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

A Review on the Performance of Linear and Mixed Integer Two-Stage Stochastic Programming Algorithms and Software

Juan J. Torres · Can Li · Robert M. Apap · Ignacio E. Grossmann

Received: date / Accepted: date

Abstract This paper presents a tutorial on the state-of-the-art methodologies for the solution of two-stage (mixed-integer) linear stochastic programs and provides a list of software designed for this purpose. The methodologies are classified according to the decomposition alternatives and the types of the variables in the problem. We review the fundamentals of Benders Decomposition, Dual Decomposition and Progressive Hedging, as well as possible improvements and variants. We also present extensive numerical results to underline the properties and performance of each algorithm using software implementations including DECIS, FORTSP, PySP, and DSP. Finally, we discuss the strengths and weaknesses of each methodology and propose future research directions.

Keywords Stochastic programming · L-shaped method · Scenario Decomposition · Software benchmark

Juan J. Torres
Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS, (UMR 7030), France
E-mail: torresfigueroa@lipn.univ-paris13.fr

Can Li
Department of Chemical Engineering, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213, USA
E-mail: canl1@cmu.edu

Robert M. Apap
Department of Chemical Engineering, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213, USA

Ignacio E. Grossmann
Department of Chemical Engineering, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213, USA
E-mail: grossmann@cmu.edu

1 Introduction

In the modeling and optimization of real-world problems, there is usually a level of uncertainty associated with the input parameters and their future outcomes. Stochastic Programming (SP) models have been widely studied to solve optimization problems under uncertainty over the past decades [5, 31]. SP is acknowledged for providing superior results than the corresponding deterministic model with nominal values for the uncertain parameters, which can lead to suboptimal or infeasible solutions. SP applications in process systems engineering include manufacturing networks and supply chain optimization [17, 37], production scheduling [63], synthesis of process networks [57].

Two-stage stochastic programming is a commonly applied framework for cases where parameter uncertainties are decision-independent (exogenous uncertainties). In stochastic models, uncertain parameters are explicitly represented by a set of scenarios. Each scenario corresponds to one possible realization of the uncertain parameters according to a discretized probability distribution. The goal of such schemes is to optimize the expected value of the objective function over the full set of scenarios, subject to the implementation of common decisions at the beginning of the planning horizon.

Stochastic programs are often difficult to solve due to their large size and complexity that grows with the number of scenarios. To overcome those problems, decomposition algorithms such as Benders decomposition [59], Lagrangean decomposition [19], and Progressive Hedging [50], have been developed to solve linear programming (LP) and mixed-integer linear programming (MILP) stochastic problems. Moreover, several modeling systems and optimization platforms have included extensions for an adequate algebraic representation of stochastic problems, as offered by major software vendors such as GAMS, LINDO, XpressMP, AIMMS, and Maximal.

In recent years, diverse commercial and open-source applications have been developed specifically to represent and solve multistage SP problems. Some of them include capabilities to read and build stochastic MPS (SMPS) files, the standard exchange format for SP applications. However, despite advances in the field and proven benefits, SP has not been widely used in industrial applications. In this paper, we review the current state-of-art of available methods and software for the solution of two-stage stochastic programming problems and evaluate their performance, using large-scale test libraries in SMPS format.

The remainder of this paper is organized as follows. In Section 2, we explain the mathematical formulation of (mixed-integer) linear stochastic problems. Section 3 describes the classical L-shaped algorithm. Section 4 summarizes enhancement strategies to improve the performance of Benders decomposition. Section 5 describes scenario decomposition methods and algorithmic modifications. In section 6, we present the software packages for Benders decomposition and show some computational results. In section 7, we describe algorithmic innovations in software packages for dual decomposition and show some computational results. Finally, in Section 8 we summarize our conclusions.

2 Problem Statement

We consider a two-stage stochastic mixed-integer linear problem (P) in the following form:

$$(P) \quad \min_{x,y} \; TC = c^T x + \sum_{\omega \in \Omega} \tau_\omega d_\omega^T y_\omega \tag{1a}$$

$$\text{s.t.} \quad Ax \le b \tag{1b}$$

$$x \in X, \quad X = \big\{x : x_i \in \{0,1\} \;\; \forall i \in I_1, \;\; x_i \ge 0 \;\; \forall i \in I \backslash I_1\big\} \tag{1c}$$

$$W_\omega y_\omega \le h_\omega - T_\omega x \quad \forall \omega \in \Omega \tag{1d}$$

$$y_\omega \in Y \quad \forall \omega \in \Omega \tag{1e}$$

where x denotes the 'here and now' decisions, taken at the beginning of the planning horizon before the uncertainties unfold, and Ω is the set of scenarios. Vector y_ω represents the recourse or corrective actions (wait and see), applied after the realization of the uncertainty. Matrix $A \in \mathbb{R}^{m_1 \times n_1}$ and vector $b \in \mathbb{R}^{m_1}$ represent the first-stage constraints. Matrices T_ω and W_ω, and vector $h_\omega \in \mathbb{R}^{m_2}$, represent the second-stage problem. Matrices $T_\omega \in \mathbb{R}^{m_2 \times n_1}$ and $W_\omega \in \mathbb{R}^{m_2 \times n_2}$ are called technological and recourse matrices, respectively. Let $I = \{1, 2, \cdots, n_1\}$ be the index set of all first-stage variables. Set $I_1 \subseteq I$ is the subset of indices for binary first-stage variables. Let $J = \{1, 2, \cdots, n_2\}$ be the index set of all second-stage variables. If the second-stage variables are mixed-integer, $Y = \big\{y : y_j \in \{0,1\}, \forall j \in J_1, \;\; y_j \ge 0 \;\; \forall j \in J \backslash J_1\big\}$, where set $J_1 \subseteq J$ is the subset of indices for binary second-stage variables. If all the second-stage variables are continuous, set $J_1 = \emptyset$ and $Y = \big\{y : y_j \ge 0 \;\; \forall j \in J\big\}$. The objective function (TC) minimizes the total expected cost with the scenario probability τ_ω, and the cost vectors c and d_ω. Equation (1) is often referred to as the deterministic equivalent, or extensive form of the SP.

Formulation (P) can be rewritten in an equivalent form (PNAC) with nonanticipativity constraints (NACs), where the first-stage variables are no longer shared, and each scenario represents an instance of a deterministic problem with a specific realization outcome [9, 52].

$$(PNAC) \quad \min_{x_\omega, y_\omega} \; TC = \sum_{\omega \in \Omega} \tau_\omega (c^T x_\omega + d_\omega^T y_\omega) \tag{2a}$$

$$s.t. \quad \sum_{\omega \in \Omega} H_\omega x_\omega = 0 \tag{2b}$$

$$(x_\omega, y_\omega) \in G_\omega \quad \forall \omega \in \Omega \tag{2c}$$

In equation (2c) , G_ω represents the feasible region for scenario ω, which is defined by constraints (1b)-(1e). Nonanticipativity constraints (2b) are added to ensure that the first-stage decisions are the same across all scenarios. Nonanticipativity constraints are represented by suitable sequence of matrices $H_\omega \in \mathrm{R}^{n_1 \cdot (|\Omega|-1) \times n_1}$. One example of such constraints is the following:

$$x_\omega = x_{\omega-1} \quad \forall \omega = 2, 3, ..., |\Omega| \tag{3}$$

Given the mathematical structure of the deterministic equivalent formulations (P) and (PNAC), (mixed-integer) linear stochastic problems can be solved using decomposition methods derived from duality theory [38]. Such methods split the deterministic equivalent into a master problem and a series of smaller subproblems to decentralize the overall computational burden. Decomposition methodologies are classified in two groups: (i) time-stage or *vertical* decomposition which includes Benders decomposition and variants, and, (ii) scenario-based or *horizontal* decomposition. In the following section 3, we provide a tutorial overview of Benders decomposition. In section 4, we also provide a tutorial review of scenario decomposition methods, including the dual decomposition algorithm and the progressive hedging algorithm.

3 L-shaped Algorithm / Benders Decomposition

If the second stage variables are all continuous (i.e, $Y = \{y : y_j \geq 0 \ \forall j \in J\}$), problem (P) can be solved with Benders decomposition. Benders decomposition (BD) was originally developed in 1962 by Benders [4] to solve large-scale mixed integer linear problems (MILP) with complicating variables. This concept has been extended to solve a broader range of optimization problems [18], including multistage, bilevel, and nonlinear programming. When applied to stochastic problems, it is commonly referred to as the L-shaped algorithm [59].

The L-shaped algorithm partitions the deterministic formulation (P) into multiple problems according to the time structure of the stochastic model: (i) a master problem (MP) that contains all the first-stage constraints and variables, which can be mixed-integer; and, (ii) a collection of subproblems that include corrective future actions for the given first-stage solution. The master problem (MP) is derived from the projection of (P) on variables x:

$$(MP) \quad \min_{x} \ TC = c^T x + Q(x) \tag{4a}$$

$$\text{s.t.} \quad Ax \leq b \tag{4b}$$

$$x \in X \tag{4c}$$

where $Q(x) = \sum_{\omega \in \Omega} \tau_\omega \theta_\omega(x)$ is defined as the recourse function (or expected second-stage value function); and $\theta_\omega(x)$ is defined by the primal second-stage program for scenario ω, $(BSPp_\omega)$:

$$(BSPp_\omega) \quad \theta_\omega(x) = \min_{y_\omega} \ d_\omega^T y_\omega \tag{5a}$$

$$\text{s.t} \quad W_\omega y_\omega \leq h_\omega - T_\omega x \tag{5b}$$

$$y_\omega \geq 0 \tag{5c}$$

Recourse functions $\theta_\omega(x)$ and $Q(x)$ are convex, differentiable, and piecewise linear, characteristics that are exploited in the BD method [5]. These conditions do not hold when integer variables are included in the second-stage program. For the case of integer recourse, a logic-based Benders framework [28], second-stage convexification techniques [16, 56, 67], specialized branch-and-bound schemes [3, 48] or dual decomposition methods [52] may be applied to solve large stochastic problems. In this section, we only focus on Benders decomposition for SP with continuous second-stage variables.

Formulation $(BSPp_\omega)$ is a linear program for any given feasible value of x. By the strong duality theorem, the second-stage program is equivalent to its dual $(BSPd_\omega)$, if $(BSPp_\omega)$ is feasible and bounded. Vector π_s represents the Lagrangean multipliers associated with the second-stage constraints given by Eq.(5b):

$$(BSPd_\omega) \quad \theta_\omega(x) = \max_{\pi_\omega} \; (h_\omega - T_\omega x)^T \pi_\omega \tag{6a}$$

$$\text{s.t.} \quad W_\omega^T \pi_\omega \leq d_\omega \tag{6b}$$

$$\pi_\omega \geq 0 \tag{6c}$$

BD introduces a set of piece-wise linear approximations of the recourse function in the problem MP, known as optimality cuts, which are built from dual solutions of the second-stage program. It is important to highlight that the dual feasible region does not depend on the value of x. Thus, the exact representation of the expected cost consists of the computation of all the extreme points of problems $(BSPd_\omega)$.

However, the second-stage program may not be feasible for some values of x. BD enforces second-stage constraints (5b) by adding feasibility cuts, which are valid inequalities that exclude infeasible first-stage solutions from the MP. Subproblem feasibility is evaluated by solving the following recourse reformulation for scenario ω:

$$V_\omega(x) = \min_{y_\omega, v^+, v^-} \quad e^T v^+ + e^T v^- \tag{7a}$$

$$\text{s.t.} \quad W_\omega y_\omega + v^+ - v^- \leq h_\omega - T_\omega x \tag{7b}$$

$$v^+ \geq 0 \quad v^- \geq 0, \quad y_\omega \geq 0 \tag{7c}$$

where $e \in \mathbb{R}^{m_2}$ is a vector with all-1 entries, and $v^+ \in \mathbb{R}^{m_2}$ and $v^- \in \mathbb{R}^{m_2}$ are the positive and negative slack of constraints (5b), respectively. The objective function $V_\omega(x)$ measures the amount by which these constraints are violated; thus, if $V_\omega(x)$ equals zero, it implies that the original subproblem (5) is feasible. To derive feasibility cuts in terms of x, BD considers the dual of problem (7) to generate an expression equivalent to Eq (7a). The optimal solution $\mu \in \mathbb{R}^{m_2}$ of the dual feasibility problem (8) corresponds to one of the extreme rays (or directions) of the recourse subproblem (6):

$$V_\omega(x) = \max_{\mu} \; (h_\omega - T_\omega x)^T \mu \tag{8a}$$

$$\text{s.t.} \quad W_\omega^T \mu \leq 0 \tag{8b}$$

$$-e \leq \mu \leq e \tag{8c}$$

The master problem (4) is linearized by: (i) substituting function $Q(x)$ with the weighted sum of the future cost estimation (6a), and (ii) applying feasibility cuts as needed. This reformulation is referred to as the multi-cut Benders master problem (BMP):

$$(BMP) \quad TC_d = \min_{x,\theta_\omega} \; c^T x + \sum_{\omega \in \Omega} \tau_\omega \theta_\omega \tag{9a}$$

$$\text{s.t.} \quad Ax \leq b, \;\; x \in X \tag{9b}$$

$$(h_j - T_j x)^T \bar{\mu}_j \leq 0 \quad \forall j \in E \tag{9c}$$

$$(h_\omega - T_\omega x)^T \bar{\pi}^k_\omega \leq \theta_\omega \quad \forall \omega \in \Omega, \quad k \in K \tag{9d}$$

where variables $\theta_\omega \in \mathbb{R}^{|\Omega|}$ represent the outer linearization of the second-stage cost $\theta_\omega(x)$. Parameters $\bar{\pi}^k_\omega$ and $\bar{\mu}_j$ represent the extreme points and rays from the dual form of the recourse program $(BSPd_\omega)$, which are stored in sets E and K, respectively. Constraints (9c) and (9d) denote the *feasibility* and *optimality* cuts, $j \in E$ and $k \in K$ respectively. Matrices h_j and T_j correspond to the matrices h_ω and T_ω for the scenario where a feasibility cut can be found.

The complete enumeration of the extreme points and rays of the dual second-stage program is impractical, if not impossible. Instead, the L-shaped algorithm relaxes the MP by initially considering a subset of the optimality and feasibility cuts. Iteratively, BD solves the relaxed problem to generate a candidate solution for the first-stage variables $(\bar{x})$ and then solves the collection of scenarios subproblems at fixed $\bar{x}$ to generate a new group of optimality or feasibility cuts. This process is repeated until the optimal solution is found [48].

The optimal solution of the relaxed Benders Master Problem provides a valid lower estimation (TC_d) of the optimal total cost TC. On the other hand, the solution of the second-stage programs $(BSPd_\omega)$ at feasible $\bar{x}$ yields an upper bound of the original objective function (TC_p), given by Eq. (10). The solution procedure terminates when the difference between the bounds is closed, as implied by Eq. (11). Algorithm 1 summarizes the procedure.

$$TC_p(\bar{x}) = c^T \bar{x} + \sum_{\omega \in \Omega} \tau_\omega \theta_\omega(\bar{x}) \tag{10}$$

$$TC_d \leq TC \leq TC_p \tag{11}$$

The L-Shapped method is summarized in Algorithm (1). It is initialized with a guess of the first-stage solution x_o and considers two stopping criteria: (i) is the optimality tolerance ϵ that limits the relative gap between the dual (z_{LB}) and primal (z_{UB}) bounds of the objective function (TC), and (ii) is the maximum number of allowed iterations (k_{max}).

Algorithm 1: Multi-cut Benders Decomposition

```
1  Set $k \longleftarrow 0$, $z_{LB} \longleftarrow -\infty$, $z_{UB} \longleftarrow \infty$, $x^k \longleftarrow x_o$ and $\epsilon > 0$
2  while $k < k_{max}$ do
3      SOLVE (6) to obtain $\theta_\omega(x^k)$ and $\pi^k_\omega$ for given $x^k$ for all $\omega \in \Omega$
4      if all subproblems (6) are feasible then
5          ADD new optimality cuts (9d) corresponding to $\pi^k_\omega$ for all $\omega \in \Omega$
6          compute $TC_p$ from $\theta(x^k)$ and $x^k$
7          if $TC_p < z_{UB}$ then
8              $z_{UB} \longleftarrow TC_p$ (upper bound)
9              $x^* \longleftarrow x^k$
10     else
11         SOLVE (8) to obtain $\mu$ for given $x^j$ and infeasible scenario $j \in \Omega$
12         ADD new feasibility cut (9c) corresponding to $\mu$
13     SOLVE (9) to obtain $(x^{k+1}, \theta^{k+1}_\omega)$ and $TC_d$
14     if $z_{LB} < TC_d$ then
15         $z_{LB} \longleftarrow TC_d$ (lower bound)
16     if $(z_{UB} - z_{LB})/\big(\max(|z_{UB}|, |z_{LB}|) + 1e - 10\big) < \epsilon$ then
17         break
18     Set $k \longleftarrow k + 1$
19 return optimal solution $x^*$ and $z_{LB}$
```

4 Benders Decomposition Enhancement Strategies

The application of BD often leads to slow convergence, long computational times, and excessive use of memory resources, particularly for the case when the MILP master problem has poor LP relaxation [20, 41, 45]. Major BD disadvantages include: time-consuming iterations, poor feasibility and optimality cuts, ineffective initial iterations; primal solutions that behave erratically, slow convergence at the end of the algorithm (tailing-off effect), and upper bounds that remain stuck in successive iterations due to second-stage degeneracy [48, 60].

Various strategies have been proposed to accelerate the convergence of the standard BD method. Enhancement strategies are mainly split into two categories: reducing the cost of each iteration or reducing the number of iterations [48, 65].

4.1 Reducing the Cost of Each Iteration

Cheaper iterations are achieved by reducing the time spent solving the MP and subproblems. The MP is often the most time-consuming part of the BD algorithm (more than 90% of the execution time), especially in the case of mixed-integer problems [41]. The overall process can be accelerated by relaxing the integrality of the MP in most of the iterations, to rapidly compute a large number of cuts [42]. A variation of this method has been proposed by Geoffrion and Graves [20], in which the MP is solved to a non-zero optimality gap. The integrality gap is continuously reduced to ensure global convergence.

Alternatively, the MP might be solved via (meta) heuristics [10, 49], which provide good approximate solutions in short time; however, it is still required to solve the MP to optimality to guarantee convergence. The application of heuristics or the LP relaxation of the MP often yields worse bounds and lack of controllability, reducing the ability of BD to generate the necessary cuts [27].

Similarly, suboptimal solutions of the dual subproblems yield valid cuts, known as Inexact Cuts. Algorithm convergence can still be guaranteed under the conditions described by Zakeri et al. [64]. Additional subproblem acceleration schemes comprise synchronous parallelization and re-optimization. The latter exploits structural similarities between scenarios to solve the subproblems in fewer solver iterations.

4.2 Reducing the Number of Iterations

The number of iterations of the L-shaped algorithm is closely related to the tightness of the LP relaxation of the first-stage problem, as well as the strength of the optimality and feasibility cuts [41]. Better candidates are computed from improvements in the MP problem, especially, in the strengthening of the representation of the recourse functions. Tighter formulations can be obtained by adding multiple cuts per iteration (multi-cut reformulation [6]); , as well as through the use of heuristics to eliminate inactive cuts and to select the fittest dual variables to be inserted in the MP (size management techniques).

Complementary strategies have been developed to generate cuts that are more efficient. One alternative is the reformulation of the subproblems to select non-dominant dual solutions from the set of optimal multipliers, known as Pareto-optimal cuts [41]. Recently, [55] proposed a methodology to compute bundles of covering cuts, designed to involve most of the first-stage variables and to carry as much information as possible.

Alternative methods tighten the MP to alleviate some of the drawbacks of BD: cross-decomposition, for instance, avoids the generation of low-quality solutions, while quadratic stabilization methods provide a solution for the tailing-off effect. Cross-decomposition [58] combines and alternates between iterations of BD and Lagrangean decomposition, to provide an additional valid lower bound of the objective function and a set of feasible deterministic solutions $(x_\omega, y_\omega) \in G_\omega$, which are used to compute Lagrangean-based cuts to strengthen the MP.

Quadratic methods have been proposed to stabilize BD, aiming to improve the quality of the initial iterations and reduce the oscillation that occurs when the algorithm is close to the optimal solution [65]. These methods encourage the generation of first-stage candidates close to stability centers (the current best solution) while reducing the original objective function value. Popular variants include Regularized Decomposition [52, 53], the Trust-Region method [39] and Level Decomposition [36, 66], which are summarized below:

– **Regularized Decomposition (also known as Proximal Bundle Method)**

$$x_{k+1} = \arg\min_{x,\theta_\omega}\{c^T x + \sum_{\omega\in\Omega} \tau_\omega\theta_\omega + \frac{1}{2t_k}\|x-\hat{x}_k\|_2^2 \quad \text{s.t.} \quad (x,\theta_\omega)\in V_k\} \tag{12}$$

– **Trust-Region Method**

$$x_{k+1} = \arg\min_{x,\theta_\omega}\{c^T x + \sum_{\omega\in\Omega} \tau_\omega\theta_\omega \quad \text{s.t.} \quad \|x-\hat{x}_k\|_2^2 \le R_k, (x,\theta_\omega)\in V_k\} \tag{13}$$

– **Level Decomposition Method**

$$x_{k+1} = \arg\min_{x,\theta_\omega}\{\|x-\hat{x}_k\|_2^2 \quad \text{s.t.} \quad c^T x + \sum_{\omega\in\Omega} \tau_\omega\theta_\omega \le L_k, (x,\theta_\omega)\in V_k\} \tag{14}$$

where t_k, R_k and L_k are real-valued, iteration-dependent parameters that balance the minimization of the relaxed MP and the distance to the stability center $\hat{x}_k$. V_k represents the feasible region of the Benders master problem at each iteration, which is defined by the optimality (9d) and feasibility cuts (9c), as well by the first-stage constraints (9b). Stabilization methods were initially introduced for BD with no integer variables; nonetheless, recent improvements have adapted the method to mixed-integer problems [65].

5 Scenario Decomposition Methods

Scenario decomposition is a popular approach to solve two-stage SP formulations with mixed-integer recourse, i.e., $Y = \{y : y_j \in \{0,1\}, \forall j \in J_1,\ y_j \ge 0\ \forall j \in J\backslash J_1\}$ in (PNAC). In contrast to the BD algorithm, scenario decomposition methods dualize the non-anticipativity constraints (NACs) to obtain lower bounds of the original formulation. Scenario-based decomposition addresses the computational difficulties associated with the solution of large stochastic problems by considering each scenario independently and solving the set of subproblems in parallel. Moreover, feasible solutions to the original problem (P) can be obtained by heuristics based on the optimal solutions of the subproblems. In this section, we describe the dual decomposition (DD) algorithm and the progressive hedging (PH) algorithm.

5.1 Dual Decomposition (DD) Method

The dual decomposition algorithm proposed by Carøe and Schultz [9] applies the Lagrangean relaxation to problem (2) and uses a *branch-and-bound* procedure to restore the non-anticipativity conditions. The Lagrangean relaxation of the NACs results in the following dual function:

$$D(\lambda) = \min_{x,y} \sum_{\omega\in\Omega} L_\omega(x_\omega, y_\omega, \lambda_\omega) \tag{15a}$$

$$\text{s.t.} \quad (x_\omega, y_\omega) \in G_\omega \quad \forall \omega \in \Omega \tag{15b}$$

where

$$L_\omega(x_\omega, y_\omega, \lambda_\omega) = \tau_\omega(c^T x_\omega + d_\omega^T y_\omega) + \lambda_\omega^T H_\omega x_\omega \tag{16}$$

In the equation (15a), vector $\lambda \in \mathbb{R}^{n_1 \times (|\Omega|-1)}$ represents the dual multipliers associated with the NACs (2b). $\lambda_\omega \in \mathbb{R}^{n_1}$ represents the Lagrangean multipliers for the NACs associated with scenario ω, as given by Eq. (3). Given the independence of the variables and constraints in each scenario, function D can be split into separate subproblems $D_\omega(\lambda_\omega)$:

$$D(\lambda) = \sum_{\omega \in \Omega} D_\omega(\lambda_\omega) \tag{17a}$$

$$D_\omega(\lambda_\omega) = \{\min_{x_\omega, y_\omega} \quad L_\omega(x_\omega, y_\omega, \lambda_\omega) \quad \text{s.t.} \quad (x_\omega, y_\omega) \in G_\omega\} \tag{17b}$$

According to the weak duality theorem, the relaxation (17) is always less than or equal to the optimal objective value of problem (2). The best lower bound of (PNAC) is computed by solving the following maximization problem, referred to as the Lagrangean dual problem:

$$Z_{LD} = \max_\lambda D(\lambda) \tag{18}$$

The Lagrangean dual is a concave non-smooth program and can be solved by subgradient methods, cutting-plane methods, or column generation methods. The details of these methods can be found in Guignard [22]. We illustrate the dual search approaches by describing the standard cutting-plane algorithm.

5.1.1 Cutting-Plane Method

The cutting-plane algorithm solves the Lagrangean problem iteratively by implementing outer approximation on (18) and solving the Lagrangean subproblems (17b) to improve the formulation of the relaxed dual function (RZ_{LD}) in equation (19a). The outer approximation is given by the Lagrangean master problem (LMP):

$$(LMP) \quad RZ_{LD} = \max_{\lambda_\omega, \phi_\omega} \sum_{\omega \in \Omega} \phi_\omega \tag{19a}$$

$$\text{s.t.} \quad \phi_\omega \leq \bar{D}_\omega^k(\lambda_\omega^k) + (H_\omega x_\omega^k)^T(\lambda_\omega - \lambda_\omega^k) \quad \forall k \in K, \omega \in \Omega \tag{19b}$$

where parameters for iteration k and scenario ω, x_ω^k and $\bar{D}_\omega^k(\lambda_\omega^k)$ represent the previous solution of subproblem (17b), and parameter λ_ω^k represents the vector of previously considered dual multipliers. The dual search is outlined in Algorithm (2).

Cutting-plane methods present similar drawbacks to the BD algorithm, such as slow convergence and strong oscillations of the dual variables. Various alternatives have been proposed to accelerate this technique, including the

Algorithm 2: Cutting-plane dual search

```
1 Set $k \longleftarrow 0$, $z_{LB} \longleftarrow -\infty$ and $\lambda^0 \longleftarrow 0$
2 repeat
3     SOLVE (17b) to obtain $(x^k_\omega, y^k_\omega)$ and $D_\omega(\lambda^k_\omega)$ for given $\lambda^k_\omega$ for each $\omega \in \Omega$
4     set $z_{LB} \longleftarrow \max\{z_{LB}, D(\lambda^k)\}$
5     ADD new optimality cut (19b) from $x^k_\omega$ and $D_\omega(\lambda^k_\omega)$
6     SOLVE (19) to obtain $\lambda^{k+1}$ and $RZ_{LD}$
7     set $k \longleftarrow k+1$
8 until $|D(\lambda^k) - RZ_{LD}|/|D(\lambda^k) + 1e-10| < \epsilon$;
9 return $x^k_\omega, \lambda^k, D(\lambda^k)$
```

bundle method and the volume algorithm [22]. Additional strategies consider the suboptimal solution of the master problem, using an interior-point method (IPM) in combination with Benders-like cuts to tighten the Lagrangean subproblems (17b) and exclude infeasible first-stage solutions (see [33, 45]). Other methodologies such as cross-decomposition, exchange information with BD to compute additional cuts derived from feasible first-stage candidates [44].

5.1.2 Branch-and-Bound Method

The DD algorithm proposed by Carøe and Schultz [9] uses the bound Z_{LD} as bounding the criterion to discard nodes from the first-stage search domain. Algorithm 3 summarizes the branch-and-bound procedure. The set $\mathcal{P}$ denotes the group of active problems and TC^i the lower bound associated with program $\mathcal{P}_i \in \mathcal{P}$. Commonly, the Lagrangean dual problem yields first-stage solutions that differ in value from one scenario to another. For those instances, a candidate $\hat{x}$ is estimated by applying a rounding heuristic on the average solution $\sum_{\omega\in\Omega} \tau_\omega x^i_\omega$. Note that Algorithm 3 can be applied not only to problems with mixed-binary variables but to problems with general mixed-integer variables as well. The branching steps assume that the integer variables can be nonbinary.

5.2 Progressive Hedging (PH) Algorithm

The Progressive Hedging (PH) algorithm [50] is a popular approximation for solving multi-stage stochastic programs. Although it was initially proposed for convex stochastic problems, it has been successfully applied as a heuristic to solve mixed-integer stochastic programs [40, 61].

To find a solution of problem (2), PH aggregates a new set of variables $\hat{x}$ (also known as a first-stage policy) that replaces the NACs (2b). Then, it solves the reformulated program (20) using a specialized variant of the alternating direction method of multipliers (ADMM) [11, 14]:

$$\min_{x_\omega, y_\omega, \hat{x}} \quad TC = \sum_{\omega\in\Omega} (c^T x_\omega + d^T_\omega y_\omega) \tag{20a}$$

Algorithm 3: DD Branch and Bound method

```
1  Set $NAC \longleftarrow$ **false**, $z_{UB}, \longleftarrow \infty$, $\mathcal{P} = \{PNAC\}$
2  **while** $|\mathcal{P}| > 0$ **do**
       /* Lower bounding procedure */
3      Select problem $\mathcal{P}_i$ from $\mathcal{P}$ and SOLVE (18) to get the lower bound $Z^i_{LD}$ and $x^i_\omega$
4      Eliminte problem $\mathcal{P}_i$ from $\mathcal{P}$
5      **if** $Z^i_{LD} = -\infty$ *(infeasibility of a subproblem) or* $Z^i_{LD} \geq z_{UB}$ **then**
6          go to line 2
7      **else if** $\sum_{\omega \in \Omega} H_\omega x^i_\omega = 0$ **then**
8          $\hat{x}^i \longleftarrow x^i_j$ for any $j \in \Omega$
9          $NAC \longleftarrow$ **true**
10     **else**
11         $NAC \longleftarrow$ **false**
12         perform rounding heuristic to obtain $\hat{x}^i$
       /* Upper bounding procedure */
13     Compute $TC^i_p$ from $\hat{x}^i$ using equation (10)
14     **if** $TC^i_p < z_{UB}$ **then**
15         $z_{UB} \longleftarrow TC^i_p$
16         $x^* \longleftarrow \hat{x}^i$
17         eliminate from $\mathcal{P}$ all the problems $\mathcal{P}_j$ with $Z^j_{LD} \geq z_{UB}$
18     **else**
19         go to line 2
       /* Branching procedure */
20     **if** *not* $NAC$ **then**
21         Select a component $x_{(k)}$ of $x$ and add two new problems to $\mathcal{P}$ by adding
             constraints:
22         $x_{\omega,(k)} \leq \hat{x}^i_{(k)} - \delta$ and $x_{\omega,(k)} \geq \hat{x}^i_{(k)} + \delta$    for all $\omega$ in $\Omega$ (if $x_{(k)}$ is
             continuous)
23         $x_{\omega,(k)} \leq \lfloor \hat{x}^i_{(k)} \rfloor$ and $x_{\omega,(k)} \geq \lceil \hat{x}^i_{(k)} \rceil$    for all $\omega$ in $\Omega$ (if $x_{(k)}$ is integer)
24     **return** *Optimal solution* $x^*$ *and* $z_{UB}$
```

$$\text{s.t. } (x_\omega, y_\omega) \in G_\omega, \;\; x_\omega = \hat{x}, \;\; \forall \omega \in \Omega, \;\; \hat{x} \in X \tag{20b}$$

Related to dual decomposition, PH relaxes the non-anticipativity restrictions on the first-stage. The augmented Lagrangean relaxation L^ρ of constraints $x_\omega = \hat{x}, \forall \omega \in \Omega$ yields a lower bound $D(\lambda)$ of the original deterministic formulation (20). The best lower bound is estimated by solving the following problem:

$$TC \geq \max_\lambda \{D(\lambda) \;\text{ s.t. } \sum_{\omega \in \Omega} \tau_\omega \lambda_\omega = 0\} \tag{21a}$$

where

$$D(\lambda) = \min_{x,\hat{x},y} \; L^\rho(x, \hat{x}, y, \lambda) \;\text{ s.t. } (x_\omega, y_\omega) \in G_\omega \;\; \forall \omega \in \Omega, \;\; \hat{x} \in X \tag{21b}$$

$$L^\rho(x, y, \hat{x}, \lambda) = \sum_{\omega \in \Omega} \tau_\omega L_\omega(x_\omega, y_\omega, \hat{x}, \lambda_\omega) \tag{21c}$$

$$L_\omega(x_\omega, y_\omega, \hat{x}, \lambda_\omega) = c^T x_\omega + d_\omega^T y_\omega + \lambda^T(x_\omega - \hat{x}) + \rho/2\|x - \hat{x}\|_2^2 \tag{21d}$$

and $\rho > 0$ is a penalty parameter. Constraints $\sum_{\omega\in\Omega} \tau_\omega \lambda_\omega = 0$ are required to make L^ρ bounded from below. To mitigate the computational difficulties of minimizing the augmented Lagrangean dual function (21b), PH decomposes the problem by scenarios. To achieve the complete separability of subproblems, Rockafellar and Wets [50] propose to fix the first-stage policy temporarily, and repeatedly solve the program (22) to update the multipliers and the value of $\hat{x}$:

$$\min_{x_\omega, y_\omega} \{c^T x_\omega + d_\omega^T y_\omega + \lambda^T x_\omega + \rho/2\|x_\omega - \hat{x}\|_2^2\} \tag{22}$$

Algorithm (4) summarizes the procedure to solve the dual problem (21).

Algorithm 4: Two-Stage Progressive Hedging Algorithm

```
1 set $k \longleftarrow 0$, $\lambda^0 = 0$
2 SOLVE $(x_\omega^1, y_\omega^1) = \arg\min_{x_\omega, y_\omega}\{c^T x_\omega + d_\omega^T y_\omega \quad s.t \quad (x_\omega, y_\omega) \in G_\omega\}$ for all $\omega$ in $\Omega$
3 repeat
4     set $k \longleftarrow k + 1$
5     set $\hat{x}^k \longleftarrow \sum_{\omega\in\Omega} \tau_\omega x_\omega^k$
6     set $\lambda_\omega^k \longleftarrow \lambda_\omega^{k-1} + \rho(x_\omega^{k-1} - \hat{x}^{k-1})$
7     SOLVE (22) for every $\omega \in \Omega$ to compute $x_\omega^{k+1}$
8 until $k > k_{max}$ or $\sqrt{\sum_{\omega\in\Omega} \tau_\omega (x_\omega^{k+1} - \hat{x})^2} < \epsilon$;
9 return $\hat{x}^k$, $x_\omega^{k+1}, y_\omega^{k+1}$
```

The termination of the algorithm is achieved when the first-stage policy is non-ancipative. In the case of convex instances, $\hat{x}^{k\longrightarrow\infty}$ is equivalent to the optimal solution of the deterministic formulation (2) and the convergence is guaranteed. These conditions do not hold for mixed-integer programs; however, a high-quality solution and upper bound can be computed from a non-convergent value of $\{\hat{x}^k\}_{k=k_{max}}$ and $TC_p(\hat{x}^k)_{k=k_{max}}$, respectively [61].

Recent investigations have focused on the improvement and acceleration of PH. Various studies identify the penalty term as a critical factor in the quality of the solution and the convergence rate: larger values of ρ can accelerate the convergence but can lead to suboptimal solutions. On the other hand, lower values can improve the quality of the solutions and lower bounds, although with a very slow convergence rate [15]. Numerous alternatives have been developed to circumvent those problems, from *per-component* and *cost-proportional* heuristics [61], to the dynamic update of the penalty parameter [21, 66].

A limitation in applying PH to stochastic mixed-integer programs is the lack of a lower bound to assess the quality of the computed solution. This disadvantage can be alleviated by estimating a valid lower bound from the non-convergent set of Lagrangean weights λ_k [15], or by combining the Frank-Wolfe and PH methods [7]. These methodologies establish relationship between dual decomposition and progressive hedging, which has motivated the development of hybrid solution strategies (see [23]).

6 Software packages for Benders Decomposition

In this section, we review two software packages, GAMS - DECIS [8, 29] and FORTSP [12]m for Benders Decomposition. Both packages are benchmarked with 20 instances from the Random [30] and SAPHIR [35] test collections, which are some of the largest instances found in the literature. All of the test problems are available in the SMPS format; however, specific modifications need to be done in order to make the format compatible with DECIS. The computational experiments are performed on a Linux machine with a 2.67 GHz Intel Xeon CPU, 128 GB of RAM, and a limit of 3 hours of walltime.

The Random collection consists of 15 instances artificially generated with the test problem generator GENSLP [32]. The instances are grouped into 3 sets of problems (rand0, rand1, rand2), each one of them having 5 instances with 2000, 4000, 6000, 8000 and 10000 scenarios. None of the instances represent a real-world problem; nonetheless, they have been successfully used to assess the performance of stochastic solvers (see [68]). All problems in this collection present uncertainty only in the right-hand side (RHS) coefficients h_ω.

The SAPHIR collection consists of 5 instances of the optimization of a gas-purchase portfolio, considering the cost of purchase, as well as underground storage capacities and transportation, under uncertain demand conditions [34]. In this family of problems, the random elements are located in both the RHS and constraint matrices W_ω and T_ω.

The sizes of all of the test problems are shown in Table 1. The size is expressed as the number of constraints (Rows) and variables (Cols) in the first stage and the second stage per scenario. None of the test instances consider integer variables in the first-stage.

Table 1 Sizes of SLP instances tested

Name	Scenarios	First Stage		Second Stage	
		Rows	Cols	Rows	Cols
rand0	2000, 4000, 6000, 8000, 10000	50	100	25	50
rand1	2000, 4000, 6000, 8000, 10000	100	200	50	100
rand2	2000, 4000, 6000, 8000, 10000	150	300	75	150
saphir	50, 100, 200, 500, 1000	32	53	8678	3924

6.1 FortSP: a stochasting programing solver

FortSP is a solver for the solution of linear and mixed-integer linear stochastic programs. It accepts input in the SMPS format, or through a separate

SAMPL translator (an AMPL extension for stochastic programming). In addition, FortSP can be used as a library with an application programming interface (API) in C. FortSP enables the user to solve stochastic two-stage linear programs with 4 variants of Benders decomposition, and provides 3 different solution approximations for mixed-integer instances.

6.1.1 Algorithmic Innovations in FortSP

The innovations in FortSP for two-stage linear and mixed-integer linear stochastic programs are described by Ellison et al. [12]. FortSP incorporates 5 methods to solve two-stage stochastic linear programs: (i) solving the deterministic equivalent via the interior-point method (IMP), (ii) Benders decomposition with aggregated cuts (see problem (23)), (iii) Regularized decomposition [51] (see problem (12)), (iv) Benders decomposition with regularization of the expected recourse by the level method [36] (see problem (14)), and (v) the Trust region (regularization) method [39] (see problem (13)).

To solve mixed-integer instances, FortSP uses the deterministic equivalent with both implicit and explicit representations for the NACs. In addition, it incorporates a specialized L-shaped algorithm based on branch-and-cut for instances with mixed-integer variables in the first-stage and continuous and complete recourse. This method might be accelerated with the Variable Neighborhood Decomposition Search heuristic (VNDS) [25].

All of the Benders variants in FortSP are formulated in the *aggregated* form shown in Eq (23). *Disagregated* formulations (i.e, problem (9)) store larger information in the master problem, which yields a reduction in the number of iterations. However, this is done at the expense of larger master problems. As a rule of thumb, the *disaggregated* approach is expected to be more effective when the number of scenarios $|\Omega|$ is not significantly larger than the number of constraints m_1 of the first-stage program [5].

$$(BMP) \quad TC_d = \min_{x,v} c^T x + v \tag{23a}$$

$$\text{s.t. } Ax \le b, \ \ x \in X, \ \ v \in \mathbb{R} \tag{23b}$$

$$(h_j - T_j x)^T \bar{\mu}_j \le 0 \quad \forall j \in E \tag{23c}$$

$$\sum_{\omega \in \Omega} \tau_\omega (h_\omega - T_\omega x)^T \pi_\omega^k \le v \quad \forall k \in K \tag{23d}$$

6.1.2 Computational results for FortSP

We use FortSP to solve the Random [30] and SAPHIR [35] test instances. The number of iterations and walltime for different solution methodologies are shown in Table 2, where IPM stands for Interior-Point Method, RD for Regularized Decomposition, and, TR for Trust Region. The CPLEX (12.5.1) linear and quadratic solver is used to solve the set of master problem and subproblems. For decomposition methodologies, a stopping optimality gap of

1×10^{-5} is used. FortSP automatically selects the methodology used to solve the set of master problem and recourse instances, from primal and dual simplex, as well as an interior-point method. In addition, FortSP considers the warm-start of linear programs.

From Table 2, one can observe that solving the deterministic equivalent via IPM is an effective alternative, outperforming BD in most of the instances considered; nonetheless, it fails to solve the larger instances in the Saphir set. Regularized Decomposition and the Trust Region method perform better than BD in the Saphir set, effectively decreasing the number of iterations and the solution time. However, RD fails on the whole set of RAND test problems. Decomposition with the Level Method presents the best performance on both of the benchmark sets, yielding computational times close to the interior-point method and effectively reducing the number iterations of the standard BD method.

Table 2 Computational results for FortSP

Instances	Scenarios	IPM		Benders		Level		RD		TR	
		Iter	Time	Iter	Time	Iter	Time	Iter	Time	Iter	Time
rand0	2000	128	38.21	80	10.57	44	7.53	-	-	103	13.56
	4000	46	26.18	69	20.02	32	11.50	-	-	84	24.60
	6000	57	46.30	108	41.10	51	21.53	-	-	136	51.36
	8000	64	66.28	127	65.34	50	34.00	-	-	159	81.33
	10000	80	95.32	230	153.99	71	53.39	-	-	311	207.46
rand1	2000	37	34.74	391	237.40	74	52.86	-	-	502	307.87
	4000	46	79.92	502	528.99	59	69.90	-	-	624	655.29
	6000	47	116.40	385	576.33	58	94.25	-	-	484	728.86
	8000	50	160.58	453	818.78	65	126.08	-	-	611	1126.22
	10000	51	414.21	430	1064.25	52	526.53	-	-	558	1388.47
rand2	2000	36	63.78	886	1643.40	65	133.59	-	-	1239	2415.88
	4000	40	140.56	414	1355.37	42	152.27	-	-	573	1936.61
	6000	48	245.89	514	3067.92	52	318.58	-	-	675	4172.58
	8000	51	329.10	454	3036.40	44	310.44	-	-	681	4638.54
	10000	51	418.11	686	6774.75	52	528.81	-	-	988	9733.37
Saphir	50	-	-	127	527.06	39	215.72	22	82.30	33	77.18
	100	-	-	122	768.42	44	503.87	29	216.37	34	97.01
	200	-	-	-	-	-	-	30	163.66	19	84.15
	500	326	555.35	122	847.17	42	426.28	29	231.10	25	85.62
	1000	-	-	138	1153.40	51	655.66	29	259.29	86	289.53

6.2 DECIS: A system for solving large-scale stochastic programs

DECIS is a software platform for the solution of large-scale two-stage stochastic programs. It accepts problems in SMPS format. To use DECIS in GAMS, the user needs to formulate the deterministic problem and time distribution of the constraints and variables in the GAMS interface, which automatically constructs the *core* and *tim* files. The uncertain components and realization probabilities are set from an external *stochastic* file (.sto extension in SMPS format), which is written by the user. Recent developments in GAMS allow to use the Extended Mathematical Programming (EMP) framework to define a stochastic program for DECIS, as well as set the optimization of two additional risk measures: Value at Risk (VaR) and Conditional Value at Risk (CVaR).

6.2.1 Algorithmic innovations in DECIS

DECIS incorporates multiple alternatives to solve linear two-stage stochastic programs, including: (i) Benders decomposition with aggregated cuts, and, (ii) a regularized decomposition variant. The latter uses MINOS to solve the quadratic master problem (12), and requires the user to select a proper constant penalty parameter ($t_k > 0$). The overall algorithm performance and convergence are strongly affected by the value of t_k.

When the number of realizations is large, DECIS can employ advanced Monte Carlo sampling techniques to compute good approximate solutions. Instead of considering the whole set of possible outcomes to estimate the expected cost, DECIS uses an independent sample drawn from the distribution of random parameters. In addition to crude Monte Carlo sampling, DECIS incorporates importance sampling and control variates, variance reduction techniques which enhance the estimation of the expected cost. In addition, DECIS computes a confidence interval in which the optimal objective function value lies.

6.2.2 Computational results for DECIS

We use DECIS to solve the RANDOM and SAPHIR test instances. The number of iterations and walltime for different solution methodologies are shown in Table 3. Two initialization strategies are tested on Benders Decomposition: (U) where the initial first-stage candidate solution is 0, and (EV+U) where BD is employed to solve the EV (expected value) problem. The EV optimal solution is then used as a starting point for the stochastic instance. Iter-EV and Iter-U stand for the number of iterations required to solve the EV and stochastic problem, respectively. A stopping optimality gap of 1×10^{-5} is considered. DECIS-CPLEX (12.7.0) uses primal simplex in both the MP and subproblems in Benders decomposition. DECIS-MINOS (5.6) is used in the quadratic MP and linear subproblems in Regularized decomposition.

To exemplify the effects of the constant penalty term on the performance of regularized decomposition, two ρ values, 1 and 10, are tested. From Table 3, it can be observed that regularized decomposition may significantly reduce the number of iterations, and thus the solution time of the overall decomposition algorithm. In addition, stronger penalization might increase the number of iterations as it restricts the movement of first-stage candidate to be close to the best incumbent solution. Furthermore, this methodology might present numerical issues such as bad scaling in the master problem, which makes the algorithm stop without closing the optimality gap. For instance, regularized decomposition fails to solve the whole set of SAPHIR problems.

Using the (EV+U) initialization can accelerate the convergence of Benders Decomposition. In 14 of 17 instances where BD converged, (EV+U) had fewer iterations than the (U) strategy, as well as less solution time. The reduction of the iteration number alleviates the time spent computing an appropriate starting point.

Table 3 Computational results for DECIS

Instances	Scenarios	Benders (U)		Benders (EV+U)			RD - 1 (U)		RD - 10 (U)	
		Iter	Time	Iter-EV	Iter-U	Time	Iter	Time	Iter	Time
rand0	2000	82	29.72	31	77	27.98	50	13.17	72	18.30
	4000	71	53.77	35	58	48.49	42	22.11	58	30.16
	6000	105	112.36	47	106	120.96	58	40.6	85	58.83
	8000	121	170.25	38	111	155.61	59	54.23	102	91.64
	10000	229	410.76	40	213	389.04	110	133.2	135	163.31
rand1	2000	391	459.29	91	384	448.74	120	264.43	255	551.64
	4000	488	1051.82	87	487	1031.35	117	448.65	296	1175.35
	6000	396	1269.56	118	363	1158.85	100	533.02	146	781.95
	8000	443	1763.46	100	436	1688.43	106	679.39	153	1004.85
	10000	449	2356.12	115	437	2353.02	113	983.68	193	1736.57
rand2	2000	885	3213.08	125	870	3225.03	142	1147.62	265	2620.33
	4000	411	2784.49	136	405	2786.91	93	1696.08	212	3879.52
	6000	496	5470.71	165	520	5764.87	132	4196.52	223	6981.10
	8000	457	6151.33	173	459	6277.49	97	3631.94	140	5224.19
	10000	-	-	-	-	-	-	-	-	-
Saphir	50	167	362.21	163	80	317.21	-	-	-	-
	100	151	568.44	151	83	539.73	-	-	-	-
	200	-	-	-	-	-	-	-	-	-
	500	138	1357.83	109	73	917.47	-	-	-	-
	1000	-	-	-	-	-	-	-	-	-

6.2.3 Computational results for FortSP in comparison with DECIS

From the results in the previous subsections, it can be observed that the algorithms implemented in FortSP (Table 2) outperforms the decomposition implementations in GAMS - DECIS (Table 3) in terms of solution time. The strength of FortSP resides in the use of multiple strategies that can accelerates the convergence of standard BD algorithm and regularization solved with MINOS. In fact we observed that the best FortSP methodology is at least 37.3% faster than the best algorithmic implementation evaluated with DECIS for each test problem (see Figure 1). In the instances in which none of the DECIS solvers converge, the solution time is noted as 3 hours of walltime.

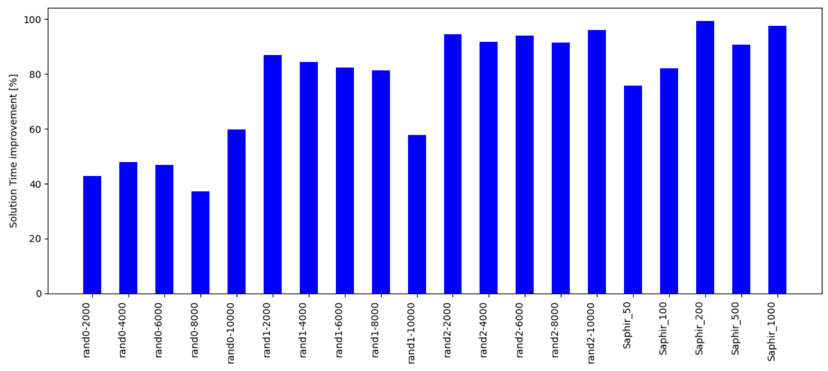

Fig. 1 Maximum relative improvement of the solution time by using FortSP's solvers over DECIS's solvers.

As expected, the performance of the BD algorithm in both FortSP and DECIS behaves similarly, having a difference of less than 10 iterations in each test instance. Both implementations use BD with aggregated cuts but differ in the initialization procedure. However, the BD algorithm is on average 2 times faster in the FortSP's implementation than DECIS's implementation.

In this particular set of instances, the most time-consuming part of the algorithm is the cumulative solution of scenario subproblems, as can be observed in Figures 2 and 3, which is explained by the large number of scenario subproblems. This difference is especially pronounced in the Saphir group, where the recourse problem is larger than the first-stage program, in terms of constraints and variables. In most of the test instances, DECIS with initialization in the EV solution is the methodology that spends more time solving the master problem, as it uses BD to get a proper starting point. Following the general trend, FortSP is faster in the solution of both the master problem and the subproblems separately, indicating that differences in the implementation play an important role in the performance of the decomposition strategies. Warm-starting and automatic selection of the linear solver might contribute to the acceleration of the convergence of BD in FortSP.

7 Software packages for scenario decomposition

In this section, we review two software packages, PySP [61, 62] and DSP [33], for scenario decomposition. The two software packages are benchmarked based on the problems in SIPLIB [2], including the SSLP [47], SSLPR [46], and DCAP [1] test problems.

The SSLP test set consists of 12 two-stage stochastic mixed-integer programs arising in stochastic server location problems (SSLPs). The base deterministic server location problem considers building servers in some potential locations to serve clients in given locations. The stochastic version of the server

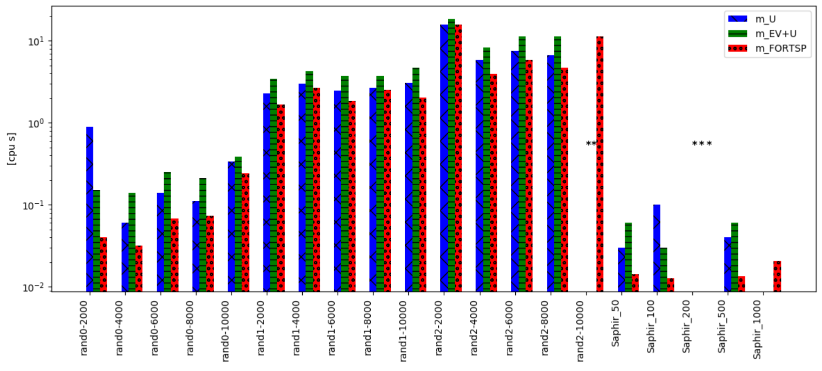

Fig. 2 Cumulative solution time of masters problem in BD, where * means the algorithm fails to solve the instance in 10800 CPU seconds

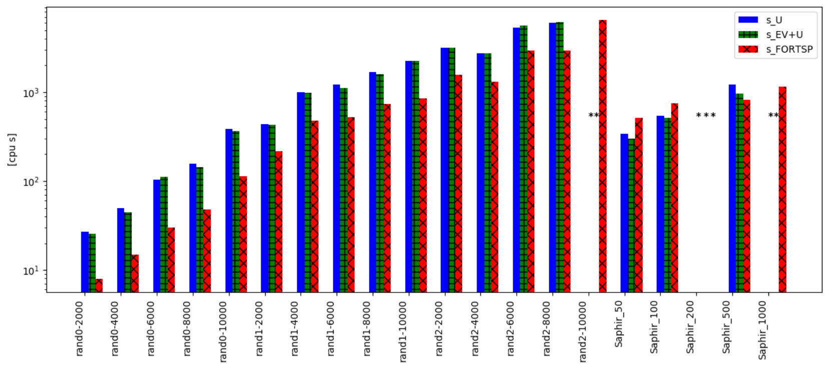

Fig. 3 Cumulative solution time of scenario instances in BD, where * means the algorithm fails to solve the instance in 10800 CPU seconds

location problem considers different realizations of client locations. Each scenario represents a set of potential clients that do materialize. The decisions in SSLP are all binary variables. In the first stage, we decide whether a server is located at each given location. The second stage (recourse) actions decide whether any given client is served by any given server. SSLPR (stochastic server location problem with random recourse) is an extension of SSLP. While SSLP assumes fixed demands for the clients, SSLPR considers the demands of the clients as uncertain parameters.

DCAP consists of 12 two-stage stochastic integer programs arising in dynamic capacity acquisition and allocation applications. The deterministic model considers a capacity expansion schedule over T time periods. In each time period, the amount of capacity expansion for each resource needs to be decided. There is a fixed and a variable cost for each capacity expansion. In each time period, each task must be assigned to one of the existing resources, which is represented by binary variables that decide whether a given task is assigned to a given resource. Since there are multiple periods, the stochastic version of this problem should in principle be formulated as a multi-stage stochastic program, which is difficult to solve. Ahmed and Garcia [1] propose to approximate the multi-stage problem with a two-stage stochastic program in which the first-stage decisions are the capacity expansions. The second-stage decisions are the assignment decisions. The uncertainties include the processing requirement for each task and the cost of processing each task.

The sizes of all the test problems are shown in Table 4. The names of the SSLP and SSLPR instances are expressed in the form sslp(rf)_m_n, where m is the number of potential server locations, and n is the number of potential clients. Each instance is tested with a different number of scenarios. The size is expressed as the number of constraints (Rows), variables (Cols), and integer variables (Ints) in the first stage and the second stage per scenario. Note that the SSLP problems have pure binary first-stage variables and the DCAP problems have mixed-binary first-stage variables. This difference affects the PH algorithm, which will be discussed in detail later.

All of the test problems are available in the SMPS format; however, we implement an interpreter to make the format compatible with PySP. All of the tests were run on a server with an Intel Xeon CPU (24 cores) at 2.67 GHz and 128 GB of RAM. The whole set of instances is solved in synchronous parallel manner to reduce the time of each iteration.

Table 4 The sizes of the problems tested

Name	Scenarios	First Stage			Second Stage		
		Rows	Cols	Ints	Rows	Cols	Ints
sslp_5_25	50, 100	1	5	5	30	130	125
sslp_10_50	50, 100, 500, 1000	1	10	10	60	510	500
sslp_15_45	5,10,15	1	15	15	60	690	675
sslprf_5_25	100	1	5	5	30	130	125
sslprf_5_50	100	1	10	10	60	510	500
dcap 233	200, 300, 500	6	12	6	15	27	27
dcap 243	200, 300, 500	6	12	6	18	36	36
dcap 332	200, 300, 500	6	12	6	12	24	24
dcap 342	200, 300, 500	6	12	6	14	32	32

7.1 PySP: Pyomo Stochastic Programming

PySP is a software package implemented in the Python programming language using Pyomo [26] as the optimization modeling framework. PySP enables the user to solve stochastic programs with a specialized Progressive Hedging algorithm for stochastic mixed-integer programs. In order to use PySP, the user only needs to write a deterministic base model and define the scenario tree structure in Pyomo. With these inputs, PySP is able to apply the Progressive Hedging algorithm as an effective heuristic for obtaining feasible solutions to multi-stage stochastic programs.

7.1.1 Algorithmic Innovations in PySP

The innovations in PySP for multi-stage mixed-integer stochastic programs are described by Watson and Woodruff [61]. Here, we briefly paraphrase those innovations. First, instead of keeping a fixed ρ value for all first-stage decisions in Algorithm 4, the authors propose several variable-dependent ρ strategies. Cost proportional (CP) strategy sets $\rho(i)$ to be proportional to the cost parameter $c(i)$, i.e., $\rho(i) = \alpha c(i)$, where α is a constant multiplier for all first-stage variables i. The other variable-dependent ρ strategy is denoted by SEP in [61], where the $\rho(i)$ for integer variables is calculated by,

$$\rho(i) := \frac{c(i)}{(x^{\max} - x^{\min} + 1)}$$

After PH iteration 0, for each variable x, $x^{\max} = \max_{\omega \in \Omega} x^0_\omega$ and $x^{\min} = \min_{\omega \in \Omega} x^0_\omega$. For continuous variables, the $\rho(i)$ is calculated with

$$\rho(i) := \frac{c(i)}{\max\Big(\big(\sum_{\omega \in \Omega} \tau_\omega |x^0_\omega - \hat{x}^0| \big), 1 \Big)}$$

where $\hat{x}^0$ is the weighted average of x^0_ω, i.e., $\hat{x}^0 = \sum_{\omega \in \Omega} \tau_\omega x^0_\omega$.

The authors also propose some heuristics for accelerating convergence. One heuristic is called "variable fixing". The values of some of the first stage decisions $x_{\omega,i}$ are fixed after they stay at a given value z_i for a few iterations for all scenarios. In order to apply this heuristic, the authors introduce a lag parameter μ. At a given PH iteration k, the value of $x^k_{\omega,i}$ will be fixed to z_i for all subsequent iterations $l > k$, if $x^{(k)}_{\omega,i} = z_i$ for all $\omega \in \Omega$ and $m \in \{k - \mu|\Omega|, \cdots, k\}$, such that $m \geq \mu|\Omega|$. Additionally, the authors propose another more aggressive variable fixing heuristic called "variable slamming" where the x^k_ω will be fixed if they are "sufficiently converged", i.e., there can be some discrepancies for x^k_ω across all scenarios. In order to decide when variable slamming should be applied, the authors propose several termination criteria based on the deviations of the first stage variables.

In solving stochastic mixed-integer programs with PH, cyclic behavior can be found in some instances. In order to detect the cyclic behavior, the authors propose a strategy based on the values of the u_ω vectors, i.e., the weights associated with the first stage decision variable x_ω. The authors propose a simple hashing scheme. Let hash value $h(i) = \sum_{\omega \in \Omega} z_\omega u_{\omega,i}$, where z_ω is an integer hash weight for each scenario $\omega \in \Omega$ when PH is initialized. If equal hash weights are detected, they are interpreted as evidence for potential cycle. Variable x_i can be fixed if cyclic behaviors are found.

The heuristics, including variable fixing and slamming, cyclic behavior detection, are denoted as WW (Watson-Woodruff) heuristics in the software distribution of PySP.

7.1.2 Computational results for PySP

We use PySP (Pyomo 5.0.0) to solve the SSLP, SSLPR, and DCAP problems. Each subproblem is solved with the CPLEX (12.7.0) quadratic solver. We use the cost-proportional (CP) heuristic to set the values of $\rho(i)$. The multipliers α in the CP heuristic are set to 0.1, 0.3, 0.5, and 1.0, respectively. Note that the main results shown in this section are not using WW-heuristics, i.e., we do not use the variable fixing and slamming, or cycle-detection heuristics. We will make a comparison of PySP with WW-heuristics and PySP without WW-heuristics at the end of this section.

The number of iterations and the walltime for different multipliers are shown in Figures 4 and 5, respectively. If the PH algorithm reaches iteration limit, there is an "(i)" label at the top of the column. If the PH algorithm reaches the time limit, there is a "(t)" label on top of the column. From Figures 4 and 5, one can observe that setting the α value to 0.1 makes PH take the largest number of iterations and largest amount of wall time to converge in most of the instances, which may be due to the small step size. On the other hand, setting α to the largest value, i.e., 1.0, takes fewer iterations and less walltime than using other α values in most instances. However, it runs out of the iteration limit in two of the instances. Overall, setting α to 0.3 seems to be a robust choice because cp-0.3 always converges within a reasonable walltime and number of iterations. The details of the SSLP and SSLPR results are shown in Tables 6 and 7 in Appendix 1.

We also apply PySP to solve DCAP instances. We observe that for all the DCAP instances, PySP is unable to converge within 300 iterations. The details of the results are shown in Table 8 in Appendix 2 where the walltime the upper bound for those instances are reported. We will compare the upper bound obtained by PySP with those obtained by DSP in the next subsection. From this experiment, we can see that it is more difficult for PySP to solve problems with mixed-binary first stage variables than problems with pure binary first stage variables because it is more difficult for the continuous variables to satisfy the NACs.

Scenario bundling [13, 24, 54] is a technique that has been used in dual decomposition algorithms. The main idea is to dualize only "some" of the non-anticipativity constraints, rather than dualizing all of them. In other words, the individual scenario subproblems are bundled into larger subproblems in which the NACs are preserved. Ryan et al. [54] use PH with scenario bundling to solve stochastic unit commitment problems. The authors show that with the use of scenario bundling, PH can obtain solutions with better optimality gap. In order to test the effectiveness of scenario bundling, we test several instances from the SSLP and DCAP libraries. The computational results are shown in Table 5. For each instance, we try a different number of bundles.

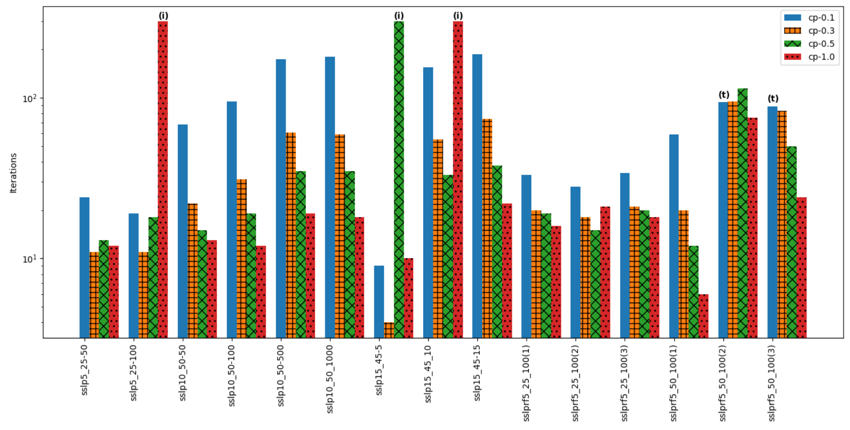

Fig. 4 Number of iterations for PH to solve SSLP instances using different cost proportional multipliers

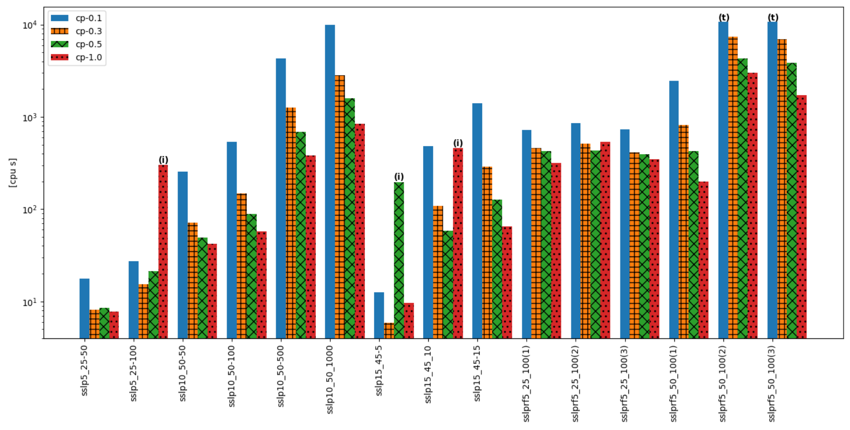

Fig. 5 Walltime for PH to solve SSLP instances using different cost proportional multipliers

For the SSLP instances, PH with a different number of bundles can obtain the same upper bound. However, the number of bundles has a significant impact on the computational time. For example, for SSLP_10_50 with 1000 scenarios, PH with 50 bundles can reduce the walltime of the original PH with 1000 bundles to 3%. Also, it only takes PH with 50 bundles one iteration to converge. For DCAP problems, PH does not converge within 300 iterations for most cases even with scenario bundling. However, PH is able to obtain better feasible solutions with scenario bundling (see UB in Table 5).

Finally, we evaluate how the use of WW-heuristics can affect the performance of PySP on the SSLP and SSLPR libraries. The results on DCAP library are omitted here since PySP does not converge for DCAP instances. The solution time improvements by using WW-heuristic for each SSLP and

Table 5 Computational results for PySP with scenario bundling

Instances	Scenarios	Bundles	Iterations	Time	UB
SSLP_5_25	50	10	4	3.73	-121.60
		50	24	17.59	-121.60
	100	10	2	3.99	-127.37
		50	7	10.03	-127.37
SSLP_10_50	50	10	4	30.55	-364.64
		50	68	254.15	-364.64
	100	10	2	83.67	-354.19
		50	49	263.59	-354.19
		100	95	540.21	-354.19
	500	10	1	476.13	-349.13
		50	2	162.54	-349.14
		500	174	4322.45	-349.14
	1000	10	1	7137.61	-351.71
		50	1	313.07	-351.71
		1000	180	9984.56	-351.71
DCAP233	200	10	>300	342.97	1854.36
		50	>300	232.21	1861.63
		200	>300	456.18	2206.68
	300	10	147	>10800	-
		50	>300	317.28	1679.80
		300	>300	1515.27	2498.12
	500	10	>300	634.60	1749.87
		50	>300	400.59	1858.98
		500	>300	1494.13	1893.83

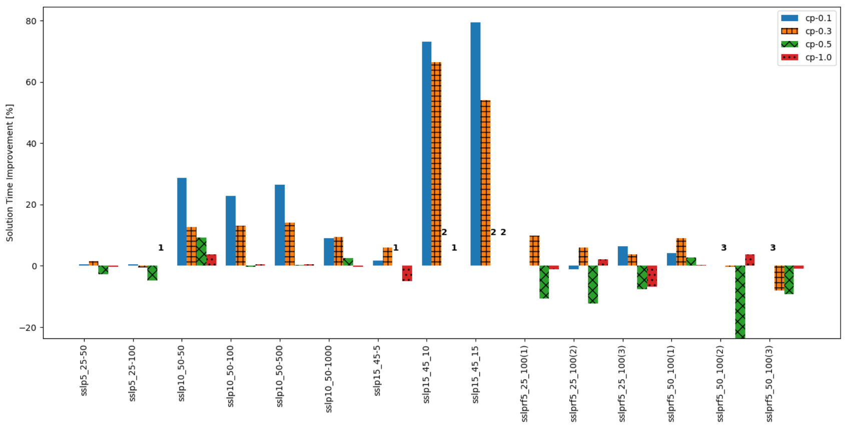

Fig. 6 Solution time improvement by using WW-heuristics for SSLP and SSLPR instances

SSLPR instances are shown in Figure 6. Note that there are three cases where the improvements are omitted in the figure: case (1): neither PH nor PH with WW-heuristics can converge in 300 iterations; case (2): only PH-WW fails to converge in 300 iterations; and case (3): both PH and PH-WW exceed the time limit of 3 hours (10800 CPU seconds). Using WW-heuristic gives significant improvements for small cost-proportional multipliers, i.e., 0.1 and 0.3. As we have described in Table 4, PH with small multipliers usually takes more iterations to converge. Therefore, the WW-heuristics can accelerate convergence for those instances more effectively. However, there also a few instances where PH can converge, but PH with WW-heuristics cannot converge, which are denoted by case (2) in Figure 6.

7.2 DSP: Decompositions for Structured Programming

DSP [33] is an open-source software package implemented in C++ that provides different types of dual decomposition algorithms to solve stochastic mixed-integer programs (SMIPs). DSP can take SMPS files, and JuMP models as input via a Julia package Dsp.jl.

7.2.1 Algorithmic innovations in DSP

From the description of the dual decomposition algorithm in section 5, one can observe that the lower bounds of the dual decomposition algorithm are affected by the way the Lagrangean multipliers are updated. One advantage of DSP is that the authors have different dual-search methods implemented including the subgradient method, the cutting plane method, and a novel interior-point cutting-plane method for the Lagrangean master problem. The authors observe that if the simplex algorithm is used, the solutions to the Lagrangean master problem can oscillate significantly, especially when the Lagrangean dual function is not well approximated. Therefore, the authors propose to solve the Lagrangean master problem suboptimally using an interior point method, which follows from the work of Mitchell [43].

The authors also propose some tightening inequalities that are valid for the Lagrangean subproblems. These valid inequalities, including feasibility and optimality cuts, are obtained from Benders subproblems where the integrality constraints are relaxed. Computational results show that the Benders-like cuts can be effective in practice.

7.2.2 Computational results for DSP in comparison with PySP

We test the dual decomposition algorithm on the SSLP, SSLPR, and DCAP libraries. Each subproblem is solved with the CPLEX (12.7.0) mixed-integer linear solver. The interior point method proposed by the authors [33] is used to solve the Lagrangean master problem, which is solved with the CPLEX as well. Benders-like cuts are not used because the implementation of Benders cuts in DSP only works with SCIP. In Figure 7 and 8, we evaluate the best feasible solution (the upper bound) obtained by PySP, and the upper and lower bound obtained by DSP. For each instance, we include three different gaps.

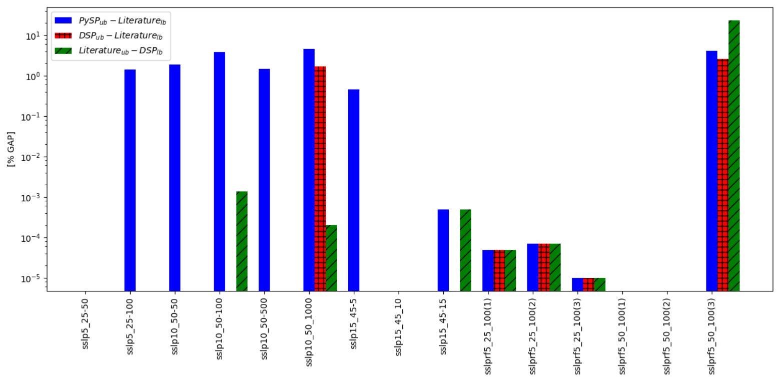

Fig. 7 Comparison of optimality gaps from PySP, DSP, and literature for SSLP and SSLPR library - Instances with only binary in the first-stage.

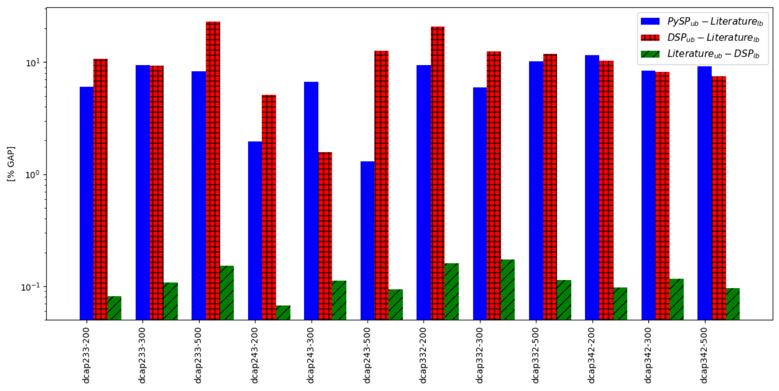

Fig. 8 Comparison of the optimality gaps from PySP, DSP, and literature for DCAP library - Instances with mixed-integer variables in the first-stage.

The upper and lower bound from literature [2] are used to evaluate the bounds from PySP and DSP. Note that the bounds from literature are close to the global optimality of each instance. The first column for each instance in both Figures 7 and 8 is the gap between the upper bound from PySP ($PySP_{ub}$) and the lower bound from literature ($Literature_{lb}$). The second column represents the gap between the upper bound from DSP (DSP_{ub}) and the lower bound from literature ($Literature_{lb}$). The third column represents the gap between the upper bound from literature ($Literature_{ub}$) and the lower bound from DSP (DSP_{lb}).

For the SSLP and SSLPR instances shown in Figure 7, although PySP can converge within the time and iteration limit, the best feasible solution obtained from PySP ($PySP_{ub}$) may not be optimal. There are about 1% gaps for some of the SSLP instances (see the first column of each instance in Figure 7). DSP can solve more instances to optimality than PySP (see the second column of each instance in Figure 7). The lower bounds obtained by DSP are also quite tight, usually less than 0.01% (see the third column of each instance in Figure 7). Note that the literature values for SSLPRF5_50_100(1), SSLPRF5_50_100(2), and SSLPRF5_50_100(3) do not match the values from our experiment. Therefore, we try to solve the deterministic equivalent of these instances to obtain bounds. The literature bounds of SSLPRF5_50_100(3) come from solving the deterministic equivalent. The gaps of SSLPRF5_50_100(1) and SSLPRF5_50_100(2) are omitted since the corresponding deterministic equivalent cannot be solved within 3 hours.

For the DCAP instances where we have mixed-integer first-stage variables, the best feasible solutions from PySP ($PySP_{ub}$) and DSP (DSP_{ub}) are quite far from optimality. The gaps of the first two columns are around 10%. On the other hand, the lower bounds obtained from DSP (DSP_{lb}) are tight. The gaps between ($Literature_{ub}$) and (DSP_{lb}) are around 0.1%. Therefore, in order to improve the relative optimality gap of DSP, the focus should be on designing more advanced heuristics to obtain better feasible solutions.

8 Conclusion

We have presented a summary of the state-of-the-art methodologies for the solution of two-stage linear stochastic problems. First, we introduced the mathematical formulation of such programs and highlighted features in their structure which enable the development of decomposition algorithms. These methodologies are classified in two groups: time-dependent decomposition and scenario-based decomposition.

For two-stage stochastic programs with continuous recourse, we have summarized Benders Decomposition, which partitions the problem according to its time structure. BD may present computational problems, which can be alleviated by reducing the cost of each iteration, and/or decreasing the number of iterations. We benchmarked standard BD and three quadratic stabilization variants in two separate software packages, DECIS and FortSP. Our results show that these methodologies are effective in reducing the solution time and overall number of iterations; however, they might fail to converge due to numerical problems. Specifically, we showed that Regularized Decomposition relies on the quality of the penalty term if it is not updated at each iteration. In addition, we found that differences in the initialization and implementation can affect the performance of BD.

Scenario decomposition methodologies are popular alternatives in the case of (mixed) integer recourse. The progressive Hedging Algorithm and Dual Decomposition relax the nonanticipatity restrictions and provide the user with valid bounds. Our numerical results show that the performance of PH is strongly affected by the constant penalty multiplier. Furthermore, its performance and the quality of the approximate solution may be enhanced by grouping the scenarios in large bundles (or scenario sets). We also have tested the dual decomposition algorithm with the DSP package. The computational results show that DSP is able to provide tight lower bound on the instances that we have tested. However, the optimality gaps can be as large as 10%, relative to the upper bound from literature. Therefore, for those tested instances, future effort should be focused on developing more advanced heuristics to improve the best feasible solution.

9 Appendix 1: Computational results for PySP

Table 6 Computational results for PySP on SSLP

Instances	Scenarios	cp multiplier	Iterations	Time	UB
SSLP_5_25	50	cp 0.1	24	17.59	-121.60
		cp 0.3	11	8.16	-121.60
		cp 0.5	13	8.53	-121.60
		cp 1.0	12	7.80	-121.60
	100	cp 0.1	19	27.38	-127.37
		cp 0.3	11	15.27	-127.37
		cp 0.5	18	21.26	-127.37
		cp 1.0	>300	303.53	-125.59
SSLP_10_50	50	cp 0.1	68	254.15	-364.64
		cp 0.3	22	71.57	-364.64
		cp 0.5	15	49.19	-364.64
		cp 1.0	13	41.91	-364.64
	100	cp 0.1	95	540.21	-354.19
		cp 0.3	31	149.33	-354.19
		cp 0.5	19	88.54	-354.19
		cp 1.0	12	57.98	-354.19
	500	cp 0.1	174	4322.45	-349.14
		cp 0.3	61	1265.18	-349.14
		cp 0.5	35	688.90	-349.14
		cp 1.0	19	379.64	-349.14
	1000	cp 0.1	180	9984.56	-351.71
		cp 0.3	59	2849.90	-357.71
		cp 0.5	35	1604.27	-357.71
		cp 1.0	18	845.16	-351.71
SSLP_15_45	5	cp 0.1	9	12.48	-262.40
		cp 0.3	4	5.85	-261.20
		cp 0.5	>300	197.35	-261.20
		cp 1.0	10	9.65	-261.20
	10	cp 0.1	155	485.38	-260.50
		cp 0.3	55	108.15	-260.50
		cp 0.5	33	58.23	-259.30
		cp 1.0	>300	463.63	-259.30
	15	cp 0.1	186	1416.21	-253.60
		cp 0.3	74	286.78	-253.60
		cp 0.5	38	126.79	-253.60
		cp 1.0	22	65.08	-253.20

Table 7 Computational results for PySP on SSLPR

Instances	cp multiplier	Iterations	Time	UB
SSLPRF_5_25_100_1	cp 0.1	33	722.61	-74005.84
	cp 0.3	20	461.1	-74005.84
	cp 0.5	19	427.04	-74005.84
	cp 1.0	16	315.67	-74005.84
SSLPRF_5_25_100_2	cp 0.1	28	852.75	-72671.95
	cp 0.3	18	517.33	-72671.95
	cp 0.5	15	436.16	-72671.95
	cp 1.0	21	539.61	-72671.95
SSLPRF_5_25_100_3	cp 0.1	34	735.9	-75664.19
	cp 0.3	21	412.91	-75664.19
	cp 0.5	20	397.01	-75664.19
	cp 1.0	18	348.75	-75664.19
SSLPRF_5_50_100_1	cp 0.1	59	2457.66	138900.12
	cp 0.3	20	814.04	138900.12
	cp 0.5	12	429.35	138900.12
	cp 1.0	6	200.07	138900.12
SSLPRF_5_50_100_2	cp 0.1	94	>10800	-
	cp 0.3	95	7382.14	245424.96
	cp 0.5	114	4315.02	245424.96
	cp 1.0	75	3008.51	500144.07
SSLPRF_5_50_100_3	cp 0.1	88	>10800	-
	cp 0.3	83	6984.09	258578.79
	cp 0.5	50	3887.54	258578.79
	cp 1.0	24	1727.73	258578.79

10 Appendix 2: Computational results for DSP

Table 8 Computational results for PySP on DCAP

Instances	Scenarios	cp multiplier	Iterations	Time	UB
DCAP 233	200	cp 0.1	>300	456.18	2206.7
		cp 0.3	>300	457.5	2431.8
		cp 0.5	>300	458.23	1966.1
		cp 1.0	>300	455.88	1952.6
	300	cp 0.1	>300	734.23	1862.0
		cp 0.3	>300	726.59	1943.3
		cp 0.5	>300	726.77	1831.6
		cp 1.0	>300	741.17	1815.4
	500	cp 0.1	>300	1515.27	2498.1
		cp 0.3	>300	1492.59	2000.0
		cp 0.5	>300	1467.64	1939.2
		cp 1.0	>300	1494.13	1893.8
DCAP 243	200	cp 0.1	>300	481.96	2465.9
		cp 0.3	>300	478.34	2454.2
		cp 0.5	>300	481.51	2369.3
		cp 1.0	>300	466.95	2383.2
	300	cp 0.1	>300	792.01	2825.8
		cp 0.3	>300	756.17	2802.8
		cp 0.5	>300	710.44	2755.0
		cp 1.0	>300	776.97	2743.1
	500	cp 0.1	>300	1690.74	2196.0
		cp 0.3	>300	1622.8	2235.2
		cp 0.5	>300	1674.76	2216.7
		cp 1.0	>300	1536.42	2323.3
DCAP 332	200	cp 0.1	>300	456.58	1362.7
		cp 0.3	>300	427.65	1529.1
		cp 0.5	>300	450.88	1278.0
		cp 1.0	>300	460.35	1171.0
	300	cp 0.1	>300	714.83	1332.1
		cp 0.3	>300	698.52	1948.9
		cp 0.5	>300	709.21	1904.4
		cp 1.0	>300	706.75	1766.3
	500	cp 0.1	>300	1464.11	1768.6
		cp 0.3	>300	1451.73	1822.8
		cp 0.5	>300	1473.66	1846.6
		cp 1.0	>300	1452.64	1861.6
DCAP 342	200	cp 0.1	>300	449.9	1993.4
		cp 0.3	>300	476.77	1990.4
		cp 0.5	>300	445.11	1870.6
		cp 1.0	>300	470.82	1830.8
	300	cp 0.1	>300	722.6	2260.3
		cp 0.3	>300	739.94	2371.1
		cp 0.5	>300	690.76	2497.7
		cp 1.0	>300	702.23	2255.9
	500	cp 0.1	>300	1582.9	2198.1
		cp 0.3	>300	1604.98	2317.5
		cp 0.5	>300	1555.9	2290.6
		cp 1.0	>300	1593.41	2097.5

Table 9 Computational results for DSP on SSLP

Instances	Scenarios	Iterations	Time	LB	UB	Gap [%]
SSLP_5_25	50	16	3.86	-121.60	-121.60	0.00
	100	17	6.00	-127.37	-125.59	1.42
SSLP_10_50	50	57	204.63	-364.64	-357.98	1.86
	100	44	213.95	-354.19	-341.33	3.77
	500	69	2439.58	-349.14	-344.02	1.49
	1000	60	2960.55	-351.71	-336.23	4.60
SSLP_15_45	5	15	14.14	-262.40	-261.20	0.46
	10	41	152.25	-260.50	-260.50	0.00
	15	44	207.39	-253.60	-253.60	0.00

Table 10 Computational results for DSP on SSLPR

Instances	Iterations	Time	LB	UB	Gap [%]
SSLPRF_5_25_100_1	36	1239.55	-74005.84	-74005.84	0.00
SSLPRF_5_25_100_2	38	1783.89	-72671.95	-72671.95	0.00
SSLPRF_5_25_100_3	40	1541.41	-75664.19	-75664.19	0.00
SSLPRF_5_50_100_1	88	6776.87	138900.12	138900.12	0.00
SSLPRF_5_50_100_2	57	9357.49	163943.96	245427.14	33.20
SSLPRF_5_50_100_3	85	>10800	189569.71	254469.62	25.50

Table 11 Computational results for DSP on DCAP

Instances	Scenarios	Iterations	Time	LB	UB	Gap [%]
DCAP 233	200	59	17.65	1833.40	2053.77	10.73
	300	69	35.45	1642.73	1812.89	9.39
	500	60	29.57	1735.09	2257.81	23.15
DCAP 243	200	54	17.81	2321.17	2447.75	5.17
	300	50	23.62	2556.68	2600.56	1.69
	500	62	58.24	2165.48	2481.84	12.75
DCAP 332	200	59	16.25	1059.09	1337.71	20.83
	300	79	39.58	1250.91	1431.11	12.59
	500	66	55.94	1587.07	1802.24	11.94
DCAP 342	200	52	14.32	1618.07	1804.57	10.34
	300	46	21.19	2065.42	2252.33	8.30
	500	56	51.59	1902.98	2059.87	7.62

References

1. Ahmed, S., Garcia, R.: Dynamic capacity acquisition and assignment under uncertainty. Annals of Operations Research **124**(1-4), 267–283 (2003)
2. Ahmed, S., Garcia, R., Kong, N., Ntaimo, L., Parija, G., Qiu, F., Sen, S.: Siplib: A stochastic integer programming test problem library. See http://www2. isye. gatech. edu/˜ sahmed/siplib (2004)
3. Ahmed, S., Tawarmalani, M., Sahinidis, N.V.: A finite branch-and-bound algorithm for two-stage stochastic integer programs. Mathematical Pro-

gramming **100**(2), 355–377 (2004)

4. Benders, J.F.: Partitioning procedures for solving mixed-variables programming problems. Numerische Mathematik **4**(1), 238–252 (1962). DOI 10.1007/BF01386316. URL https://doi.org/10.1007/BF01386316
5. Birge, J.R., Louveaux, F.: Introduction to stochastic programming. Springer Science & Business Media (2011)
6. Birge, J.R., Louveaux, F.V.: A multicut algorithm for two-stage stochastic linear programs. European Journal of Operational Research **34**(3), 384–392 (1988). DOI https://doi.org/10.1016/0377-2217(88)90159-2. URL http://www.sciencedirect.com/science/article/pii/0377221788901592
7. Boland, N., Christiansen, J., Dandurand, B., Eberhard, A., Linderoth, J., Luedtke, J., Oliveira, F.: Combining progressive hedging with a frank–wolfe method to compute lagrangian dual bounds in stochastic mixed-integer programming. SIAM Journal on Optimization **28**(2), 1312–1336 (2018)
8. Bussieck, M.R., Meeraus, A.: General Algebraic Modeling System (GAMS). In: Modeling Languages in Mathematical Optimization, J. Kallrath (Ed.), *Applied Optimization*, vol. 88, pp. 137–157. Springer US (2004)
9. Carøe, C.C., Schultz, R.: Dual decomposition in stochastic integer programming. Operations Research Letters **24**(1-2), 37–45 (1999)
10. Costa, A.M., Cordeau, J.F., Gendron, B., Laporte, G.: Accelerating Benders decomposition with heuristic master problem solutions. Pesquisa Operacional **32**(1), 03–20 (2012)
11. Eckstein, J., Bertsekas, D.P.: On the douglasrachford splitting method and the proximal point algorithm for maximal monotone operators. Mathematical Programming **55**(1-3), 293–318 (1992)
12. Ellison, F., Mitra, G., Zverovich, V.: Fortsp: a stochastic programming solver. OptiRisk Systems (2010)
13. Escudero, L.F., Garín, M.A., Unzueta, A.: Scenario cluster lagrangean decomposition for risk averse in multistage stochastic optimization. Computers & Operations Research **85**, 154–171 (2017)
14. Gabay, D., Mercier, B.: A dual algorithm for the solution of non linear variational problems via finite element approximation. Institut de recherche d'informatique et d'automatique (1975)
15. Gade, D., Hackebeil, G., Ryan, S.M., Watson, J.P., Wets, R.J.B., Woodruff, D.L.: Obtaining lower bounds from the progressive hedging algorithm for stochastic mixed-integer programs. Mathematical Programming **157**(1), 47–67 (2016)
16. Gade, D., Küçükyavuz, S., Sen, S.: Decomposition algorithms with parametric gomory cuts for two-stage stochastic integer programs. Mathematical Programming **144**(1-2), 39–64 (2014)
17. Garcia-Herreros, P., Wassick, J.M., Grossmann, I.E.: Design of resilient supply chains with risk of facility disruptions. Industrial & Engineering Chemistry Research **53**(44), 17,240–17,251 (2014). DOI 10.1021/ie5004174. URL http://dx.doi.org/10.1021/ie5004174

18. Geoffrion, A.M.: Generalized Benders decomposition. Journal of Optimization Theory and Applications **10**(4), 237–260 (1972). DOI 10.1007/BF00934810. URL https://doi.org/10.1007/BF00934810
19. Geoffrion, A.M.: Lagrangean relaxation for integer programming. In: Approaches to integer programming, pp. 82–114. Springer (1974)
20. Geoffrion, A.M., Graves, G.W.: Multicommodity distribution system design by Benders decomposition. Management science **20**(5), 822–844 (1974)
21. Gonçalves, R.E.C., Finardi, E.C., da Silva, E.L.: Applying different decomposition schemes using the progressive hedging algorithm to the operation planning problem of a hydrothermal system. Electric Power Systems Research **83**(1), 19–27 (2012)
22. Guignard, M.: Lagrangean relaxation. Top **11**(2), 151–200 (2003)
23. Guo, G., Hackebeil, G., Ryan, S.M., Watson, J.P., Woodruff, D.L.: Integration of progressive hedging and dual decomposition in stochastic integer programs. Operations Research Letters **43**(3), 311–316 (2015). DOI https://doi.org/10.1016/j.orl.2015.03.008. URL http://www.sciencedirect.com/science/article/pii/S0167637715000462
24. Gupta, V., Grossmann, I.E.: A new decomposition algorithm for multistage stochastic programs with endogenous uncertainties. Computers & Chemical Engineering **62**, 62–79 (2014)
25. Hansen, P., Mladenović, N., Perez-Britos, D.: Variable neighborhood decomposition search. Journal of Heuristics **7**(4), 335–350 (2001). DOI 10.1023/A:1011336210885. URL https://doi.org/10.1023/A:1011336210885
26. Hart, W.E., Laird, C.D., Watson, J.P., Woodruff, D.L., Hackebeil, G.A., Nicholson, B.L., Siirola, J.D.: Pyomo-optimization modeling in python, vol. 67. Springer (2012)
27. Holmberg, K.: On using approximations of the Benders master problem. European Journal of Operational Research **77**(1), 111–125 (1994). DOI https://doi.org/10.1016/0377-2217(94)90032-9. URL http://www.sciencedirect.com/science/article/pii/0377221794900329
28. Hooker, J.N., Ottosson, G.: Logic-based Benders decomposition. Mathematical Programming **96**(1), 33–60 (2003). DOI 10.1007/s10107-003-0375-9. URL https://doi.org/10.1007/s10107-003-0375-9
29. Infanger, G.: Gams/decis users guide (1999)
30. Kall, P., Mayer, J.: On testing slp codes with slp-ior. In: New trends in mathematical programming, pp. 115–135. Springer (1998)
31. Kall, P., Wallace, S.W.: Stochastic programming. Wiley, Chichester (1996)
32. Keller, E.: Genslp: A program for generating input for stochastic linear programs with complete fixed recourse. Manuscript, IOR, University of Zurich (1984)
33. Kim, K., Zavala, V.M.: Algorithmic innovations and software for the dual decomposition method applied to stochastic mixed-integer programs. Mathematical Programming Computation (2017). DOI 10.1007/s12532-017-0128-z. URL https://doi.org/10.1007/s12532-017-0128-z

34. Koberstein, A., Lucas, C., Wolf, C., König, D.: Modeling and optimizing risk in the strategic gas-purchase planning problem of local distribution companies. The Journal of Energy Markets **4**(3), 47 (2011)
35. Konig, D., Suhl, L., Koberstein, A.: Optimierung des gasbezugs im liberalisierten gasmarkt unter berucksichtigung von rohren-und untertagespeichern. VDI BERICHTE **2018**, 83 (2007)
36. Lemaréchal, C., Nemirovskii, A., Nesterov, Y.: New variants of bundle methods. Mathematical programming **69**(1), 111–147 (1995)
37. Li, C., Grossmann, I.E.: An improved L-shaped method for two-stage convex 0–1 mixed integer nonlinear stochastic programs. Computers & Chemical Engineering **112**, 165–179 (2018)
38. Lim, C., Cochran, J.J., Cox, L.A., Keskinocak, P., Kharoufeh, J.P., Smith, J.C.: Relationship among Benders, Dantzig-Wolfe, and Lagrangian Optimization. John Wiley & Sons, Inc. (2010). DOI 10.1002/9780470400531.eorms0717. URL http://dx.doi.org/10.1002/9780470400531.eorms0717
39. Linderoth, J., Wright, S.: Decomposition algorithms for stochastic programming on a computational grid. Computational Optimization and Applications **24**(2), 207–250 (2003). DOI 10.1023/A:1021858008222. URL https://doi.org/10.1023/A:1021858008222
40. Lubin, M., Martin, K., Petra, C.G., Sandıkçı, B.: On parallelizing dual decomposition in stochastic integer programming. Operations Research Letters **41**(3), 252–258 (2013). DOI https://doi.org/10.1016/j.orl.2013.02.003. URL http://www.sciencedirect.com/science/article/pii/S0167637713000242
41. Magnanti, T.L., Wong, R.T.: Accelerating Benders decomposition: Algorithmic enhancement and model selection criteria. Operations research **29**(3), 464–484 (1981)
42. McDaniel, D., Devine, M.: A modified Benders' partitioning algorithm for mixed integer programming. Management Science **24**(3), 312–319 (1977)
43. Mitchell, J.E.: Computational experience with an interior point cutting plane algorithm. SIAM Journal on Optimization **10**(4), 1212–1227 (2000)
44. Mitra, S., Garcia-Herreros, P., Grossmann, I.E.: A cross-decomposition scheme with integrated primaldual multi-cuts for two-stage stochastic programming investment planning problems. Mathematical Programming **157**(1), 95–119 (2016)
45. Naoum-Sawaya, J., Elhedhli, S.: An interior-point Benders based branch-and-cut algorithm for mixed integer programs. Annals of Operations Research **210**(1), 33–55 (2013)
46. Ntaimo, L.: Disjunctive decomposition for two-stage stochastic mixed-binary programs with random recourse. Operations research **58**(1), 229–243 (2010)
47. Ntaimo, L., Sen, S.: The million-variable march for stochastic combinatorial optimization. Journal of Global Optimization **32**(3), 385–400 (2005)
48. Rahmaniani, R., Crainic, T.G., Gendreau, M., Rei, W.: The Benders decomposition algorithm: A literature review. European Journal of Opera-

tional Research (2016)
49. Raidl, G.R., Baumhauer, T., Hu, B.: Speeding up logic-based Benders decomposition by a metaheuristic for a bi-level capacitated vehicle routing problem. In: International Workshop on Hybrid Metaheuristics, pp. 183–197. Springer (2014)
50. Rockafellar, R.T., Wets, R.J.B.: Scenarios and policy aggregation in optimization under uncertainty. Mathematics of Operations Research **16**(1), 119–147 (1991). DOI 10.1287/moor.16.1.119. URL https://doi.org/10.1287/moor.16.1.119
51. Ruszczyński, A.: A regularized decomposition method for minimizing a sum of polyhedral functions. Mathematical programming **35**(3), 309–333 (1986)
52. Ruszczyński, A.: Decomposition methods in stochastic programming. Mathematical Programming **79**(1), 333–353 (1997). DOI 10.1007/BF02614323. URL https://doi.org/10.1007/BF02614323
53. Ruszczyński, A., Świętanowski, A.: Accelerating the regularized decomposition method for two stage stochastic linear problems. European Journal of Operational Research **101**(2), 328–342 (1997). DOI https://doi.org/10.1016/S0377-2217(96)00401-8. URL http://www.sciencedirect.com/science/article/pii/S0377221796004018
54. Ryan, S.M., Wets, R.J.B., Woodruff, D.L., Silva-Monroy, C., Watson, J.P.: Toward scalable, parallel progressive hedging for stochastic unit commitment. In: Power and Energy Society General Meeting (PES), 2013 IEEE, pp. 1–5. IEEE (2013)
55. Saharidis, G.K., Minoux, M., Ierapetritou, M.G.: Accelerating Benders method using covering cut bundle generation. International Transactions in Operational Research **17**(2), 221–237 (2010)
56. Sherali, H.D., Fraticelli, B.M.P.: A modification of Benders' decomposition algorithm for discrete subproblems: An approach for stochastic programs with integer recourse. Journal of Global Optimization **22**(1), 319–342 (2002). DOI 10.1023/A:1013827731218. URL https://doi.org/10.1023/A:1013827731218
57. Tarhan, B., Grossmann, I.E.: A multistage stochastic programming approach with strategies for uncertainty reduction in the synthesis of process networks with uncertain yields. Computers & Chemical Engineering **32**(4-5), 766–788 (2008)
58. Van Roy, T.J.: Cross decomposition for mixed integer programming. Mathematical programming **25**(1), 46–63 (1983)
59. Van Slyke, R., Wets, R.: L-shaped linear programs with applications to optimal control and stochastic programming. SIAM Journal on Applied Mathematics **17**(4), 638–663 (1969). DOI 10.1137/0117061. URL https://doi.org/10.1137/0117061
60. Vanderbeck, F., Wolsey, L.A.: Reformulation and decomposition of integer programs (2009)
61. Watson, J.P., Woodruff, D.L.: Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems. Computational

Management Science **8**(4), 355–370 (2011)
62. Watson, J.P., Woodruff, D.L., Hart, W.E.: Pysp: modeling and solving stochastic programs in python. Mathematical Programming Computation **4**(2), 109–149 (2012)
63. Ye, Y., Li, J., Li, Z., Tang, Q., Xiao, X., Floudas, C.A.: Robust optimization and stochastic programming approaches for medium-term production scheduling of a large-scale steelmaking continuous casting process under demand uncertainty. Computers & Chemical Engineering **66**, 165–185 (2014)
64. Zakeri, G., Philpott, A.B., Ryan, D.M.: Inexact cuts in Benders decomposition. SIAM Journal on Optimization **10**(3), 643–657 (2000)
65. Zaourar, S., Malick, J.: Quadratic stabilization of Benders decomposition (2014)
66. Zehtabian, S., Bastin, F.: Penalty parameter update strategies in progressive hedging algorithm (2016)
67. Zhang, M., Küçüukyavuz, S.: Finitely convergent decomposition algorithms for two-stage stochastic pure integer programs. SIAM Journal on Optimization **24**(4), 1933–1951 (2014)
68. Zverovich, V., Fábián, C.I., Ellison, E.F., Mitra, G.: A computational study of a solver system for processing two-stage stochastic lps with enhanced Benders decomposition. Mathematical Programming Computation **4**(3), 211–238 (2012)