Math. Prog. Comp. (2018) 10:225–266
https://doi.org/10.1007/s12532-017-0128-z

FULL LENGTH PAPER

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

Kibaek Kim[1] · Victor M. Zavala[2]

Received: 23 June 2015 / Accepted: 20 September 2017 / Published online: 3 November 2017

Abstract We present algorithmic innovations for the dual decomposition method to address two-stage stochastic programs with mixed-integer recourse and provide an open-source software implementation that we call DSP. Our innovations include the incorporation of Benders-like cuts in a dual decomposition framework to tighten Lagrangian subproblems and aid the exclusion of infeasible first-stage solutions for problems without (relative) complete recourse. We also use an interior-point cutting-plane method with new termination criteria for solving the Lagrangian master problem. We prove that the algorithm converges to an optimal solution of the Lagrangian dual problem in a finite number of iterations, and we also prove that convergence can be achieved even if the master problem is solved suboptimally. DSP can solve instances specified in C code, SMPS files, and Julia script. DSP also implements a standard

This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357. V. M. Zavala acknowledges funding from the early career program of the U.S. Department of Energy under grant DE-SC0014114. 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 also thank Julie Bessac for providing wind speed prediction data, Sven Leyffer for providing feedback on an earlier version of the manuscript, the SCIP team at the Zuse Institute Berlin for valuable comments and support, and the anonymous reviewers for constructive suggestions that considerably improved the original version of this article.

✉ Kibaek Kim
kimk@anl.gov

Victor M. Zavala
victor.zavala@wisc.edu

[1] Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Lemont, IL 60439, USA

[2] Department of Chemical and Biological Engineering, University of Wisconsin-Madison, 1415 Engineering Drive, Madison, WI 53706, USA

Benders decomposition method and a dual decomposition method based on subgradient dual updates that we use to perform benchmarks. We present extensive numerical results using SIPLIB instances and a large unit commitment problem to demonstrate that the proposed innovations provide significant improvements in the number of iterations and solution times. The software reviewed as part of this submission has been given the Digital Object Identifier (DOI) https://doi.org/10.5281/zenodo.998971.

Keywords Stochastic mixed-integer programming · Decomposition · Parallel · Large-scale · Open-source software

Mathematics Subject Classification 90C15 · 90C11 · 49M27 · 65Y05

1 Problem statement

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

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

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

$$\phi(t) := \min_y \{ q(\xi)^T y : \; W(\xi)y \geq t, \; y \in Y \}, \; t \in \mathbb{R}^{m_2}. \tag{2}$$

We assume that the random parameter ξ follows a discrete distribution with finite support $\{\xi_1, \ldots, \xi_r\}$ and corresponding probabilities $p_1, \ldots, p_r$ (continuous distributions can be handled by using a sample-average approximation [32]). The sets $X \subseteq \mathbb{R}^{n_1}$ and $Y \subseteq \mathbb{R}^{n_2}$ represent integer or binary restrictions on a subset of the decision variables x and y, respectively. The first-stage problem data comprise $A \in \mathbb{R}^{m_1 \times n_1}$, $b \in \mathbb{R}^{m_1}$, and $c \in \mathbb{R}^{n_1}$. The second-stage data are given by $T(\xi_j) \in \mathbb{R}^{m_2 \times n_1}$, $W(\xi_j) \in \mathbb{R}^{m_2 \times n_2}$, $h(\xi_j) \in \mathbb{R}^{m_2}$, and $q(\xi_j) \in \mathbb{R}^{n_2}$ for $j = 1, \ldots, r$. For simplicity, we use the notation (T_j, W_j, h_j, q_j) for $j = 1, \ldots, r$ to represent the scenario data.

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

$$z = \min_{x_j, y_j} \quad \sum_{j=1}^{r} p_j \left(c^T x_j + q_j^T y_j \right) \tag{3a}$$

$$\text{s.t.} \quad \sum_{j=1}^{r} H_j x_j = 0 \tag{3b}$$

$$(x_j, y_j) \in G_j, \quad \forall j = 1, \ldots, r, \tag{3c}$$

where the scenario feasibility set is defined as

$$G_j := \{(x_j, y_j) \; : \; Ax_j \geq b, \; T_j x_j + W_j y_j \geq h_j, \; x_j \in X, \; y_j \in Y\}. \tag{4}$$

The nonanticipativity constraints in (3b) represent the equations $x_1 = x_r$ and $x_j = x_{j-1}$ for $j = 2, \ldots, r$, and H_j is a suitable $r \cdot n_1 \times n_1$ matrix. We assume that SMIP does not necessarily have relatively complete recourse. We recall that, without this property, there can exist an $\hat{x} \in X$ satisfying $A\hat{x} \geq b$ for which there does not exist a recourse $y \in \mathbb{R}^{m_2}$ satisfying $(\hat{x}, y) \in G_j$ for some j. 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 [6,33,50]. When integer variables appear in the second stage, the recourse value function is nonconvex and discontinuous in the first-stage variables; consequently, more sophisticated approaches are needed. These include convexification of the recourse function (e.g., [16,30,51,52,59]) or specialized branch-and-bound schemes (e.g., [3,48]). These approaches are not able to tackle the general SMIP (1), except [48] that may not be computationally practical. For instance, convexification techniques are based on either finite cutting-plane methods (i.e., lift-and-project cuts [52], Gomory cuts [16,59], and no-good cuts [2]) or disjunctive programming [51]. Consequently, these approaches are limited to certain problem classes (i.e., mixed-binary second-stage variables [52], pure-integer variables [2,16,59], integral technology matrix [52], and special second-stage matrix structures [30]). For example, the method proposed in [2] cannot be applied for some of the standard benchmark problems (the `dcap` problem instances) used in this paper.

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 [55] 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 often appear 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.

Dual variables can be updated in a dual decomposition method by using subgradient methods [2,8,45,46,55], cutting-plane methods [38], or column-generation methods [38,40]. 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, and subgradient schemes require ad-hoc step-length selection criteria and can exhibit slow convergence [12,43].

Progressive hedging is a popular and flexible method for solving SMIPs and can handle problems with mixed-integer recourse [9,26,36,56]. Connections between progressive hedging and dual decomposition have also been established recently to create hybrid strategies [26]. Progressive hedging has no convergence guarantees. Consequently, it is often used as a heuristic to find approximate solutions. Moreover, this method is not guaranteed to find a feasible solution for problems without relatively complete recourse.

Few software packages are available for solving large-scale stochastic programs. SMI [31] 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 [44,60]. The C package ddsip [41] implements dual decomposition for SMIPs and uses the ConicBundle [28] package for updating dual variables. This package was unable to solve many small-size SIPLIB instances [38]. 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 [38,39]. 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 [57].

Open source software packages are also available to decompose general MIPs. GCG is a generic branch-and-cut solver based on Dantzig–Wolfe decomposition [17]. DIP implements Dantzig–Wolfe decomposition and Lagrangian relaxation [47]. Both packages can automatically detect decomposition structure of MIP problem and find an optimal solution using branch-and-cut and branch-and-price methods. Hence, these solvers can potentially be used for solving SMIP problems. However, none of these packages provide parallel implementations.

In this work, we present algorithmic innovations for the dual decomposition method. We develop a procedure to generate Benders-like valid inequalities (i.e., Benders feasibility and optimality cuts) that tighten Lagrangian subproblems and that aid the exclusion of infeasible first-stage solutions. The Benders-like cuts are derived from Lagrangian subproblem solutions and 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 use an interior-point cutting-plane method with new termination criteria for solving the Lagrangian master problem. We prove that the dual decomposition algorithm terminates finitely even without solving the master problem to optimality using the interior point method. We demonstrate that this approach reduces oscillations of the dual variables and ultimately aids convergence. We also introduce DSP, an open-source, object-oriented, 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 [5,18]. DSP can also read models expressed in JuMP [37], a Julia-based algebraic modeling package, which can be used to represent large-scale stochastic programs using a compact syntax. 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 computing cluster at Argonne National Laboratory. We demonstrate that our innovations yield important improvements in robustness and solution time.

The contributions of this work can be summarized as follows.

– We develop a procedure to generate the Benders-like valid inequalities in the dual decomposition framework.

- We use an interior-point cutting-plane method with new termination criteria for solving the Lagrangian master problem, which allows the finite termination of the dual decomposition even without solving the master problem to optimality.
- We develop an open-source software package `DSP` that implements several decomposition methods capable of running on high-performance computing systems by using the MPI library.

The paper is structured as follows. In Sect. 2 we present the standard dual decomposition method and discuss different approaches for updating dual variables. In Sect. 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 (Sect. 3.1). We then present an interior-point cutting-plane method and termination criteria for the master problem (Sect. 3.2). In Sect. 4 we describe the design of `DSP` and the modeling interfaces. In Sect. 5 we present our benchmark results. In Sect. 6 we summarize our innovations and provide directions for future work.

2 Dual decomposition

We describe a standard dual decomposition method for two-stage SMIPs of the form (3). We apply a Lagrangian relaxation of these constraints to obtain the Lagrangian dual function of (3):

$$D(\lambda) := \min_{x_j, y_j} \left\{ \sum_{j=1}^{r} L_j(x_j, y_j, \lambda) : (x_j, y_j) \in G_j, \ \forall j = 1, \dots, r \right\}, \tag{5}$$

where

$$L_j(x_j, y_j, \lambda) := p_j \left(c^T x_j + q_j^T y_j \right) + \lambda^T (H_j x_j). \tag{6}$$

Here, $\lambda \in \mathbb{R}^{r \cdot n_1}$ are the dual variables of the nonanticipativity constraints (3b). For fixed λ, the Lagrangian dual function can be decomposed as

$$D(\lambda) = \sum_{j=1}^{r} D_j(\lambda), \tag{7}$$

where

$$D_j(\lambda) := \min_{x_j, y_j} \left\{ L_j(x_j, y_j, \lambda) : (x_j, y_j) \in G_j \right\}. \tag{8}$$

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

$$z_{\text{LD}} := \max_{\lambda} \sum_{j=1}^{r} D_j(\lambda). \tag{9}$$

Proposition 1 is a well known result of Lagrangian relaxation that shows the tightness of the lower bound z_{LD} [19].

Proposition 1 *The optimal value z_{LD} of the Lagrangian dual problem* (9) *equals the optimal value of the linear program (LP):*

$$\min_{x_j, y_j} \left\{ \sum_{j=1}^{r} p_j \left(c^T x_j + q_j^T y_j \right) : (x_j, y_j) \in conv(G_j), \right.$$
$$\left. \sum_{j=1}^{r} H_j x_j = 0, \; \forall j = 1, \ldots, r \right\}, \tag{10}$$

where $conv(G_j)$ denotes the convex hull of G_j. Moreover, $z_{LD} \geq z_{LP}$ holds, where z_{LP} is the optimal value of the LP 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 computationally prohibitive.

The dual decomposition method can be seen as a primal multistart heuristic procedure [15,38]. With different objective functions $L_j(x_j, y_j, \lambda)$, the dual decomposition finds multiple primal solutions $(x_j, y_j) \in conv(G_j)$ at each iteration. Particularly for stochastic programs with pure integer first-stage variables, the method can effectively find an optimal solution (e.g., [2]). Note, however, that the original problem (3) may be infeasible for all candidates x_j if the problem does not have relatively complete recourse. We provide a remedy for this in Sect. 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_j^k be an optimal solution of (8) for given λ^k. The dual variable is updated as

$$\lambda^{k+1} = \lambda^k - \alpha_k \sum_{j=1}^{r} H_j x_j^k, \tag{11}$$

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 $\sum_{j=1}^{r} H_j x_j^k$. The step size α_k is given by

$$\alpha_k := \beta_k \frac{z_{\mathrm{UB}} - D(\lambda^k)}{\left\| \sum_{j=1}^{r} H_j x_j^k \right\|_2^2}, \tag{12}$$

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 $k \leftarrow 0$, $z_{\mathrm{LB}} \leftarrow -\infty$, $z_{\mathrm{UB}} \leftarrow \infty$ and $\gamma \leftarrow 0$.
2: repeat
3:    SOLVE (8) to obtain $D_j(\lambda^k)$ and $(x_j^k, y_j^k)$ for given $\lambda^k$ and for all $j = 1, \ldots, r$
4:    if $D(\lambda^k) > z_{\mathrm{LB}}$ then
5:       $z_{\mathrm{LB}} \leftarrow D(\lambda^k)$
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 $z_{\mathrm{UB}}$ by solving (3) for given $x_j^k$
13:   $k \leftarrow k + 1$
14: until $(z_{\mathrm{UB}} - z_{\mathrm{LB}})/|10^{-10} + z_{\mathrm{UB}}| < 10^{-5}$, $\beta_k < 10^{-6}$, or time limit is reached.
```

Algorithm 1 is initialized with user-defined parameters λ^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_j^k (line 12). An important limitation of subgradient methods is that enforcing convergence using line-search schemes can be computationally impractical, since each trial step requires the solution of the Lagrangian subproblems [11]. Therefore, heuristic step-size rules are typically used.

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:

$$m_k := \max_{\theta_j, \lambda} \sum_{j=1}^{r} \theta_j \tag{13a}$$

$$\text{s.t.} \quad \theta_j \le D_j(\lambda^l) + \left(H_j x_j^l\right)^T (\lambda - \lambda^l), \; j = 1, \ldots, r, \; l = 0, 1, \ldots, k. \tag{13b}$$

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, $\theta := (\theta_1, ..., \theta_r)$ and $\pi := (\pi_1^0, ..., \pi_1^k, ..., \pi_r^0, ..., \pi_r^k)$, where π_j^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: loop
3:     SOLVE (8) to obtain $D_j(\lambda^k)$ and $(x_j^k, y_j^k)$ for given $\lambda^k$ and for all $j = 1, \ldots, r$.
4:     SET $z_{LB} \leftarrow \max\{z_{LB}, D(\lambda^k)\}$.
5:     ADD (13b) for given $D(\lambda^k)$ and $x_j^k$
6:     SOLVE (13) to obtain $m_k$ and $(\theta^{k+1}, \lambda^{k+1})$
7:     STOP if $m_k \leq D(\lambda^k)$
8:     SET $k \leftarrow k + 1$
9: end loop
```

The function $D_j(\lambda)$ 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-plane method a natural termination criterion (i.e., $m_{k-1} \leq D(\lambda^k)$). 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

$$\sum_{j=1}^{r} \theta_j \leq \sum_{j=1}^{r} D_j(\lambda^l) + \left(\sum_{j=1}^{r} H_j x_j^l \right)^T (\lambda - \lambda^l) \tag{14}$$

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

Remark 2 The column generation method [38,40] is a variant of the cutting-plane method that solves the dual of the Lagrangian master problem (13). Lubin et al. [38] demonstrated that the dual of the Lagrangian master has a dual angular structure that can be exploited by using decomposition methods.

3 Algorithmic innovations for dual decomposition

We now develop innovations for the dual decomposition method based on cutting planes. In Sect. 3.1 we present a procedure to construct Benders-like 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 Sect. 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 $(\hat{x}, \hat{y}) \in G_j$ for some j, and assume that for some scenario j' there does not exist $y \in \mathbb{R}^{n_2}$ such $(\hat{x}, y) \in G_{j'}$ for fixed $\hat{x}$. Let $\mu_{j'}$ be an optimal solution of the LP*

$$\max_{\mu} \left\{ \mu^T \left(h_{j'} - T_{j'}\hat{x} \right) : \mu^T W_{j'} = 0,\ 0 \le \mu \le 1 \right\}. \tag{15}$$

The inequality

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

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

Proof From Farkas' lemma, there exists a $\mu_{j'} \in \mathbb{R}^{m_2}$ such that $\mu_{j'}^T W_{j'} = 0$ and $\mu_{j'}^T (h_{j'} - T_{j'}\hat{x}) > 0$, and thus the hyperplane $\mu_{j'}^T \left(h_{j'} - T_{j'}x \right) \le 0$ separates $\hat{x}$ from $\{x : (x, y) \in G_{j'}\}$. 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 for fixed $\hat{x}$, for all $j = 1, \ldots, r$ there exists $y \in \mathbb{R}^{n_2}$ such that $(\hat{x}, y) \in G_j$. Let π_j be the optimal solution of the following recourse problem for each $j = 1, \ldots, r$:*

$$\max_{\pi} \left\{ \pi^T \left(h_j - T_j\hat{x} \right) : \pi^T W_j = q_j,\ \pi \ge 0 \right\}. \tag{17}$$

The inequality

$$c^T x + \sum_{j=1}^{r} \pi_j^T \left(h_j - T_j x \right) \le z_{UB} \tag{18}$$

is valid for SMIP (3).

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

$$\begin{aligned} \min_x \quad & c^T x + \sum_{j=1}^{r} p_j q_j^T \hat{y}_j(x) \\ \text{s.t.} \quad & Ax \geq b, \ x \in X, \end{aligned}$$

where $\hat{y}_j(x) := \text{argmin}\{q_j^T y : W_j y \geq h_j - T_j x\}$. This is equivalent to

$$\begin{aligned} \min_x \quad & c^T x + \sum_{j=1}^{r} p_j \hat{\pi}_j(x)^T (h_j - T_j x) \\ \text{s.t.} \quad & Ax \geq b, \ x \in X, \end{aligned}$$

where $\hat{\pi}_j(x) := \text{argmax}\{\pi^T(h_j - T_j x) : \pi^T W_j = q_j, \ \pi \geq 0\}$. By assumption, there exists a solution π_s for (17) for each $j = 1, \ldots, r$. Because

$$\begin{aligned} c^T x + \sum_{j=1}^{r} p_j \pi_j^T \left(h_j - T_j x\right) &\leq c^T x + \sum_{j=1}^{r} p_j^T \hat{\pi}_j(x)^T (h_j - T_j x) \\ &\leq z_{\text{UB}} \end{aligned}$$

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

The inequalities (16) and (18) are feasibility cuts and optimality cuts, respectively, obtained from Benders decomposition of the scenario subproblems for each $j = 1, \ldots, r$. These Benders-like cuts are generated by using a relaxation of second-stage integrality. Consequently the cuts may be loose and are not guaranteed to exclude $\hat{x}$ or tighten the objective value. Computational experiments provided in Sect. 5, however, provide evidence that the inequalities generated by Proposition 2 are effective in practice.

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 line 3 of the standard dual decomposition method of Algorithm 2. In other words, Procedure 1 is called in each iteration in order to generate feasibility and optimality cuts (16) and (18).

We explain the proposed cutting-plane procedure as follows. The situation stated in Proposition 2 can occur when SMIP does not have relatively complete recourse. Because the 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 the inequality (16) can be added only when the LP relaxation of the subproblem (8) is infeasible. We emphasize that the inequalities (16) and (18) generated in lines 6 and 14, respectively, *are added to all the Lagrangian subproblems* (lines 8 and 15). In other words, the inequalities generated for one scenario are *shared* with all the other scenarios. This approach seeks to prevent a given subproblem from visiting 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

Procedure 1 Cutting-Plane Procedure for Lagrangian Subproblems

```
Require: $\lambda^k$
1: for all $j = 1, \dots, r$ do
2:   repeat
3:     SOLVE subproblem (8) to obtain $D_j(\lambda^k)$ and $(x_j^k, y_j^k)$ for $\lambda^k$
4:     isFeasible $\leftarrow$ true
5:     for all $j' \neq j$ do
6:       SOLVE feasibility cut generator (15) to obtain $\mu_{j'}$ for $x_j^k$
7:       if $\mu_{j'}^T(h_{j'} - T_{j'} x_j^k) > 0$ 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 $z_{\mathrm{UB}}$ by solving (3) for fixed $x_j^k$
14:  GENERATE optimality cut (18) by solving (17) for $x_j^k$ and for all $j = 1, \dots, r$
15:  ADD optimality cut (18) to all the subproblems (8)
16: end for
```

by the best-known upper bound z_{UB} and thus can be tightened 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 the 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.

One can easily prove that Procedure 1 terminates in a finite number of steps by showing that only a finite number of feasibility cuts (16) are generated. This holds because a finite number of bases exist in each one of the cut generation problems (15). Note also that (15) is an LP, 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

The simplex method is an efficient algorithm for solving the Lagrangian master problems (13). This efficiency is due to 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) [21,43]. The oscillations can in turn slow convergence (we illustrate this behavior in Fig. 6 of Sect. 5). To avoid this situation, we solve the Lagrangian master problems *suboptimally* using an interior-point method (IPM). As noted in [43], *early termination* on an IPM can enable us to find stronger cuts and to avoid degeneracy. The use of an IPM with early termination has been applied in the context of cutting-plane and column-generation methods [21,23,25,43]. Our IPM implementation follows the lines of the work of [43], which uses Mehrotra's primal-dual predictor corrector method to deliver suboptimal but strictly feasible solutions of the master problem. Our implementation specializes the relative duality gap criterion of [23] to a dual decomposition setting. In addition to satisfying this criterion, the basic requirement for any IPM used is that it returns a feasible point for the master problem. From a practical standpoint, we also require that the IPM delivers well-centered iterates (i.e., which have well-balanced complementarity products) [22]. This requirement guarantees that the computed solution is an approximate analytic center that is not too close to the boundary of the primal-dual feasible set and ensures that the oscillations of the dual solutions will be relatively small [23]. We also propose a new termination criterion specific to a dual decomposition setting that uses upper-bound information to determine whether the IPM should be terminated.

The IPM checks the termination criteria in the following order:

$$\sum_{j=1}^{r} \theta_j^k \geq z_{\text{UB}}, \tag{19a}$$

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

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

$$g_k(\theta, \pi) := \frac{\sum_{j=1}^{r} \sum_{l=1}^{k} \pi_j^l \left(D_j(\lambda^l) - (H_j x_j^l)^T \lambda^l \right) - \sum_{j=1}^{r} \theta_j}{1 + \left| \sum_{j=1}^{r} \theta_j \right|}. \tag{20}$$

This quantity is the difference between the dual and primal objectives of the master problem scaled by the primal objective. 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 $g_k(\tilde{\theta}^k, \tilde{\pi}^k) < \epsilon_{\text{IPM}}^k$ for some duality gap tolerance $\epsilon_{\text{IPM}}^k > 0$ or $\sum_{j=1}^{r} \tilde{\theta}_j^k \geq z_{\text{UB}}$ for the current upper bound $z_{\text{UB}} < \infty$.

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

$$\epsilon_{\text{IPM}}^k := \min \left\{ \epsilon_{\text{IPM}}^{max}, \frac{g_{k-1}(\tilde{\theta}^{k-1}, \tilde{\pi}^{k-1})}{\delta} + \frac{\tilde{m}_{k-1} - \sum_{j=1}^{r} D_j(\tilde{\lambda}^{k-1})}{1 + |\tilde{m}_{k-1}|} \right\}, \tag{21}$$

where $\tilde{m}_{k-1} := \sum_{j=1}^{r} \tilde{\theta}_j^{k-1}$, $\epsilon_{\text{IPM}}^{max}$ is the maximum tolerance, and parameter $\delta > 1$ is the degree of optimality (defined as in [23]). Parameter δ controls the tolerance ϵ_{IPM}^k relative to the duality gap $g_k(\theta, \pi)$. This tolerance control (21) mechanism is different from that defined in [23]. In particular, the tolerance update in [23] depends on parameter κ specifically defined for a particular problem instance, whereas (21) does not require such a parameter.

At first sight it seems possible that Algorithm 2 may not generate any cut for the Lagrangian master problem because the solution of the master is terminated early. The following propositions show, however, that Algorithm 2 does not stall and eventually generates cutting planes for the Lagrangian master problem 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* $\tilde{\theta}_j^k \leq D_j(\tilde{\lambda}^k)$ *for all* $j = 1, \ldots, r$*, then* $\epsilon_{IPM}^{k+1} \leq g_k(\tilde{\theta}^k, \tilde{\pi}^k) < \epsilon_{IPM}^k$*.*

Proof Suppose that

$$\epsilon_{\text{IPM}}^{k+1} = \frac{g_k(\tilde{\theta}^k, \tilde{\pi}^k)}{\delta} + \frac{\tilde{m}_k - \sum_{j=1}^r D_j(\tilde{\lambda}^k)}{1 + |\tilde{m}_k|} < \epsilon_{\text{IPM}}^{max}.$$

Because $\tilde{\theta}_j^k \leq D_j(\tilde{\lambda}^k)$ holds for all $j = 1, \ldots, r$, we have $\tilde{m}_j^k - \sum_{j=1}^r D_j(\tilde{\lambda}^k) = \sum_{j=1}^r \left(\tilde{\theta}_j^k - D_j(\tilde{\lambda}^k)\right) \leq 0$ and $\epsilon_{\text{IPM}}^{k+1} \leq g_k(\tilde{\theta}^k, \tilde{\pi}^k)/\delta < g_k(\tilde{\theta}^k, \tilde{\pi}^k) < \epsilon_{\text{IPM}}^k$. □

Note that $(\tilde{\theta}^k, \tilde{\lambda}^k, \tilde{\pi}^k)$ in Proposition 4 are feasible solutions of the Lagrangian master problem at any iteration k. Proposition 4 also states that the relative tolerance is decreased at each iteration. This implies that a feasible solution at iteration k will not be the same as that at iteration $k+1$ obtained 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 also highlight that the proposition requires feasibility of the subptimal solution of the master problem. This requirement is fulfilled in practice by using an IPM that delivers a feasible solution at each iterate.

We are interested only in feasible solutions satisfying $\sum_{j=1}^r \tilde{\theta}_j < z_{UB}$ for a given $z_{UB} < \infty$. Consequently, the IPM can be terminated with a feasible solution $(\tilde{\theta}, \tilde{\lambda}, \tilde{\pi})$ whenever $\sum_{j=1}^r \tilde{\theta}_j \geq z_{UB}$.

Proposition 5 *Let* $(\tilde{\theta}^k, \tilde{\lambda}^k, \tilde{\pi}^k)$ *be a feasible solution of the master (13) satisfying* $\sum_{j=1}^r \tilde{\theta}_j^k \geq z_{UB}$ *at iteration* k *for a given* $z_{UB} < \infty$*. If* $\tilde{\theta}_j^k \leq D_j(\tilde{\lambda}^k)$ *for all* $j = 1, \ldots, r$*, then* $\sum_{j=1}^r D_j(\tilde{\lambda}^k) = z_{UB}$*.*

Proof Because $\tilde{\theta}_j^k \leq D_j(\tilde{\lambda}^k)$ holds for all $j = 1, \ldots, r$, we have

$$z_{\text{UB}} \leq \sum_{j=1}^r \tilde{\theta}_j^k \leq \sum_{j=1}^r D_j(\tilde{\lambda}^k) \leq z_{\text{LD}}.$$

Because $z_{\text{LD}} \leq z_{\text{UB}}$, we have $z_{\text{LD}} = \sum_{j=1}^r D_j(\tilde{\lambda}^k) = z_{\text{UB}}$. □

Proposition 5 states that a feasible master solution $(\tilde{\theta}^k, \tilde{\lambda}^k, \tilde{\pi}^k)$ satisfying $\sum_{j=1}^r \tilde{\theta}_j^k \geq z_{\text{UB}}$ at iteration k is an optimal solution of the Lagrangian dual problem if no cut is generated at such a point. The following corollary also suggests that at least one cut is generated for the feasible solution if $\sum_{j=1}^r \tilde{\theta}_j^k > z_{\text{UB}}$; that is, in the case that strict inequality holds.

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

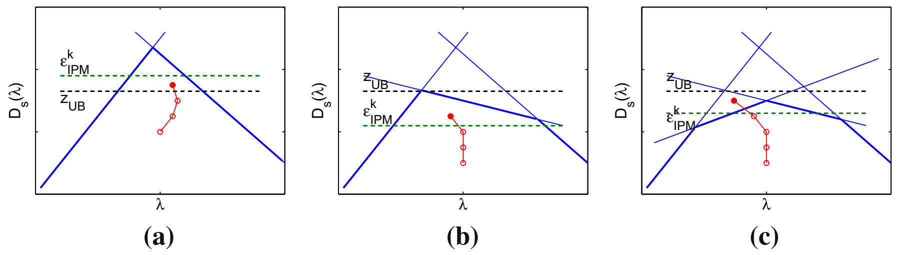

Fig. 1 Conceptual illustration of different termination cases of the IPM with the proposed criteria. **a** Terminated by criterion (19a). **b** Terminated by criterion (19b), but no cut generation. **c** Terminated by criterion (19b) with decreased ϵ^k_{IPM}

Figure 1 provides a conceptual illustration of the 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 Fig. 1b, the IPM terminated by criterion (19b), but no cut was generated at the solution. Hence, by decreasing the duality gap tolerance ϵ^k_{IPM} from Proposition 4, the interior-point solution moves toward an extreme point (see Fig. 1b, c). In Fig. 1c the IPM terminated with the decreased tolerance at the next iteration.

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 $g_k(\theta^k, \pi^k) < \epsilon_{\text{Opt}}$.

Theorem 1 *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 $\sum_{j=1}^{r} \tilde{\theta}_j^{k+1} \geq z_{\text{UB}}$ for given $z_{\text{UB}} < \infty$. We have two subcases:
  - If $\tilde{\theta}_j^k \leq D_j(\tilde{\lambda}^k)$ holds for all $j = 1, \ldots, r$, we have from Proposition 5 that the algorithm terminates with optimality (line 8).
  - Otherwise, from Corollary 1, we must have that the algorithm excludes the current solution by adding cutting-planes (13b) (line 10).
- Assume that a feasible solution of the master problem is found that satisfies $g_k(\tilde{\theta}^k, \tilde{\lambda}^k) < \epsilon^k_{\text{IPM}}$. We have two subcases:
  - If $\tilde{\theta}_j^k \leq D_j(\tilde{\lambda}^k)$ holds for all $j = 1, \ldots, r$, we have from Proposition 4 that the algorithm reduces ϵ^k_{IPM} by a factor of δ (line 15). An optimal master solution can thus be obtained after a finite number of reductions in ϵ^k_{IPM}. (line 17)
  - Otherwise, the algorithm excludes the current solution by adding (13b) (line 20).

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

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

```
1: $k \leftarrow 0$, $\lambda^0 \leftarrow 0$ and $z_{\rm UB} \leftarrow \infty$
2: loop
3:   CALL Procedure 1 to obtain $D_j(\lambda^{k+1})$ and $(x_j^{k+1}, y_j^{k+1})$ for given $\lambda^{k+1}$
4:   if $z_{\rm LB} = \infty$ then
5:     ADD cutting-planes (13b) to the master (13) for given $D(\lambda^k)$ and $x_j^k$
6:   else if $(\theta^{k+1}, \lambda^{k+1})$ is obtained from (19a) then
7:     if $\theta_j^{k+1} \le D_j(\lambda^{k+1})$ for all $j = 1, \ldots, r$ then
8:       STOP
9:     else
10:      ADD cutting-planes (13b) to the master (13) for given $D(\lambda^{k+1})$ and $x_j^{k+1}$
11:    end if
12:  else if $(\theta^{k+1}, \lambda^{k+1})$ is obtained from (19b) then
13:    if $\theta_j^{k+1} \le D_j(\lambda^{k+1})$ for all $j = 1, \ldots, r$ then
14:      if $\epsilon_{\rm IPM}^k > \epsilon_{\rm Opt}$ then
15:        UPDATE $\epsilon_{\rm IPM}^{k+1}$ from (21)
16:      else
17:        STOP
18:      end if
19:    else
20:      ADD cutting-planes (13b) to the master (13) for given $D(\lambda^{k+1})$ and $x_j^{k+1}$
21:    end if
22:  end if
23:  $z_{\rm LB} \leftarrow \max\{z_{\rm LB}, D(\lambda^k)\}$.
24:  SOLVE the master (13) by the IPM to obtain $(\theta^k, \lambda^k)$
25:  $k \leftarrow k + 1$
26: end loop
```

terminates in a finite number of steps, the algorithm terminates after a finite number of steps. □

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 note that DSP does not implement a branch-and-bound method to exploit Lagrangian lower bounds (as in [8,17,47]) . Consequently, our current implementation cannot be guaranteed to obtain an optimal solution for SMIP.

4.1 Software design

We first provide software implementation details and different avenues for the user to call the solver. 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.In Fig. 2 we illustrate the core classes in DSP using an UML class diagram.

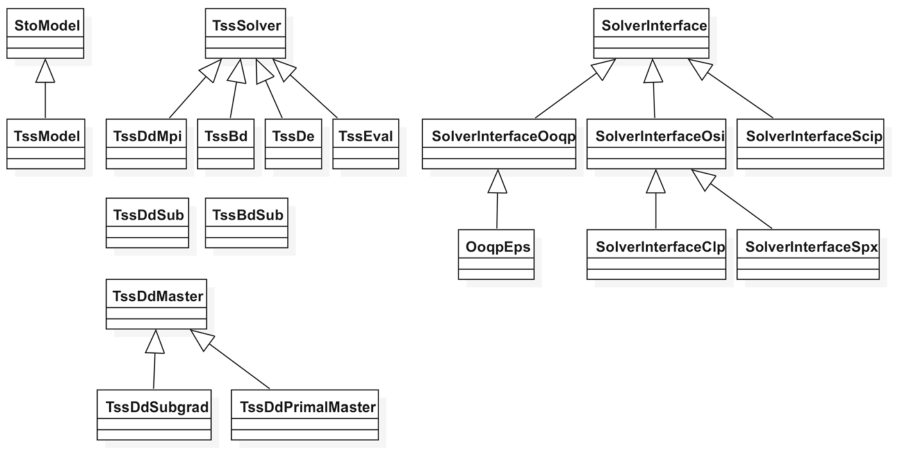

Fig. 2 UML (unified modeling language) class diagram for `DSP`

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 can derive new classes for their own purposes and efficiently manage model structure provided from several interfaces (e.g., `JuMP` and SMPS; see Sect. 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 the 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 $\widetilde{\mathcal{J}} \subseteq \mathcal{J} := \{1, \dots, r\}$ of scenarios to accelerate convergence.

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 [16,51,52,59]. An l_∞-norm trust region is also added to the master problems of Algorithm 2 in order to stabilize the cutting-plane method. The rule of updating the trust region follows that proposed in [35]. 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` [58], and `OOQP` [20]) and a mixed-integer solver (`SCIP` [1]). Users familiar with the COIN-OR Open Solver Interface [49] should easily be able to use the *SolverInterfaceOsi* class to derive classes for other solvers (e.g., `CPLEX` [29], `Gurobi` [27]).

4.1.4 Parallelization

The proposed dual decomposition method can be run on distributed-memory and shared-memory computing systems with multiple cores. The implementation protocol is MPI. In a distributed-memory environment, the scenario data are distributed to multiple processorsin a round-robin fashion. For the distributed scenario data, each processor solves the Lagrangian subproblems, generates Benders-type cuts, and solves upper-bounding problems (i.e., evaluating feasible solutions) in sequence at each iteration. For example, if a processor received the scenario data for scenarios 1 and 3, it solves the Lagrangian subproblems, cut-generation problems, and the upper-bounding problems corresponding to scenarios 1 and 3. The resulting partial information for lower bounds, cuts, and upper bounds will then be combined at the master processor (this prevents each processor from loading all the scenario data). Therefore, each iteration solves many MIP problems (i.e., Lagrangian subproblems and upper-bounding problems) and LP problems (i.e., cut-generation problems) in parallel. The root processor also solves the master problem to update the dual variables. By distributing scenario data, we can store and solve large-scale SMIP problem. As long as a single-scenario problem fits in a computing node, the number of scenarios can be increased to as many computing nodes as are available. However, each computing node requires enough memory to store at least one scenario problem in extensive form.

In a parallel dual decomposition setting, significant amounts of data need to be shared among processors. Shared data include subproblem solutions, dual variables, valid inequalities, best upper bound, and best lower bound. Data are communicated among the processors based on a master-worker framework. That is, the root processor collects and distributes data to all processors in a synchronous manner.

4.2 Interfaces for C, `JuMP`, 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 `JuMP` [4,37] and SMPS files [5] via a `Julia` package `Dsp.jl`.

The `Dsp.jl` package exploits the algebraic modeling capabilities of `JuMP`, which is a `Julia`-based modeling language. The use of `Julia` facilitates data handling and the use of supporting tools (e.g., statistical analysis and plotting). To illustrate these capabilities, we consider the two-stage stochastic integer program presented in [8]:

$$\min \left\{ -1.5\,x_1 - 4\,x_2 + \sum_{s=1}^{3} p_j\, Q(x_1, x_2, \xi_{1,s}, \xi_{2,s}) : x_1, x_2 \in \{0, \ldots, 5\} \right\},$$

where

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

and $(\xi_{1,s}, \xi_{2,s}) \in \{(7, 7), (11, 11), (13, 13)\}$ with probability 1/3. The corresponding `JuMP` model is given by:

```
using JuMP, Dsp, MPI; # Load packages
MPI.Init() # Initialize MPI
m = Model(3) # Create a model object with three scenarios
xi = [[7,7] [11,11] [13,13]] # Define random parameter
@variable(m, 0 <= x[i=1:2] <= 5, Int)
@objective(m, Min, -1.5*x[1]-4*x[2])
for s = 1:3
    q = Model(m, s, 1/3)
    @variable(q, y[j=1:4], Bin)
    @objective(q, Min, -16*y[1]+19*y[2]+23*y[3]+28*y[4])
    @constraint(q, 2*y[1]+3*y[2]+4*y[3]+5*y[4]<=xi[1,j]-
       x[1])
    @constraint(q, 6*y[1]+1*y[2]+3*y[3]+2*y[4]<=xi[2,s]-
       x[2])
end
solve(m, solve_type = :Dual, param = "myparam.txt")
MPI.Finalize(); # Finalize MPI
```

In the first line of this script we call the required `Julia` packages: `JuMP`, `Dsp`, and `MPI`. The `JuMP` model is defined in lines 3 to 14. The `Dsp.jl` package redefines the functions `JuMP.Model` and `JuMP.solve` in order to interface with the `DSP` solver. 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 line 14 with optional specifications for the solution method (dual decomposition, `:Dual`) and the algorithmic parameter file `myparam.txt`. More details are available in the software repositories. Note that only one line of code is required to invoke the parallel decomposition method. In other words, the user does not need to write any `MPI` code.

In the script provided we solve the problem using the dual decomposition method described in Sect. 3 (line 14).Arguments can be passed to the function `solve` to specify the desired solution method. For instance, if the user wishes to solve SMIP with continuous second-stage variables using Benders decomposition, then line 14 is replaced with

```
solve(m, solve_type = :Benders, param = "myparam.txt")
```

Note that this finds an optimal solution when the second stage has continuous variables only (otherwise it provides a lower bound). If the user desires to find an optimal solution for the extensive form of SMIP, then line 15 is replaced with

```
solve(m, solve_type = :Extensive, param = "myparam.txt")
```

`DSP` can also read a model provided as SMPS files. 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
readSmps("example")
optimize(solve_type = :Dual, param = "myparam.txt") #
    Solve
```

Remark 3 The `DSP` software package and the `Julia` packages (`JuMP` and `Dsp.jl`) are under active development. The up-to-date syntax and features are available in the project repositories.

5 Computational experiments

We present computational experiments to demonstrate the capabilities of `DSP`. We solve publicly available test problem instances (Sect. 5.1) and a stochastic unit commitment problem (Sect. 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 at 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 r 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.

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, $(z_{\text{UB}} - z_{\text{LB}})/|10^{-10} + z_{\text{UB}}|$, is less than 10^{-5}; (ii) $\beta_k \leq 10^{-6}$; (iii) the solution time exceeds 6 hours;and (iv) the number of iterations reaches 10^4. DDCP solves the master problem by using the simplex method implemented in Soplex-2.0.1 [58]; taking advantage of warm-start information from the previous iteration. DSP solves the master problem by using Mehrotra's predictor–corrector algorithm [42] implemented in OOQP-0.99.25 [20].In each iteration, the three methods evaluate no more than 100 primal solutions for upper bounds z_{UB}. 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.Appendix A provides the computational results obtained with FiberSCIP [54] for the extensive form of the test instances.

Figure 3 presents the optimality gap obtained by different dual decomposition methods for each SIPLIB instance. DDCP and DSP found optimal lower bounds

Table 1 Characteristics of the SIPLIB instances

		First stage			Second stage		
Name	Scen (r)	Rows	Cols	Ints	Rows	Cols	Ints
dcap233_r	200, 300, 500	6	12	6	15	27	27
dcap243_r	200, 300, 500	6	12	6	18	36	36
dcap332_r	200, 300, 500	6	12	6	12	24	24
dcap342_r	200, 300, 500	6	12	6	14	32	32
sslp_5_25_r	50, 100	1	5	5	30	130	125
sslp_10_50_r	50, 100, 500, 1000, 2000	1	10	10	60	510	500
sslp_15_45_r	5, 10, 15	1	15	15	60	690	675

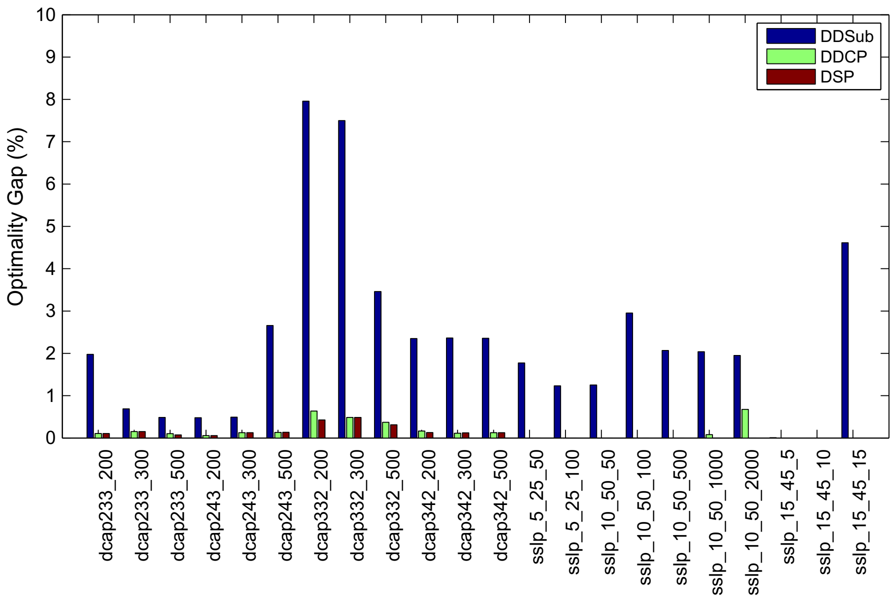

Fig. 3 Optimality gap obtained with different dual decomposition methods

z_{LD} for all the test problems, whereas DDSub found only suboptimal values.DSP also found tighter upper bounds for instances `dcap233_500`, `dcap332_200`, `dcap332_500`, and `dcap342_200` than did DDCP. Moreover, DDCP returned larger optimality gaps than did DSP for the `sslp_10_50_1000` and `sslp_10_50_2000` instances, since it did not terminate after 6 hours of solution time (see Table 2).

Figure 4 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. DSP reduced the solution time and number of iterations (relative to DDCP) 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 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.

Figure 5 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 6 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 the IPM and early termination.

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

Instance	Scen (r)	Method	Iter	UB	LB	Gap (%)	Time (s)
dcap233	200	DDSub	10,000	1835.34	1799.08	1.97	3853
		DDCP	85	1835.34	1833.38	0.11	935
		DSP	36	1835.34	1833.36	0.11	584
	300	DDSub	1503	1645.22	1633.88	0.68	5564
		DDCP	93	1645.22	1642.73	0.15	1705
		DSP	43	1645.22	1642.74	0.15	1145
	500	DDSub	2628	1737.94	1729.49	0.48	16,791
		DDCP	117	1738.47	1736.66	0.10	9451
		DSP	44	1737.94	1736.66	0.10	2155
dcap243	200	DDSub	1995	2322.50	2311.37	0.47	3157
		DDCP	52	2322.50	2321.18	0.05	843
		DSP	37	2322.50	2321.18	0.05	694
	300	DDSub	10,000	2559.48	2546.91	0.49	8869
		DDCP	53	2559.92	2556.66	0.12	1313
		DSP	36	2559.92	2556.67	0.12	1167
	500	DDSub	126	2167.97	2110.37	2.65	1992
		DDCP	64	2168.38	2165.47	0.13	2648
		DSP	39	2168.38	2165.46	0.13	2290
dcap332	200	DDSub	126	1068.24	983.23	7.95	443
		DDCP	85	1065.87	1059.08	0.64	708
		DSP	43	1063.52	1059.08	0.41	512
	300	DDSub	126	1260.42	1165.89	7.52	733
		DDCP	80	1257.01	1250.90	0.49	1298
		DSP	48	1257.01	1250.91	0.49	986
	500	DDSub	126	1596.49	1541.24	3.46	1333
		DDCP	76	1593.00	1587.06	0.37	2000
		DSP	43	1592.02	1587.06	0.31	1496
dcap342	200	DDSub	126	1620.19	1582.10	2.35	505
		DDCP	76	1620.76	1618.07	0.16	734
		DSP	37	1620.18	1618.07	0.13	577
	300	DDSub	126	2069.00	2020.11	2.36	881
		DDCP	73	2067.76	2065.43	0.11	1271
		DSP	37	2067.96	2065.42	0.12	930
	500	DDSub	126	1906.18	1861.25	2.36	1716
		DDCP	87	1905.38	1902.98	0.12	2400
		DSP	44	1905.36	1902.97	0.12	1788
sslp_5_25	50	DDSub	645	– 121.60	– 123.76	1.77	244
		DDCP	26	– 121.60	−121.60	0.00	14
		DSP	5	– 121.60	−121.60	0.00	6

Table 2 continued

Instance	Scen (r)	Method	Iter	UB	LB	Gap (%)	Time (s)
	100	DDSub	995	– 127.37	−128.94	1.23	643
		DDCP	28	– 127.37	−127.37	0.00	32
		DSP	5	– 127.37	−127.37	0.00	19
sslp_10_50	50	DDSub	774	– 364.64	−369.21	1.25	3105
		DDCP	62	– 364.64	−364.64	0.00	402
		DSP	11	– 364.64	−364.64	0.00	174
	100	DDSub	1224	– 354.19	−364.64	2.95	9591
		DDCP	72	– 354.19	−354.19	0.00	1545
		DSP	12	– 354.19	−354.19	0.00	777
	500	DDSub	344	– 349.13	−356.35	2.06	> 21,600
		DDCP	107	– 349.13	−349.13	0.00	10,771
		DSP	17	– 349.13	−349.13	0.00	1614
	1000	DDSub	177	– 351.71	−358.88	2.03	> 21,600
		DDCP	66	– 351.71	−351.99	0.07	> 21,600
		DSP	11	– 351.71	−351.71	0.00	3251
	2000	DDSub	115	– 347.26	−354.03	1.94	> 21,600
		DDCP	54	– 347.26	−349.60	0.67	> 21,600
		DSP	10	– 347.26	−347.26	0.00	4762
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	> 21,600
		DDCP	89	– 253.60	−253.60	0.00	14917
		DSP	17	– 253.60	−253.60	0.00	3092

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 7 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 approximates lower bounds poorly 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.

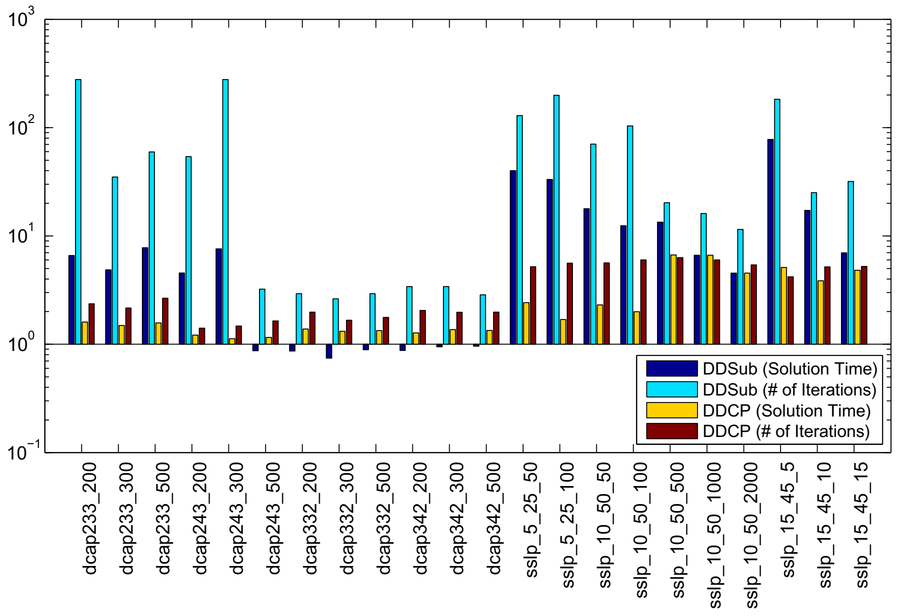

Fig. 4 Solution time and number of iterations ratios (relative to those obtained with DSP)

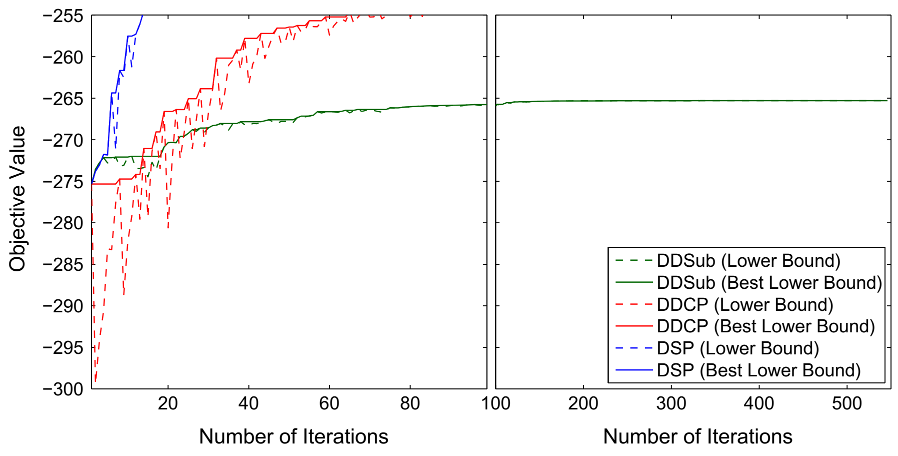

Fig. 5 Dual objective values and best lower bounds for SIPLIB instance `sslp_15_45_15` under different dual search methods

5.1.3 Scaling experiments

We present parallel scalability results for `DSP` running with up to 200 computing cores. We use the `sslp_10_50_r` instances where r presents the number of scenarios used (50, 100, 500, 1000, and 2000). We also present scaling profiles for solution times spent on different algorithmic steps and tasks for the `sslp_10_50_2000` instance. Figure 8 shows strong scaling performance results of `DSP`. We define the speedup as

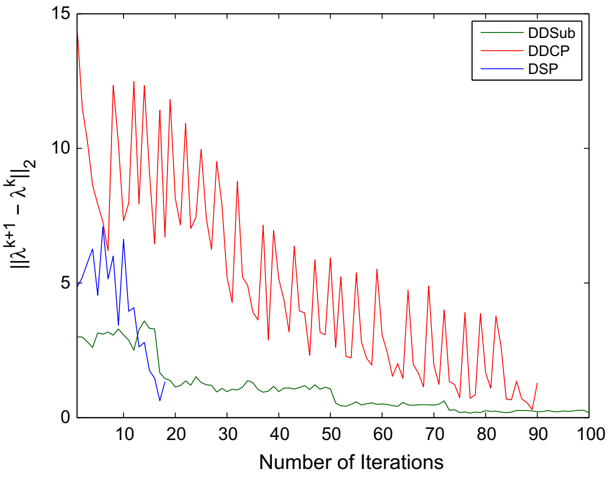

Fig. 6 Evolution of dual variables under different methods applied to `sslp_15_45_15`

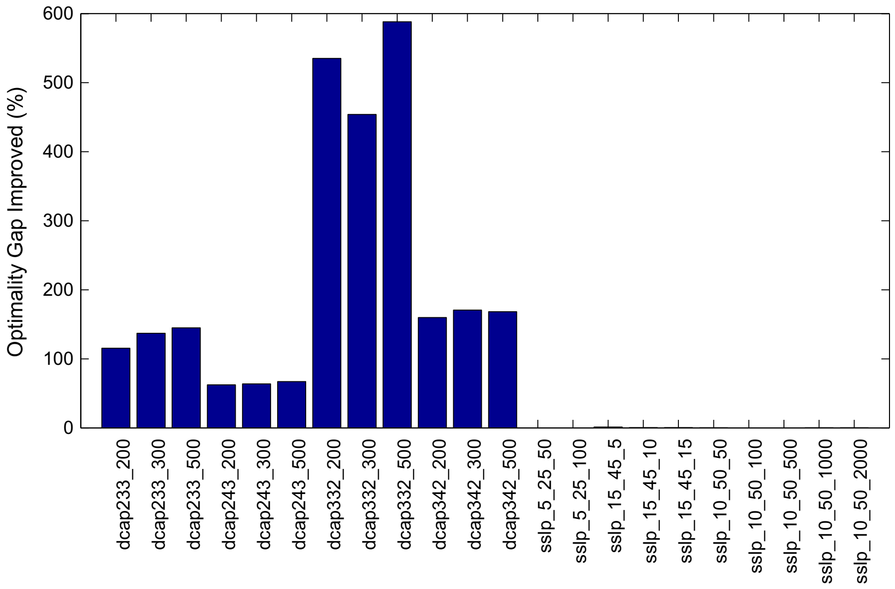

Fig. 7 Improvement in optimality gaps obtained with DSP over Benders decomposition

the ratio of the solution time with N cores to the solution time with five cores. We observe that DSP scales up as the number of cores increases for all the instances. As expected, scaling efficiency is lower for instances with fewer numbers of scenarios because subproblem solutions become less dominant. Figure 9 itemizes time spent on each of the algorithmic steps in addition to the MPI communication tasks. We see that time spent for solving lower and upper bounding problems dominates. We note that the

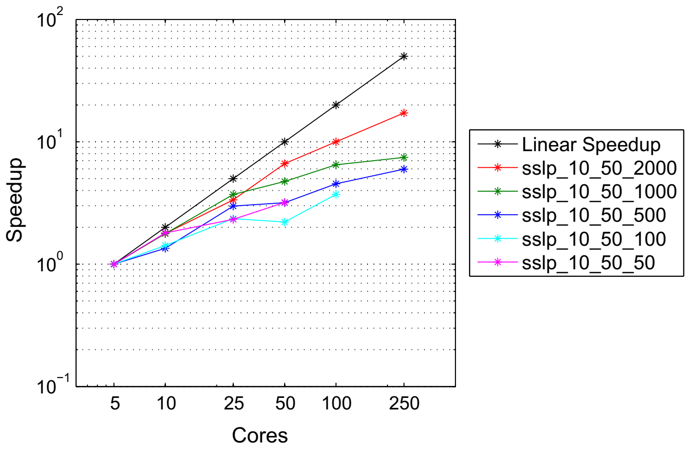

Fig. 8 Strong scaling results of DSP for the `sslp_10_50_`r instances with 5, 10, 25, 50, 100, and 200 cores, where $r \in \{50, 100, 500, 1000, 2000\}$

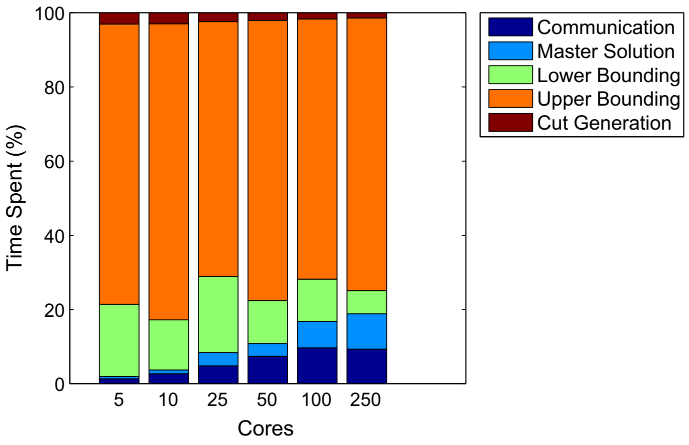

Fig. 9 Parallel scaling profile of DSP for `sslp_10_50_2000` instance with 5, 10, 25, 50, 100, and 250 cores

cut generation times are only a small fraction. We also observe that the master solution time and communication time increases with the number of cores, as expected.

5.2 Large-scale stochastic unit commitment

We test performance of different methods using 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 [34]. 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

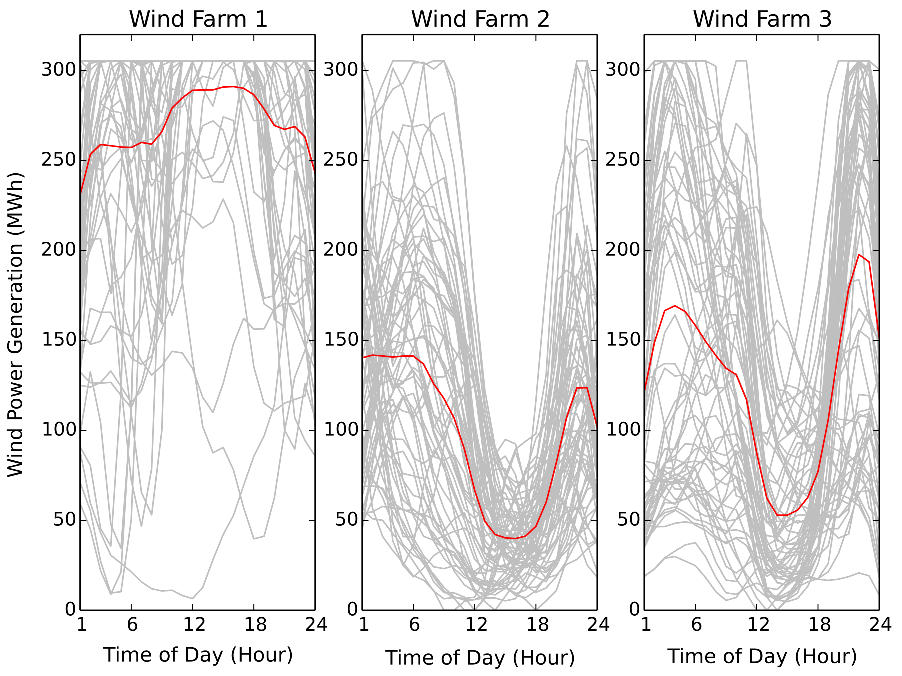

Fig. 10 Wind generation scenarios used in stochastic unit commitment model

Table 3 Sizes for stochastic unit commitment instances

Scen (r)	# Rows	# Columns	# Integers
4	120,015	38,880	2592
8	229,303	75,168	4320
16	447,879	147,744	7776
32	885,031	292,896	14,688
64	1,759,335	583,200	28,512

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 10 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.

The mathematical formulation and the `JuMP` modeling script of the two-stage stochastic unit commitment model is provided in Appendices B, C, and D.

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 summarizes the numerical results. Each instance uses the same number of computing cores as scenarios. We add only

Table 4 Numerical results for stochastic unit commitment problem under DSP

Scen (r)	Iter	LB	UB	Gap (%)	Time (s)
4	1	906,979.1	907,046.1	< 0.01	590
8	1	903,953.5	904,006.6	< 0.01	785
16	1	900,650.7	900,706.3	< 0.01	1293
32	19	903,149.9	903,227.7	< 0.01	19,547
64	6	894,756.5	895,118.0	0.04	21,600

one aggregated cut to the master problem at each iteration (see Remark 1). DSP found upper and lower bounds with $< 0.01\%$ optimality gap for the 4-, 8-, 16-, and 32-scenario instances. Most notably, DSP terminated after the first iteration for the 4-, 8- and 16-scenario instances because of the addition of valid inequalities in Procedure 1 to the Lagrangian subproblems. This fast convergence behavior becomes evident in Sect. 5.2.1, where we compare the LB and Initial LB columns for the instances terminating in one iteration. We observe, however, 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 does not represent a bottleneck.

5.2.1 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 11 shows the best upper bound and the best lower boundof the 8-scenario stochastic unit commitment problem 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.

Table 5 reports the lower and upper bounds obtained at the first iteration by DSP-P1 and DSP+P1. Procedure 1 improves lower bound by up to 0.7% by adding valid inequalities. More importantly, it allowed us to find upper bounds at the first iteration, which then can be used as a termination criterion (19a) of the interior-point cutting-plane method.

Table 6 presents detailed computational results for four methods: DDCP-P1 (Algorithm 2 without Procedure 1), DDCP+P1 (Algorithm 2 with Procedure 1), DSP-P1, and

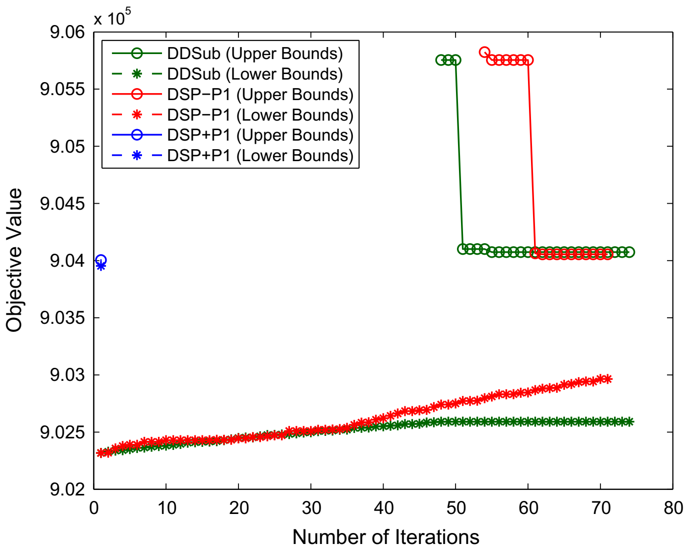

Fig. 11 Upper bounds and lower boundsof the stochastic unit commitment problem with 8 scenarios, obtained with DSP with and without Procedure 1 and with the subgradient method

Table 5 Upper bounds and lower bounds of the stochastic unit commitment problem resulting from DSP with and without Procedure 1 at the first iteration

	DSP-P1		DSP+P1		
Scen (r)	LB	UB	LB	UB	LB Improved (%)
4	905,613.8	∞	906,979.1	907,046.1	0.15
8	902,319.4	∞	903,953.5	904,004.6	0.18
16	899,000.3	∞	900,650.7	900,708.4	0.18
32	897,114.5	903227.7	902,902.3	903,227.7	0.65
64	888,418.6	∞	894,643.6	895,118.0	0.70

DSP+P1 (as defined previously). These results seek to highlight the positive impact of using the interior-point cutting-plane method and Procedure 1. For the 4-, 8, and 16-scenario instances, both DDCP+P1 and DSP+P1 found upper and lower bounds with $< 0.01\%$ optimality gaps. However, without Procedure 1, DDCP-P1 did not terminate within the 6-hour time limit for all the instances, of which no upper bound was found for the 4- and 8-scenario instances. The reasons is that the problem does not have relative complete recourse. DSP-P1 did not terminate within the time limit for all the instances except the 4-scenario instance. Note that DDCP+1 and DSP+P1 are exactly the same method when they terminate at the first iteration for the 4-, 8-, and 16-scenario instances. For the 32-scenario instance, DSP+P1 terminated with $< 0.01\%$ optimality gap, whereas DDCP+P1 took more iterations and reached the time limit with a larger gap. For the 64-scenario instance, DSP+P1 found a better lower bound than did DDCP+P1. However, both DDCP+P1 and DSP+P1 did not terminate after 6

Table 6 Computational results from DDCP (Algorithm 2) and DSP (Algorithm 3) with and without Procedure 1

Scen (r)	Iter	Method	LB	UB	Gap (%)	Time (s)
4	153	DDCP-P1	905,708.5	∞	∞	$> 21{,}600$
	1	DDCP+P1	906,979.1	907,046.1	< 0.01	566
	35	DSP-P1	906,974.7	907,036.0	< 0.01	6161
	1	DSP+P1	906,979.1	907,046.1	< 0.01	590
8	55	DDCP-P1	902,385.2	∞	∞	$> 21{,}600$
	1	DDCP+P1	903,953.5	904,004.6	< 0.01	810
	73	DSP-P1	902,974.0	904,054.9	0.12	$> 21{,}600$
	1	DSP+P1	903,953.5	904,004.6	< 0.01	785
16	49	DDCP-P1	899,046.9	900,767.2	0.19	$> 21{,}600$
	1	DDCP+P1	900,650.7	900,708.4	< 0.01	1298
	41	DSP-P1	899,578.1	900,732.1	0.13	$> 21{,}600$
	1	DSP+P1	900,650.7	900,706.3	< 0.01	1293
32	34	DDCP-P1	897,083.7	904,440.9	0.81	$> 21{,}600$
	19	DDCP+P1	902,902.3	903,227.7	0.04	$> 21{,}600$
	22	DSP-P1	898,404.1	903,227.7	0.53	$> 21{,}600$
	19	DSP+P1	903,149.9	903,227.7	< 0.01	19,547
64	18	DDCP-P1	888,571.5	895,682.2	0.79	$> 21{,}600$
	5	DDCP+P1	894,655.3	895,118.0	0.05	$> 21{,}600$
	10	DSP-P1	891,820.9	895,256.5	0.38	$> 21{,}600$
	6	DSP+P1	894,756.6	895,118.0	0.04	$> 21{,}600$

hours. We attribute this behavior to the fact that the interior-point cutting-plane method computed lower bounds more effectively and ameliorated dual oscillations.

5.2.2 Extensive form solutions

We conclude this section by reporting the computational results from solving the extensive form of the stochastic unit commitment problems. The results are summarized in Table 7. 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 a 6-hour time limit, upper and lower bounds obtained from the extensive forms were not better than those obtained with DSP.We also tested `ParaSCIP` [53] for solving the extensive forms, but the solver ran out of memory in all instances.

6 Summary and directions of future work

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

Table 7 Numerical results for the extensive form of the stochastic unit commitment problems

Scen (r)	Number of B&C Nodes	UB	LB	Gap (%)	Time (s)
4	88,831	907,035.3	906,089.9	0.01	6632
8	58,235	904,068.1	903,567.8	0.05	$>$ 21,600
16	3505	900,806.1	900,200.3	0.07	$>$ 21,600
32	9	907,536.0	901,759.8	0.64	$>$ 21,600
64	1	∞	33,605.4	∞	$>$ 21,600

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 proved that the dual decomposition method incorporating our innovations converge in a finite number of iterations to an optimal solution of the Lagrangian dual problem. 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, Julia, and SMPS formats. The object-oriented design of the implementation enables users to customize decomposition techniques. We use DSP to benchmark different methods on standard problem instances and stochastic unit commitment problems. The numerical results show significant improvements in terms of the quality of upper bounds, number of iterations, and time for all instances. We also verified that the solver can achieve strong scaling.

We will seek to scale our method to solve problems with a larger number of scenarios. In particular, we have observed load imbalances because each Lagrangian subproblem is a large mixed-integer program with unpredictable solution times. Such imbalances can be alleviated by asynchronous parallel implementation that enable early subproblem termination. Moreover, solving the master problem can become a bottleneck as cuts are accumulated. A warm-start technique for the primal-dual interior-point method [22,24] and a parallelization of the master problem as proposed in [38] can ameliorate such effects. We will address these issues in future work.

A Computational results for SIPLIB test instances using FiberSCIP

We present the computational results from FiberSCIP (compiled with ug-0.7.5) [54] for solving SIPLIB test instances using 16 computing cores in extensive form. FiberSCIP terminates after 6 hours of solution. Results are reported in Table 8.

For all the dcap instances, FiberSCIP found upper and lower bounds with small duality gap after 6 hours of solution. An optimal solution is found for dcap243_200 instance. Optimal solutions were also found for the sslp_5_25 and sslp_15_45 instances within the 6-hour time limit. However, most sslp_10_50 instances were not able to be solved by FiberSCIP, because of insufficient memory. Note that

Table 8 Computational results for SIPLIB instances using `FiberSCIP` for solving the extensive forms

Instance	Scen	UB	LB	Gap (%)	Time (s)
`dcap233`	200	1834.56	1834.38	0.01	21,600
	300	1644.55	1642.92	0.10	21,600
	500	1737.52	1737.31	0.01	21,600
`dcap243`	200	2322.49	2322.49	0.00	1444
	300	2559.23	2558.58	0.02	21,600
	500	2167.37	2166.93	0.02	21,600
`dcap332`	200	1060.74	1060.13	0.06	21,600
	300	1252.93	1252.26	0.05	21,600
	500	1588.84	1588.12	0.05	21,600
`dcap342`	200	1619.54	1619.41	0.01	21,600
	300	2068.93	2066.07	0.14	21,600
	500	1908.55	1903.46	0.27	21,600
`sslp_5_25`	50	−121.60	−121.60	0.00	338
	100	−127.37	−127.37	0.00	1662
`sslp_10_50`	50	−364.30	−365.36	0.29	21,600
	100	N/A	N/A	N/A	N/A
	500	N/A	N/A	N/A	N/A
	1000	N/A	N/A	N/A	N/A
	2000	N/A	N/A	N/A	N/A
`sslp_15_45`	5	−262.40	−262.40	0.00	180
	10	−260.50	−260.50	0.00	555
	15	−253.60	−265.60	0.00	2525

memory was not sufficient for solving the stochastic unit commitment problems by using `ParaSCIP`.

B Notations: two-stage stochastic unit commitment

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

C 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. Notations are given in Table 9. 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.

Table 9 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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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 a *nonanticipativity* constraint. Equations (22p)–(22s) represent the initial conditions of generators and production level.

D `Julia` model script: two-stage stochastic unit commitment

We provide the `Julia` model script for the stochastic unit commitment problem.

```
using JuMP, Dsp, MPI

# Initialize MPI
MPI.Init()

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

# JuMP object
m = Model(nScenarios);

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

# First-stage Objective function
@objective(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
@constraint(m, LINKING_SHUT_DOWN0[i=SLOWGENS],
       Down[i,1] <= use_0[i])
@constraint(m, LINKING_SHUT_DOWN[i=SLOWGENS, t=2:
    nPeriods],
       Use[i,t-1] >= Down[i,t])
@constraint(m, LINKING_START_UP0[i=SLOWGENS],
        Up[i,1] <= 1 - use_0[i])
@constraint(m, LINKING_START_UP[i=SLOWGENS, t=2:nPeriods
    ],
        1 - Use[i,t-1] >= Up[i,t])
@constraint(m, LINKING_BOTH0[i=SLOWGENS],
        Use[i,1] - use_0[i] == Up[i,1] - Down[i,1])
@constraint(m, LINKING_BOTH[i=SLOWGENS, t=2:nPeriods],
        Use[i,t] - Use[i,t-1] == Up[i,t] - Down[i,t])

# Min down time
@constraint(m, MIN_DOWN_INIT[i=SLOWGENS, t=1:min(
    downtime_init[i],nPeriods)],
        Use[i,t] == 0)
@constraint(m, MIN_DOWN_S1[i=SLOWGENS, t=PERIODS, s=max
    (1,t-downtime[i]+1):t],
        1 - Use[i,t] >= Down[i,s])
@constraint(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
@constraint(m, MIN_UP_INIT[i=SLOWGENS, t=1:min(
    uptime_init[i],nPeriods)],
        Use[i,t] == 1)
@constraint(m, MIN_UP_S1[i=SLOWGENS, t=PERIODS, s=max(1,
    t-uptime[i]+1):t],
        Use[i,t] >= Up[i,s])
@constraint(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
for s in blockids()

        # Second-stage momdel
        sb = Model(m, s, prob[s])

        @variable(sb, UseF[i=FASTGENS, t=PERIODS], Bin) #
            Generator on/off indicator
        @variable(sb, 0 <= UpF[i=FASTGENS, t=PERIODS] <=
            1) # Start up
            indicator
        @variable(sb, 0 <= DownF[i=FASTGENS, t=PERIODS]
            <= 1) # Shut down
            indicator
        @variable(sb, 0 <= Gen[i=GENERATORS, t=PERIODS]
            <= max_gen[i]) # Power
            generation
        @variable(sb, 0 <= Gen_Sgmt[i=GENERATORS, k=
            SEGMENTS, t=PERIODS] <= max_gen_sgmt[i,k])
        @variable(sb, 0 <= Spin_Resv[i=GENERATORS, t=
            PERIODS] <= spin_notice / 60. * ramp_rate[i])
```

```
          # Spinning
        reserve

    # Second-stage Objective function
    @objective(sb, Min,
            sum{cost_start[i] * UpF[i,t], i=FASTGENS,
                t=PERIODS}
            + sum{fixed_cost_gen[i] * UseF[i,t], i=
                FASTGENS, t=PERIODS}
            + sum{cost_gen[i,k] * Gen_Sgmt[i,k,t], i=
                GENERATORS, k=SEGMENTS, t=PERIODS})

    # Linking Use / Up / Down variables
    @constraint(sb, FAST_LINKING_SHUT_DOWN0[i=
        FASTGENS],
            DownF[i,1] <= use_0[i])
    @constraint(sb, FAST_LINKING_SHUT_DOWN[i=FASTGENS
        , t=2:nPeriods],
            UseF[i,t-1] >= DownF[i,t])
    @constraint(sb, FAST_LINKING_START_UP0[i=FASTGENS
        ],
            UpF[i,1] <= 1 - use_0[i])
    @constraint(sb, FAST_LINKING_START_UP[i=FASTGENS,
         t=2:nPeriods],
            1 - UseF[i,t-1] >= UpF[i,t])
    @constraint(sb, FAST_LINKING_BOTH0[i=FASTGENS],
            UseF[i,1] - use_0[i] == UpF[i,1] - DownF[i
                ,1])
    @constraint(sb, FAST_LINKING_BOTH[i=FASTGENS, t
        =2:nPeriods],
            UseF[i,t] - UseF[i,t-1] == UpF[i,t] -
                DownF[i,t])

    # Min down time
    @constraint(sb, FAST_MIN_DOWN_INIT[i=FASTGENS, t
        =1:min(downtime_init[i],nPeriods)],
            UseF[i,t] == 0)
    @constraint(sb, FAST_MIN_DOWN_S1[i=FASTGENS, t=
        PERIODS, tt=max(1,t-downtime[i]+1):t],
            1 - UseF[i,t] >= DownF[i,tt])
    @constraint(sb, FAST_MIN_DOWN_S2[i=FASTGENS, t=
        PERIODS],
            1 - UseF[i,t] >= sum{DownF[i,tt], tt=max
                (1,t-downtime[i]+1):t})

    # Min up time
    @constraint(sb, FAST_MIN_UP_INIT[i=FASTGENS, t=1:
        min(uptime_init[i],nPeriods)],
            UseF[i,t] == 1)
    @constraint(sb, FAST_MIN_UP_S1[i=FASTGENS, t=
        PERIODS, tt=max(1,t-uptime[i]+1):t],
            UseF[i,t] >= UpF[i,tt])
    @constraint(sb, FAST_MIN_UP_S2[i=FASTGENS, t=
        PERIODS],
            UseF[i,t] >= sum{UpF[i,tt], tt=max(1,t-
                uptime[i]+1):t})


    # Ramping rate in normal operating status
    @constraint(sb, RAMP_DOWN0[i=GENERATORS],
            gen_0[i] - Gen[i,1] <= ramp_rate[i])
    @constraint(sb, RAMP_DOWN[i=GENERATORS, t=2:
        nPeriods],
            Gen[i,t-1] - Gen[i,t] <= ramp_rate[i])
    @constraint(sb, RAMP_UP0[i=GENERATORS],
            Gen[i,1] - gen_0[i] + Spin_Resv[i,1] <=
                ramp_rate[i])
    @constraint(sb, RAMP_UP[i=GENERATORS, t=2:
        nPeriods],
            Gen[i,t] - Gen[i,t-1] + Spin_Resv[i,t] <=
                ramp_rate[i])

    # Spinning reserve requirement for system
    @constraint(sb, SPIN_RESV_REQ[t=PERIODS],
            sum{Spin_Resv[i,t], i=GENERATORS}
            >= spin_resv_rate * (total_demand[t] -
                total_wind_scen[t,s]))

    # Spinning reserve requirement for system
    @constraint(sb, SPIN_RESV_REQ[t=PERIODS],
            sum{Spin_Resv[i,t], i=GENERATORS}
            >= spin_resv_rate * (total_demand[t] -
                total_wind_scen[t,s]))

    # Spinning reserve capacity for individual unit
    @constraint(sb, SPIN_RESV_MAX_SLOW[i=SLOWGENS, t=
        PERIODS],
            Spin_Resv[i,t] <= spin_notice / 60. *
                ramp_rate[i] * Use[i,t])
    @constraint(sb, SPIN_RESV_MAX_FAST[i=FASTGENS, t=
        PERIODS],
            Spin_Resv[i,t] <= spin_notice / 60. *
                ramp_rate[i] * UseF[i,t])

    # Power output capacity constraints
    @constraint(sb, POWER_OUTPUT_SLOW[i=SLOWGENS, t=
        PERIODS],
            Gen[i,t] == min_gen[i] * Use[i,t] + sum{
                Gen_Sgmt[i,k,t], k=SEGMENTS})
    @constraint(sb, POWER_SEGMENT_SLOW[i=SLOWGENS, k=
        SEGMENTS, t=PERIODS],
            Gen_Sgmt[i,k,t] <= max_gen_sgmt[i,k] * Use
                [i,t])
    @constraint(sb, POWER_MAX_SLOW[i=SLOWGENS, t=
        PERIODS],
            Gen[i,t] + Spin_Resv[i,t] <= max_gen[i] * Use[i,t])
    @constraint(sb, POWER_OUTPUT_FAST[i=FASTGENS, t=PERIODS],
            Gen[i,t] == min_gen[i] * UseF[i,t] + sum{Gen_Sgmt[i,k,t], k=SEGMENTS})
    @constraint(sb, POWER_SEGMENT_FAST[i=FASTGENS, k=SEGMENTS, t=PERIODS],
            Gen_Sgmt[i,k,t] <= max_gen_sgmt[i,k] * UseF[i,t])
    @constraint(sb, POWER_MAX_FAST[i=FASTGENS, t=PERIODS],
            Gen[i,t] + Spin_Resv[i,t] <= max_gen[i] * UseF[i,t])

    # Power balance constraints for system
    @constraint(sb, POWER_BALANCE[t=PERIODS],
            sum{Gen[i,t], i=GENERATORS} == total_demand[t] - total_wind_scen[t,s])

    # Transmission constraints with load shift factor (These can be lazy constraints.)
    @constraint(sb, FLOW_BRANCH_LSF_LB[l=BRANCHES, t=PERIODS],
            sum{load_shift_factor[n,l] * Gen[i,t], n=BUSES, i=GENERATORS; gen_bus_id[i] == n}
            >= sum{load_shift_factor[n,l] * demand[n,t], n=BUSES}
            - sum{load_shift_factor[n,l] * wind_scen[wn,t,s], n=BUSES, wn=WINDS; wind_bus_id[wn] == n}
            - flow_max[l])
    @constraint(sb, FLOW_BRANCH_LSF_UB[l=BRANCHES, t=PERIODS],
            sum{load_shift_factor[n,l] * Gen[i,t], n=BUSES, i=GENERATORS; gen_bus_id[i] == n}
            <= sum{load_shift_factor[n,l] * demand[n,t], n=BUSES}
            - sum{load_shift_factor[n,l] * wind_scen[wn,t,s], n=BUSES, wn=WINDS; wind_bus_id[wn] == n}
            + flow_max[l])
end

status = solve(m, solver_type = :Dual); # solve model

# print out some results
println("Solution status: ", status);
println("Primal Bound : ", getprimobjval());
println("Dual Bound : ", getdualobjval());

MPI.Finalize();
```

References

1. Achterberg, T.: SCIP: solving constraint integer programs. Math. Program. Comput. **1**(1), 1–41 (2009)
2. Ahmed, S.: A scenario decomposition algorithm for 0–1 stochastic programs. Oper. Res. Lett. **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. Math. Program. **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, Berlin (2011)
7. Bixby, R.E.: Solving real-world linear programs: a decade and more of progress. Oper. Res. **50**(1), 3–15 (2002)
8. Carøe, C.C., Schultz, R.: Dual decomposition in stochastic integer programming. Oper. Res. Lett. **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. Oper. Res. **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. Manag. Sci. **50**(12–supplement), 1861–1871 (2004)
13. Forrest, J.: Cbc. https://projects.coin-or.org/Cbc
14. Forrest, J.: Clp. https://projects.coin-or.org/Clp
15. Frangioni, A.: About lagrangian methods in integer optimization. Ann. Oper. Res. **139**(1), 163–193 (2005)
16. Gade, D., Küçükyavuz, S., Sen, S.: Decomposition algorithms with parametric gomory cuts for two-stage stochastic integer programs. Math. Program. **144**(1–2), 39–64 (2014)
17. Gamrath, G., Lübbecke, M.E.: Experiments with a generic Dantzig–Wolfe decomposition for integer programs. In: International Symposium on Experimental Algorithms, pp. 239–252. Springer (2010)
18. Gassmann, H.I., Schweitzer, E.: A comprehensive input format for stochastic linear programs. Ann. Oper. Res. **104**(1–4), 89–125 (2001)
19. Geoffrion, A.M.: Lagrangean Relaxation for Integer Programming. Springer, Berlin (1974)
20. Gertz, E.M., Wright, S.J.: Object-oriented software for quadratic programming. ACM Trans. Math. Softw. (TOMS) **29**(1), 58–81 (2003)
21. Goffin, J.L., Vial, J.P.: Cutting planes and column generation techniques with the projective algorithm. J. Optim. Theory Appl. **65**(3), 409–429 (1990)
22. Gondzio, J.: Warm start of the primal-dual method applied in the cutting-plane scheme. Math. Program. **83**(1–3), 125–143 (1998)
23. Gondzio, J., Gonzalez-Brevis, P., Munari, P.: New developments in the primal-dual column generation technique. Eur. J. Oper. Res. **224**(1), 41–51 (2013)
24. Gondzio, J., Grothey, A.: A new unblocking technique to warmstart interior point methods based on sensitivity analysis. SIAM J. Optim. **19**(3), 1184–1210 (2008)
25. Gondzio, J., Sarkissian, R.: Column generation with a primal-dual method. Relatorio tecnico, University of Geneva **102** (1996)
26. Guo, G., Hackebeil, G., Ryan, S.M., Watson, J.P., Woodruff, D.L.: Integration of progressive hedging and dual decomposition in stochastic integer programs. Oper. Res. Lett. **43**(3), 311–316 (2015)
27. Gurobi Optimization, Inc.: Gurobi optimizer reference manual (2015). http://www.gurobi.com
28. Helmberg, C.: ConicBundle. https://www-user.tu-chemnitz.de/~helmberg/ (2004)

29. IBM Corp.: IBM ILOG CPLEX Optimization Studio 12.6.1 (2014). http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html
30. Kim, K., Mehrotra, S.: A two-stage stochastic integer programming approach to integrated staffing and scheduling with application to nurse management. Oper. Res. **63**(6), 1431–1451 (2015). https://doi.org/10.1287/opre.2015.1421
31. King, A.: Stochastic Modeling Interface (2007). https://projects.coin-or.org/Smi
32. Kleywegt, A.J., Shapiro, A., Homem-de Mello, T.: The sample average approximation method for stochastic discrete optimization. SIAM J. Optim. **12**(2), 479–502 (2002)
33. Laporte, G., Louveaux, F.V.: The integer L-shaped method for stochastic integer programs with complete recourse. Oper. Res. Lett. **13**(3), 133–142 (1993)
34. Lee, C., Liu, C., Mehrotra, S., Shahidehpour, M.: Modeling transmission line constraints in two-stage robust unit commitment problem. IEEE Trans. Power Syst. **29**(3), 1221–1231 (2014)
35. Linderoth, J., Wright, S.: Decomposition algorithms for stochastic programming on a computational grid. Comput. Optim. Appl. **24**(2–3), 207–250 (2003)
36. Løkketangen, A., Woodruff, D.L.: Progressive hedging and tabu search applied to mixed integer (0, 1) multistage stochastic programming. J. Heuristics **2**(2), 111–128 (1996)
37. Lubin, M., Dunning, I.: Computing in operations research using Julia. arXiv preprint arXiv:1312.1431 (2013)
38. Lubin, M., Martin, K., Petra, C.G., Sandıkçı, B.: On parallelizing dual decomposition in stochastic integer programming. Oper. Res. Lett. **41**(3), 252–258 (2013)
39. Lubin, M., Petra, C.G., Anitescu, M., Zavala, V.: Scalable stochastic optimization of complex energy systems. In: 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pp. 1–10. IEEE (2011)
40. Lulli, G., Sen, S.: A branch-and-price algorithm for multistage stochastic integer programming with application to stochastic batch-sizing problems. Manag. Sci. **50**(6), 786–796 (2004)
41. 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)
42. Mehrotra, S.: On the implementation of a primal-dual interior point method. SIAM J. Optim. **2**(4), 575–601 (1992)
43. Mitchell, J.E.: Computational experience with an interior point cutting plane algorithm. SIAM J. Optim. **10**(4), 1212–1227 (2000)
44. OptiRisk Systems: FortSP: a stochastic programming solver, version 1.2 (2014). http://www.optirisk-systems.com/manuals/FortspManual.pdf
45. Papavasiliou, A., Oren, S.S.: Multiarea stochastic unit commitment for high wind penetration in a transmission constrained network. Oper. Res. **61**(3), 578–592 (2013)
46. Papavasiliou, A., Oren, S.S., O'Neill, R.P.: Reserve requirements for wind power integration: a scenario-based stochastic programming framework. IEEE Trans. Power Syst. **26**(4), 2197–2206 (2011)
47. Ralphs, T.K., Galati, M.V.: Decomposition in integer linear programming. Integer Program. Theory Pract. **3**, 57–110 (2005)
48. Ralphs, T.K., Hassanzadeh, A.: A generalization of Benders algorithm for two-stage stochastic optimization problems with mixed integer recourse. Technical Report 14T-005, Department of Industrial and Systems Engineering, Lehigh University (2014)
49. Saltzman, M., Ladányi, L., Ralphs, T.: The COIN-OR open solver interface: technology overview. In: CORS/INFORMS Conference. Banff (2004)
50. Santoso, T., Ahmed, S., Goetschalckx, M., Shapiro, A.: A stochastic programming approach for supply chain network design under uncertainty. Eur. J. Oper. Res. **167**(1), 96–115 (2005)
51. Sen, S., Higle, J.L.: The C^3 theorem and a D^2 algorithm for large scale stochastic mixed-integer programming: set convexification. Math. Program. **104**(1), 1–20 (2005)
52. Sherali, H.D., Fraticelli, B.M.: A modification of Benders' decomposition algorithm for discrete subproblems: an approach for stochastic programs with integer recourse. J. Global Optim. **22**(1–4), 319–342 (2002)
53. Shinano, Y., Achterberg, T., Berthold, T., Heinz, S., Koch, T.: ParaSCIP: a parallel extension of SCIP. In: Competence in High Performance Computing 2010, pp. 135–148. Springer (2011)
54. Shinano, Y., Heinz, S., Vigerske, S., Winkler, M.: Fiberscip-a shared memory parallelization of scip, pp. 13–55. Zuse Institute Berlin, Technical Report ZR (2013)
55. Tarhan, B., Grossmann, I.E.: Improving dual bound for stochastic MILP models using sensitivity analysis. Working paper (2015)

56. Watson, J.P., Woodruff, D.L.: Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems. CMS **8**(4), 355–370 (2011)
57. Watson, J.P., Woodruff, D.L., Hart, W.E.: PySP: modeling and solving stochastic programs in Python. Math. Program. Comput. **4**(2), 109–149 (2012)
58. Wunderling, R.: Paralleler und objektorientierter Simplex-Algorithmus. Ph.D. Thesis, Technische Universität Berlin (1996). http://www.zib.de/Publications/abstracts/TR-96-09/
59. Zhang, M., Kucukyavuz, S.: Finitely convergent decomposition algorithms for two-stage stochastic pure integer programs. SIAM J. Optim. **24**(4), 1933–1951 (2014)
60. 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. Math. Program. Comput. **4**(3), 211–238 (2012)