \quad [\mu] \\
 & \sum_{k=1}^K \gamma_k = 1, \quad [\varphi] \\
 & \gamma_k, \gamma^+, \gamma^- \geq 0, \quad k = 1, \dots, K.
 \end{aligned} \tag{11}$$

Here, the outer minimization problem—involving the variables x and θ , and constants α and β —forms an outer approximation of the convex dimension of the cost-to-go function, $V_2(\cdot, \lambda)$, given a fixed value of λ . The inner maximization problem—involving variables γ , constants λ_k , and dual variables μ and φ —forms an inner approximation of the concave dimension of the cost-to-go function, $V_2(x, \cdot)$, given a fixed value of x . Taken together, these two approximations provide a lower bound. The scalar constants M and L are sufficiently large and ensure that the problem has a finite optimal solution.

From the point of view of the inner maximization, θ_k are fixed estimates for $V_2(\lambda_k, x)$, so the inner maximization is a standard formulation for constructing an inner approximation of a uni-variate function. Moreover, θ_k are the only linkages between the min problem and the max problem. Therefore, and because of Assumption 6, the second step of the reformulation is to take the dual of the inner maximization problem to form the following pure minimization problem:

$$\begin{aligned}
 V_1^K(\lambda) = \min_{x, \mu, \varphi} & \lambda c_1^\top x + (1 - \lambda)c_2^\top x + \lambda \mu + \varphi & (12a) \\
 \text{s.t. } & Ax = b & (12b) \\
 & x \geq 0 & (12c) \\
 & \lambda_k \mu + \varphi \geq \alpha_k + \beta_k^\top x, \quad k = 1, \dots, K & (12d) \\
 & -L \leq \mu \leq L & (12e) \\
 & \varphi \geq -M. & (12f)
 \end{aligned}$$

Note that we have projected out the θ_k variables by replacing one set of primal and one set of dual inequalities involving θ_k by a single combined set of inequalities. The simple bounds on μ can be avoided if we initialize the master program with solutions for $\lambda = 0$ and $\lambda = 1$ for some x . Following [11], we call the cuts in this reformulated problem *saddle cuts*, since they form piecewise-saddle function lower bounds in the original primal space.

In order to understand the algorithm that we are about to propose to generate the saddle cuts, it helps to observe two facts. First, given a fixed set of cuts, the master problem is a bi-objective linear program that can be solved via the bi-objective simplex

method. Thus, we can use the update method of Eq. (3) to efficiently modify λ . Second, the cuts generated for a given value of λ are valid for all other values of λ , even if the chosen value of λ is not associated with an extreme point of the weight-space.

Therefore, we propose an algorithm that iteratively applies one iteration of Benders decomposition, and then takes one step along the Pareto frontier using the bi-objective simplex method. If the scalarization weight reaches 0, we reset the scalarization weight to $\lambda = 1$ and begin a new sweep along the frontier. To ensure that we visit all supported non-dominated objective vectors, we compute two steps, one each for the master (λ_m) and subproblem (λ_s), and take the smaller step. This is represented by $\max\{\lambda_m, \lambda_s\}$ in the algorithm with the “max” because we iteratively decrease λ . To ensure convergence, it is also necessary to add a cut at the new scalarization weight, resulting in two cuts being added at each step. The reasons for taking the smaller step and adding both cuts are non-trivial, and so we delay discussion until Theorem 2. Pseudo-code is given in Algorithm 2. In the algorithm’s statement, for short-hand we say, e.g., “solve $V_1^K(\lambda_K)$ ” to mean solve master problem (12) with cuts accumulated through the K -th iteration, and with input $\lambda = \lambda_K$.

Algorithm 2: Bi-objective Benders decomposition algorithm

```

Set  $K \leftarrow 1$ 
while not converged do
  Set  $\lambda_K \leftarrow 1$ 
  repeat
    /* Forward pass */
    Solve  $V_1^{K-1}(\lambda_K)$ , i.e., (12), and obtain optimal extreme point primal solution  $\hat{x}_K$ 
    /* Backward pass */
    Solve  $V_2(\hat{x}_K, \lambda_K)$ , i.e., (10), and obtain optimal extreme point dual solution  $\hat{v}_K$  and
    objective value  $\hat{V}_K$ 
    Set  $\alpha_K \leftarrow \hat{V}_K - \hat{v}_K^\top \hat{x}_K$ 
    Set  $\beta_K \leftarrow \hat{v}_K$ 
    Add the cut  $\lambda_K \mu + \varphi \geq \alpha_K + \beta_K^\top x$  to  $V_1^{K-1}$ , creating  $V_1^K$ 
    /* Update weight and add second cut */
    Solve  $V_1^K(\lambda_K)$  to obtain an optimal basis  $\mathcal{B}_m$  and decision  $\hat{x}_{K+1}$ 
    Set  $\lambda_m \leftarrow \Lambda(\lambda_K, \mathcal{B}_m)$ 
    Solve  $V_2(\hat{x}_{K+1}, \lambda_K)$  to obtain an optimal basis  $\mathcal{B}_s$ 
    Set  $\lambda_s \leftarrow \Lambda(\lambda_K, \mathcal{B}_s)$ 
    Set  $\lambda_{K+1} \leftarrow \max\{\lambda_m, \lambda_s\}$ 
    Solve  $V_2(\hat{x}_{K+1}, \lambda_{K+1})$  and obtain optimal extreme point dual solution  $\hat{v}_{K+1}$  and objective
    value  $\hat{V}_{K+1}$ 
    Set  $\alpha_{K+1} \leftarrow \hat{V}_{K+1} - \hat{v}_{K+1}^\top \hat{x}_{K+1}$ 
    Set  $\beta_{K+1} \leftarrow \hat{v}_{K+1}$ 
    Add the cut  $\lambda_{K+1} \mu + \varphi \geq \alpha_{K+1} + \beta_{K+1}^\top x$  to  $V_1^K$ , creating  $V_1^{K+1}$ 
    Update  $V_1^{K+1}$ , removing cuts that are convex combinations of other cuts
    Set  $\lambda_{K+2} \leftarrow \lambda_{K+1}$ 
    Set  $K \leftarrow K + 2$ 
  until  $\lambda_K = 0$ ;
end

```

Note that the formulation of problems (9) and (10) and the saddle-cut reformulation (12) naturally extend to o objectives using a convex combination of $o - 1$ weights. However, we focus on the bi-objective case because we do not have a convergent algorithm for exploring the multi-dimensional space of λ when $o > 2$.

3.1 Bounds

Unlike single-objective Benders decomposition, the concept of lower and upper bounds in bi-objective Benders decomposition is not straightforward. This is because the choice of the appropriate trade-off in the objective is left to the decision maker. If we knew the location of all extreme vertices in weight-space, we could compute an “expected” lower bound by averaging the lower bound $V_1^K(\lambda)$ at all extreme points λ . However, in initial iterations of the algorithm, we do not know the location of these vertices. Therefore, we need a surrogate measure for the lower bound with two properties: (i) it is non-decreasing as K increases; and (ii) it has a finite maximum value.

As our measure, we choose the integral of the function $V_1^K(\lambda)$ over the domain $\lambda \in [0, 1]$. It is easy to show that this measure has the two properties outlined above since adding a cut can only increase the value of $V_1^K(\lambda)$, and therefore the integral’s area must increase, satisfying i); and since $V_1^K(\lambda) \leq V_1(\lambda)$ for all $\lambda \in [0, 1]$, it is bounded above by the area of the true function, satisfying (ii).

To compute the surrogate lower bound, we perform a sweep along λ space in $V_1^K(\lambda)$ using the weight-update (3). This results in a sequence of weights $\lambda_1, \lambda_2, \dots, \lambda_N$, where $\lambda_1 = 1, \lambda_N = 0$, and $\lambda_i > \lambda_{i+1}$. Then, the surrogate lower bounding area can be easily computed using the trapezoid rule:

$$\underline{V}^K = \sum_{i=1}^{N-1} \left(\frac{V_1^K(\lambda_i) + V_1^K(\lambda_{i+1})}{2} - \underline{Z} \right) (\lambda_i - \lambda_{i+1}), \tag{13}$$

where $\underline{Z} = \min\{V_1^0(0), V_1^0(1)\}$.

Given a value of λ , we can obtain an upper bound on the optimal value of the weighted-sum problem by solving $V_1^K(\lambda)$ to obtain an optimal decision x^* , and then solving $V_2(x^*, \lambda)$ to obtain an optimal decision y^* . This feasible solution (x^*, y^*) defines a linear function:

$$\overline{V}_{x^*, y^*}^K(\lambda) = \lambda(c_1^\top x^* + d_1^\top y^*) + (1 - \lambda)(c_2^\top x^* + d_2^\top y^*), \tag{14}$$

which is an upper bound on $V_1(\lambda)$ at any value $\lambda \in [0, 1]$. A collection of these upper bounding functions can be obtained by computing the set of feasible solutions, denoted \mathcal{X} , one corresponding to each weight $\lambda = \lambda_1, \dots, \lambda_N$ obtained from the lower-bounding sweep. Then, a valid upper bound can be computed by calculating the area between the point-wise minimum of these upper bounding linear functions and the value \underline{Z} :

$$\overline{V}^K = \int_0^1 \min_{(x,y) \in \mathcal{X}} \left\{ \overline{V}_{x,y}^K(\lambda) - \underline{Z} \right\} d\lambda. \tag{15}$$

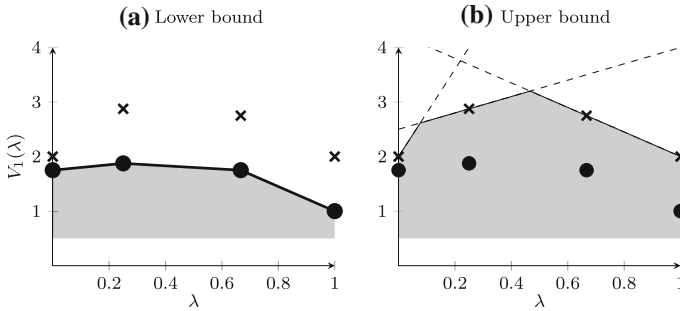


Fig. 1 Visualization of lower and upper bounds in a fictional example. The upper bounding linear functions are shown as dashed lines in Fig. 1b, and the upper bounding area (shaded) is formed by the area between the piecewise minima of these functions on $[0, 1]$ and \underline{Z} , rather than the convex hull of the upper bounding points, which are shown as crosses

The integral (15) is easily computed using the trapezoid rule with the intersection of the upper bounding functions (14), as opposed to the upper bounding points. A pictorial representation of these bounds is given in Fig. 1 for a fictional example (assuming that $\underline{Z} = 0.5$). In Fig. 1a, the lower bounding points $V_1^K(\lambda)$ are shown as dark circles, and the lower bounding area (shaded) is computed by (13), incorporating $\underline{Z} = 0.5$.

By themselves, the lower and upper bounds have limited meaning, especially if the objectives have different units of measurement. However, the difference between the upper and lower bounds can be interpreted as a proxy for the average distance between the lower-bounding approximation and the upper-bounding approximation of the Pareto frontier. When this distance reaches zero, Algorithm 2 has converged to the Pareto frontier.

3.2 Convergence

We now discuss finite convergence of Algorithm 2 applied to model (8). The main thrust of our argument is that there are a finite number of cuts required to define the complete Pareto frontier, and that the procedure in Algorithm 2 is sufficient to find them.

Lemma 2 *Let Assumptions 1, 2, and 6 hold. Consider a fixed value of λ . Then, ignoring the weight update so that $\lambda_K = \lambda$ for all K , consider running Algorithm 2. There exists some $K < \infty$ such that $V_1^K(\lambda) = V_1(\lambda)$.*

Proof Fixing λ and ignoring the weight update transforms Algorithm 2 into traditional Benders decomposition with known finite convergence [3, 34]. □

Lemma 3 *Let Assumption 6 hold. There are a finite number of extreme weights λ in weight-space.*

Proof We can re-express model (9) as a linear program with the feasible region of model (8) and objective function, $\lambda(c_1^\top x + d_1^\top y) + (1 - \lambda)(c_2^\top x + d_2^\top y)$. The result then

follows immediately from the fact there are finite number of basic feasible solutions to this linear program. □

Lemma 4 *Consider problem (2). Assume there exists a solution x^* and two values $\lambda_0, \lambda_1 \in [0, 1]$, $\lambda_0 < \lambda_1$, such that x^* is optimal to problem (2) at $\lambda = \lambda_0$ and at $\lambda = \lambda_1$. Then x^* is optimal to problem (2) for all $\lambda \in [\lambda_0, \lambda_1]$.*

Proof Let X denote the feasible region of model (2). By hypothesis x^* satisfies:

$$\lambda_0 c_1^\top x^* + (1 - \lambda_0) c_2^\top x^* \leq \lambda_0 c_1^\top x + (1 - \lambda_0) c_2^\top x, \quad \forall x \in X \tag{16a}$$

$$\lambda_1 c_1^\top x^* + (1 - \lambda_1) c_2^\top x^* \leq \lambda_1 c_1^\top x + (1 - \lambda_1) c_2^\top x, \quad \forall x \in X. \tag{16b}$$

The result is immediate for $\lambda = \lambda_0, \lambda_1$. Restricting to $\lambda \in (\lambda_0, \lambda_1)$, there exists $\gamma \in (0, 1)$ such that $\lambda = \gamma \lambda_0 + (1 - \gamma) \lambda_1$. Multiplying inequality (16a) by γ , inequality (16b) by $1 - \gamma$, and summing yields the desired result. □

Theorem 2 *Let Assumptions 1, 2, and 6 hold. Then, Algorithm 2 finds a set of Pareto-optimal solutions corresponding to the Pareto frontier of model (8) in a finite number of iterations.*

We preface the main argument that we use to prove the theorem. A complicating factor is that it does not suffice to prove convergence of the value function at the points identified in weight-space, *if we only add one cut at each step*. This is because we will have proved that x_1 is an optimal solution at λ_1 , and that x_2 is an optimal solution at $\lambda_2 = A(\lambda_1, \cdot) < \lambda_1$, but we have *not* proved that x_1 is an optimal solution for $\lambda = \lambda_2$, and therefore an optimal solution for all $\lambda \in [\lambda_2, \lambda_1]$. We resolve this issue by adding a second cut at x_1 and λ_2 in each step, as indicated in Algorithm 2.

Proof Each inner loop of Algorithm 2 terminates after a finite number of steps from decrementing λ until $\lambda = 0$ due to a finite number of possible bases. To show this, we will argue that both the subproblem (10) and master program (12) have a finite number of bases. Finiteness of the inner loop then follows from algorithm’s decrement, $\max\{A(\lambda_K, B_m), A(\lambda_K, B_s)\}$. The fact that the subproblem has a finite number of bases is immediate from the linear programming constraints of (10). The master program (12)’s cuts (12d) are indexed by λ explicitly via the $\lambda_k \mu$ term and implicitly because the cut coefficients α_k and β_k depend on λ . Suppose that the subproblem has been solved at two distinct values of $\lambda' < \lambda''$ via the *same basis* of (10). Then, for any $\lambda_k \in [\lambda', \lambda'']$ that also yields the same optimal basis of (10), the cut coefficients λ_k, α_k , and β_k can be expressed as a convex combination of those of λ' and λ'' . Algorithm 2 removes cuts that are redundant in this way. As a result, a finite number of bases for the master (12) follows from the finite number of bases for the subproblem (10).

By design, Algorithm 2 visits $\lambda = 1$ at the start of every outer (while) loop. Therefore, by Lemma 2 there exists some number of iterations $\kappa(1) < \infty$ after which $V_1^{\kappa(1)}(1) = V_1(1)$. Thus, by Assumptions 1 and 2, every time we solve $V_1^{\kappa(1)}(1)$ and the corresponding V_2 we obtain the same pair of optimal bases $B_{\kappa(1)}^* = \{B_{\kappa(1),m}^*, B_{\kappa(1),s}^*\}$ with optimal solutions $x_{\kappa(1)}^*$ and $y_{\kappa(1)}^*$.

Now, consider computing λ_s and λ_m as in Algorithm 2 using $B_{\kappa(1)}^*$ at some $K > \kappa(1)$. There are two cases: (i) $\lambda_s \geq \lambda_m$; and (ii) $\lambda_m > \lambda_s$.

In case (i), $\lambda_s \geq \lambda_m$. This case implies that, conditioned on $x_{\kappa(1)}^*$ remaining optimal, λ_s is the first weight at which y changes. If we repeatedly visit λ_s during iterations of the algorithm, then there exists a new value for K , $\kappa(2) < \infty$ at which $V_1^{\kappa(2)}(\lambda_s) = V_1(\lambda_s)$, and we obtain a new optimal basis $\mathcal{B}_{\kappa(2)}^*$ with optimal solutions $x_{\kappa(2)}^*$ and $y_{\kappa(2)}^*$. Because there is no approximation when solving the second-stage problem, by Lemma 4, $\mathcal{B}_{\kappa(1)}^*$ is an optimal basis for all $\lambda \in [\lambda_s, 1]$.

In case (ii), $\lambda_m > \lambda_s$. This case implies that, given the current approximation of the cost-to-go function, λ_m is the first weight at which a decision variable in the first-stage problem (i.e., x , μ , or φ) changes. If we repeatedly visit λ_m during iterations of the algorithm, then there exists a new value $\kappa(2) < \infty$ at which $V_1^{\kappa(2)}(\lambda_m) = V_1(\lambda_m)$, and we obtain a new optimal basis $\mathcal{B}_{\kappa(2)}^*$ with optimal solutions $x_{\kappa(2)}^*$ and $y_{\kappa(2)}^*$.

However, because of the presence of multiple optimal solutions at vertices in weight-space, it may be that even after we have converged the cost-to-go function at λ_m , there exists a weight $\lambda \in (\lambda_m, 1)$ at which the basis changes in the “true” problem $V_1(\lambda)$, and that we have missed identifying this basis because of the under-approximation of the cost-to-go function. However, the second cut computed at $x_{\kappa(1)}^*$ in Algorithm 2 ensures that there exists some value $\kappa(2) < \infty$ at which the approximation of the cost-to-go function is tight at both $x_{\kappa(2)}^*$ and at $x_{\kappa(1)}^*$. If the approximation of the cost-to-go function is tight at both values of x , then by Lemma 4—and the fact that V_2 is not approximated and so the second stage variables must remain optimal—we have that $\mathcal{B}_{\kappa(1)}^*$ is an optimal basis for all $\lambda \in [\lambda_m, 1]$.

The remainder of the proof follows an inductive argument: (a) beginning at the new weight λ^* (either λ_s from case (i) or λ_m from case (ii)) and basis $\mathcal{B}_{\kappa(2)}^*$, there is a finite number of iterations until we obtain a basis $\mathcal{B}_{\kappa(3)}^*$ at a new weight λ ; and (b) by Lemma 3 there are a finite number of optimal bases in the Pareto frontier. \square

Corollary 1 *Assume the hypotheses of Theorem 2. Then, the theorem still holds if we initialize the master problem (12) with a set of valid cuts.*

Proof Introducing an initial set of cuts does not change the validity of Theorem 2’s proof. Notably, the sequence of weights λ that are visited may differ, and therefore the set of discovered cuts may also differ, but Algorithm 2 still obtains the Pareto frontier in a finite number of iterations. \square

We now consider a random sampling variant of the algorithm in which, instead of choosing $\lambda_{K+2} \leftarrow \max\{\lambda_m, \lambda_s\}$, we randomly and independently choose either λ_m or λ_s at each iteration. We call this algorithm Algorithm 2a. Although not needed for Benders decomposition, this algorithm foreshadows the random sampling in stochastic dual dynamic programming.

Theorem 3 *Assume the hypotheses of Theorem 2. Then, Algorithm 2a converges almost surely in a finite number of iterations, again finding a set of Pareto-optimal solutions corresponding to the Pareto frontier of model (8).*

Proof The result follows directly from application of the second Borel–Cantelli lemma [19]. In particular, we first observe that Theorem 2 relies on a deterministic sequence of weights λ , and second note that by Corollary 1 the master problem may be initialized

with an arbitrary finite set of valid cuts. Then, if we independently sample the choice of λ_m and λ_s at each iteration, we will almost surely eventually sample the deterministic sequence that would have been obtained from Theorem 2, if it were initialized with the same set of cuts as described in Corollary 1. □

4 Bi-objective multistage stochastic programming

In Sect. 3, we extended the bi-objective simplex method to linear programs solved using Benders decomposition. Using those results, we now show how to extend the bi-objective simplex method to the stochastic dual dynamic programming algorithm.

We start by formulating a bi-objective multistage stochastic program in recursive form:

$$\begin{aligned} \vec{V}_1 = \min_{x_1} & C_1 x_1 + \mathbb{E}_{\omega_2} \left[\vec{V}_2(x_1, \omega_2) \right] \\ \text{s.t. } & A_1 x_1 = b_1 \\ & x_1 \geq 0, \end{aligned} \tag{17}$$

where for $t = 2, 3, \dots, T$:

$$\begin{aligned} \vec{V}_t(x_{t-1}, \omega_t) = \min_{\bar{x}, x_t} & C_t(\omega_t) x_t + \mathbb{E}_{\omega_{t+1}} \left[\vec{V}_{t+1}(x_t, \omega_{t+1}) \right] \\ \text{s.t. } & \bar{x} = x_{t-1} \\ & T_t(\omega_t) \bar{x} + W_t(\omega_t) x_t = g_t(\omega_t) \\ & x_t \geq 0, \end{aligned} \tag{18}$$

with the convention that $\vec{V}_{T+1}(\cdot, \cdot) = 0$. Here, the expectations of \vec{V}_{t+1} (now a vector) are taken component-wise. Notation from the formulation of model (4) carries over to (17), except that $C_t(\omega_t)$ are matrices with two rows, $c_{t,1}^\top(\omega_t)$ and $c_{t,2}^\top(\omega_t)$, for the respective objective function coefficients. Assumptions 3–5 also carry over directly.

Just as bi-objective linear programs have Pareto-optimal solutions, bi-objective stochastic programs require the definition of Pareto-optimal *policies*. Similar to how we defined a policy for single-objective multistage stochastic programs in Sect. 2.2, let $\vec{V}_t(\pi)$ be identical to \vec{V}_t , $t = 1, 2, \dots, T$, except that a decision-rule $\pi_t(x_{t-1}, \omega_t) \rightarrow x_t$ replaces the minimization operators in (17) and (18), again respecting the dynamics of the linear programming constraints of (17)–(18). In other words, instead of solving an optimization problem, we evaluate the decision rule to obtain a new state x_t , and return the objective value. We define Pareto-optimality of policies using $\vec{V}_1(\pi)$.

Definition 2 A policy $\hat{\pi}$, composed of decision rules $\hat{\pi}_t(x_{t-1}, \omega_t) \rightarrow x_t$ for each stage t , is *Pareto-optimal* if there exists no policy π with $\vec{V}_1(\pi) \neq \vec{V}_1(\hat{\pi})$ such that $\vec{V}_1(\pi) \leq \vec{V}_1(\hat{\pi})$.

Proceeding in parallel to the development of Sect. 3, we use the weighted-sum method and rewrite problem (17):

$$\begin{aligned}
 V_1(\lambda) = \min_{x_1} & [\lambda c_{1,1} + (1 - \lambda)c_{1,2}]^\top x_1 + \mathbb{E}_{\omega_2} [V_2(x_1, \lambda, \omega_2)] \\
 \text{s.t. } & A_1 x_1 = b_1 \\
 & x_1 \geq 0,
 \end{aligned}
 \tag{19}$$

and problem (18):

$$\begin{aligned}
 V_t(x_{t-1}, \lambda, \omega_t) = \min_{\bar{x}, x_t} & [\lambda c_{t,1}(\omega_t) + (1 - \lambda)c_{t,2}(\omega_t)]^\top x_t + \\
 & \mathbb{E}_{\omega_{t+1}} [V_{t+1}(x_t, \lambda, \omega_{t+1})] \\
 \text{s.t. } & \bar{x} = x_{t-1} \\
 & T_t(\omega_t)\bar{x} + W_t(\omega_t)x_t = g_t(\omega_t) \\
 & x_t \geq 0,
 \end{aligned}
 \tag{20}$$

where $c_{t,i}$ is the i^{th} objective vector in stage t . Then, like we did in Sect. 3, we approximate the cost-to-go term by the pointwise maximum of a collection of saddle cuts. This results in the approximated subproblems:

$$\begin{aligned}
 V_1^K(\lambda) = \min_{x_1, \mu_1, \varphi_1} & [\lambda c_{1,1} + (1 - \lambda)c_{1,2}]^\top x_1 + \lambda\mu_1 + \varphi_1 \\
 \text{s.t. } & A_1 x_1 = b_1 \\
 & x_1 \geq 0 \\
 & \lambda_k \mu_1 + \varphi_1 \geq \alpha_{1,k} + \beta_{1,k}^\top x_1, \quad k = 1, \dots, K \\
 & -L_1 \leq \mu_1 \leq L_1 \\
 & \varphi_1 \geq -M_1,
 \end{aligned}
 \tag{21}$$

and:

$$\begin{aligned}
 V_t^K(x_{t-1}, \lambda, \omega_t) = \min_{\bar{x}, x_t, \mu_t, \varphi_t} & [\lambda c_{t,1}(\omega_t) + (1 - \lambda)c_{t,2}(\omega_t)]^\top x_t + \lambda\mu_t + \varphi_t \\
 \text{s.t. } & \bar{x} = x_{t-1} \\
 & T_t(\omega_t)\bar{x} + W_t(\omega_t)x_t = g_t(\omega_t) \\
 & x_t \geq 0 \\
 & \lambda_k \mu_t + \varphi_t \geq \alpha_{t,k} + \beta_{t,k}^\top x_t, \quad k = 1, \dots, K \\
 & -L_t \leq \mu_t \leq L_t \\
 & \varphi_t \geq -M_t.
 \end{aligned}
 \tag{22}$$

With the approximated subproblems, we can combine the stochastic dual dynamic programming algorithm (Algorithm 1) and the bi-objective Benders decomposition algorithm (Algorithm 2) to create a bi-objective stochastic dual dynamic programming algorithm. Pseudo-code is given in Algorithm 3. Just like it was necessary to compute $\max\{\lambda_s, \lambda_m\}$ in the bi-objective Benders case, in theory it is necessary to compute the minimum step in weight-space across all stages and outcomes of the stagewise-independent random variable ω_t . However, like Algorithm 2a, we select a subproblem at random (from any stage) to compute the change in basis, independently across

iterations. Finally, note the distinction that Algorithms 2 and 2a construct the Pareto frontier using Pareto-optimal *solutions*, but Algorithm 3 instead constructs the Pareto frontier using Pareto-optimal *policies*.

Algorithm 3: Bi-objective stochastic dual dynamic programming algorithm

```

Set  $K \leftarrow 1$ 
while not converged do
  Set  $\lambda_K \leftarrow 1$ 
  repeat
    /* Forward pass */
    Solve  $V_1^{K-1}(\lambda_K)$ , i.e., (21), and obtain optimal extreme point primal solution  $\hat{x}_1$ 
    for  $t = 2, 3, \dots, T - 1$  do
      Sample  $\hat{\omega}_t$  from  $\Omega_t$ 
      Solve  $V_t^{K-1}(x_{t-1}, \lambda_K, \hat{\omega}_t)$ , i.e., (22), and obtain optimal extreme point primal solution  $\hat{x}_t$ 
    end
    /* Backward pass */
    for  $t = T - 1, T - 2, \dots, 1$  do
      for  $\omega \in \Omega_{t+1}$  do
        Solve  $V_{t+1}^{K-1}(\hat{x}_t, \lambda_K, \omega)$  and obtain optimal extreme point dual solution  $\hat{v}_{t+1,\omega}$  and objective value  $\hat{V}_{t+1,\omega}$ 
      end
      Set  $\alpha_{t,K} \leftarrow \mathbb{E}_\omega [\hat{V}_{t+1,\omega} - \hat{v}_{t+1,\omega}^\top \hat{x}_t]$ 
      Set  $\beta_{t,K} \leftarrow \mathbb{E}_\omega [\hat{v}_{t+1,\omega}]$ 
      Add the cut  $\lambda_K \mu_t + \varphi_t \geq \alpha_{t,K} + \beta_{t,K}^\top x_t$  to  $V_t^{K-1}$ , creating  $V_t^K$ 
    end
    /* Update weight and add second cuts */
    Sample  $t \in \{1, \dots, T\}$ 
    Solve  $V_t^K(\hat{x}_{t-1}, \lambda_K, \hat{\omega}_t)$  and obtain optimal basis  $\mathcal{B}$ 
    Set  $\lambda_{K+1} \leftarrow \Lambda(\lambda_K, \mathcal{B})$ 
    for  $t = T - 1, T - 2, \dots, 1$  do
      for  $\omega \in \Omega_{t+1}$  do
        Solve  $V_{t+1}^K(\hat{x}_t, \lambda_{K+1}, \omega)$  and obtain optimal extreme point dual solution  $\hat{v}_{t+1,\omega}$  and objective value  $\hat{V}_{t+1,\omega}$ 
      end
      Set  $\alpha_{t,K+1} \leftarrow \mathbb{E}_\omega [\hat{V}_{t+1,\omega} - \hat{v}_{t+1,\omega}^\top \hat{x}_t]$ 
      Set  $\beta_{t,K+1} \leftarrow \mathbb{E}_\omega [\hat{v}_{t+1,\omega}]$ 
      Add the cut  $\lambda_{K+1} \mu_t + \varphi_t \geq \alpha_{t,K+1} + \beta_{t,K+1}^\top x_t$  to  $V_t^K$ , creating  $V_t^{K+1}$ 
      Update  $V_t^{K+1}$ , removing cuts that are convex combinations of other cuts
    end
    Set  $\lambda_{K+2} \leftarrow \lambda_{K+1}$ 
    Set  $K \leftarrow K + 2$ 
  until  $\lambda_K = 0$ ;
end

```

4.1 Convergence

The proof of convergence for Algorithm 3 is an extension of the results in Sect. 3.2 to the multistage stochastic setting, analogous to the way that single-objective Benders decomposition results have been extended to the multistage stochastic setting.

Our results parallel those obtained in [11], which involves a single-objective problem but with inter-stage dependence in objective function coefficients. As a result, the key difference is that in [11] the analog of λ form a stochastic process that is revealed over time. Here, the weight λ specifies the trade-off between two objectives and is iteratively updated by Algorithm 3.

First, analogous to Lemma 1, we have a saddle function result for the cost-to-go function $V_t(x_{t-1}, \lambda, \omega_t)$.

Lemma 5 *Let $V_t(x_{t-1}, \lambda, \omega_t)$ be defined by subproblem (20), and let Assumptions 3–6 hold. Then, given fixed ω_t the function $V_t(x_{t-1}, \lambda, \omega_t)$ is piecewise-linear convex with respect to x_{t-1} on the convex hull of achievable x_{t-1} 's and piecewise-linear concave with respect to λ on $[0, 1]$, and each piecewise-linear function comprises a finite number of pieces.*

Proof The proof follows directly from Lemma 1 of [11]. \square

Second, analogous to Lemma 2, given a fixed λ we obtain an optimal policy almost surely in a finite number of iterations.

Lemma 6 *Let Assumptions 1–6 hold. Consider a fixed value of λ . Then, ignoring the weight update so that $\lambda_K = \lambda$ for all K , consider running Algorithm 3. There almost surely exists some $K < \infty$ such that $V_1^K(\lambda) = V_1(\lambda)$.*

Proof Given Lemma 5, this follows directly from existing convergence proofs for the stochastic dual dynamic programming algorithm in the literature; see, e.g., Theorem 2 of [11]. \square

Finally, analogous to Theorems 2 and 3, we have a convergence result for Algorithm 3.

Theorem 4 *Let Assumptions 1–6 hold, and assume that the choice of $t \in \{1, \dots, T\}$ for the weight update is made randomly and independently across the iterations. Then, Algorithm 3 converges almost surely to a set of Pareto-optimal policies corresponding to the Pareto frontier of model (17) in a finite number of iterations.*

Proof Like Theorem 2, proof of convergence requires a sequence of iterations at which we update the scalarizing weight using the minimum decrease until the basis changes. In the two-stage deterministic case, this involved computing the basis change for two linear programs. In the multistage stochastic setting, we must compute the change for every possible stage and sequence of realizations of the noise-terms ω_t . However, as in the proof of Theorem 3, we can apply the second Borel–Cantelli lemma [19] so that we will, almost surely, sample the requisite deterministic sequence for convergence in a finite number of iterations.

When combined with the independent sampling of the forward pass in stochastic dual dynamic programming, this implies that we almost surely sample a deterministic sequence of iterations in which we realize a sequence $\omega_2, \dots, \omega_T$ and choice of stage t , required for convergence. The remainder of the proof can be argued via induction in a manner identical to Theorem 2 using Lemmas 5 and 6. \square

It is important to note that, like the bi-objective linear programming case, our algorithm does not find *all* Pareto optimal policies because there may be multiple optimal policies corresponding to a given weight λ . However, assuming the decision-maker uses the same trade-off weight λ for all stages, we will have found all supported and non-dominated objective vectors \vec{V}_1 . The assumption that the decision-maker uses the same trade-off weight is a strong one, but we know of no quantitative way to incorporate the subjective choices of a decision-maker that changes their trade-off weights over time.

4.2 Bounds

Since the first-stage is deterministic, a proxy lower bound for the multistage stochastic bi-objective problem can be constructed in a manner identical to the lower bound for the bi-objective Benders decomposition algorithm. However, determining an upper bound requires a full enumeration of the scenarios. This quickly becomes intractable as the number of stages and size of Ω_t grow, and so we do not attempt to construct a deterministically valid upper bound. We note that such challenges with upper bounds in stochastic dual dynamic programming also arise when in risk averse and distributionally robust settings; i.e., such issues are not unique to the bi-objective context. That said, we could take a commonly used single-objective approach and conduct a Monte Carlo simulation of the policy at different values of λ in order to obtain a set of confidence intervals for the expected cost of the policy at different places along the Pareto frontier. While this could be done with a relatively large set of independent forward paths—*independent of the paths used in Algorithm 3, along which cuts are computed*—using common random numbers across all values of λ , we do not pursue this here.

5 Examples

We implemented Algorithms 2, 2a, and 3 in SDDP.jl [13], an open-source multistage stochastic programming solver built on JuMP [15] in the Julia Language [4]. All code and data for our experiments are available at <https://github.com/odow/SDDP.jl>.

5.1 A simple example

To gain intuition, we test our code on the following problem:

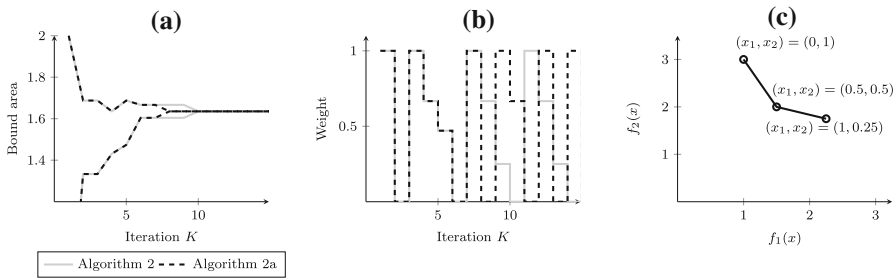


Fig. 2 Solution metrics of problem (23) against the number of iterations K : (a) The lower and upper (dashed) bounds; (b) The sequence of weights visited by iteration; and (c) The optimal solution in objective-space, with annotated solutions for decision-space

$$\begin{aligned}
 \min \quad & f_1 = 2x_1 + x_2 \\
 & f_2 = x_1 + 3x_2 \\
 \text{s.t.} \quad & x_1 + x_2 \geq 1 \\
 & 0.5x_1 + x_2 \geq 0.75 \\
 & x_2 \geq 0.25 \\
 & x_1, x_2 \geq 0.
 \end{aligned} \tag{23}$$

In Fig. 2a we plot evolution of the lower bound \underline{V}^K and upper bound \overline{V}^K against the number of iterations K . As a valid lower bound, we set $\underline{Z} = 0$. The solid, gray lines are the traces using Algorithm 2, and the dashed, black lines are the traces using Algorithm 2a. The lower and upper bounds of both algorithms converge to the same (optimal) value of 1.635417.

To explain the (small) difference in convergence, we plot, in Fig. 2b, the scalarizing weight λ_K used in iteration K . Because it does not take the minimum step, the sampling version of the algorithm skips some values in weight-space (e.g., $\lambda = 0.25$) at which the solution is already optimal and is therefore able to add more cuts at $\lambda = 0$ and $\lambda = 1$, where the solution is sub-optimal. Finally, the optimal Pareto frontier is shown in Fig. 2c.

5.2 Hydro-thermal scheduling

We consider a hydro-thermal scheduling problem in the Brazilian interconnected power system [33], using the open-source model and data provided by [10]. We now sketch the main details of the model; see [10, 33] for a full description.

The hydro-thermal scheduling problem for the Brazilian interconnected power system seeks to manage the medium-term electricity generation of the Brazilian national grid. The country is divided into four regions, each with an aggregate reservoir, hydro-power station, and thermal power stations. Energy can be transferred between regions. The goal of the agent is to construct a policy for hydro and thermal electricity generation with minimum expected cost under uncertain inflows into the four reservoirs. The cost is composed of two main components: (i) deficit cost, which is incurred if

there is insufficient electricity generated to meet demand; and (ii) thermal cost, which is incurred when thermal generation is used. Thermal cost is a convex increasing function of the quantity of thermal generation.

Although both objectives have the same units in this example, this is not a limitation of our method. We choose these objectives because they are the objectives from [33]. With suitable data, the thermal cost could, for example, be replaced by the tonnes of CO₂ emitted, and the deficit cost could similarly be replaced with the quantity of unmet demand (MWh). The important point is that there be a trade-off between the two selected objectives.

As originally formulated, the objective in each stage is composed by the addition of these terms, therefore, the problem can be stated as:

$$V_t(x_{t-1}, \omega_t) = \min_{x_t} c^d(x_t) + c^t(x_t) + \mathbb{E}[V_{t+1}(x_t, \omega_{t+1})]$$

$$\text{s.t. } x_t \in \mathcal{X}(x_{t-1}, \omega_t),$$

where x are the decision variables, c^d is the deficit cost, c^t is the thermal cost, and \mathcal{X} are the various constraints of the problem.

Rather than solve the problem with a fixed objective, in this example we solve the bi-objective problem in which one objective is the deficit cost and the other objective is the thermal cost. Therefore, we have:

$$V_t(x_{t-1}, \lambda, \omega_t) = \min_{x_t} \lambda c^d(x_t) + (1 - \lambda)c^t(x_t) + \mathbb{E}[V_{t+1}(x_t, \lambda, \omega_{t+1})]$$

$$\text{s.t. } x_t \in \mathcal{X}(x_{t-1}, \omega_t).$$

For numerical stability reasons, we scaled c^d by 1/100 and c^t by 1/10. Therefore, $\lambda = 10/11$ corresponds to the original formulation of the problem (scaled by 1/110). We use 12 monthly stages to represent one year of decision making, and there are 82 stagewise-independent realizations of inflow in each stage.

We trained a policy for 2000 iterations of Algorithm 3. Then, we selected three λ weights and performed a Monte Carlo simulation of the policy given those scalarizing weights, using common random numbers with a total of 1000 forward paths. The weights were $\lambda = 0.1$, $\lambda = 0.7$, and $\lambda = 0.9$. Visualizations of these Monte Carlo simulations are given in Fig. 3 in terms of objective-space (Fig. 3a) and weight-space (Fig. 3b). The figures show the spread of the cost under the 1000 simulated realizations, along with the mean of each of the three λ weights.

As Fig. 3a shows, with $\lambda = 0.1$ we prefer low thermal costs and allow high deficit costs, and $\lambda = 0.9$ does the opposite. The figure also shows that $\lambda = 0.7$ provides a less extreme trade-off between the two costs. In Fig. 3b, observe that the lower bound $V_1^K(\lambda)$ is a concave function of λ . Cuts (not shown) are being added with respect to x (into the page), and interpolated via the dual variables μ and φ between different weights λ .

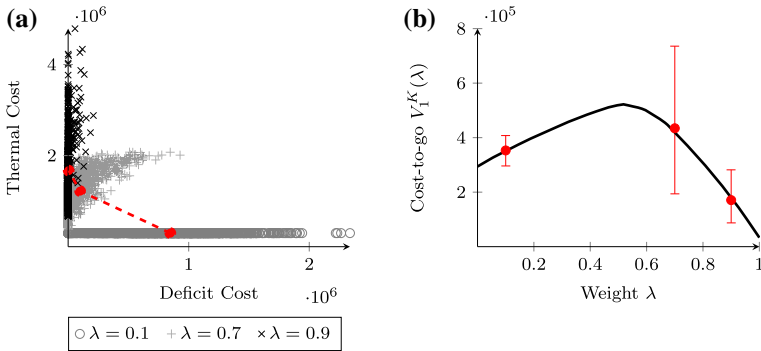


Fig. 3 Visualization of three policies for $\lambda = 0.1$, $\lambda = 0.7$, and $\lambda = 0.9$: **(a)** Objective-space; **(b)** Weight-space. In **(a)**, Solid red dots are the sample mean of 1000 simulated sample paths, with dashed red lines representing the estimated Pareto frontier. In **(b)**, Solid black line is the deterministic lower-bound $V_1^K(\lambda)$ as a function of λ , and red circles are the sample mean upper bound under 1000 simulated realizations. The error bars are the empirical 10th and 90th percentiles (i.e., red bars represent prediction, not confidence, intervals)

6 Heuristic solution methods

When implementing Algorithm 3, we encountered various numerical and computational challenges. In this section, we present a heuristic solution method that is numerically stable and easier to implement. While the heuristic lacks a convergence guarantee with a finite number of iterations, lower and upper bounds can still be computed at specific values of λ , per the discussion in Sect. 4.2.

6.1 Motivation

In our experimentation, numerical issues were more frequent when the two objective functions in the model were of different orders of magnitude, which suggests appropriate re-scaling should be performed. In addition, we encountered cases in which the optimal basis would change with very small changes in the weight λ (for example, a cut leaving the basis and being replaced by a cut that is nearly co-linear). Therefore, we sought a heuristic solution method that did not have these numerical challenges.

Moreover, computing the λ update step, $A(\lambda_K, \mathcal{B})$, can become a significant computational bottleneck. In particular, the weight update either requires computing reduced costs for coefficients not present in the current problem instance (see Eq. (3)) and/or requires solver-specific functionality that modeling languages such as JuMP do not easily provide. For the experiments in the previous section, we hard-coded support for Gurobi’s basis routines into SDDP.jl. However, due to licensing requirements of Gurobi, it is not practical to distribute this code, and it limits users of SDDP.jl to the problem types that Gurobi supports. Therefore, we sought a heuristic solution method that did not require solver-specific functionality.

6.2 Heuristic

In standard bi-objective optimization, a common solution approach is the non-inferior set estimation method [8]. The method starts by solving for $\lambda = 0$ and $\lambda = 1$, and then chooses weights so that the new objective function is perpendicular to the line joining the solutions in objective space. If no new solution is found for this new objective function, the algorithm stops. Otherwise, the new solution is an extreme point on the Pareto frontier, and we partition the previous line into two segments. We then repeat the process of setting the weights so that the scalarized objective function is perpendicular to each segment.

This results in a binary search procedure. However, a direct translation to the multi-stage stochastic case is difficult for two reasons. First, we cannot compute exact points in objective space; instead, we need a simulation of the policy to estimate $\vec{V}_1(\pi^K)$ given the value of λ . Second, without the saddle cuts, this would require solving a new instance SDDP to provable optimality for each λ .

As an alternative, computationally efficient heuristic, we consider a modified version of Algorithm 3. The resulting Algorithm 4 starts by solving $\lambda = 0$ and $\lambda = 1$ to near optimality, but then instead of performing a binary search in objective-space, we conduct a binary search in weight-space. Thus, the next weight we consider is $\lambda = 0.5$, and then $\lambda = 0.25$ and $\lambda = 0.75$, and so on. In addition, we maintain the saddle-cut reformulation. The benefit of this approach is that each successive solve is warm-started by the saddle-cuts of previous solves. Moreover, the algorithm works for any solver integrated into SDDP.jl—including conic and nonlinear solvers—and does not require solver-specific functionality.

Algorithm 4: Heuristic solution method

Input: $N > 2$
 Run Algorithm 3 given (i.e., for a fixed value of) $\lambda_K = 0$ for all K
 Run Algorithm 3 given $\lambda_K = 1$ for all K
 Initialize $\mathcal{Q} = \{(0, 1)\}$
for $i = 3, \dots, N$ **do**
 Take an element (l, u) from \mathcal{Q}
 Run Algorithm 3 given $\lambda_K = \frac{l+u}{2}$ for all K
 Add $(l, \frac{l+u}{2})$ and $(\frac{l+u}{2}, u)$ to \mathcal{Q}
end

6.3 Experiment

To investigate the performance of our heuristic, we implemented Algorithm 4 in SDDP.jl and applied it to the same hydro-thermal scheduling instance as in the previous section, only this time with 60 stages instead of 12.

As a baseline, we chose nine equidistant points in weight space, and then solved the nine resulting instances separately with SDDP. These weights were visited in the same order that they would be in Algorithm 4, i.e., 0, 1, 0.5, 0.25, and so on. For a

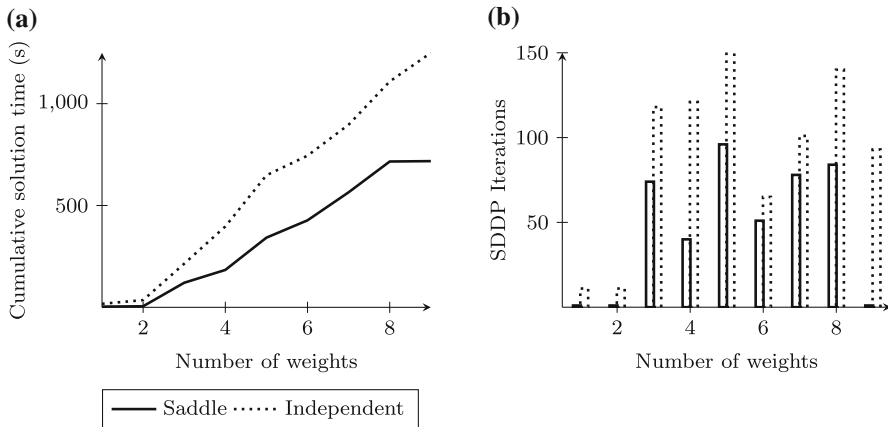


Fig. 4 Cumulative solution time using Algorithm 4 (“Saddle”), and independent instances of SDDP without the saddle-cut formulation (“Independent”)

stopping rule, we used the `BoundStalling` termination rule in `SDDP.jl`, which stops when the lower bound fails to improve by more than 10 units in absolute terms ($\approx 10^{-4}$ in relative terms) in each iteration for 10 consecutive iterations. Then, we timed how long it would take Algorithm 4 to solve the same nine-weight values to the same value of the lower bound using the `BoundLimit` stopping rule in `SDDP.jl`. In Fig. 4, we present the cumulative solution time (a) and number of iterations to converge (b) against the number of weights we have visited.

Because Algorithm 4 uses the saddle cuts, it needs fewer iterations to converge at a given weight than the equivalent run of regular SDDP. This is particularly noticeable for the $\lambda = 0.875$ case, which is visited last; with regular SDDP it takes 93 iterations to converge, while with the saddle cut we converge after a single iteration because the interpolated value function has the same objective value.

7 Conclusion

This paper has presented two algorithms for solving bi-objective multistage stochastic linear programs, and demonstrated the algorithms on an example from hydro-thermal scheduling. Many real-world examples are naturally bi-objective. In energy systems alone, future applications of our approach could include capacity expansion problems, where the capital costs are one objective and the operational costs are the second objective. Another extension could be to model the quantity of CO_2 emitted in tonnes against the quantity of unmet demand in MWh. We encourage other researchers to experiment with these ideas, using the open-source code we have provided in `SDDP.jl`.

Acknowledgements This research was supported, in part, by Northwestern University’s Center for Optimization & Statistical Learning (OSL).

References

1. Artzner, P., Delbaen, F., Eber, J.M., Heath, D.: Coherent measures of risk. *Math. Financ.* **9**(3), 203–228 (1999)
2. Baucke, R., Downward, A., Zakeri, G.: A deterministic algorithm for solving multistage stochastic minimax dynamic programmes. *Optimization Online* (2018). http://www.optimization-online.org/DB_HTML/2018/02/6449.html
3. Benders, J.: Partitioning procedures for solving mixed-variables programming problems. *Numer. Math.* **4**, 238–252 (1962)
4. Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B.: Julia: A Fresh Approach to Numerical Computing. *SIAM Rev.* **59**(1), 65–98 (2017)
5. Bland, R.G.: New finite pivoting rules for the simplex method. *Math. Oper. Res.* **2**(2), 103–107 (1977)
6. Boland, N., Charkhgard, H., Savelsbergh, M.: A criterion space search algorithm for biobjective integer programming: The balanced box method. *INFORMS J. Comput.* **27**(4), 735–754 (2015)
7. Cardona-Valdés, Y., Álvarez, A., Ozdemir, D.: A bi-objective supply chain design problem with uncertainty. *Trans. Res. Part C: Emerg. Technol.* **19**(5), 821–832 (2011)
8. Cohon, J.L., Church, R.L., Sheer, D.P.: Generating multiobjective trade-offs: An algorithm for bicriterion problems. *Water Resour. Res.* **15**(5), 1001–1010 (1979)
9. Dantzig, G.B.: *Linear programming and extensions*. Princeton University Press, Princeton, NJ (1963)
10. Ding, L., Ahmed, S., Shapiro, A.: A Python package for multi-stage stochastic programming. *Optimization*, pp. 1–45 (2019). Available online: <https://optimization-online.org/2019/05/7199/>. Accessed 2 Aug 2022
11. Downward, A., Dowson, O., Baucke, R.: Stochastic dual dynamic programming with stagewise-dependent objective uncertainty. *Oper. Res. Lett.* **48**, 33–39 (2020)
12. Dowson, O.: The policy graph decomposition of multistage stochastic programming problems. *Networks* **76**, 3–23 (2020)
13. Dowson, O., Kapelevich, L.: SDDP.jl: a Julia package for stochastic dual dynamic programming. *INFORMS J. Comput.* **33**(1), 27–33 (2021)
14. Dowson, O., Morton, D., Pagnoncelli, B.: Partially observable multistage stochastic programming. *Oper. Res. Lett.* **48**, 505–512 (2020)
15. Dunning, I., Huchette, J., Lubin, M.: JuMP: A Modeling Language for Mathematical Optimization. *SIAM Rev.* **59**(2), 295–320 (2017)
16. Ehrgott, M.: *Multicriteria Optimization*, 2nd edn. Springer, Berlin; New York (2005)
17. Geoffrion, A.M.: Proper efficiency and the theory of vector maximization. *J. Math. Anal. Appl.* **22**(3), 618–630 (1968)
18. Girardeau, P., Leclère, V., Philpott, A.B.: On the Convergence of Decomposition Methods for Multistage Stochastic Convex Programs. *Math. Oper. Res.* **40**(1), 130–145 (2015)
19. Grimmett, G., Stirzaker, D.: *Probability and Random Processes*, 2nd edn. Oxford University Press, Oxford (1992)
20. Guigues, V.: Convergence Analysis of Sampling-Based Decomposition Methods for Risk-Averse Multistage Stochastic Convex Programs. *SIAM J. Optim.* **26**(4), 2468–2494 (2016)
21. Gutjahr, W.J., Pichler, A.: Stochastic multi-objective optimization: A survey on non-scalarizing methods. *Ann. Oper. Res.* **236**(2), 475–499 (2016)
22. Jacobs, J., Freeman, G., Grygier, J., Morton, D., Schultz, G., Staschus, K., Stedinger, J.: Socrates: a system for scheduling hydroelectric generation under uncertainty. *Ann. Oper. Res.* **59**, 99–133 (1995)
23. Mardan, E., Govindan, K., Mina, H., Gholami-Zanjani, S.M.: An accelerated Benders decomposition algorithm for a bi-objective green closed loop supply chain network design problem. *J. Clean. Prod.* **235**, 1499–1514 (2019)
24. Moradi, S., Raith, A., Ehrgott, M.: A bi-objective column generation algorithm for the multi-commodity minimum cost flow problem. *Eur. J. Oper. Res.* **244**(2), 369–378 (2015)
25. Pereira, M., Pinto, L.: Multi-stage stochastic optimization applied to energy planning. *Math. Program.* **52**, 359–375 (1991)
26. Pflug, G.C., Pichler, A.: Time-Consistent Decisions and Temporal Decomposition of Coherent Risk Functionals. *Math. Oper. Res.* **41**(2), 682–699 (2016)
27. Philpott, A.: On the Marginal Value of Water for Hydroelectricity. In: Terlaky, T., Anjos, M.F., Ahmed, S. (eds.) *Advances and Trends in Optimization with Engineering Applications*, pp. 405–425. Society for Industrial and Applied Mathematics, Philadelphia, PA (2017)

28. Philpott, A., de Matos, V., Finardi, E.: On Solving Multistage Stochastic Programs with Coherent Risk Measures. *Oper. Res.* **61**(4), 957–970 (2013)
29. Philpott, A.B., Guan, Z.: On the convergence of sampling-based methods for multi-stage stochastic linear programs. *Oper. Res. Lett.* **36**, 450–455 (2008)
30. Rahmanniyay, F., Yu, A.J., Seif, J.: A multi-objective multi-stage stochastic model for project team formation under uncertainty in time requirements. *Comput. Ind. Eng.* **132**, 153–165 (2019)
31. Razmi, J., Zahedi-Anaraki, A.H., Zakerinia, M.S.: A bi-objective stochastic optimization model for reliable warehouse network redesign. *Math. Comput. Model.* **58**(11–12), 1804–1813 (2013)
32. Shapiro, A.: Analysis of stochastic dual dynamic programming method. *Eur. J. Oper. Res.* **209**(1), 63–72 (2011)
33. Shapiro, A., Tekaya, W., da Costa, J.P., Soares, M.P.: Risk neutral and risk averse Stochastic Dual Dynamic Programming method. *Eur. J. Oper. Res.* **224**(2), 375–391 (2013)
34. Van Slyke, R.M., Wets, R.: L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM J. Appl. Math.* **17**(4), 638–663 (1969)
35. Wolfe, P.: A technique for resolving degeneracy in linear programming. *J. Soc. Ind. Appl. Math.* **11**(2), 205–211 (1963)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.