5-2015

Integration of progressive hedging and dual decomposition in stochastic integer programs

Ge Guo
Iowa State University, geguo@iastate.edu

Gabriel Hackebeil
Sandia National Laboratories

Sarah M. Ryan
Iowa State University, smryan@iastate.edu

Jean-Paul Watson
Sandia National Laboratories

David L. Woodruff
University of California - Davis

Integration of progressive hedging and dual decomposition in stochastic integer programs

Abstract
We present a method for integrating the Progressive Hedging (PH) algorithm and the Dual Decomposition (DD) algorithm of Carøe and Schultz for stochastic mixed-integer programs. Based on the correspondence between lower bounds obtained with PH and DD, a method to transform weights from PH to Lagrange multipliers in DD is found. Fast progress in early iterations of PH speeds up convergence of DD to an exact solution. We report computational results on server location and unit commitment instances.

Keywords
Stochastic programming, Mixed-integer programming, Progressive hedging, Dual decomposition, Lower bounding

Disciplines
Industrial Engineering | Systems Engineering

Integration of progressive hedging and dual decomposition in stochastic integer programs

Ge Guo[a], Gabriel Hackebeil[b], Sarah M. Ryan[a,*], Jean-Paul Watson[b], David L. Woodruff[c]

[a] Department of Industrial and Manufacturing Systems Engineering, Iowa State University, Ames, IA 50011, USA
[b] Discrete Math and Complex Systems Department, Sandia National Laboratories, Albuquerque, NM 87185, USA
[c] Graduate School of Management, University of California Davis, Davis, CA 95616, USA
* Corresponding author. Tel.: +1515 294 4347. E-mail address: smryan@iastate.edu (S.M. Ryan)

Abstract

We present a method for integrating the Progressive Hedging (PH) algorithm and the Dual Decomposition (DD) algorithm of Carøe and Schultz for stochastic mixed-integer programs. Based on the correspondence between lower bounds obtained with PH and DD, a method to transform weights from PH to Lagrange multipliers in DD is found. Fast progress in early iterations of PH speeds up convergence of DD to an exact solution. We report computational results on server location and unit commitment instances.

Keywords: Stochastic programming; Mixed-integer programming; Progressive hedging; Dual decomposition; Lower bounding

1. Introduction

Stochastic mixed-integer programs find a broad application in energy, facility location, production scheduling and other areas where a set of decisions must be taken before full information is revealed on some random events and some of the decisions are required to be integer [1]. The combination of uncertainty and discrete decisions leads to the difficulty in solving stochastic mixed-integer programs.

Until now much progress has been made in developing algorithms to solve these problems, extending from special instances [12, 13, 23] to more general stochastic mixed-integer programs [2, 20]. Carøe and Schultz [3] developed a dual decomposition (DD) algorithm based on scenario decomposition and Lagrangian relaxation. Lubin et al. [14] demonstrated the potential for parallel speedup by addressing the bottleneck of parallelizing dual decomposition. Originally proposed by Rockafellar and Wets [19] for stochastic programs with only continuous variables, progressive hedging (PH) has been successfully applied by Listes and Dekker [17], Fan and Liu [6], Watson and Woodruff [25], and many others as a heuristic to solve stochastic mixed-integer programs. To assess the quality of the solutions generated by PH relative to the optimal solution, Gade et al. [8] presented a lower bounding technique for the PH algorithm and showed that the best possible lower bound obtained from PH is as tight as the lower bound obtained using DD.

The PH algorithm can find high-quality solutions within a reasonable number of iterations, but is not guaranteed to converge to a globally optimal solution in the case of mixed-integer problems. The DD algorithm, on the other hand, will achieve convergence combined with branch and bound but may be slow. This paper combines advantages of both scenario decomposition methods. By transforming PH weights into Lagrangian multipliers as a starting point for DD, the convergence of DD can be sped up considerably.

The remainder of this paper is organized as follows. In Section 2 we describe the PH and DD algorithms, two scenario-based decomposition algorithms for stochastic mixed-integer programs. Our integration approach to transfer information from PH to DD is developed in Section 3. In Section 4, we document the implementation of our integration method and in Section 5, provide experimental results on a set of stochastic server location instances and two stochastic unit commitment instances.

2. Scenario Decomposition Algorithms for Stochastic Mixed Integer Programs

Decomposition methods for stochastic programs generally fall into two groups: stage-based methods and scenario-based methods [18]. The exemplary stage-based decomposition method is the L-shaped method, or Benders decomposition [21]. Paradigms of scenario-based decomposition include the PH algorithm [19] and the DD algorithm [3]. One advantage of scenario-based decomposition methods over the stage-based ones is their mitigation of the computational difficulty associated with large problem instances by decomposing the problem by scenario and solving the subproblems in parallel. In practical applications, PH can easily be implemented as a “wrapper” for existing software for large-scale implementation of the deterministic scenario problems. In this section, we will discuss these two scenario-based decomposition methods for stochastic mixed-integer programs in detail.

2.1. Two-Stage Stochastic Mixed-Integer Program

We consider the following two-stage stochastic mixed-integer program:

$$z = \min\left\{cx + Q(x) : Ax = b, x \in X\right\}, \tag{1}$$

where $Q(x) = \mathbb{E}_{\xi}\phi(x,\xi)$ and $\phi(x,\xi) = \min\left\{q(\xi)y : Wy = h(\xi) - T(\xi)x, y \in Y\right\}$. Here $c^T \in \mathbb{R}^{n_1}$ and $b \in \mathbb{R}^{m_1}$ are known vectors, while $A \in \mathbb{R}^{m_1 \times n_1}$ and $W \in \mathbb{R}^{m_2 \times n_2}$ are known matrices. The vector ξ is a random variable defined on some probability space $(\Xi, \Im, P)$ and for each $\xi \in \Xi$, the vectors $q(\xi)^T \in \mathbb{R}^{n_2}$ and $h(\xi) \in \mathbb{R}^{m_2}$ and the matrix $T(\xi) \in \mathbb{R}^{m_2 \times n_1}$. The sets $X \subseteq \mathbb{R}^{n_1}_{+}$ and $Y \subseteq \mathbb{R}^{n_2}_{+}$ denote the mixed-integer requirements on the first-stage and second-stage variables. The decisions are two-stage in the sense that first-stage decisions x have to be taken without full information on some random events while second-stage decisions y are taken after full

information is received on the realization of the random vector ξ. The notation $\mathbb{E}_{\xi}$ denotes expectation with respect to the distribution of ξ.

To avoid complications when computing the integral behind $\mathbb{E}_{\xi}$ we assume that we have only a finite number of realizations of ξ, known as scenarios ξ^j, $j=1,...,r$, with corresponding probabilities p^j. Then problem (1) can be written as a large-scale deterministic mixed-integer linear program with a block-angular structure called the extensive form of the deterministic equivalent:

$$z = \min\left\{cx + \sum_{j=1}^{r} p^j q^j y^j : (x, y^j) \in S^j,\ j = 1,...,r\right\}, \quad \textbf{(2)}$$

where $S^j = \left\{(x, y^j) : Ax = b, x \in X, Wy^j = h^j - T^j x, y^j \in Y\right\}$.

The block-angular structure of Eq. (2) enables the decomposition methods to split it into scenario subproblems by introducing copies of the first-stage variables. This idea leads to the so-called scenario formulation of the stochastic program:

$$z = \min\left\{\sum_{j=1}^{r} p^j (cx^j + q^j y^j) : (x^j, y^j) \in S^j,\ j = 1,...,r, x^1 = ... = x^r\right\}. \quad \textbf{(3)}$$

The subproblems are coupled by the non-anticipativity constraints, $x^1 = ... = x^r$, which force the first-stage decisions to be scenario-independent.

2.2. Dual Decomposition

The dual decomposition (DD) algorithm of Carøe and Schultz relaxes the non-anticipativity constraints and uses branch and bound to restore non-anticipativity. DD obtains lower bounds on the optimal value of problem (3) by solving the Lagrangian dual obtained by relaxing the non-anticipativity constraints.

The non-anticipativity requirement of problem (3) can be expressed by several equivalent representations. Lulli and Sen [15] as well as Lubin and Martin [14] introduce an additional variable $x.$ and model non-anticipativity as

$$x_j - x. = 0,\ j = 1,...r, \quad \textbf{(4)}$$

while Carøe and Schultz represent non-anticipativity by

$$\sum_{j=1}^{r} H^j x^j = 0, \quad \textbf{(5)}$$

where the matrix $H^j \in \mathbb{R}^{n_1(r-1)\times n_1}$.

Using non-anticipativity representation (4), the Lagrangian relaxation of non-anticipativity constraints may be written as

$$P(\lambda) = \min\left\{\sum_{j=1}^{r}[R_j(x^j, y^j, \mu^j) - \mu^j x.] : (x^j, y^j) \in S^j\right\}, \quad \textbf{(6)}$$

where $R_j(x^j, y^j, \mu^j) = p^j(cx^j + q^j y^j) + \mu^j x^j$ for $j = 1,...,r$ and the parameter $(\mu^j)^T \in \mathbb{R}^{n_1}$. The Lagrangian (6) is separable into $P(\mu^1,...,\mu^r) = \sum_{j=1}^{r} P_j(\mu^j)$, where

$$P_j(\mu^j) = \min\left\{R_j(x^j, y^j, \mu^j) : (x^j, y^j) \in S^j\right\}, \quad \textbf{(7)}$$

with the condition $\sum_{j=1}^{r} \mu^j = 0$ required for boundedness of the Lagrangian. The Lagrangian dual is expressed as

$$c_{LD} = \max_{\mu^1,...,\mu^r}\left\{P(\mu^1,...,\mu^r) : \sum_{j=1}^{r} \mu^j = 0\right\}. \quad \textbf{(8)}$$

The non-anticipativity representation (5), on the other hand, leads to the Lagrangian relaxation in the form

$$D(\lambda) = \min\left\{\sum_{j=1}^{r} L_j(x^j, y^j, \lambda) : (x^j, y^j) \in S^j\right\}, \quad \textbf{(9)}$$

where $L_j(x^j, y^j, \lambda) = p^j(cx^j + q^j y^j) + \lambda(H^j x^j)$ for $j = 1,...,r$, where the vector $\lambda = (\lambda^1,...,\lambda^{r-1})$ and the vector $(\lambda^j)^T \in \mathbb{R}^{n_1}$. The Lagrangian (9) is separable into $D(\lambda) = \sum_{j=1}^{r} D_j(\lambda)$, where

$$D_j(\lambda) = \min\left\{L_j(x^j, y^j, \lambda) : (x^j, y^j) \in S^j\right\}. \quad \textbf{(10)}$$

The Lagrangian dual problem then becomes the problem

$$z_{LD} = \max_{\lambda} D(\lambda). \quad \textbf{(11)}$$

The Lagrangian dual (11) is a convex non-smooth program and can be solved using subgradient methods.

Due to the integer requirements in Eq. (2), a duality gap may occur between the optimal value of the Lagrangian dual (11) and the optimal value of Eq. (2) as described in the proof of Proposition 2 in [3]. The Lagrangian dual (11) provides lower bounds on the optimal value of Eq. (2) and the optimal solutions of the Lagrangian relaxation. In general, these first-stage solutions will not coincide unless the duality gap vanishes. The DD algorithm employs a branch and bound procedure that uses Lagrangian relaxation of non-anticipativity constraints as lower bounds [3].

STEP 1 Initialization: Set $z^* = \infty$ and let $\mathcal{P}$ consist of problem (2).

STEP 2 Termination: If $\mathcal{P} = \varnothing$ and $z^* < \infty$, then x^* with $z^* = cx^* + Q(x^*)$ is optimal.

STEP 3 Node selection: Select and delete a problem P from $\mathcal{P}$, solve its Lagrangian dual (11). If the associated optimal value $z_{LD}(P)$ equals infinity go to *STEP 2*.

STEP 4 Bounding: If $z_{LD}(P)$ is greater than z^* go to *STEP 2*. Otherwise proceed as follows; if the first-stage solutions x^j, $j = 1,...,r$, of the subproblems are

(1) identical, then set $z^* := \min\left\{z^*, cx^j + Q(x^j)\right\}$.

(2) not identical, then compute a suggestion $\hat{x} = Heu(x^1,...,x^r)$ using some heuristic. If $\hat{x}$ is feasible then let $z^* := \min\left\{z^*, c\hat{x} + Q(\hat{x})\right\}$. Go to Step 5.

STEP 5 Branching: Select a component $x_{(k)}$ of $\hat{x}$ and add two new problems to $\mathcal{P}$ that differ from P by the additional constraint $x_{(k)} \le \lfloor \hat{x}_{(k)} \rfloor$ and $x_{(k)} \ge \lfloor \hat{x}_{(k)} \rfloor + 1$, respectively, if $x_{(k)}$ is integer, or $x_{(k)} \le \hat{x}_{(k)} - \varepsilon$ and $x_{(k)} \ge \hat{x}_{(k)} + \varepsilon$, respectively, if $x_{(k)}$ is continuous. The value of $\varepsilon > 0$ must be chosen such that the two new problems have disjoint subdomains. Go to *STEP 3*.

2.3. Progressive Hedging

Proposed by Rockafellar and Wets [19], the progressive hedging (PH) algorithm is a scenario decomposition method for stochastic programs motivated by augmented Lagrangian theory. By decomposing the extensive form into scenario subproblems, the PH algorithm effectively reduces the computational burden of solving extensive forms directly, especially for large-scale problem. Solving scenario subproblems separately can also take advantage of any special structures that are present.

A scenario solution is said to be admissible if it is feasible in one scenario; a scenario solution is said to be implementable or non-anticipative if its first-stage decision is scenario-independent; a solution is feasible if it is both admissible and implementable. The idea of the PH algorithm is to aggregate the admissible solutions of modified scenario subproblems, which progressively

causes the aggregated solution to be non-anticipative and optimal. The modified scenario subproblem comes from scenario decomposition of the augmented Lagrangian as a close approximation of problem (3). The modified cost function includes a penalty term relative to the non-anticipativity constraint and a proximal term that measures the deviation of the scenario solution from the aggregated solution for first-stage decisions. The weight vector $w \in \mathbb{R}^{n_1 \times s}$ is updated by the penalty parameter (vector) $\rho > 0$ in each iteration. This weight update rule is essential to the proofs of the convergence theorems [19].

The PH algorithm has been proven to converge when all decision variables are continuous and can serve as a heuristic in the mixed-integer case. The basic PH algorithm for two-stage stochastic mixed-integer programs proceeds as follows [8]:

STEP 1 Initialization: Let $v := 0$ and $w_v^j := 0$, $j = 1,...,r$. For each $j = 1,...,r$, compute

$$(x_{v+1}^j, y_{v+1}^j) := \arg\min_{x^j, y^j} \left\{ cx^j + q^j y^j : (x^j, y^j) \in S^j \right\}$$

STEP 2 Iteration update: $v \leftarrow v + 1$

STEP 3 Non-anticipative policy: $\hat{x}_v := \sum_{j=1}^{r} p^j x_v^j$

STEP 4 Weight update: $w_v^j := w_{v-1}^j + \rho(x_v^j - \hat{x}_v),\ j = 1,...,r$

STEP 5 Decomposition: For each $j = 1,...,r$, compute

$$(x_{v+1}^j, y_{v+1}^j) := \arg\min_{x^j, y^j} \left\{ cx^j + q^j y^j + w_v^j x + \frac{\rho}{2} \left\| x - \hat{x}_v \right\|^2 : (x^j, y^j) \in S^j \right\}$$

STEP 6 Termination: If all the first-stage scenario solutions x_{v+1}^j agree, then stop. Otherwise, return to Step 2.

While convergence is not guaranteed for mixed-integer problems, computational studies have shown that the PH algorithm can find high-quality solutions within a reasonable number of iterations [25]. The PH algorithm also applies to multi-stage stochastic programs with discrete variables in any stage.

3. Integration of PH and DD

In view of the fact that the PH algorithm can find high-quality solutions within a reasonable number of iterations but is not guaranteed to converge in the mixed-integer case and the DD algorithm is exact but may be slow, fast progress in early iterations of PH could speed up

convergence of DD to an exact solution if the PH algorithm can be combined with the DD algorithm. We now demonstrate how PH and DD can be integrated through their lower bounds. We first review the lower bounding technique for the PH algorithm proposed by Gade et al. [8] and recall equivalence between the best lower bounds obtained by the PH algorithm and the Lagrangian dual from the DD algorithm. Finally, we establish relationships between PH weights and DD multipliers.

3.1. Lower bounds for PH

Although the PH algorithm has been successfully applied as a heuristic to solve multi-stage stochastic mixed-integer programs, it is limited by the lack of convergence guarantee as well as the lack of information to evaluate solution quality relative to the optimal objective. Gade et al. [8] corrected this deficiency of the PH algorithm by presenting a method to compute lower bounds in PH for two-stage and multi-stage stochastic mixed-integer programs. This not only allows us to assess the quality of the solutions in each iteration, but also can provide lower bounds for solution methods, such as branch-and-bound, that rely on them. We restate Proposition 1 of [8], which shows that the weights w define implicit lower bounds, $D(w)$, on the optimal objective value of denoted by z^*.

Proposition 1 [8]. Let $w^j, j=1,...r,$ satisfy $\sum_{j=1}^{r} p^j w^j = 0$. Let

$$D_j(w^j) := \min\left\{ p^j(cx^j + q^j y^j + w^j x^j) : (x^j, y^j) \in S^j \right\}. \quad \textbf{(12)}$$

Then $D(w) := \sum_{j=1}^{r} D_j(w^j) \le z^*$.

It can be verified $\sum_{j=1}^{r} p^j w^j = 0$ is maintained in every iteration by the weight update rule.

Proposition 1 indicates that one can compute a lower bound on z^* in any iteration of the PH algorithm using the current weights with approximately the same effort as one PH iteration.

3.2. Information exchange between PH and DD

Theorem 5.1. of Rockafellar and Wets [19] states that, in the convex case, the sequence $\{(\hat{x}^v, \rho^{-1}w^v)\}_{v=1}^{\infty}$ from PH converges to a pair $(x^*, \rho^{-1}w^*)$ such that x^* solves the primal problem and w^* solves the dual problem. In the mixed-integer case, however, a duality gap may occur because of the introduced nonconvexity. We restate Proposition 2 in [3], which follows from Theorem II.3.6.2 in [27], to provide insight into why this duality gap arises.

Proposition 2. The optimal value z_{LD} of the Lagrangian dual (11) equals the optimal value of the linear program

$$\min\left\{\sum_{j=1}^{r} p^j(cx^j + q^j y^j) : (x^j, y^j) \in conv\, S^j,\ j = 1,...,r, x^1 = ... = x^r\right\}, \quad \textbf{(13)}$$

where *conv* denotes convex hull.

Gade et al. [8] show that by applying the PH algorithm to the linear program (13), one can recover both primal and dual optimal solutions to (13) and (11), respectively. Furthermore, the best PH lower bound $D(w)$ obtained from (12) equals the Lagrangian dual z_{LD} from (11) and c_{LD} obtained from (8). Since both PH and DD can decompose by scenario, the equivalence between $D(w)$ and z_{LD} can be realized by the equivalence for each scenario, that is, $D_j(w^j)$ from (12) equals $Q_j(\mu^j)$ from (7) and $D_j(\lambda)$ from (10). Based on this observation, the equivalence can be established by letting $p^j w^j = \lambda^j$ for the non-anticipativity representation of Lulli and Sen and Lubin et al. and $p^j w^j = \lambda H^j$ for that of Carøe and Schultz. More generally, this information exchange can be applied in any iteration of the PH algorithm to obtain a starting point for solving the Lagrangian relaxation in the DD algorithm. We will illustrate a software implementation of the weight exchange method in detail in the next section.

4 Implementation

4.1 DDSIP – Implementation of DD

DDSIP [16] is a C package for the Dual Decomposition algorithm of Carøe and Schultz for two-stage stochastic mixed-integer programs. Its main idea is the Lagrangian relaxation of the non-anticipativity constraints and it uses a branch-and-bound algorithm to reestablish non-anticipativity. The dual optimization employs ConicBundle [10] provided by C. Helmberg as an implementation of the proximal bundle method [11]. The mixed-integer scenario subproblems in the branch-and-bound tree are solved using CPLEX [28].

4.2 PySP – Implementation of PH

PySP [26] is an open-source software package for modeling and solving stochastic programs by leveraging the combination of a high-level programming language (Python) and the embedding of the base deterministic model in that language (Pyomo [9]). It provides an implementation of PH for stochastic programs. One must specify both the deterministic base model and the scenario tree model to formulate a stochastic program in PySP. The PySP library also provides a generic

implementation of the lower bounding method for the PH algorithm in a plugin called phboundextension.py.

In the application of PH, a significant trade-off in terms of the speed of convergence and quality of the solution is observed as the PH parameter, ρ , is varied, indicating that larger values of a scalar ρ can accelerate the convergence of PH while lower values of ρ can improve the quality of solutions and lower bounds [8]. Watson and Woodruff [25] developed a heuristic method for selecting per-element $\rho(i)$ called SEP that will allow the updates to proceed more quickly to a "good" value w^* of the weight w. The value of the ρ component for an integer variable with index i is determined after PH iteration 0 by setting $\rho(i) = c(i)/(x^{\max} - x^{\min} + 1)$, where $c(i)$ is the corresponding cost coefficient, $x^{\max}(i) = \max_j x_1^j(i)$ and $x^{\min}(i) = \min_j x_1^j(i)$. The primary advantage of the SEP selection heuristic is its problem-independent nature. However, there is a high likelihood that more effective methods exist for any specific problem. For instance, Watson and Woodruff [25] have observed that the best performing alternative for a class of stochastic mixed-integer resource allocation programs is a straightforward yet effective "cost-proportional" method that sets $\rho(i)$ equal to a multiple $k > 0$ of the element unit cost $c(i)$. This method is denoted by $CP(k)$. As a control measure in our computational results, various fixed, global values of ρ denoted by $FX(\cdot)$ are used. The FX stands for fixed and the argument gives the scalar value of ρ.

4.3 Weight exchange between PySP and DDSIP

DDSIP allows three ways to represent the non-anticipativity constraints in problem (3):

NONANT1: $x^1 = x^2, x^1 = x^3, ..., x^1 = x^r$ **(14)**

NONANT2: $x^1 = x^2, x^2 = x^3, ..., x^{r-1} = x^r$ **(15)**

NONANT3: $x^i = \sum_{j=1}^{r} p^j x^j, \forall i = 1, ..., r-1$ **(16)**

By writing the three sets of equalities in the form $\sum_{j=1}^{r} H^j x^j = 0$ as in Lagrangian relaxation (5), the matrices H^j for representation (14) are $H^1 = \begin{bmatrix} I_{n_1} \\ \vdots \\ I_{n_1} \end{bmatrix}, H^2 = \begin{bmatrix} -I_{n_1} \\ 0_{n_1} \\ \vdots \\ 0_{n_1} \end{bmatrix}, \ldots, H^r = \begin{bmatrix} 0_{n_1} \\ \vdots \\ 0_{n_1} \\ -I_{n_1} \end{bmatrix}$, the

matrices H^j for representation (15) are $H^1 = \begin{bmatrix} I_{n_1} \\ 0_{n_1} \\ \vdots \\ 0_{n_1} \end{bmatrix}, H^2 = \begin{bmatrix} -I_{n_1} \\ I_{n_1} \\ 0_{n_1} \\ \vdots \\ 0_{n_1} \end{bmatrix}, \ldots, H^n = \begin{bmatrix} 0_{n_1} \\ \vdots \\ 0_{n_1} \\ -I_{n_1} \end{bmatrix}$ and the

matrices H^j for representation (16) are

$H^1 = \begin{bmatrix} diag(p^1 - 1) \\ diag(p^1) \\ \vdots \\ diag(p^1) \end{bmatrix}, H^2 = \begin{bmatrix} diag(p^2) \\ diag(p^2 - 1) \\ diag(p^2) \\ \vdots \\ diag(p^2) \end{bmatrix}, \ldots, H^{n-1} = \begin{bmatrix} diag(p^{n-1}) \\ \vdots \\ diag(p^{n-1}) \\ diag(p^{n-1} - 1) \end{bmatrix}$ and

$H^n = \begin{bmatrix} diag(p^n) \\ \vdots \\ diag(p^n) \end{bmatrix}$ where $diag(x)$ is a $(n_1 \times n_1)$ matrix with x on the main diagonal.

Motivated by the equivalence between the best PH lower bound and the Lagrangian dual of the linear programming relaxation, we equate the corresponding objective function coefficients in the bounding subproblems for each scenario; i.e., $p^j w^j = \lambda H^j$. This equation enables the information exchange between PH and the Lagrangian dual. Given a weight w from PH, the corresponding Lagrangian multiplier vector $\lambda = [-p^2 w^2, \ldots, -p^n w^n]$ for representation (14), $\lambda = [\sum_{j=1}^{1} p^j w^j, \sum_{j=1}^{2} p^j w^j, \ldots, \sum_{j=1}^{n-1} p^j w^j]$ for representation (15) and $\lambda = [p^1(w^1 - w^n), \ldots, p^{n-1}(w^{n-1} - w^n)]$ for representation (16).

A model-dependent user-defined PySP extension called ddextension.py is used to create input files for DDSIP from the PySP input files and the PH results. While DDSIP allows the specification of various types of starting information such as an initial feasible solution or cost bound, in this paper we focus on providing starting values of the multipliers for solving the Lagrangian dual.

5 Numerical Results

In this section, we study the impact of DDSIP starting multipliers on the run-time of DDSIP for stochastic mixed-integer instances. We consider summary results of the performance of DDSIP starting multipliers on a number of stochastic server location instances. We investigate the interaction between the strategies for choosing the PH ρ parameter and the quality of DDSIP starting multipliers on a stochastic unit commitment problem. We further examine various types of starting information such as multipliers combined with initial solutions for DDSIP on a stochastic modified WECC-240 instance. All the experiments are conducted on Linux Mint 13 running as a virtual machine (3.7 GB RAM with one core at 3.1 GHz).

5.1 Server Location

The stochastic server location problem (SSLP) is a two-stage stochastic mixed-integer program widely applied in a variety of domains such as network design of electric power, internet server and telecommunications systems. The goal is to find the optimal server locations to minimize the investment costs minus the revenue while satisfying the clients' demand and not exceeding the servers' capacities. First-stage variables decide whether to locate a server at each potential position and second-stage variables assign the clients to the servers. A "scenario" specifies a subset of potential clients that are present. As we examine the following empirical results, SSLP instances are named m.n.s, where m is the number of potential server locations, n is the number of potential clients and s is the number of scenarios. The data for each instance are available as three text files in SMPS format (http://www2.isye.gatech.edu/~sahmed/siplib/sslp/sslp.html).

We compare DDSIP run-times required to reduce the relative duality gap below 0.001 with and without starting multipliers on a set of SSLP instances. Several parameters can be set to tune the performance of DDSIP for a particular problem or instance, including the frequency with which the Lagrangian dual is solved in the branch-and-bound tree and the number of iterations for which ConicBundle is allowed to run. We first experimented with these DDSIP parameters. The DDSIP performs the best with regard to the running time without starting multipliers when the Lagrangian dual is solved in every 10^{th} node and the Lagrangian dual is allowed to run for 2 iterations for each SSLP instance. Therefore, this DDSIP parameter setting is used for each run of SSLP instances. The PH ρ parameter selection methods are explored for each SSLP instance and the PH algorithm is allowed to converge. The DDSIP run-time results in Table 4 are obtained using the best PH ρ parameter selection method for each instance, which is specified in the second column of Table 1. As demonstrated in Table 1, starting multipliers derived from PH weights can reduce DDSIP run-time by up to 50% in stochastic server location instances.

Table 1: DDSIP run-time results on a set of SSLP instances.

DDSIP run-time (seconds)		Non-anticipativity representation					
		NONANT1		NONANT2		NONANT3	
SSLP instance	ρ selection method	With or without starting multipliers from PH					
		Without	With	Without	With	Without	With
5.50.500	FX(10)	181	140	148	133	154	81
5.50.1000	FX(10)	651	534	700	500	567	342
5.50.1500	FX(10)	1088	974	1060	959	1071	963
10.50.50	CP(1)	112	74	98	77	99	78
10.50.100	CP(1)	238	175	238	200	240	193
10.50.500	CP(1)	1777	1221	1367	1033	1476	1122
15.45.10	CP(1)	96	46	95	46	95	45
15.45.15	CP(1)	259	123	246	169	281	235

5.2 Unit Commitment

The unit commitment problem to schedule electricity generating units over a given time horizon is extensively used in daily system operation. The uncertainty in net load associated with inaccurate demand forecasts and unpredictable power output from variable generation units has traditionally been managed by deterministically derived reserve margins [18]. Stochastic unit commitment explicitly accounts for the uncertainty via probabilistic scenarios. The objective is to minimize the expected total operational cost such that load is satisfied in all scenarios, subject to operational constraints such as ramp rate limits, minimum startup and shutdown times, and power flow limits on transmission lines. The first-stage variables are on/off decisions for the generators which incur startup, no-load and shutdown costs. The second-stage variables include scenario-specific power output levels. We use the model of Carrión and Arroyo [4] as our core deterministic optimization model [7].

We first execute on a 5 bus test case of the AMES wholesale power market test bed system [22], augmented with additional unit commitment extensions [5]. The instance includes 5 generators, 5 buses and 6 transmission lines with a scheduling horizon of 24 hours in hourly increments. We consider 10 equally likely scenarios for the sequence of hourly loads. The extensive form of this instance has 16,194 variables (1,200 binary) and 24,092 constraints.

Table 2 shows the running times required for DDSIP to reduce the relative duality gap below 0.001 for different parameter values, both without any starting information and with starting multipliers obtained from the final weights obtained by fixing the PH penalty parameter $\rho = 1$ and allowing the PH algorithm to converge. In Table 2, CBFREQ specifies the frequency of solving the Lagrangian dual using ConicBundle, and CBITLI specifies the limit for the number of descent steps in solving the Lagrangian dual.

Table 2: DDSIP run-time with different ConicBundle parameters for the 5-bus instance.

DDSIP run-time (seconds)	Non-anticipativity representation					
	NONANT1		NONANT2		NONANT3	
ConicBundle parameter (CBFREQ, CBITLI)	With or without starting multipliers from PH					
	Without	With	Without	With	Without	With
(1, 5)	2164	237	2179	249	2624	278
(1, 1000)	714	203	2014	263	477	165
(100, 20)	527	156	603	139	149	141
(50, 10)	271	73	426	102	654	102

Table 2 displays only a selection of the DDSIP parameters we have explored. Among all the DDSIP parameters we have experimented with, the DDSIP parameters of (50, 10) perform the best with regard to DDSIP run-time without starting multipliers. Therefore, we adopt (50, 10) as the DDSIP parameter setting for further experiments on this 5 bus test case.

Next, we consider the interaction between the value of PH parameter ρ and the quality of DDSIP starting multipliers derived from PH weights. We vary the strategy to compute PH ρ values for DDSIP starting multipliers. The results are shown in Table 3, where data in the row labeled FX(1) are repeated from Table 2. Even though we chose the DDSIP parameters with the shortest DDSIP running time without starting multipliers, the starting multipliers transformed from PH weights can reduce the DDSIP running time by roughly an order of magnitude in this instance as demonstrated by Table 2.

Table 3: DDSIP run-time with starting multipliers from PH using different ρ computation strategies for the 5-bus instance.

DDSIP run-time (seconds)	Non-anticipativity representation		
PH ρ value selection method	NONANT1	NONANT2	NONANT3
No starting multipliers	271	426	654
FX(1)	73	102	102
FX(10)	94	85	48
FX(30)	77	90	69
CP(10)	32	163	40
SEP(10)	75	121	76

To assess the performance of DDSIP starting multipliers on utility-scale systems, we test on a stochastic WECC-240 instance with 5 scenarios. The WECC-240 instance is introduced in [24], which provides a simplified description of the western US interconnection. This instance consists of a single bus and 85 generators with a scheduling horizon of 48 hours in hourly increments. Because it was originally introduced to assess market design alternatives, we have modified this instance to capture characteristics more relevant to reliability assessment, including startup, shutdown, and nominal ramping limits, startup cost curves, and minimum up and down times. The full set of modifications and the case itself can be obtained by contacting the authors. The instance has 31,674 variables (4,080 binaries) and 59,374 constraints for a single scenario problem.

Table 4 reports the DDSIP run-time required to reduce the optimality gap below 2% and the optimality gap of the resulting solution with or without DDSIP starting multipliers on the WECC-240 stochastic instance. The DDSIP parameter is set to be (50, 10) for each run. Moreover, we study various types of DDSIP starting information by providing both starting multipliers and initial solutions for solving the Lagrangian dual from the final iteration of PH. Based on extensive exploration of ρ-setting strategies, in the PH run we choose CP(0.1) to compute PH parameter ρ value and limit the number of PH iterations to 100. Without starting information, DDSIP cannot reduce the optimality gap below 99% within 24 hours. Supplying starting multipliers derived from PH weights, however, allows DDSIP to converge to a near-

optimal solution within minutes. By also supplying the primal solution from PH, the DDSIP run-time is further reduced by up to an order of magnitude.

Table 4: DDSIP run-time and optimality gap on WECC-240 stochastic instance with various starting information. The symbol * denotes failure to converge within 24 hours.

DDSIP run-time (seconds) and optimality gap	Non-anticipativity representation					
	NONANT1		NONANT2		NONANT3	
DDSIP Starting information	Run-time	Opt. Gap	Run-time	Opt. Gap	Run-time	Opt. Gap
None	*	-	*	-	*	-
Multipliers only	877	1.98%	1937	1.82%	5056	1.82%
Both multipliers and solutions	671	1.88%	777	1.99%	646	1.94%

Acknowledgments

This work was funded by the US Department of Energy's Advanced Research Projects Agency – Energy. We are grateful to Ralf Gollmer for his consistent assistance with DDSIP software. Sandia is a multi-program laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

References

[1] J.R. Birge, F. Louveaux, Introduction to Stochastic Programming, second ed., Springer, New York, 2011

[2] C.C. Carøe, J. Tind, L-shaped decomposition of two-stage stochastic programs with integer recourse, Technical Report, Institute of Mathematics, University of Copenhagen, 1995

[3] C.C. Carøe, R. Schultz, Dual decomposition in stochastic integer programming, Operations Research Letters 24 (1-2) (1999) 37-45

[4] M. Carrión, J.M. Arroyo, A computationally efficient mixed-integer linear formulation for the thermal unit commitment problem, IEEE Trans. Power Systems 21 (2006) 1371-1378

[5] E. Ela, M. Milligan, M. O'Malley, A flexible power system operations simulation model for assessing wind integration, Power and Energy Society General Meeting, 2011 IEEE

[6] Y. Fan, C. Liu, Solving stochastic transportation network protection problem using the progressive hedging-based method, Networks and Spatial Economics 10 (2) (2010) 193–208

[7] Y. Feng, S.M. Ryan, Solution sensitivity-based scenario reduction for stochastic unit commitment, Computational Management Science (2014) 1-34

[8] D. Gade, G. Hackebeil, S.M. Ryan, C. J.-P. Watson, R.J.B. Wets, D.L. Woodruff, Obtaining lower bounds from the progressive hedging algorithm for stochastic mixed-integer programs, under review

[9] W.E. Hart, J.-P. Watson, D.L. Woodruff, Pyomo: Modeling and solving mathematical programs in Python, Mathematical Programming Computation 3 (3) (2011) 219-260

[10] C. Helmberg, The ConicBundle library for convex optimization, Available at: http://www-user.tu-chemnitz.de/helmberg/ConicBundle/

[11] K.C. Kiwiel, Proximity control in bundle methods for convex nondifferentiable optimization, Mathematical Programming 46 (1990) 105-122

[12] G. Laporte, F.V. Louveaux, The integer L-shaped method for stochastic integer programs with complete recourse, Operations Research Letters 13 (1993) 133-142

[13] A. Løkketangen, D.L. Woodruff, Progressive hedging and tabu search applied to mixed integer (0,1) multi-stage stochastic programming, Journal of Heuristics, 2 (2) (1996) 111-128

[14] M. Lubin, K. Martin, C. Petra, B. Sandıkçı, On parallezing dual decomposition in stochastic integer programming, Operations Research Letters 41 (3) (2013) 252-258

[15] G. Lulli, S. Sen, A branch-and-price algorithm for multistage stochastic integer programming with application to stochastic batch-sizing problems, Management Science 50 (6) (2004) 786-796

[16] A. Märkert, R. Gollmer, User's Guide to ddsip – A C Package for the Dual Decomposition of Two-Stage Stochastic Programs with Mixed-Integer Recourse, Available at: http://plato.asu.edu/ddsip-man.pdf

[17] O. Listes, R. Dekker, A scenario aggregation based approach for determining a robust airline fleet composition, Transportation Science 39 (3) (2005) 367–382

[18] S.M. Ryan, C. Silva-Monroy, J.P. Watson, R.J.B. Wets, D.L. Woodruff, Toward scalable, parallel progressive hedging for stochastic unit commitment, in Proceedings of the 2013 IEEE Power and Energy Society General Meeting

[19] R.T. Rockafellar, R.J.-B. Wets, Scenarios and policy aggregation in optimization under uncertainty, Mathematics of Operations Research 16 (1) (1991) 119-147

[20] S. Sen, Algorithms for stochastic mixed-integer programming models, in: K. Aardal, G.L. Nemhauser, R. Weismantel (Eds.), Discrete Optimization, Elsevier, 2005, pp. 515-558.

[21] R. Van Slyke, R.J.B. Wets, L-Shaped Linear Programs with Applications to Optimal Control and Stochastic Programming, SIAM Journal of Applied Mathematics 17 (4) (1969) 638-663.

[22] J. Sun, L. Tesfatsion, Dynamic testing of wholesale power market designs: An open-source agent-based framework, Computational Economics 30 (3) (2007) 291-327

[23] S. Takriti, J.R. Birge, E. Long, A stochastic model for the unit commitment problem, IEEE Trans. Power Systems 11 (1996) 1497-1508

[24] J.E. Price, J. Goodin, Reduced network modeling of WECC as a market design prototype, Proceedings of the 2011 IEEE Power and Energy Society General Meeting

[25] J.-P. Watson, D.L. Woodruff, Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems, Computational Management Science 8 (4) (2011) 355-370

[26] J.-P. Watson, D.L. Woodruff, W.E. Hart, PySP: Modeling and solving stochastic programs in Python, Mathematical Programming Computation 4 (2) (2012) 109-149

[27] L.A. Wolsey, G.L. Nemhauser, Integer and Combinatorial Optimization, Wiley-Interscience, New York, 1988

[28] IBM ILOG CPLEX Optimization Studio V12.6, Available at: http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud