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

Algorithmic innovations and software for the dual decomposition method applied to stochastic mixed-integer programs

Kibaek Kim · Victor M. Zavala

Submitted: June 30, 2015

Abstract We develop algorithmic innovations for the dual decomposition method to address two-stage stochastic programs with mixed-integer recourse and provide a parallel software implementation that we call `DSP`. Our innovations include the derivation of valid inequalities that tighten Lagrangian subproblems and that aid the exclusion of infeasible first-stage solutions for problems without (relative) complete recourse. We also propose an interior-point cutting-plane method to solve the Lagrangian master problem and provide termination criteria that guarantee finite termination of the overall algorithm. `DSP` can solve instances specified in C code, SMPS files, and `StochJump` (a `Julia`-based and parallel algebraic modeling language). `DSP` also implements a standard Benders decomposition method and a dual decomposition method based on subgradient dual updates that we use to perform benchmarks. We present numerical results using standard SIPLIB instances and a large-scale unit commitment problem to demonstrate that the innovations provide significant improvements in the number of iterations and solution times.

Keywords stochastic mixed-integer programming · decomposition · parallel · large-scale · open-source software

This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357. We gratefully acknowledge the computing resources provided on Blues, a high-performance computing cluster operated by the Laboratory Computing Resource Center at Argonne National Laboratory. We thank Julie Bessac for providing wind speed prediction data, and Sven Leyffer for providing feedback on an earlier version of the manuscript.

Kibaek Kim
Mathematics and Computer Science Division
Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439, USA
E-mail: kimk@anl.gov

Victor M. Zavala
Mathematics and Computer Science Division
Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439, USA

1 Problem Statement

We are interested in computing solutions for two-stage stochastic mixed-integer programs (SMIPs) of the form

$$z := \min_{x} \left\{ c^T x + \mathcal{Q}(x) : \; Ax = b, \; x \in X \right\}, \tag{1}$$

where $\mathcal{Q}(x) = \mathbb{E}_\xi Q(x, \xi)$ is the recourse function and

$$Q(x, \xi) := \min_{y} \{ q(\xi)^T y : \; W(\xi) y = h(\xi) - T(\xi) x, \; y \in Y \}. \tag{2}$$

We assume that the random parameter ξ follows a discrete distribution with finite support $\{\xi^1, \ldots, \xi^S\}$ and corresponding probabilities $p_1, \ldots, p_S$ (continuous distributions can be handled by using a sample-average approximation). The sets $X \subseteq \mathbb{R}^{n_1}_+$ and $Y \subseteq \mathbb{R}^{n_2}_+$ represent integer restrictions on a subset of the decision variables x and y, respectively. The first-stage problem data comprises $A \in \mathbb{R}^{m_1 \times n_1}$, $b \in \mathbb{R}^{m_1}$, and $c \in \mathbb{R}^{n_1}$. The second-stage data are given by $T(\xi^s) \in \mathbb{R}^{m_2 \times n_1}$, $W(\xi^s) \in \mathbb{R}^{m_2 \times n_2}$, $h(\xi^s) \in \mathbb{R}^{m_2}$, and $q(\xi^s) \in \mathbb{R}^{n_2}$. For simplicity, we use the notation (T_s, W_s, h_s, q_s) for $s \in \mathcal{S} := \{1, \ldots, S\}$.

The SMIP (1) can be rewritten in the extensive form

$$z = \min_{x_s, y_s} \quad \sum_{s \in \mathcal{S}} p_s \left(c^T x_s + q_s^T y_s \right) \tag{3a}$$

$$\text{s.t.} \quad \sum_{s \in \mathcal{S}} H_s x_s = 0 \tag{3b}$$

$$(x_s, y_s) \in G_s, \quad \forall s \in \mathcal{S} \tag{3c}$$

where the scenario feasibility set is defined as

$$G_s := \{ (x_s, y_s) \; : \; Ax_s = b, \; T_s x_s + W_s y_s = h_s, \; x_s \in X, \; y_s \in Y \}, \tag{4}$$

(3b) are the nonanticipativity constraints representing the equations $x_1 = x_S$ and $x_s = x_{s-1}$ for $s = 2, \ldots, S$, and H_s is a suitable $S \cdot n_1 \times n_1$ matrix. We assume that SMIP does not necessarily have relatively complete recourse. We recall that, without this property, there can exist a $\hat{x} \in X$ satisfying $A\hat{x} = b$ for which there does not exist a recourse $y \in \mathbb{R}^{m_2}$ satisfying $(\hat{x}, y) \in G_{s'}$ for some $s' \in \mathcal{S}$. In other words, not every choice of the first-stage variables is guaranteed to have feasible recourse for all scenarios.

Different techniques exist for solving SMIPs. Benders decomposition (also known as the L-shaped method) can be applied for problems in which integer variables appear only in the first stage [6,30]. When integer variables appear in the second stage the recourse value function is nonconvex and discontinuous in the first-stage variables and, consequently, other approaches are needed. These include convexification of the recourse function [46,45,15,51, 28] or specialized branch-and-bound schemes [3]. These approaches, however, are limited to certain problem classes.

Carøe and Schultz [8] proposed a dual decomposition method for SMIPs. Dual decomposition solves a Lagrangian dual problem by dualizing (relaxing)

the nonanticipativity constraints to obtain lower bounds. Such lower bounds are often tight and can be used to guide the computation of feasible solutions (and thus upper bounds) using heuristics or branch-and-bound procedures. Tarhan and Grossmann [47] applied mixed-integer programming sensitivity analysis [10] within a dual decomposition method to improve bounds during the solution and this approach was shown to reduce the number of iterations. A crucial limitation of dual decomposition is that it is not guaranteed to recover feasible solutions for problems without (relatively) complete recourse which appear commonly in applications. As a result, the method may not be able to obtain an upper bound and it is thus difficult to estimate the optimality gap and stop the search. Ahmed [2] proposed a dual decomposition method that is guaranteed to recover an optimal feasible solution for two-stage stochastic programs with pure binary first-stage variables by iteratively adding *cover inequalities*. This approach is computationally attractive, but it is applicable only to problems with pure binary first stages.

Dual variables can be updated in a dual decomposition method by using subgradient methods [2,8,43,42,47], cutting-plane methods [35], or column-generation methods [35,37]. The efficiency of dual decomposition strongly depends on the update scheme used and existing schemes are limited. For instance, it is well known that cutting-plane schemes can lead to strong oscillations of the dual variables while subgradient schemes require of ad-hoc steplength selection criteria and can exhibit slow convergence [40,12].

Progressive hedging is a popular and flexible method for solving SMIPs and can handle problems with mixed-integer recourse [9,33,48,23]. Connections between progressive hedging and dual decomposition have also been established recently to create hybrid strategies [23]. It is well known that progressive hedging has no convergence guarantees. Consequently, it is often used as a heuristic to find approximate solutions. We also note that this method is not guaranteed to find a feasible solution for problems without relatively complete recourse.

Few software packages are currently available for solving large-scale stochastic programs. `SMI` [29] is an open-source software implementation that can read SMPS files and solve the extensive form of the problem by using COIN-OR solvers such as `Cbc` [13]. `FortSP` is a commercial solver that implements variants of Benders decomposition [41,52]. The C package `ddsip` [38] implements dual decomposition for SMIPs and uses the `ConicBundle` [25] package for updating dual variables. This package was unable to solve many small-size SIPLIB instances [35]. Moreover, `ddsip` does not support parallelism and does not support model specification through SMPS files and algebraic modeling languages. `PIPS` provides a parallel interior-point method for solving continuous stochastic programs and provides a basic implementation of the dual decomposition method for solving SMIPs [36,35]. `PySP` is a `Python`-based open-source software package that can model and solve SMIPs in parallel computing environments by using progressive hedging and Benders decomposition [49].

In this work, we present algorithmic innovations for the dual decomposition method. We develop a procedure to generate valid inequalities that tighten Lagrangian subproblems and that aid the exclusion of infeasible first-stage so-

lutions. The inequalities are Benders-type cuts (i.e., Benders feasibility and optimality cuts) that are derived from Lagrangian subproblem solutions and that are shared among subproblems. We demonstrate that this procedure enables us to accelerate solutions and to find upper bounds for problems without relatively complete recourse. We also present an interior-point cutting-plane method for solving the Lagrangian master problem and provide termination criteria that guarantee finite convergence of the dual decomposition algorithm. We demonstrate that this approach enables us to drastically reduce oscillations of the dual variables and ultimately aids convergence. To the best of our knowledge, the derivation of valid inequalities from Lagrangian subproblems is new and the proposed termination criteria for the Lagrangian master problem are new. We also introduce `DSP`, an open-source, object-oriented, and parallel software package that enables the implementation and benchmarking of different dual decomposition strategies and of other standard techniques such as Benders decomposition. `DSP` provides interfaces that can read models expressed in C code and the SMPS format [5,16]. Most notably, `DSP` can read models expressed in `StochJuMP` [26], an extension of the `Julia`-based algebraic modeling language package `JuMP` [34]), that can be used to compactly represent large-scale stochastic programs. We benchmark `DSP` using SIPLIB instances and large-scale unit commitment problems with up to 1,700,000 rows, 583,200 columns, and 28,512 integer variables on a 310-node parallel computing cluster at Argonne National Laboratory. We demonstrate that our innovations yield important improvements in robustness and solution time.

The paper is structured as follows. In Section 2 we present the standard dual decomposition method and discuss different approaches for updating dual variables. In Section 3 we present our algorithmic innovations. Here, we first present valid inequalities for the Lagrangian subproblems to eliminate infeasible first-stage solutions and to tighten the subproblems (Section 3.1). We then present an interior-point cutting-plane method and termination criteria for the master problem (Section 3.2). In Section 4 we describe the design of `DSP` and the modeling interfaces. In Section 5 we present our benchmark results. In Section 6 we summarize our innovations and provide directions of future work.

2 Dual Decomposition

We describe a standard dual decomposition method for two-stage SMIPs of the form (3). Let $\lambda \in \mathbb{R}^{S \cdot n_1}$ be the dual variables of the nonanticipativity constraints (3b). We apply a Lagrangian relaxation of these constraints to obtain the Lagrangian dual function of (3):

$$D(\lambda) := \min_{x_s, y_s} \left\{ \sum_{s \in \mathcal{S}} L_s(x_s, y_s, \lambda) : (x_s, y_s) \in G_s, \ \forall s \in \mathcal{S} \right\}, \tag{5}$$

where

$$L_s(x_s, y_s, \lambda) := p_s \left(c^T x_s + q_s^T y_s\right) + \lambda^T (H_s x_s). \tag{6}$$

For fixed λ, the Lagrangian dual function can be decomposed as

$$D(\lambda) = \sum_{s \in \mathcal{S}} D_s(\lambda), \tag{7}$$

where

$$D_s(\lambda) := \min_{x_s, y_s} \{L_s(x_s, y_s, \lambda) : (x_s, y_s) \in G_s\}. \tag{8}$$

We thus seek to obtain the best lower bound for (3) by solving the maximization problem (the Lagrangian dual problem):

$$z_{\text{LD}} := \max_{\lambda} \sum_{s\mathcal{S}} D_s(\lambda). \tag{9}$$

The following Proposition 1 is a well known result of Lagrangian relaxation that shows the tightness of the lower bound z_{LD} [17].

Proposition 1 *The optimal value z_{LD} of the Lagrangian dual problem (9) is equal to the optimal value of the following linear program,*

$$\min_{x_s, y_s} \left\{ \sum_{s \in \mathcal{S}} p_s \left(c^T x_s + q_s^T y_s\right) : \sum_{s \in \mathcal{S}} H_s x_s = 0,\ (x_s, y_s) \in conv(G_s),\ \forall s \in \mathcal{S} \right\}, \tag{10}$$

where $conv(G_s)$ denotes the convex hull of G_s. Moreover, $z_{LD} \geq z_{LP}$ holds, where z_{LP} is the optimal value of the linear programming relaxation of SMIP (3).

We highlight that the solution of the maximization problem (9) provides only a lower bound for SMIP (albeit this one is often tight). When the problem at hand has no duality gap, the solution of (9) is feasible for SMIP and thus optimal. When this is not the case, an upper bound z_{UB} for SMIP may be obtained and refined by finding feasible solutions during the dual decomposition procedure. Finding the best upper bound and thus computing the actual duality (optimality) gap $z_{\text{UB}} - z_{\text{LB}}$ can be done by performing a branch-and-bound search but this is often prohibitive. In our case we find a first stage decision that is feasible for SMIP by solving (3) for fixed candidates x_s obtained from the solution of the subproblems (8). Note that (3) may be infeasible for all candidates x_s if the problem does not have relatively complete recourse. We provide a remedy for this in Section 3.

2.1 Dual-Search Methods

In a dual decomposition method, we iteratively search for dual values λ that maximize the Lagrangian dual function (5). We now present a conventional subgradient method and a cutting-plane method to perform such a search.

2.1.1 Subgradient Method

Subgradient methods have been widely used in nonsmooth optimization. We describe a conventional method with a step-size rule described in [11]. Let λ^k be the dual variable at iteration $k \geq 0$, and let x_s^k be an optimal solution of (8) for given λ^k. The dual variable is updated as

$$\lambda^{k+1} = \lambda^k - \alpha_k \sum_{s \in \mathcal{S}} H_s x_s^k, \tag{11}$$

where $\alpha_k \in (0, 1]$ is the step size. This method updates the duals by using a subgradient of $D(\lambda)$ at λ^k, denoted by $\sum_{s \in \mathcal{S}} H_s x_s^k$. The step size α_k is given by

$$\alpha_k := \beta_k \frac{z_{\text{UB}} - D(\lambda^k)}{\left\| \sum_{s=1}^{S} H_s x_s^k \right\|_2^2}, \tag{12}$$

where z_{UB} is the objective value of the best-known feasible solution to (1) up to iteration k and β_k is a user-defined positive scalar. Algorithm 1 summarizes the procedure.

Algorithm 1 Dual Decomposition Based on Subgradient Method (DDSub)

1: Set $k \leftarrow 0, z_{\text{LB}} \leftarrow -\infty, z_{\text{UB}} \leftarrow \infty$ and $\gamma \leftarrow 0$.
2: **loop**
3: SOLVE (8) to obtain $D_s(\lambda^k)$ and (x_s^k, y_s^k) for given λ^k and for all $s \in \mathcal{S}$
4: **if** $D(\lambda^k) > z_{\text{LB}}$ **then**
5: $z_{\text{LB}} \leftarrow D(\lambda^k)$
6: **else**
7: $\gamma \leftarrow \gamma + 1$
8: **if** $\gamma = \gamma^{\max}$ **then**
9: $\beta_k \leftarrow 0.5\beta_k$ and $\gamma \leftarrow 0$
10: **end if**
11: **end if**
12: UPDATE z_{UB} for given x_s^k
13: $k \leftarrow k + 1$
14: **end loop**

Algorithm 1 is initialized with user-defined parameters $\lambda^0, \gamma^{\max}$, and β_0 and reduces β_k by a half when the best lower bound z_{LB} is not improved for the last $\gamma^{\max}$ iterations (lines 8-10). The best upper bound z_{UB} can be obtained by solving (3) for fixed x_s^k (line 12). An important limitation of subgradient methods is that it is not possible to prove finite termination [11].

2.1.2 Cutting-Plane Method

The cutting-plane method is an outer approximation scheme that solves the Lagrangian dual problem by iteratively adding linear inequalities. The outer

approximation of (9) at iteration k is given by the Lagrangian master problem:

$$m_k := \max_{\theta_s,\lambda} \quad \sum_{s\in\mathcal{S}} \theta_s \tag{13a}$$

$$\text{s.t.} \quad \theta_s \le D_s(\lambda^l) + \left(H_s x_s^l\right)^T (\lambda - \lambda^l),\ s \in \mathcal{S},\ l = 0, 1, \ldots, k. \tag{13b}$$

The dual variable λ^{k+1} is obtained by solving the approximation (13) at iteration k. We define the primal-dual solution of the Lagrangian master problem as the triplet (θ, λ, π). Here, $\theta := (\theta_1, ..., \theta_S)$ and $\pi := (\pi_1^0, ..., \pi_1^k, ..., \pi_S^0, ..., \pi_S^k)$, where π_s^l are the dual variables of (13b). The procedure is summarized in Algorithm 2.

Algorithm 2 Dual Decomposition Based on Cutting-Plane Method (DDCP)

1: $k \leftarrow 0$ and $\lambda^0 \leftarrow 0$
2: SOLVE (8) to obtain $D_s(\lambda^k)$ and (x_s^k, y_s^k) for given λ^k and for all $s \in \mathcal{S}$
3: $z_{\text{LB}} \leftarrow D(\lambda^k)$.
4: **repeat**
5: ADD (13b) for given $D(\lambda^k)$ and x_s^k
6: SOLVE (13) to obtain m_k and $(\theta^{k+1}, \lambda^{k+1})$
7: SOLVE (8) to obtain $D_s(\lambda^{k+1})$ and (x_s^{k+1}, y_s^{k+1}) for given λ^{k+1} and for all $s \in \mathcal{S}$; and obtain $D(\lambda^{k+1})$ from (7).
8: $z_{\text{LB}} \leftarrow \max\{z_{\text{LB}}, D(\lambda^{k+1})\}$.
9: $k \leftarrow k + 1$
10: **until** $m_{k-1} \le D(\lambda^k)$

The function $D_s(\lambda)$ is piecewise linear concave in λ supported by the linear inequalities (13b). Assuming that the master problem (13) and the subproblem (8) can be solved to optimality, Algorithm 2 terminates with an *optimal* solution of (9) after a finite number of steps because the number of linear inequalities required to approximate $D(\lambda)$ is finite. This gives the cutting-plame method a natural termination criterion (i.e., $m_{k-1} \le D(\lambda^k)$). In other words, this criterion indicates that m_{k-1} matches the Lagrangian dual function $D(\lambda^k)$ and thus the maximum of the Lagrangian master problem matches the maximum of the Lagrangian dual problem.

Remark 1 Instead of adding the linear inequalities (13b) for each $s \in \mathcal{S}$, one can add a single aggregated cut

$$\sum_{s\in\mathcal{S}} \theta_s \le \sum_{s\in\mathcal{S}} D_s(\lambda^l) + \left(\sum_{s\in\mathcal{S}} H_s x_s^l\right)^T (\lambda - \lambda^l) \tag{14}$$

per iteration $l = 0, 1, \ldots, k$. While the convergence will slow down using aggregated cuts, the master problem can maintain a smaller number of constraints.

Remark 2 The column generation method [35,37] is a variant of the cutting-plane method that solves the dual of the Lagrangian master problem (13). Lubin et al. [35] demonstrated that the dual of the Lagrangian master has a dual angular structure that can be exploited using decomposition methods.

3 Algorithmic Innovations for Dual Decomposition

We now develop innovations for the dual decomposition method based on cutting planes. In Section 3.1 we present a procedure to construct valid inequalities that aid the elimination of infeasible first-stage solutions and that tighten the Lagrangian subproblems. As a byproduct, the procedure enables us to obtain upper bounds for SMIP (3). In Section 3.2 we present an interior-point method and termination criteria to stabilize the solutions of the Lagrangian master problem (13).

3.1 Tightening Inequalities for Subproblems

We consider two cases in which a set of valid inequalities can be generated to exclude a subset of (suboptimal) solutions for the subproblems. In the first case we consider a feasible Lagrangian subproblem solution that is infeasible for SMIP.

Proposition 2 *Let $(\hat{x}, \hat{y}) \in G_s$ for some $s \in \mathcal{S}$ and assume that there exists an scenario $s' \neq s$ such that there does not exist $y \in \mathbb{R}^{n_2}$ satisfying $(\hat{x}, y) \in G_{s'}$ for fixed $\hat{x}$. Let $\mu_{s'}$ be an optimal solution of the linear programming problem*

$$\max_{\mu} \left\{ \mu^T \left(h_{s'} - T_{s'} \hat{x} \right) : \mu^T W_{s'} \leq 0,\ |\mu| \leq 1 \right\}, \tag{15}$$

where the absolute value $|\cdot|$ is taken componentwise. The inequality

$$\mu_{s'}^T \left(h_{s'} - T_{s'} x \right) \leq 0 \tag{16}$$

excludes $\hat{x}$ from the set $\{x : (x, y) \in G_{s'}\}$ and is also valid for SMIP (3).

Proof From Farkas' lemma, there exists a $\mu_{s'} \in \mathbb{R}^{m_2}$ such that $\mu_{s'}^T W_{s'} \leq 0$ and $\mu_{s'}^T (h_{s'} - T_{s'} \hat{x}) > 0$, and thus hyperplane $\mu_{s'}^T \left(h_{s'} - T_{s'} x \right) \leq 0$ separates $\hat{x}$ from $\{x : (x, y) \in G_{s'}\}$. Moreover, this is valid for G_s and thus for SMIP (3). □

In the second case we consider a feasible subproblem solution that is also feasible with respect to SMIP. For this case, we present a set of valid inequalities that can tighten the subproblems by using an upper bound z_{UB} of SMIP.

Proposition 3 *Assume that there exists $y \in \mathbb{R}^{n_2}$ such that $(\hat{x}, y) \in G_{s'}$ for all $s' \in \mathcal{S}$ and for fixed $\hat{x}$. Let π_s be the optimal solution of the following recourse problem for each $s \in \mathcal{S}$:*

$$\max_{\pi} \left\{ \pi^T \left(h_s - T_s \hat{x} \right) : \pi^T W_s \leq q_s \right\}. \tag{17}$$

The inequality

$$c^T x + \sum_{s \in \mathcal{S}} \pi_s^T \left(h_s - T_s x\right) \leq z_{UB} \tag{18}$$

is valid for SMIP (3).

Proof Consider a Benders-like decomposition of SMIP with relaxation of second-stage integrality,

$$\begin{aligned} \min_x \quad & c^T x + \sum_{s \in \mathcal{S}} p_s q_s^T \hat{y}_s(x) \\ \text{s.t.} \quad & Ax = b,\ x \in X, \end{aligned}$$

where $\hat{y}_s(x) := \text{argmin}_y \{q_s^T y : W_s y = h_s - T_s x,\ y \geq 0\}$. This is equivalent to

$$\begin{aligned} \min_x \quad & c^T x + \sum_{s \in \mathcal{S}} p_s \hat{\pi}_s(x)^T (h_s - T_s x) \\ \text{s.t.} \quad & Ax = b,\ x \in X, \end{aligned}$$

where $\hat{\pi}_s(x) := \text{argmax}_\pi \{\pi^T (h_s - T_s x) : \pi^T W_s \leq q_s\}$. By assumption, there exists a solution π_s for (17) for each $s \in \mathcal{S}$. Because

$$\begin{aligned} c^T x + \sum_{s \in \mathcal{S}} \pi_s^T \left(h_s - T_s x\right) &\leq c^T x + \sum_{s \in \mathcal{S}} p_s^T \hat{\pi}_s(x)^T (h_s - T_s x) \\ &\leq z_{\text{UB}} \end{aligned}$$

holds for any feasible x, the inequality (18) is valid for SMIP (3). □

Procedure 1 summarizes the proposed cutting-plane procedure for solving the Lagrangian subproblems (8) by adding the valid inequalities (16) and (18). This procedure replaces lines 2 and 7 of the standard dual decomposition method of Algorithm 2.

We explain the cutting-plane procedure as follows. The situation in Proposition 2 can occur when SMIP does not have relatively complete recourse. Because inequality (16) is analogous to the feasibility cut of the L-shaped method [6], we call it a feasibility cut. This cut eliminates a candidate first-stage solution $\hat{x}$ that does not have a feasible recourse for the subproblem. We highlight, however, that infeasibility can only be guaranteed to be eliminated if it is detected at the root node of the branch-and-bound tree of the subproblem (8). It is important to emphasize that inequalities (16) and (18) generated in lines 6 and 14, respectively, *are added to all the Lagrangian subproblems* (lines 8 and 14). In other words, the inequalities generated for one scenario are *shared* with all the other scenarios. This seeks to prevent that a given subproblem visits a first-stage solution that is infeasible for another subproblem.

The inequality (18) of Proposition 3 is a supporting hyperplane that lower approximates the objective function of SMIP (3). Moreover, the inequality is parameterized by the best known upper bound z_{UB} and thus can be tightened

Procedure 1 Cutting-Plane Procedure for Lagrangian Subproblems

Require: λ^k
1: **for all** $s \in \mathcal{S}$ **do**
2: **repeat**
3: SOLVE subproblem (8) to obtain $D_s(\lambda^k)$ and (x_s^k, y_s^k) for λ^k
4: $isFeasible \leftarrow true$
5: **for all** $s' \in \mathcal{S} \backslash \{s\}$ **do**
6: SOLVE feasibility cut generator (15) to obtain $\mu_{s'}$ for x_s^k
7: **if** $\mu_{s'}^T(h_{s'} - T_{s'} x_s^k) > 0$ **then**
8: ADD feasibility cut (16) to all the subproblems (8)
9: $isFeasible \leftarrow false$
10: **end if**
11: **end for**
12: **until** $isFeasible = true$
13: UPDATE z_{UB} by solving (3) for fixed x_s^k
14: GENERATE optimality cut (18) by solving (17) for x_s^k and for all $s \in \mathcal{S}$
15: ADD optimality cut (18) to all the subproblems (8)
16: **end for**

as better upper bounds are obtained. In other words, the optimality cut seeks to eliminate first-stage solutions that go above a known upper bound. We call inequality (18) an optimality cut, because it is analogous to the optimality cut of the L-shaped method [6]. We also note that the same optimality cut is used for all the subproblems.

The next theorem shows that Procedure 1 terminates in a finite number of steps.

Theorem 1 *Procedure 1 terminates in a finite number of steps.*

Proof We need to show that only a finite number of feasibility cuts (16) are generated and this holds because there exists a finite number of bases in each one of the cut generation problems (15). Note that (15) is a linear program, where the objective function is parameterized by $\hat{x}$, and the corresponding bases do not change in $\hat{x}$. Consequently, only a finite number of cuts can be generated in the loop of lines 2-12 before finding a feasible solution (i.e., $isFeasible = true$ in line 12). □

3.2 Interior-Point Cutting-Plane Method for the Lagrangian Master Problem

From a computational standpoint, the simplex method is an efficient algorithm for solving the Lagrangian master problems. This efficiency is mostly because of its warm-starting capabilities [7]. The solutions of the Lagrangian master problem, however, oscillate significantly when the epigraph of the Lagrangian dual function is not well approximated because many near-optimal solutions of the master problem are present [19,40]. This oscillation can make the dual decomposition method numerically unstable and can lead to slow convergence (we illustrate this behavior in Figure 5 of Section 5). To avoid this situation, we solve the Lagrangian master problems *suboptimally* using an

interior-point method (IPM). As noted in [40], *early termination* can enable us to find stronger cuts and to avoid degeneracy.

IPM with early termination has been applied in the context of cutting-plane and column-generation methods [19,40,22,21]. We propose a new termination criterion specific to the dual decomposition context that uses upper-bound information to determine whether the IPM should be terminated. The IPM checks the termination criteria in the following order:

$$\sum_{s\in\mathcal{S}} \theta_s^k \geq z_{\mathrm{UB}}, \tag{19a}$$

$$g_k(\theta^k, \pi^k) < \epsilon_{\mathrm{IPM}}^k. \tag{19b}$$

Here, we define the relative duality gap of the primal-dual feasible solution (θ, λ, π) of the master (13) at iteration k as

$$g_k(\theta,\pi) := \frac{\sum_{s\in\mathcal{S}}\sum_{l=1}^{k} \pi_s^l \left(D_s(\lambda^l) - (H_s x_s^l)^T \lambda^l\right) - \sum_{s\in\mathcal{S}} \theta_s}{1 + \left|\sum_{s\in\mathcal{S}} \theta_s\right|}. \tag{20}$$

We denote $(\tilde{\theta}^k, \tilde{\lambda}^k, \tilde{\pi}^k)$ as a primal-dual feasible solution of the master (13) obtained at iteration k such that $g_k(\tilde{\theta}^k, \tilde{\pi}^k) < \epsilon_{\mathrm{IPM}}^k$ for some duality gap tolerance $\epsilon_{\mathrm{IPM}}^k > 0$ or $\sum_{s\in\mathcal{S}} \tilde{\theta}_s^k \geq z_{\mathrm{UB}}$ for the current upper bound $z_{\mathrm{UB}} < \infty$.

We adjust the tolerance $\epsilon_{\mathrm{IPM}}^k$ at each iteration of the dual decomposition procedure. The tolerance $\epsilon_{\mathrm{IPM}}^k$ can be made loose when the duality gap of the dual decomposition method is large and it is updated as follows:

$$\epsilon_{\mathrm{IPM}}^k := \min\left\{ \epsilon_{\mathrm{IPM}}^{max}, \frac{g_{k-1}(\tilde{\theta}^{k-1}, \tilde{\pi}^{k-1})}{\delta} + \frac{\tilde{m}_{k-1} - \sum_{s\in\mathcal{S}} D_s(\tilde{\lambda}^{k-1})}{1 + |\tilde{m}_{k-1}|} \right\}, \tag{21}$$

where $\tilde{m}_{k-1} := \sum_{s\in\mathcal{S}} \tilde{\theta}_s^{k-1}$ and $\delta > 1$ is the *degree of optimality* [21].

At first sight it seems possible that Algorithm 2 may not generate any cut because the solution of the master is terminated early. Propositions 4 and 5 show, however, that Algorithm 2 does not stall and eventually generates cuts or terminates with optimality.

Proposition 4 *Let $(\tilde{\theta}^k, \tilde{\lambda}^k, \tilde{\pi}^k)$ be a feasible suboptimal solution of the master (13) satisfying $g_k(\tilde{\theta}^k, \tilde{\pi}^k) < \epsilon_{IPM}^k$ at iteration k with ϵ_{IPM}^k defined in (21). If $\tilde{\theta}_s^k \leq D_s(\tilde{\lambda}^k)$ for all $s \in \mathcal{S}$, then $\epsilon_{IPM}^{k+1} \leq g_k(\tilde{\theta}^k, \tilde{\pi}^k) < \epsilon_{IPM}^k$.*

Proof Suppose that

$$\epsilon_{\mathrm{IPM}}^{k+1} = \frac{g_k(\tilde{\theta}^k, \tilde{\pi}^k)}{\delta} + \frac{\tilde{m}_k - \sum_{s\in\mathcal{S}} D_s(\tilde{\lambda}^k)}{1 + |\tilde{m}_k|} < \epsilon_{\mathrm{IPM}}^{max}.$$

Because $\tilde{\theta}_s^k \leq D_s(\tilde{\lambda}^k)$ holds for all $s \in \mathcal{S}$, we have $\tilde{m}_s^k - \sum_{s\in\mathcal{S}} D_s(\tilde{\lambda}^k) = \sum_{s\in\mathcal{S}} \left(\tilde{\theta}_s^k - D_s(\tilde{\lambda}^k)\right) \leq 0$ and $\epsilon_{\text{IPM}}^{k+1} \leq g_k(\tilde{\theta}^k, \tilde{\pi}^k)/\delta < g_k(\tilde{\theta}^k, \tilde{\pi}^k) < \epsilon_{\text{IPM}}^k$. □

Proposition 4 implies that the feasible solution at iteration k will not be the same as that at iteration $k+1$ with the new tolerance $\epsilon_{\text{IPM}}^{k+1}$ (even if no cut was generated at iteration k). In fact, the next iteration finds a feasible solution that, if not optimal, is closer to optimality.

We are interested only in feasible solutions satisfying $\sum_{s\in\mathcal{S}} \tilde{\theta}_s < z_{UB}$ for a given $z_{UB} < \infty$. Consequently, the IPM can be terminated with a feasible solution $(\tilde{\theta}, \tilde{\lambda}, \tilde{\pi})$ whenever $\sum_{s\in\mathcal{S}} \tilde{\theta}_s \geq z_{UB}$.

Proposition 5 *Let $(\tilde{\theta}^k, \tilde{\lambda}^k, \tilde{\pi}^k)$ be a feasible solution of the master (13) satisfying $\sum_{s\in\mathcal{S}} \tilde{\theta}_s^k \geq z_{UB}$ at iteration k for a given $z_{UB} < \infty$. If $\tilde{\theta}_s^k \leq D_s(\tilde{\lambda}^k)$ for all $s \in \mathcal{S}$, then $\sum_{s\in\mathcal{S}} D_s(\tilde{\lambda}^k) = z_{UB}$.*

Proof Because $\tilde{\theta}_s^k \leq D_s(\tilde{\lambda}^k)$ holds for all $s \in \mathcal{S}$, we have

$$z_{\text{UB}} \leq \sum_{s\in\mathcal{S}} \tilde{\theta}_s^k \leq \sum_{s\in\mathcal{S}} D_s(\tilde{\lambda}^k) \leq z_{\text{LD}}.$$

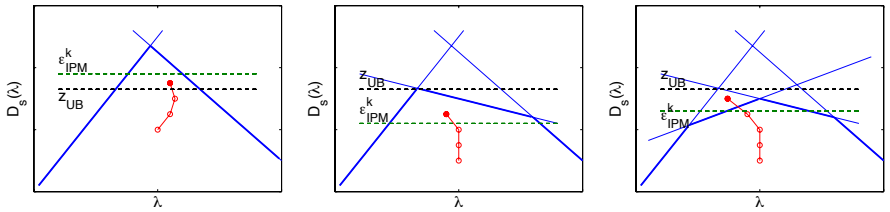
Because $z_{\text{LD}} \leq z_{\text{UB}}$, we have $z_{\text{LD}} = \sum_{s\in\mathcal{S}} D_s(\tilde{\lambda}^k) = z_{\text{UB}}$. □

Proposition 5 states that a feasible master solution $(\tilde{\theta}^k, \tilde{\lambda}^k, \tilde{\pi}^k)$ satisfying $\sum_{s\in\mathcal{S}} \tilde{\theta}_s^k \geq z_{\text{UB}}$ at iteration k is optimal if no cut is generated for such a solution. The following corollary also suggests that at least a cut is generated for the feasible solution if $\sum_{s\in\mathcal{S}} \tilde{\theta}_s^k > z_{\text{UB}}$.

Corollary 1 *Let $(\tilde{\theta}^k, \tilde{\lambda}^k, \tilde{\pi}^k)$ be a feasible solution of the master (13) satisfying $\sum_{s\in\mathcal{S}} \tilde{\theta}_s^k > z_{UB}$ at iteration k for given ϵ_{IPM}^k and z_{UB}. There exists some $s \in \mathcal{S}$ such that $\tilde{\theta}_s^k > D_s(\tilde{\lambda}^k)$.* □

Figure 1 illustrates different termination cases for the IPM under the proposed criteria (19). Figure 1a shows the IPM terminated by criterion (19a) prior to satisfying condition (19b). In Figure 1b, the IPM terminated by criterion (19b), but no cut was generated at the solution. Hence, from Proposition 4, we decrease the tolerance ϵ_{IPM}^k by the rule (21). The IPM terminated with the decreased tolerance at the next iteration in Figure 1c.

We modify the dual decomposition method of Algorithm 2 in order to use the interior-point cutting-plane method with the proposed termination criteria (19) for the master problems (13). Moreover, we solve the Lagrangian subproblems (8) by using Procedure 1. We denote the duality gap tolerance for optimality of the IPM by ϵ_{Opt}; that is, (θ^k, π^k) is an optimal solution for the master (13) at iteration k if $g_k(\theta^k, \pi^k) < \epsilon_{\text{Opt}}$.



(a) Terminated by criterion (19a)

(b) Terminated by criterion (19b), but no cut generation

(c) Terminated by criterion (19b) with decreased ϵ^k_{IPM}

Fig. 1: Illustration of different termination cases of the IPM with the proposed criteria

Algorithm 3 Dual Decomposition Based on Interior-Point Cutting-Plane Method (DSP)

$k \leftarrow 0, \lambda^0 \leftarrow 0$ and $z_{\text{UB}} \leftarrow \infty$
CALL Procedure 1 to obtain $D_s(\lambda^k)$ and (x_s^k, y_s^k) for given λ^k
ADD cutting-planes (13b) to the master (13) for given $D(\lambda^k)$ and x_s^k
$z_{\text{LB}} \leftarrow D(\lambda^k)$.
loop
  SOLVE the master (13) by the IPM to obtain $(\theta^{k+1}, \lambda^{k+1})$
  CALL Procedure 1 to obtain $D_s(\lambda^{k+1})$ and (x_s^{k+1}, y_s^{k+1}) for given λ^{k+1}
  if $(\theta^{k+1}, \lambda^{k+1})$ is obtained from (19a) **then**
    if $\theta_s^{k+1} \leq D_s(\lambda^{k+1})$ for all $s \in \mathcal{S}$ **then**
      STOP
    else
      ADD cutting-planes (13b) to the master (13) for given $D(\lambda^{k+1})$ and x_s^{k+1}
    end if
  else if $(\theta^{k+1}, \lambda^{k+1})$ is obtained from (19b) **then**
    if $\theta_s^{k+1} \leq D_s(\lambda^{k+1})$ for all $s \in \mathcal{S}$ **then**
      if $\epsilon^k_{\text{IPM}} > \epsilon_{\text{Opt}}$ **then**
        UPDATE $\epsilon^{k+1}_{\text{IPM}}$ from (21)
      else
        STOP
      end if
    else
      ADD cutting-planes (13b) to the master (13) for given $D(\lambda^{k+1})$ and x_s^{k+1}
    end if
  end if
  $z_{\text{LB}} \leftarrow \max\{z_{\text{LB}}, D(\lambda^k)\}$.
  $k \leftarrow k + 1$
end loop

Theorem 2 *Algorithm 3 terminates after a finite number of iterations with an optimal solution of the Lagrangian dual problem (9).*

Proof We consider the following two cases:

– Assume that a feasible solution of the master problem is found that satisfies $\sum_{s\in\mathcal{S}} \tilde{\theta}_s^{k+1} \geq z_{\text{UB}}$ for given $z_{\text{UB}} < \infty$. We have two subcases:

  - If $\tilde{\theta}_s^k \leq D_s(\tilde{\lambda}^k)$ holds for all $s \in \mathcal{S}$, we have from Proposition 5 that the algorithm terminates with optimality (line 10).
  - Otherwise, from Corollary 1, we must have that the algorithm excludes the current solution by adding cutting-planes (13b) (line 12).
- Assume that a feasible solution of the master problem is found that satisfies $g_k(\tilde{\theta}^k, \tilde{\lambda}^k) < \epsilon_{\text{IPM}}^k$. We have two subcases:
  - If $\tilde{\theta}_s^k \leq D_s(\tilde{\lambda}^k)$ holds for all $s \in \mathcal{S}$, we have from Proposition 4 that the algorithm reduces ϵ_{IPM}^k by a factor of δ (line 17). An optimal master solution can thus be obtained after a finite number of reductions in ϵ_{IPM}^k. (line 19)
  - Otherwise, the algorithm excludes the current solution by adding (13b) (line 22).

Each case shows either a finite termination or the addition of cuts (13b). Because a finite number of inequalities are available for (13b) and because Procedure 1 also terminates in a finite number of steps from Theorem 1, the algorithm terminates after a finite number of steps. □

We highlight that Procedure 1 does not interfere with the finite termination of DSP because it always adds valid inequalities.

4 DSP: An Open-Source Package for SMIP

We now introduce `DSP`, an open-source software package that provides serial and parallel implementations of different decomposition methods for solving SMIPs. `DSP` implements the dual decomposition method of Algorithm 3 as well as standard dual decomposition methods (Algorithms 1 and 2) and a standard Benders decomposition method. We describe software implementation details and different avenues for the user to call the solver.

4.1 Software Design

The software design is object-oriented and implemented in C++. It consists of *Model* classes and *Solver* classes for handling optimization models and scenario data.

4.1.1 Model Classes

An abstract *Model* class is designed to define a generic optimization model data structure. The *StoModel* class defines the data structure for generic stochastic programs, including two-stage stochastic programs and multistage stochastic programs. The underlying data structure of *StoModel* partially follows the SMPS format. The class also defines core functions for problem decomposition. The *TssModel* class derived defines the member variables and functions specific to two-stage stochastic programs and decompositions. Following the design of

the model classes, users are able to derive new classes for their own purposes and efficiently manage model structure provided from several interfaces (e.g., `StochJuMP` and SMPS; see Section 4.2).

4.1.2 Solver Classes

An abstract *Solver* class is designed to provide different algorithms for solving stochastic programming problems defined in the *Model* class. `DSP` implements the *TssSolver* class to define solvers specific to two-stage stochastic programs. From the *TssSolver* class, three classes are derived for each method: *TssDe*, *TssBd*, and *TssDd*.

The *TssDe* class implements a wrapper of external solvers to solve the extensive form of two-stage stochastic programs. The extensive form is constructed and provided by the *TssModel* class.

The *TssBd* class implements a Benders decomposition method for solving two-stage stochastic programs with continuous recourse. A proper decomposition of the model is performed and provided by the *TssModel* class, while the second-stage integrality restriction is automatically relaxed. Depending on parameters provided, *TssModel* can make a different form of the problem decomposition for *TssBd*. For example, the user can specify the number of cuts added per iteration, which determines the number of auxiliary variables in the master problem of Benders decomposition. Moreover, the Benders master can be augmented for a subset $\widetilde{\mathcal{S}}$ of scenarios as follows:

$$
\begin{aligned}
\min \quad & c^T x + \sum_{s \in \widetilde{\mathcal{S}}} p_s q_s^T y_s + \theta \\
\text{s.t.} \quad & Ax = b, \\
& T_s x + W_s y_s = h_s, \ \forall s \in \widetilde{\mathcal{S}}, \\
& x \in X, \ y_s \in Y, \ \forall s \in \widetilde{\mathcal{S}}, \\
& \theta \geq \sum_{s \in S \setminus \widetilde{\mathcal{S}}} p_s Q(x, \omega_s),
\end{aligned}
$$

where $\widetilde{\mathcal{S}}$ is given be user.

The *TssDd* class implements the proposed dual decomposition method for solving two-stage stochastic programs with mixed-integer recourse. For this method, an abstract *TssDdMaster* class is designed to implement methods for updating the dual variables. The subgradient method in Algorithm 1 and the cutting-plane method in Algorithm 2 are implemented in such derived classes. Moreover, a subclass derived from the *TssBd* is reused for implementing the cutting-plane procedure from Procedure 1. Users can customize this cutting-plane procedure by incorporating more advanced Benders decomposition techniques such as convexification of the recourse function [46,45,15,51]. An l_∞-norm trust region is also applied to Algorithm 2 in order to stabilize the cutting-plane method. The rule of updating the trust region follows that

proposed in [32]. Users can also implement their own method for updating the dual variables.

4.1.3 External Solver Interface Classes

DSP uses external MIP solvers to solve subproblems under different decomposition methods. The *SolverInterface* class is an abstract class to create interfaces to the decomposition methods implemented. Several classes are derived from the abstract class in order to support specific external solvers. The current implementation supports three LP solvers (Clp [14], SoPlex [50], and OOQP [18]) and a mixed-integer solver (SCIP [1]). Users familiar with the COIN-OR Open Solver Interface [44] should easily be able to use the *SolverInterfaceOsi* class to derive classes for other solvers (e.g., CPLEX [27], Gurobi [24]).

4.1.4 Parallelization

The proposed dual decomposition method can be run on distributed memory and on shared memory computing systems with multiple cores. The implementation protocol is MPI. In a distributed memory environment, the scenario data and corresponding Lagrangian subproblems are distributed to multiple processors based on scenario indices. The root processor updates the Lagrangian multipliers and solves a subset of the subproblems. The users can also compile the code with the flag `-DUSE_ROOT_PROCESSOR=0` in order to require the root processor to update only the dual variables. When solving the subproblems in distributed computing nodes, subproblem solutions and the dual variables must be communicated with the root processor. In addition, each computing node communicates the primal first-stage solutions and the valid inequalities generated for a subproblem with the rest of the nodes.

4.2 Interfaces for C, StochJuMP, and SMPS

The source code of DSP is compiled to a shared object library with C API functions defined in *StoCInterface.h*. Users can load the shared object library with the implementation of the model to call the API functions. We also provide interfaces to StochJuMP and SMPS files.

StochJuMP is a scalable algebraic modeling package for stochastic programming problems based on the mathematical programming modeling package JuMP [4,34,26]. StochJuMP enables the generation of large-scale problems in parallel environments and thus overcomes memory and timing bottlenecks. Moreover, it exploits the algebraic modeling capabilities of JuMP to specify problems in a concise format and Julia programming language, which enables the easy handling of data and the use of other tools (e.g., statistical analysis and plotting tools). To illustrate these capabilities, we present a StochJuMP implementation of the two-stage stochastic integer program with integer recourse presented in [8].

$$\min \left\{ -1.5\, x_1 - 4\, x_2 + \sum_{s=1}^{3} p_s Q(x_1, x_2, \xi_1^s, \xi_2^s) : x_1, x_2 \in \{0, \ldots, 5\} \right\},$$

where

$$\begin{aligned} Q(x_1, x_2, \xi_1^s, \xi_2^s) = \min_{y_1, y_2, y_3, y_4} \quad & -16y_1 + 19y_2 + 23y_3 + 28y_4 \\ \text{s.t.} \quad & 2y_1 + 3y_2 + 4y_3 + 5y_4 \leq \xi_1^s - x_1 \\ & 6y_1 + y_2 + 3y_3 + 2y_4 \leq \xi_2^s - x_2 \\ & y_1, y_2, y_3, y_4 \in \{0, 1\} \end{aligned}$$

and $(\xi_1^s, \xi_2^s) \in \{(7,7), (11,11), (13,13)\}$ with probability 1/3. The corresponding StochJuMP model reads:

```
1  using DSPsolver, StochJuMP, MPI; # Load packages
2  MPI.Init(); # Initialize MPI
3  m = StochasticModel(3); # Create a Model object with three scenarios
4  xi = [[7,7] [11,11] [13,13]]; # random parameter
5  @defVar(m, 0 <= x[i=1:2] <= 5, Int);
6  @setObjective(m, Min, -1.5*x[1]-4*x[2]);
7  @second_stage m s begin
8      q = StochasticBlock(m, 1/3);
9      @defVal(q, y[j=1:4], Bin);
10     @setObjective(q, Min, -16*y[1]+19*y[2]+23*y[3]+28*y[4]);
11     @addConstraint(q, 2*y[1]+3*y[2]+4*y[3]+5*y[4]<=xi[1,s]-x[1]);
12     @addConstraint(q, 6*y[1]+1*y[2]+3*y[3]+2*y[4]<=xi[2,s]-x[2]);
13 end
14 DSPsolver.loadProblem(m); # Load model m to DSP
15 DSPsolver.solve(DSP_SOLVER_DD); # Solve problem using dual decomposition
16 MPI.Finalize(); # Finalize MPI
```

In the first line of this script we include DSP, StochJuMP and MPI packages. The StochJuMP model is given in lines 3 to 13. The first-stage is defined in lines 5 and 6 and the second stage in lines 8 to 12 for each scenario. DSP reads and solves the model in lines 14 and 15, respectively. Note that only two lines of code (14 and 15) are required to invoke the parallel decomposition method.

In the script provided we solve the problem using the dual decomposition method described in Section 3 (line 15). Alternatively, users can use Benders decomposition by replacing line 15 with

```
DSPsolver.solve(DSP_SOLVER_BD);
```

or directly solve the extensive form by replacing line 15 with

```
DSPsolver.solve(DSP_SOLVER_DE);
```

DSP can also read a model provided in SMPS files [5]. In this format, a model is defined by three files: core, time, and stochastic with file extensions of `.cor`, `.tim`, and `.sto`, respectively. The core file defines the deterministic version of the model with a single reference scenario, the time file indicates a

row and a column that split the deterministic data and stochastic data in the constraint matrix, and the stochastic file defines random data. The user can load `SMPS` files and call `DSP` using the `Julia` interface as follows.

```
# Read SMPS files: example.cor, example.tim and example.sto
DSPsolver.readSmps("example");
```

5 Computational Experiments

We present computational experiments to demonstrate the capabilities of `DSP`. We solve publicly available test problem instances (Section 5.1) and a stochastic unit commitment problem (Section 5.2). All experiments were run on *Blues*, a 310-node computing cluster at Argonne National Laboratory. Blues has a QLogic QDR InfiniBand network, and each node has two octo-core 2.6 GHz Xeon processors and 64 GB of RAM.

5.1 Test Problem Instances

We use `dcap` and `sslp` problem instances from the SIPLIB test library available in http://www2.isye.gatech.edu/~sahmed/siplib/. Characteristics of the problem instances are described in Table 1. The first column of the table lists the names of the problem instances, in which S is substituted by the number of scenarios in the second column. The other columns present the number of rows, columns, and integer variables for each stage, respectively. Note that the `dcap` instances have a mixed-integer first stage and a pure-integer second stage, whereas the `sslp` instances have a pure-integer first stage and a mixed-integer second stage.

Table 1: Characteristics of the SIPLIB instances

		First Stage			Second Stage		
Name	Scenarios (S)	Rows	Cols	Ints	Rows	Cols	Ints
`dcap233_`S	$200, 300, 500$	6	12	6	15	27	27
`dcap243_`S	$200, 300, 500$	6	12	6	18	36	36
`dcap332_`S	$200, 300, 500$	6	12	6	12	24	24
`dcap342_`S	$200, 300, 500$	6	12	6	14	32	32
`sslp_5_25_`S	$50, 100$	1	5	5	30	130	125
`sslp_10_50_`S	$5, 10, 15$	1	10	10	60	510	500
`sslp_15_45_`S	$50, 100$	1	15	15	60	690	675

5.1.1 Benchmarking Dual-Search Strategies

We experiment with different methods for updating the dual variables: DDSub of Algorithm 1, DDCP of Algorithm 2, and DSP of Algorithm 3. In this experiment, DDSub initializes parameters $\beta_0 = 2$ and $\gamma^{\max} = 3$ for the step-size rule

and terminates if one of the following conditions is satisfied: (i) the optimality gap, $(z_{\mathrm{UB}} - z_{\mathrm{LB}})/|10^{-10} + z_{\mathrm{UB}}|$, is less than 10^{-5}; (ii) $\beta_k \leq 10^{-6}$; and (iii) the solution time exceeds 6 hours. DDCP solves the master problem by using the simplex method implemented in `Soplex-2.0.1` [50] by taking advantage of warm-start information from the previous iteration. DSP solves the master problem by using Mehrotra's predictor-corrector algorithm [39] implemented in `OOQP-0.99.25` [18]. Each Lagrangian subproblem is solved by `SCIP-3.1.1` with `Soplex-2.0.1`. All methods use the initial dual values $\lambda^0 = 0$. This experiment was run on a single node with 16 cores.

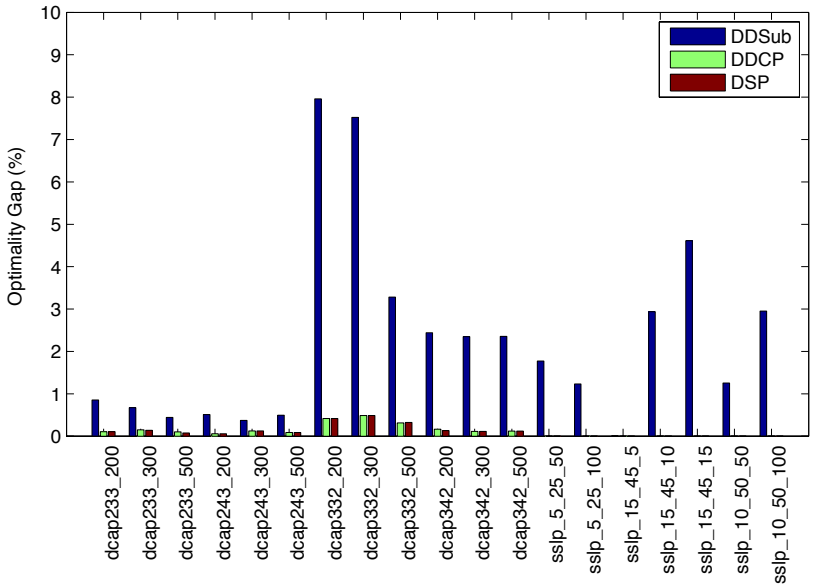


Fig. 2: Optimality gap obtained by different dual decomposition methods

Figure 2 presents the optimality gap obtained by different dual decomposition methods for each SIPLIB instance. DDCP and DSP found optimal lower bounds z_{LD} for all the test problems, whereas DDSub found only suboptimal values. DSP also found tighter upper bounds for the instances `dcap233_300`, `dcap233_500`, `dcap332_200` and `dcap342_200` than DDCP (see Table 2).

Figure 3 presents the solution time and the number of iterations relative to those from DSP for each SIPLIB instance. DSP reduced the solution time and the number of iterations (relative to DDSub) by factors of up to 199 and 78, respectively. Relative to DDCP, the reductions in solution time and number of iterations by factors of up to 7 and 5 achieved, respectively. DDSub terminated earlier than DSP for instances `dcap_332_x` and `dcap_342_x` because of the termination criterion $\beta_k < 10^{-6}$. For these instances poor-quality lower

Table 2: Computational results for SIPLIB instances using different dual decomposition methods.

Instance	Scenarios	Method	Iter	Upper Bound	Lower Bound	Gap (%)	Wall time (sec.)
dcap233	200	DDSub	2024	1835.34	1819.66	0.85	3349
		DDCP	71	1835.34	1833.38	0.11	1365
		DSP	36	1835.34	1833.38	0.11	901
	300	DDSub	1218	1645.22	1634.25	0.67	6263
		DDCP	93	1645.22	1642.73	0.15	3412
		DSP	40	1645.01	1642.73	0.14	2094
	500	DDSub	2135	1737.95	1730.31	0.44	19184
		DDCP	115	1738.18	1736.68	0.09	9451
		DSP	40	1737.95	1736.64	0.08	5390
dcap243	200	DDSub	1742	2322.51	2310.71	0.51	3489
		DDCP	48	2322.51	2321.15	0.06	1241
		DSP	33	2322.51	2321.19	0.06	936
	300	DDSub	2995	2559.49	2549.60	0.39	7782
		DDCP	56	2559.86	2556.68	0.12	2814
		DSP	43	2559.86	2556.69	0.12	2186
	500	DDSub	2489	2167.36	2156.76	0.49	19099
		DDCP	67	2167.36	2165.47	0.09	8831
		DSP	36	2167.36	2165.47	0.09	6540
dcap332	200	DDSub	126	1068.25	983.24	7.96	660
		DDCP	88	1065.88	1059.09	0.64	1174
		DSP	43	1063.53	1059.09	0.42	916
	300	DDSub	126	1260.73	1165.90	7.52	1304
		DDCP	85	1257.02	1250.91	0.49	2951
		DSP	53	1257.02	1250.91	0.49	2194
	500	DDSub	126	1593.52	1541.25	3.28	3637
		DDCP	76	1592.22	1587.05	0.32	6732
		DSP	44	1592.22	1587.07	0.32	5312
dcap342	200	DDSub	126	1621.69	1582.10	2.44	735
		DDCP	79	1620.77	1618.08	0.17	1251
		DSP	42	1620.19	1618.08	0.13	949
	300	DDSub	126	2068.69	2020.11	2.35	1644
		DDCP	81	2067.77	2065.42	0.11	2808
		DSP	41	2067.77	2065.44	0.11	2203
	500	DDSub	126	1906.18	1861.25	2.36	4451
		DDCP	87	1905.27	1902.99	0.12	7824
		DSP	42	1905.27	1902.99	0.12	5970
sslp_5_25	50	DDSub	645	-121.60	-123.76	1.77	246
		DDCP	29	-121.60	-121.60	0.00	15
		DSP	5	-121.60	-121.60	0.00	7
	100	DDSub	995	-127.37	-128.94	1.23	648
		DDCP	36	-127.37	-127.37	0.00	39
		DSP	5	-127.37	-127.37	0.00	20
sslp_15_45	5	DDSub	914	-262.40	-262.42	0.01	2490
		DDCP	21	-262.40	-262.40	0.00	164
		DSP	5	-262.40	-262.40	0.00	32
	10	DDSub	427	-260.50	-268.15	2.94	8867
		DDCP	88	-260.50	-260.50	0.00	1988
		DSP	17	-260.50	-260.50	0.00	515
	15	DDSub	541	-253.60	-265.30	4.61	> 21600
		DDCP	89	-253.60	-253.60	0.00	14917
		DSP	17	-253.60	-253.60	0.00	3092
sslp_10_50	50	DDSub	774	-364.64	-369.21	1.25	3105
		DDCP	62	-364.64	-364.64	0.00	410
		DSP	11	-364.64	-364.64	0.00	174
	100	DDSub	582	-354.19	-365.09	3.08	4676
		DDCP	80	-354.19	-354.19	0.00	1143
		DSP	12	-354.19	-354.19	0.00	594

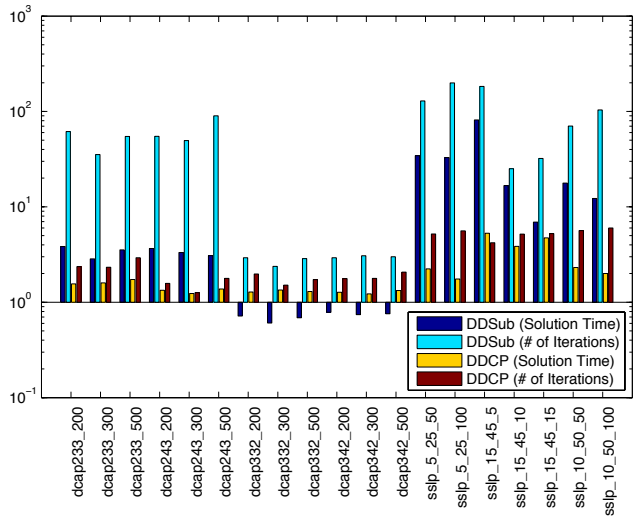


Fig. 3: Solution time and number of iterations ratios relative to those of DSP.

bounds are provided. DDSub did not terminate within the time limit of 6 hours for instance `sslp_15_45_15`. Detailed numerical results are given in Table 2.

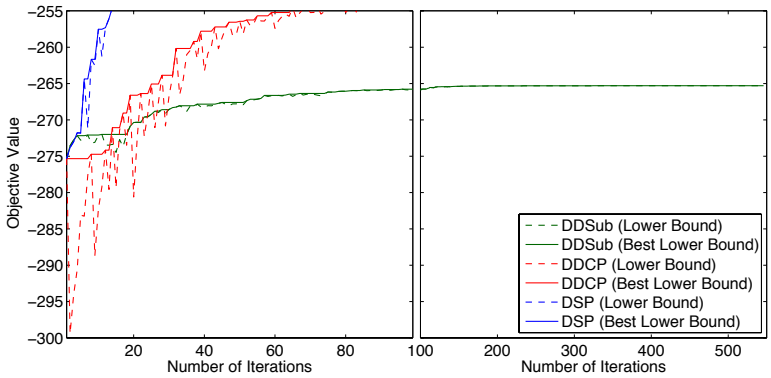


Fig. 4: Dual objective values and best lower bounds for SIPLIB instance `sslp_15_45_15` under different dual search methods.

Figure 4 shows the Lagrangian dual values obtained in each iteration for `sslp_15_45_15` by using DDSub, DDCP, and DSP. The lower bound is -275.7 at iteration $k = 0$. Fewer oscillations in the lower bounds and a faster solution

were obtained with DSP compared with the standard cutting-plane method DDCP. Figure 5 presents the Euclidean distance of the dual variable values between consecutive iterations ($\|\lambda^{k+1} - \lambda^k\|_2$). The figure is truncated at iteration 100. As can be seen, DSP dual updates oscillate less compared with DDCP updates. DDSub updates seem stable, but this is because of slow progress in the dual variables. These results highlight the benefits gained by the use of IPM and early termination

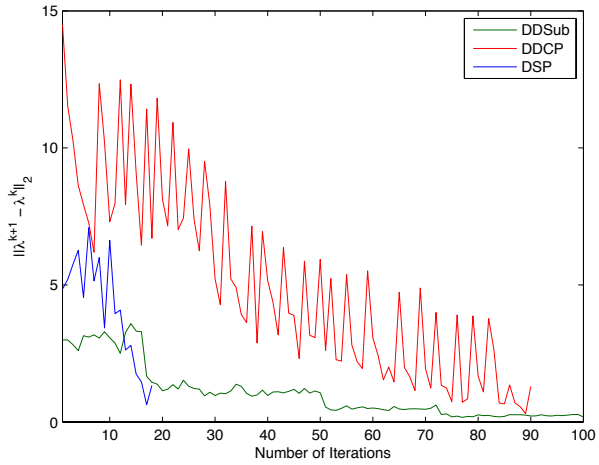


Fig. 5: Changes of the dual variables by different methods for SIPLIB instance `sslp_15_45_15`.

5.1.2 Impact of Second-Stage Integrality

We now use `DSP` to analyze the impact of relaxing recourse integrality in Benders decomposition (this approach is often used as a heuristic). Figure 6 shows that the optimality gap improved (i.e., the DSP optimality gap - the Benders optimality gap). Upper bounds for Benders decomposition were calculated by evaluating the first-stage solutions obtained from relaxation of the recourse function.

For the `dcap` instances, Benders decomposition poorly approximates lower bounds by relaxing the second-stage integrality. The largest gap was 86% obtained for `dcap332_500` with Benders decomposition, compared with 0.32 % obtained with dual decomposition. Benders solutions for `sslp_15_45_x` and `sslp_10_50_x` have duality gaps $> 0.2\%$, whereas the dual decomposition solutions are optimal for all problem instances.

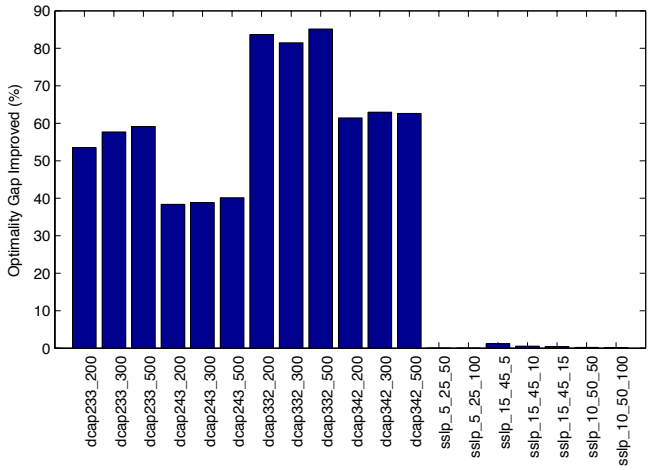


Fig. 6: Improvement in optimality gaps with DSP relative to Benders decomposition.

5.2 Large-Scale Stochastic Unit Commitment

We test the dual decomposition method on a large-scale day-ahead stochastic unit commitment model. In this model, thermal power generators are scheduled over a day. The schedules are subjected to uncertainty in wind power. We use a modified IEEE 188-bus system with 54 generators, 118 buses, and 186 transmission lines provided in [31]. We assume that 17 of the 54 generators are allowed to start on demand (second stage) whereas the other generators should be scheduled in advance (first stage). We also consider 3 identical wind farms each consisting of 120 wind turbines. The demand load is 3,095 MW on average, with a peak of 3,733 MW. The wind power generation level is 494 MW on average, with a peak of 916 MW for the 64 scenarios generated. Figure 7 shows the 64 scenarios (grey lines) of wind power generation and the mean levels (red lines). We used real wind speed data predicted from the observations of 31 weather stations in Illinois, USA. The mathematical formulation and the `StochJuMP` modeling script of the two-stage stochastic unit commitment model is provided in Appendices A, B and C.

Table 3 presents the size of the stochastic unit commitment instances with 4, 8, 16, 32, and 64 scenarios. The first stage has 10,727 constraints and 2,592 variables including 864 integer variables, and the second stage has 27,322 constraints and 9,072 variables including 432 integer variables.

Table 4 presents the numerical results. Each instance uses the same number of computing cores as scenarios. We add only one aggregated cut to the master problem in each iteration (see Remark 1). DSP found upper and lower bounds with $< 0.01\%$ optimality gap for the 4-, 8-, 16-, and 32-scenario in-

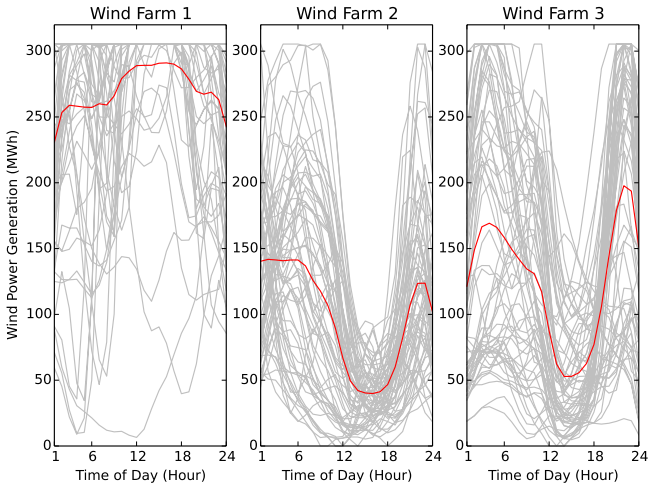


Fig. 7: Wind power generation scenarios for wind farms considered in the stochastic unit commitment model.

Table 3: Characteristics of stochastic unit commitment instances

Scenarios	# Rows	# Columns	# Integers
4	120,015	38,880	2,592
8	229,303	75,168	4,320
16	447,879	147,744	7,776
32	885,031	292,896	14,688
64	1,759,335	583,200	28,512

Table 4: Numerical results for stochastic unit commitment problem under DSP.

Scenarios	Iter	Upper Bound	Lower Bound	Gap (%)	Wall time (sec.)
4	1	907046.1	906979.1	< 0.01	590
8	1	904006.6	903953.5	< 0.01	785
16	1	900706.3	900650.7	< 0.01	1293
32	18	903227.7	903149.9	< 0.01	19547
64	5	895118.0	894756.5	0.04	> 21600

stances. Moreover, DSP terminated after the first iteration for the 4-, 8- and 16-scenario instances because of the addition of valid inequalities to the subproblems. Most notably, these results are not possible without Procedure 1, which is evident in Figure 8. We observe that the solution time per iteration increases as the number of scenarios increases. The reason is that the method generates more valid inequalities in Procedure 1 and evaluates more solutions to update upper bounds, causing imbalanced computing load among scenarios. In particular, each node needs to evaluate the recourse function for its local first-stage variables and for all scenarios in order to determine the best

possible upper bound. This step is currently done sequentially and its time can be reduced by considering additional computing nodes. The master problem solution time was less than 2 seconds per iteration and thus not a bottleneck in parallel solution time.

We now illustrate the impact of Procedure 1. For the stochastic unit commitment with 8 scenarios, we use DDSub, DSP without Procedure 1 (DSP-P1), and DSP with Procedure 1 (DSP+P1). Figure 8 shows the best upper bound and the best lower bound at each iteration. As can be seen, DSP+P1 obtained upper and lower bounds with $< 0.01\%$ duality gap at the first iteration and terminated, whereas DSP-P1 and DDSub were not able to find upper bounds for the first 53 iterations and the first 47 iterations, respectively, *because the problem does not have relatively complete recourse.* Moreover, DSP+P1 found tighter lower bounds than did DDSub and DSP-P1, because of the ability to tighten the subproblems by Procedure 1. The figure also shows that DSP-P1 still found better lower and upper bounds than did DDSub.

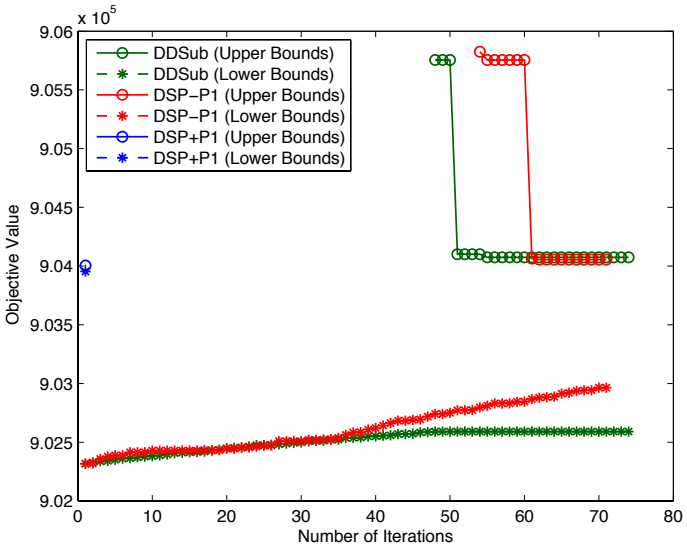


Fig. 8: Upper bounds and lower bounds obtained with DSP with and without Procedure 1 and with the subgradient method.

We conclude this section by reporting the computational results from solving the extensive form of the stochastic unit commitment problems. Th results are summarized in Table 5. The extensive form of each instance is solved by `SCIP` on a single node with a single core. For all the instances (except the 4-scenario instance) with 6-hour time limit, upper and lower bounds obtained from the extensive forms were not better than those obtained with DSP.

Table 5: Numerical results for the extensive form of the stochastic unit commitment problems

Scenarios	Number of B&C nodes	Upper Bound	Lower Bound	Gap (%)	Wall time (sec.)
4	88831	907035.3	906089.9	0.01	6632
8	58235	904068.1	903567.8	0.05	> 21600
16	3505	900806.1	900200.3	0.07	> 21600
32	9	907536.0	901759.8	0.64	> 21600
64	1	∞	33605.4	∞	> 21600

6 Summary

We have provided algorithmic innovations for the dual decomposition method. Our first innovation is a procedure to generate valid inequalities that exclude infeasible first-stage solutions in the absence of relatively complete recourse and that tighten subproblem solutions. Hence, we incorporate the Benders-type cut generation within the dual decomposition method. Our second innovation is an interior-point cutting-plane method with early termination criteria to solve the Lagrangian master problem. We have proved that the dual decomposition method incorporating our innovations converge in a finite number of iterations. We have introduced `DSP`, a software package that provides implementations of the dual decomposition method proposed as well as other standard methods such as Benders decomposition and a subgradient-based dual decomposition method. `DSP` also allows users to specify large-scale models in C, `StochJuMP`, and SMPS formats. The object-oriented design of the implementation allows users to customize decomposition techniques. We use `DSP` to benchmark different methods on standard problem instances and stochastic unit commitment problems. The numerical results showed significant improvements in terms of the quality of upper bounds, number of iterations, and time for all instances.

As part of future work, we will seek to scale our method to solve problems with a larger number of scenarios. In particular, we have observed load imbalances in the subproblem solution times, because each Lagrangian subproblem is a large mixed-integer program. These can be alleviated by asynchronous parallel implementation. Moreover, solving the master problem can be a bottleneck as cuts are accumulated. A warm-start technique for the primal-dual interior-point method [20] and a parallelization of the master problem as proposed in [35] can ameliorate this effect. We will address these issues in future work.

A Notations: Two-Stage Stochastic Unit Commitment

We present notations for the two-stage stochastic unit commitment considered in Section 5.2.

Table 6: Notations for the stochastic unit commitment model

Sets:	
$\mathcal{G}$	Set of all generators
$\mathcal{G}_s$	Set of slow generators
$\mathcal{G}_f$	Set of fast generators
$\mathcal{K}$	Set of linear segments of the piece-wise linear power generation cost
$\mathcal{L}$	Set of transmission lines
$\mathcal{N}$	Set of buses
$\mathcal{S}$	Set of scenarios
$\mathcal{T}$	Set of time periods
$\mathcal{W}$	Set of wind power generators
Parameters:	
C_g^{up}	Start-up cost of generator g
C_g^{dn}	Shut-down cost of generator g
C_g^{fx}	Fixed cost of operating the generator g
C_{gk}^{mar}	kth marginal cost of production of generator g
X_g^{init}	Initial on/off status of generator g
UT_g^{init}	Initial minimum uptime of generator g
UT_g	Minimum uptime of generator g
DT_g^{init}	Initial minimum downtime of generator g
DT_g	Minimum downtime of generator g
RU_g	Ramp-up limit of generator g
RD_g	Ramp-down limit of generator g
RC_g	Ramping capacity of generator g
P_g^{init}	Initial power output of generator g
P_g^{min}	Minimum power output of generator g
P_g^{max}	Maximum power output of generator g
Q_{gk}^{max}	Maximum power output of generator g with the kth marginal cost
SR_t	Spinning reserve required at time t
F_l^{max}	Maximum power flow of transmission line l
LSF_{ln}	Load-shift factor of transmission line l with respect to bus n
π_s	Probability of scenario s
D_{jnt}	Demand load at bus n at time t in scenario j
W_{jwt}	Wind power generation from generator w at time t in scenario j
Decision variables:	
x_{jgt}	On/off indicator of generator g at time t in scenario j
u_{jgt}	Start-up indicator of generator g at time t in scenario j
v_{jgt}	Shut-down indicator of generator g at time t in scenario j
p_{jgt}	Power output of generator g at time t in scenario j
q_{jgkt}	Power output of generator g at time t with the kth marginal cost in scenario j

B Formulation: Two-Stage Stochastic Unit Commitment

We present a two-stage stochastic unit commitment model formulation, where the commitment decisions for slow generators are made in the first stage and the commitment decisions for fast generators and the power dispatch decision are made in the second stage. In the

model, we consider ramping constraints, reserve constraints and transmission line capacity constraints. We assume that the power generation cost is piecewise linear convex.

$$\min \sum_{j\in\mathcal{S}}\sum_{t\in\mathcal{T}}\sum_{g\in\mathcal{G}} \pi_j \left(C_g^{\mathrm{fx}} x_{jgt} + C_g^{\mathrm{up}} u_{jgt} + C_g^{\mathrm{dn}} v_{jgt} + \sum_{k\in K} C_{gk}^{\mathrm{mar}} q_{jgkt} \right) \tag{22a}$$

$$\text{s.t. } 1 - x_{jg(t-1)} \geq u_{jgt},\ \forall j \in \mathcal{S}, g \in \mathcal{G}, t \in \mathcal{T}, \tag{22b}$$

$$x_{jg(t-1)} \geq v_{jgt},\ \forall j \in \mathcal{S}, g \in \mathcal{G}, t \in \mathcal{T}, \tag{22c}$$

$$x_{jgt} - x_{jg(t-1)} = u_{jgt} - v_{jgt},\ \forall j \in \mathcal{S}, g \in \mathcal{G}, t \in \mathcal{T}, \tag{22d}$$

$$x_{jgt} \geq \sum_{\tau=\max\{1,t-UT_g+1\}}^{t} u_{jg\tau},\ \forall j \in \mathcal{S}, g \in \mathcal{G}, t \in \mathcal{T}, \tag{22e}$$

$$1 - x_{jgt} \geq \sum_{\tau=\max\{1,t-DT_g+1\}}^{t} u_{jg\tau},$$
$$\forall j \in \mathcal{S}, g \in \mathcal{G}, t \in \mathcal{T}, \tag{22f}$$

$$-RD_g \leq p_{jgt} - p_{jg(t-1)} \leq RU_g - s_{jgt},$$
$$\forall j \in \mathcal{S}, g \in \mathcal{G}, t \in \mathcal{T}, \tag{22g}$$

$$s_{jgt} \leq RC_g x_{jgt},\ \forall j \in \mathcal{S}, g \in \mathcal{G}, t \in \mathcal{T}, \tag{22h}$$

$$\sum_{g\in\mathcal{G}} s_{jgt} \geq SR_t,\ \forall j \in \mathcal{S}, t \in \mathcal{T}, \tag{22i}$$

$$p_{jgt} = P_g^{\min} x_{jgt} + \sum_{k\in\mathcal{K}} q_{jgkt},\ \forall j \in \mathcal{S}, g \in \mathcal{G}, t \in \mathcal{T}, \tag{22j}$$

$$p_{jgt} + s_{jgt} \leq P_g^{\max} x_{jgt},\ \forall j \in \mathcal{S}, g \in \mathcal{G}, t \in \mathcal{T}, \tag{22k}$$

$$q_{jgkt} \leq Q_{gk}^{\max} x_{jgt},\ \forall j \in \mathcal{S}, g \in \mathcal{G}, k \in \mathcal{K}, t \in \mathcal{T}, \tag{22l}$$

$$\sum_{g\in\mathcal{G}} p_{jgt} = \sum_{n\in\mathcal{N}} D_{jnt} - \sum_{w\in\mathcal{W}} W_{jwt},\ \forall j \in \mathcal{S}, t \in \mathcal{T}, \tag{22m}$$

$$-F_l^{\max} \leq \sum_{g\in\mathcal{G}} LSF_{lg} p_{jgt} - \sum_{n\in\mathcal{N}} LSF_{ln} D_{jnt}$$
$$+ \sum_{w\in\mathcal{W}} LSF_{lw} W_{jwt} \leq F_l^{\max},\ \forall j \in \mathcal{S}, l \in \mathcal{L}, t \in \mathcal{T}, \tag{22n}$$

$$x_{igt} = x_{jgt},\ u_{igt} = u_{jgt},\ v_{igt} = v_{jgt},$$
$$\forall i, j \in \mathcal{S}, g \in \mathcal{G}_s, t \in \mathcal{T} \tag{22o}$$

$$x_{jg0} = X_g^{\mathrm{init}},\ \forall j \in \mathcal{S}, g \in \mathcal{G}, \tag{22p}$$

$$x_{jgt} = 1,\ \forall j \in \mathcal{S}, g \in \mathcal{G}, t \in \{1, \ldots, UT_g^{\mathrm{init}}\}, \tag{22q}$$

$$x_{jgt} = 0,\ \forall j \in \mathcal{S}, g \in \mathcal{G}, t \in \{1, \ldots, DT_g^{\mathrm{init}}\}, \tag{22r}$$

$$p_{jg0} = P_g^{\mathrm{init}},\ \forall j \in \mathcal{S}, g \in \mathcal{G}, \tag{22s}$$

$$x_{jgt} \in \{0,1\},\ 0 \leq u_{jgt}, v_{jgt} \leq 1,\ \forall j \in \mathcal{S}, g \in \mathcal{G}, t \in \mathcal{T} \tag{22t}$$

$$p_{jgt}, q_{jgkt}, s_{jgt} \geq 0,\ \forall j \in \mathcal{S}, g \in \mathcal{G}, t \in \mathcal{T} \tag{22u}$$

The objective (22a) is to minimize the expected value of the sum of operating, start-up, shut-down, and production cost. Equations (22b)-(22d) ensure the logical relation of the commitment, start-up and shut-down decisions. Equations (22e) and (22f) respectively represent the minimum downtime and uptime of generators in each time period. Equations (22g) and (22h) are ramping constraints and equation (22i) is a spinning reserve constraint. Equations (22j) and (22k) are the constraints for minimum power generation and maximum

power generation, respectively. Equation (22l) represents the piecewise linearized power generation cost. Equations (22m) and (22n) are the flow balance constraint and the transmission line flow constraint, respectively. Equation (22o) ensures that the decisions for slow generators does not change for scenarios. This is also called *nonanticipativity* constraint. Equations (22p)-(22s) represent the initial conditions of generators and production level.

C StochJuMP Model Script: Two-Stage Stochastic Unit Commitment

We provide the StochJuMP model script for the stochastic unit commitment problem.

```
using StochJuMP, MPI, DSPsolver

# Initialize MPI
MPI.Init()

# Data is processed in a separate file.
include("suc_data.jl");

# StochJuMP object
m = StochasticModel(nScenarios);

# First-stage Variables
@defVar(m, Use[i=SLOWGENS, t=PERIODS], Bin) # Generator on/off indicator
@defVar(m, 0 <= Up[i=SLOWGENS, t=PERIODS] <= 1) # Start up indicator
@defVar(m, 0 <= Down[i=SLOWGENS, t=PERIODS] <= 1) # Shut down indicator

# First-stage Objective function
@setObjective(m, Min,
       sum{cost_start[i] * Up[i,t], i=SLOWGENS, t=PERIODS}
       + sum{fixed_cost_gen[i] * Use[i,t], i=SLOWGENS, t=PERIODS})

# Linking Use / Up / Down variables
@addConstraint(m, LINKING_SHUT_DOWN0[i=SLOWGENS],
       Down[i,1] <= use_0[i])
@addConstraint(m, LINKING_SHUT_DOWN[i=SLOWGENS, t=2:nPeriods],
       Use[i,t-1] >= Down[i,t])
@addConstraint(m, LINKING_START_UP0[i=SLOWGENS],
       Up[i,1] <= 1 - use_0[i])
@addConstraint(m, LINKING_START_UP[i=SLOWGENS, t=2:nPeriods],
       1 - Use[i,t-1] >= Up[i,t])
@addConstraint(m, LINKING_BOTH0[i=SLOWGENS],
       Use[i,1] - use_0[i] == Up[i,1] - Down[i,1])
@addConstraint(m, LINKING_BOTH[i=SLOWGENS, t=2:nPeriods],
       Use[i,t] - Use[i,t-1] == Up[i,t] - Down[i,t])

# Min down time
@addConstraint(m, MIN_DOWN_INIT[i=SLOWGENS, t=1:min(downtime_init[i],nPeriods)],
       Use[i,t] == 0)
@addConstraint(m, MIN_DOWN_S1[i=SLOWGENS, t=PERIODS, s=max(1,t-downtime[i]+1):t],
       1 - Use[i,t] >= Down[i,s])
@addConstraint(m, MIN_DOWN_S2[i=SLOWGENS, t=PERIODS],
       1 - Use[i,t] >= sum{Down[i,s], s=max(1,t-downtime[i]+1):t})

# Min up time
@addConstraint(m, MIN_UP_INIT[i=SLOWGENS, t=1:min(uptime_init[i],nPeriods)],
       Use[i,t] == 1)
@addConstraint(m, MIN_UP_S1[i=SLOWGENS, t=PERIODS, s=max(1,t-uptime[i]+1):t],
       Use[i,t] >= Up[i,s])
@addConstraint(m, MIN_UP_S2[i=SLOWGENS, t=PERIODS],
       Use[i,t] >= sum{Up[i,s], s=max(1,t-uptime[i]+1):t})

# For each scenario s
@second_stage m s begin

       # Second-stage momdel
```

```
56          sb = StochasticBlock(m, prob[s])
57
58          @defVar(sb, UseF[i=FASTGENS, t=PERIODS], Bin) # Generator on/off indicator
59          @defVar(sb, 0 <= UpF[i=FASTGENS, t=PERIODS] <= 1) # Start up indicator
60          @defVar(sb, 0 <= DownF[i=FASTGENS, t=PERIODS] <= 1) # Shut down indicator
61          @defVar(sb, 0 <= Gen[i=GENERATORS, t=PERIODS] <= max_gen[i]) # Power
                generation
62          @defVar(sb, 0 <= Gen_Sgmt[i=GENERATORS, k=SEGMENTS, t=PERIODS] <=
                max_gen_sgmt[i,k])
63          @defVar(sb, 0 <= Spin_Resv[i=GENERATORS, t=PERIODS] <= spin_notice / 60. *
                ramp_rate[i]) # Spinning
                reserve
64
65          # Second-stage Objective function
66          @setObjective(sb, Min,
67                 sum{cost_start[i] * UpF[i,t], i=FASTGENS, t=PERIODS}
68                 + sum{fixed_cost_gen[i] * UseF[i,t], i=FASTGENS, t=PERIODS}
69                 + sum{cost_gen[i,k] * Gen_Sgmt[i,k,t], i=GENERATORS, k=SEGMENTS, t=
                       PERIODS})
70
71          # Linking Use / Up / Down variables
72          @addConstraint(sb, FAST_LINKING_SHUT_DOWN0[i=FASTGENS],
73                 DownF[i,1] <= use_0[i])
74          @addConstraint(sb, FAST_LINKING_SHUT_DOWN[i=FASTGENS, t=2:nPeriods],
75                 UseF[i,t-1] >= DownF[i,t])
76          @addConstraint(sb, FAST_LINKING_START_UP0[i=FASTGENS],
77                 UpF[i,1] <= 1 - use_0[i])
78          @addConstraint(sb, FAST_LINKING_START_UP[i=FASTGENS, t=2:nPeriods],
79                 1 - UseF[i,t-1] >= UpF[i,t])
80          @addConstraint(sb, FAST_LINKING_BOTH0[i=FASTGENS],
81                 UseF[i,1] - use_0[i] == UpF[i,1] - DownF[i,1])
82          @addConstraint(sb, FAST_LINKING_BOTH[i=FASTGENS, t=2:nPeriods],
83                 UseF[i,t] - UseF[i,t-1] == UpF[i,t] - DownF[i,t])
84
85          # Min down time
86          @addConstraint(sb, FAST_MIN_DOWN_INIT[i=FASTGENS, t=1:min(downtime_init[i],
                nPeriods)],
87                 UseF[i,t] == 0)
88          @addConstraint(sb, FAST_MIN_DOWN_S1[i=FASTGENS, t=PERIODS, tt=max(1,t-
                downtime[i]+1):t],
89                 1 - UseF[i,t] >= DownF[i,tt])
90          @addConstraint(sb, FAST_MIN_DOWN_S2[i=FASTGENS, t=PERIODS],
91                 1 - UseF[i,t] >= sum{DownF[i,tt], tt=max(1,t-downtime[i]+1):t})
92
93          # Min up time
94          @addConstraint(sb, FAST_MIN_UP_INIT[i=FASTGENS, t=1:min(uptime_init[i],
                nPeriods)],
95                 UseF[i,t] == 1)
96          @addConstraint(sb, FAST_MIN_UP_S1[i=FASTGENS, t=PERIODS, tt=max(1,t-uptime[i
                ]+1):t],
97                 UseF[i,t] >= UpF[i,tt])
98          @addConstraint(sb, FAST_MIN_UP_S2[i=FASTGENS, t=PERIODS],
99                 UseF[i,t] >= sum{UpF[i,tt], tt=max(1,t-uptime[i]+1):t})
100
101         # Ramping rate in normal operating status
102         @addConstraint(sb, RAMP_DOWN0[i=GENERATORS],
103                gen_0[i] - Gen[i,1] <= ramp_rate[i])
104         @addConstraint(sb, RAMP_DOWN[i=GENERATORS, t=2:nPeriods],
105                Gen[i,t-1] - Gen[i,t] <= ramp_rate[i])
106         @addConstraint(sb, RAMP_UP0[i=GENERATORS],
107                Gen[i,1] - gen_0[i] + Spin_Resv[i,1] <= ramp_rate[i])
108         @addConstraint(sb, RAMP_UP[i=GENERATORS, t=2:nPeriods],
109                Gen[i,t] - Gen[i,t-1] + Spin_Resv[i,t] <= ramp_rate[i])
110
111         # Spinning reserve requirement for system
112         @addConstraint(sb, SPIN_RESV_REQ[t=PERIODS],
113                sum{Spin_Resv[i,t], i=GENERATORS}
114                >= spin_resv_rate * (total_demand[t] - total_wind_scen[t,s]))
```

```
115
116         # Spinning reserve requirement for system
117         @addConstraint(sb, SPIN_RESV_REQ[t=PERIODS],
118                sum{Spin_Resv[i,t], i=GENERATORS}
119                >= spin_resv_rate * (total_demand[t] - total_wind_scen[t,s]))
120
121         # Spinning reserve capacity for individual unit
122         @addConstraint(sb, SPIN_RESV_MAX_SLOW[i=SLOWGENS, t=PERIODS],
123                Spin_Resv[i,t] <= spin_notice / 60. * ramp_rate[i] * Use[i,t])
124         @addConstraint(sb, SPIN_RESV_MAX_FAST[i=FASTGENS, t=PERIODS],
125                Spin_Resv[i,t] <= spin_notice / 60. * ramp_rate[i] * UseF[i,t])
126
127         # Power output capacity constraints
128         @addConstraint(sb, POWER_OUTPUT_SLOW[i=SLOWGENS, t=PERIODS],
129                Gen[i,t] == min_gen[i] * Use[i,t] + sum{Gen_Sgmt[i,k,t], k=SEGMENTS})
130         @addConstraint(sb, POWER_SEGMENT_SLOW[i=SLOWGENS, k=SEGMENTS, t=PERIODS],
131                Gen_Sgmt[i,k,t] <= max_gen_sgmt[i,k] * Use[i,t])
132         @addConstraint(sb, POWER_MAX_SLOW[i=SLOWGENS, t=PERIODS],
133                Gen[i,t] + Spin_Resv[i,t] <= max_gen[i] * Use[i,t])
134         @addConstraint(sb, POWER_OUTPUT_FAST[i=FASTGENS, t=PERIODS],
135                Gen[i,t] == min_gen[i] * UseF[i,t] + sum{Gen_Sgmt[i,k,t], k=SEGMENTS})
136         @addConstraint(sb, POWER_SEGMENT_FAST[i=FASTGENS, k=SEGMENTS, t=PERIODS],
137                Gen_Sgmt[i,k,t] <= max_gen_sgmt[i,k] * UseF[i,t])
138         @addConstraint(sb, POWER_MAX_FAST[i=FASTGENS, t=PERIODS],
139                Gen[i,t] + Spin_Resv[i,t] <= max_gen[i] * UseF[i,t])
140
141         # Power balance constraints for system
142         @addConstraint(sb, POWER_BALANCE[t=PERIODS],
143                sum{Gen[i,t], i=GENERATORS} == total_demand[t] - total_wind_scen[t,s])
144
145         # Transmission constraints with load shift factor (These can be lazy constraints.)
146         @addConstraint(sb, FLOW_BRANCH_LSF_LB[l=BRANCHES, t=PERIODS],
147                sum{load_shift_factor[n,l] * Gen[i,t], n=BUSES, i=GENERATORS;
                       gen_bus_id[i] == n}
148                >= sum{load_shift_factor[n,l] * demand[n,t], n=BUSES}
149                - sum{load_shift_factor[n,l] * wind_scen[wn,t,s], n=BUSES, wn=WINDS;
                       wind_bus_id[wn] == n}
150                - flow_max[l])
151         @addConstraint(sb, FLOW_BRANCH_LSF_UB[l=BRANCHES, t=PERIODS],
152                sum{load_shift_factor[n,l] * Gen[i,t], n=BUSES, i=GENERATORS;
                       gen_bus_id[i] == n}
153                <= sum{load_shift_factor[n,l] * demand[n,t], n=BUSES}
154                - sum{load_shift_factor[n,l] * wind_scen[wn,t,s], n=BUSES, wn=WINDS;
                       wind_bus_id[wn] == n}
155                + flow_max[l])
156 end
157
158 DSPsolver.loadProblem(m); # load model
159 DSPsolver.solve(); # solve model
160
161 # print out some results
162 println("Solution status: ", DSPsolver.getSolutionStatus());
163 println("Primal Bound : ", DSPsolver.getPrimalBound());
164 println("Dual Bound : ", DSPsolver.getDualBound());
165
166 MPI.Finalize();
```

References

1. Achterberg, T.: Scip: solving constraint integer programs. Mathematical Programming Computation **1**(1), 1–41 (2009)
2. Ahmed, S.: A scenario decomposition algorithm for 0–1 stochastic programs. Operations Research Letters **41**(6), 565–569 (2013)

3. Ahmed, S., Tawarmalani, M., Sahinidis, N.V.: A finite branch-and-bound algorithm for two-stage stochastic integer programs. Mathematical Programming **100**(2), 355–377 (2004)
4. Bezanson, J., Karpinski, S., Shah, V.B., Edelman, A.: Julia: A fast dynamic language for technical computing. arXiv preprint arXiv:1209.5145 (2012)
5. Birge, J.R., Dempster, M.A., Gassmann, H.I., Gunn, E.A., King, A.J., Wallace, S.W.: A standard input format for multiperiod stochastic linear programs. IIASA Laxenburg Austria (1987)
6. Birge, J.R., Louveaux, F.: Introduction to stochastic programming. Springer (2011)
7. Bixby, R.E.: Solving real-world linear programs: A decade and more of progress. Operations research **50**(1), 3–15 (2002)
8. Carøe, C.C., Schultz, R.: Dual decomposition in stochastic integer programming. Operations Research Letters **24**(1-2), 37–45 (1999)
9. Crainic, T.G., Fu, X., Gendreau, M., Rei, W., Wallace, S.W.: Progressive hedging-based metaheuristics for stochastic network design. Networks **58**(2), 114–124 (2011)
10. Dawande, M., Hooker, J.N.: Inference-based sensitivity analysis for mixed integer/linear programming. Operations Research **48**(4), 623–634 (2000)
11. Fisher, M.L.: An applications oriented guide to lagrangian relaxation. Interfaces **15**(2), 10–21 (1985)
12. Fisher, M.L.: The lagrangian relaxation method for solving integer programming problems. Management science **50**(12_supplement), 1861–1871 (2004)
13. Forrest, J.: Cbc. URL https://projects.coin-or.org/Cbc
14. Forrest, J.: Clp. URL https://projects.coin-or.org/Clp
15. Gade, D., Küçükyavuz, S., Sen, S.: Decomposition algorithms with parametric gomory cuts for two-stage stochastic integer programs. Mathematical Programming **144**(1-2), 39–64 (2014)
16. Gassmann, H.I., Schweitzer, E.: A comprehensive input format for stochastic linear programs. Annals of Operations Research **104**(1-4), 89–125 (2001)
17. Geoffrion, A.M.: Lagrangean relaxation for integer programming. Springer (1974)
18. Gertz, E.M., Wright, S.J.: Object-oriented software for quadratic programming. ACM Transactions on Mathematical Software (TOMS) **29**(1), 58–81 (2003)
19. Goffin, J.L., Vial, J.P.: Cutting planes and column generation techniques with the projective algorithm. Journal of Optimization Theory and Applications **65**(3), 409–429 (1990)
20. Gondzio, J.: Warm start of the primal-dual method applied in the cutting-plane scheme. Mathematical Programming **83**(1-3), 125–143 (1998)
21. Gondzio, J., Gonzalez-Brevis, P., Munari, P.: New developments in the primal–dual column generation technique. European Journal of Operational Research **224**(1), 41–51 (2013)
22. Gondzio, J., Sarkissian, R.: Column generation with a primal-dual method. Relatorio tecnico, University of Geneva **102** (1996)
23. Guo, G., Hackebeil, G., Ryan, S.M., Watson, J.P., Woodruff, D.L.: Integration of progressive hedging and dual decomposition in stochastic integer programs. Operations Research Letters **43**(3), 311–316 (2015)
24. Gurobi Optimization, Inc.: Gurobi optimizer reference manual (2015). URL http://www.gurobi.com
25. Helmberg, C.: ConicBundle. https://www-user.tu-chemnitz.de/~helmberg/ (2004)
26. Huchette, J., Lubin, M., Petra, C.: Parallel algebraic modeling for stochastic optimization. In: Proceedings of the 1st First Workshop for High Performance Technical Computing in Dynamic Languages, pp. 29–35. IEEE Press (2014)
27. IBM Corp.: IBM ILOG CPLEX Optimization Studio 12.6.1 (2014). URL http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html
28. Kim, K., Mehrotra, S.: A two-stage stochastic integer programming approach to integrated staffing and scheduling with application to nurse management. Optimization Online (2014)
29. King, A.: Stochastic Modeling Interface (2007). URL https://projects.coin-or.org/Smi
30. Laporte, G., Louveaux, F.V.: The integer L-shaped method for stochastic integer programs with complete recourse. Operations research letters **13**(3), 133–142 (1993)

31. Lee, C., Liu, C., Mehrotra, S., Shahidehpour, M.: Modeling transmission line constraints in two-stage robust unit commitment problem. Power Systems, IEEE Transactions on **29**(3), 1221–1231 (2014)
32. Linderoth, J., Wright, S.: Decomposition algorithms for stochastic programming on a computational grid. Computational Optimization and Applications **24**(2-3), 207–250 (2003)
33. Løkketangen, A., Woodruff, D.L.: Progressive hedging and tabu search applied to mixed integer (0, 1) multistage stochastic programming. Journal of Heuristics **2**(2), 111–128 (1996)
34. Lubin, M., Dunning, I.: Computing in operations research using Julia. arXiv preprint arXiv:1312.1431 (2013)
35. Lubin, M., Martin, K., Petra, C.G., Sandıkçı, B.: On parallelizing dual decomposition in stochastic integer programming. Operations Research Letters **41**(3), 252–258 (2013)
36. Lubin, M., Petra, C.G., Anitescu, M., Zavala, V.: Scalable stochastic optimization of complex energy systems. In: High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for, pp. 1–10. IEEE (2011)
37. Lulli, G., Sen, S.: A branch-and-price algorithm for multistage stochastic integer programming with application to stochastic batch-sizing problems. Management Science **50**(6), 786–796 (2004)
38. Märkert, A., Gollmer, R.: Users Guide to ddsip–A C package for the dual decomposition of two-stage stochastic programs with mixed-integer recourse (2014)
39. Mehrotra, S.: On the implementation of a primal-dual interior point method. SIAM Journal on Optimization **2**(4), 575–601 (1992)
40. Mitchell, J.E.: Computational experience with an interior point cutting plane algorithm. SIAM Journal on Optimization **10**(4), 1212–1227 (2000)
41. OptiRisk Systems: FortSP: A stochastic programming solver, version 1.2 (2014). URL http://www.optirisk-systems.com/manuals/FortspManual.pdf
42. Papavasiliou, A., Oren, S.S.: Multiarea stochastic unit commitment for high wind penetration in a transmission constrained network. Operations Research **61**(3), 578–592 (2013)
43. Papavasiliou, A., Oren, S.S., O'Neill, R.P.: Reserve requirements for wind power integration: A scenario-based stochastic programming framework. Power Systems, IEEE Transactions on **26**(4), 2197–2206 (2011)
44. Saltzman, M., Ladányi, L., Ralphs, T.: The COIN-OR open solver interface: Technology overview. In: CORS/INFORMS Conference. Banff (2004)
45. Sen, S., Higle, J.L.: The C^3 theorem and a D^2 algorithm for large scale stochastic mixed-integer programming: set convexification. Mathematical Programming **104**(1), 1–20 (2005)
46. Sherali, H.D., Fraticelli, B.M.: A modification of Benders' decomposition algorithm for discrete subproblems: An approach for stochastic programs with integer recourse. Journal of Global Optimization **22**(1-4), 319–342 (2002)
47. Tarhan, B., Grossmann, I.E.: Improving dual bound for stochastic MILP models using sensitivity analysis. Working paper (2015)
48. Watson, J.P., Woodruff, D.L.: Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems. Computational Management Science **8**(4), 355–370 (2011)
49. Watson, J.P., Woodruff, D.L., Hart, W.E.: PySP: modeling and solving stochastic programs in Python. Mathematical Programming Computation **4**(2), 109–149 (2012)
50. Wunderling, R.: Paralleler und objektorientierter Simplex-Algorithmus. Ph.D. thesis, Technische Universität Berlin (1996). http://www.zib.de/Publications/abstracts/TR-96-09/
51. Zhang, M., Kucukyavuz, S.: Finitely convergent decomposition algorithms for two-stage stochastic pure integer programs. SIAM Journal on Optimization **24**(4), 1933–1951 (2014)
52. Zverovich, V., Fábián, C.I., Ellison, E.F., Mitra, G.: A computational study of a solver system for processing two-stage stochastic LPs with enhanced Benders' decomposition. Mathematical Programming Computation **4**(3), 211–238 (2012)