Parallel Scenario Decomposition of Risk Averse 0-1 Stochastic Programs

Yan Deng* Shabbir Ahmed† Siqian Shen‡

January 29, 2016

Abstract

In this paper, we extend a recently proposed scenario decomposition algorithm (Ahmed (2013)) for risk-neutral 0-1 stochastic programs to the risk-averse setting. Specifically, we consider risk-averse 0-1 stochastic programs with objective functions based on coherent risk measures. Using a dual representation of a coherent risk measure, we first derive an equivalent minimax reformulation of the considered problem. We then develop three variants of the scenario decomposition algorithm for this minimax formulation based on different relaxations of the nonanticipaticity constraints. The algorithms proceed by solving scenario subproblems to obtain candidate solutions and bounds, and subsequently cutting off the candidate solutions from the search space to achieve convergence to an optimal solution. We design three parallelization schemes for implementing the algorithms with different tradeoffs between communication time and computation time. Our computational results with risk-averse extensions of two standard stochastic 0-1 programming test instances demonstrate the scalability of the proposed decomposition and parallelization framework.

Keywords: risk-averse 0-1 stochastic programs; conditional value-at-risk (CVaR); minimax optimization; dual decomposition; distributed algorithms; parallel computing

1 Introduction

Stochastic programming is valuable framework for decision-making under uncertainty in many application areas. In the two-stage setting, typically the uncertainty is modeled as a set of scenarios and the decisions are in two stages - some that are made *here-and-now*, and others that can be determined after observing realizations of the uncertain scenarios, i.e., *wait-and-see* decisions (see, e.g., Birge and Louveaux, 2011). In many applications, the here-and-now variables are

*Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI, USA 48109; email: yandeng@umich.edu

†School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, USA 30332; email: sahmed@isye.gatech.edu

‡Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI, USA 48109; email: siqian@umich.edu

often binary, representing "yes" or "no" decisions, e.g., in facility location analysis, production planning, and project selection. The computation of 0-1 stochastic programs is challenging, due to the discrete nature of variables and dimension of the realizations of uncertainty. Among many decomposition-based algorithms for solving stochastic programs, one approach constructs a reformulation of the considered problem, which makes copies of the here-and-now decision variables, one copy for each scenario, and formulates "nonanticipativity constraints" to require these copies to take the same solution values. A Lagrangian relaxation can then be attained by relaxing the nonanticipativity constraints, whose computation is decomposable by scenario subproblems and solvable in a distributed framework.

The above technique is known as scenario decomposition or dual decomposition, and was first introduced by Rockafellar and Wets (1976). We refer to Shapiro et al. (2009) for other general results on dual decomposition of stochastic programs. Carøe and Schultz (1999) apply dual decomposition to obtain strong relaxations of two-stage stochastic integer programs, and generate bounds within a branch-and-bound framework. Dentcheva and Römisch (2004) analyze the Lagrangian relaxation of multistage stochastic programs with nonconvex constraints arising from logical and integrality conditions, and demonstrate the efficacy of scenario decomposition as compared to other decomposition algorithms. Heuristic approaches, including variants of the "progressive hedging" algorithm (see Watson and Woodruff, 2011), are designed based on creating subproblems that can be decomposed by scenario and penalizing violations of nonanticipativity constraints via updating the Lagrangian dual to achieve consensus among all the subproblems to seek feasible primal solutions. Watson et al. (2010) generalize the approach to solving chance-constrained programs, where they relax both the nonanticipativity constraints and the knapsack constraint for reformulating a chance constraint as an extended mixed-integer linear program. For 0-1 stochastic problems, due to the absence of strong duality, these Lagrangian-based scenario decomposition heuristics do not have theoretical convergence guarantees. However, they have been demonstrated to have good empirical performance for solving stochastic programs in a variety of applications (see, e.g., Crainic et al., 2014, 2011).

Recently, Ahmed (2013) proposes a scenario decomposition variant for solving 0-1 stochastic programs with expectation-based objective functions. The key idea of the algorithm is to identify an optimal solution by iteratively bounding the optimal objective value and cutting-off candidate solutions obtained from scenario subproblems. The author describes serial and synchronous distributed implementations of the algorithm, and demonstrates the efficacy of the approach by solving instances from the SIPLIB test library (see Ahmed et al., 2015a). Ryan et al. (2015) extend the work of Ahmed (2013) by developing an asynchronous parallel implementation of the algorithm, which has better performance on the same set of test instances. Ahmed et al. (2015b) generalize the scenario decomposition approach for chance-constrained binary programs and derive the Lagrangian dual problems based on relaxing nonanticipativity constraints. In particular, they demonstrate the strength of the dual bounds and efficient ways of obtaining them in a distributed algorithm, although parallel implementation of the proposed algorithms are not explored.

Parallel implementations of scenario decomposition have been discussed in Mulvey and Ruszczyński

(1995); Ryan et al. (2015); Ahmed (2013); Lubin et al. (2013). Mulvey and Ruszczyński (1995) proposes a novel parallel decomposition algorithm for multistage stochastic optimization problems, in which the subproblems are solved using a nonlinear interior point algorithm, and modified by separable quadratic terms to coordinate the scenario solutions. Ahmed (2013) develops a synchronous parallel implementation scheme for a proposed scenario decomposition approach for 0-1 stochastic programs. There a synchronization barrier is placed across the processes after solving the scenario subproblems and evaluating candidate solutions before coordinating scenario solutions. Ryan et al. (2015) then proposes an asynchronous scheme that removes the barriers and instead relies on a master-worker structure, which demonstrate better parallel efficiency on the same set of test instances. Lubin et al. (2013) revisit the dual decomposition algorithm of Carøe and Schultz (1999) from a computational perspective, and develop a synchronous parallel implementation scheme where load imbalance is identified as a barrier to parallel scalability. We further refer the interested readers to Ruszczyński (1993); Birge and Rosa (1996); Nielsen and Zenios (1997); Birge and Rosa (1996); Linderoth and Wright (2003, 2005); Birge et al. (1996); Fragniere et al. (2000) for parallel implementation of other decomposition methods.

The scenario decomposition approaches discussed above focus on risk-neutral expected value objectives. Focusing on scenario decomposition of risk-averse stochastic programs, Miller and Ruszczyński (2011) consider a risk-averse two-stage stochastic linear program, in which the objective function is given as a composition of conditional risk measures. They develop two decomposition algorithms, of which one uses a generic cutting plane approach and the other exploits the composite structure of the objective function. Collado et al. (2012) propose a scenario decomposition algorithm with convergence guarantee for a risk-averse multistage stochastic program with a finite scenario tree. The main idea of the algorithm lies on the construction of risk-neutral approximations of the program by exploiting specific structure of dynamic risk measures. However, both papers did not consider integer variables, and neither explored procedures for implementing the decomposition algorithms in parallel.

In this paper, we consider risk-averse stochastic 0-1 programs (see, e.g., Ruszczyński, 2013) with uncertain parameter that follows a discrete probability distribution. In particular, we study coherent risk measures for measuring uncertain outcomes and evaluating the performance of decisions under uncertainty. Our main contributions are as follows. First, we utilize a connection between coherent risk measures and corresponding sets of probability measures in their dual representation (see Shapiro, 2012; Shapiro and Ahmed, 2004) to derive equivalent minimax reformulations of risk-averse 0-1 stochastic programs. Second, we develop three variants of the scenario decomposition algorithm for this minimax formulation based on different relaxations of the nonanticipaticity constraints. The algorithms proceed by solving scenario subproblems to obtain candidate solutions and bounds, and subsequently cutting off the candidate solutions from the search space to achieve convergence to an optimal solution. The algorithms are also applicable for solving distributionally robust risk-averse problems with 0-1 variables, and can be implemented in a distributed framework. Third, we investigate three parallelization schemes, with or without barriers, by trading off communication time and computational time. The goal is to explore special

structures in risk-averse 0-1 stochastic programs to design effective and efficient parallelization algorithms for implementing scenario decomposition. Fourth, we test two classes of 0-1 stochastic programming test problems with diverse problem sizes from the SIPLIB test library, for which we use conditional value-at-risk (CVaR) and mean-risk measures to build their risk-averse variants. Our results demonstrate the computational efficacy of scenario decomposition and also show that the proposed parallel implementation schemes can achieve near-linear speedup.

The remainder of the paper is organized as follows. In Section 2, we present risk-averse 0-1 stochastic programs and their equivalent minimax reformulations, and propose three variants of the scenario decomposition algorithm. In Section 3, we focus on designing parallel schemes to implement the algorithms in a distributed framework and propose improvement strategies by eliminating barriers. In Section 4, we report the computational results of the proposed algorithms on two sets of two-stage 0-1 stochastic programs with different risk objectives and their distributionally robust variants. Section 5 concludes the paper and states future research directions.

2 Scenario Decomposition for Risk-averse 0-1 Stochastic Programs

We consider risk-averse stochastic 0-1 programs of the form

$$\min_{x} \left\{ \rho(f(x,\xi)) : x \in X \subseteq \{0,1\}^d \right\}. \tag{1}$$

where x is a d-dimensional binary decision vector, ξ is a multivariate random vector, and the cost function $f(x,\xi)$ is convex in x for any realized value of ξ. We assume a finite support $\{\xi^1, \ldots, \xi^K\}$ of the random variable ξ, having the corresponding probabilities $p_1, \ldots, p_K$. The vector $p = (p_1, \ldots, p_K)^\top$ belongs to the polyhedral set

$$\mathcal{M} := \left\{ p = (p_1, \ldots, p_K)^\top : \sum_{k=1}^{K} p_k = 1,\ p_k \geq 0,\ \forall k = 1, \ldots, K \right\}. \tag{2}$$

Here $\rho(\cdot)$ is a generic measure that returns a scalar metric of the random cost $f(x,\xi)$. In this paper, we consider $\rho(\cdot)$ to be a coherent risk measure with an equivalent dual representation of the form (see, e.g., Artzner et al., 1999; Shapiro and Ahmed, 2004),

$$\rho(f(x,\xi)) = \max_{q \in \mathcal{Q}_\rho(p)} \left\{ \mathbb{E}_q \left[f(x,\xi) \right] \right\}, \tag{3}$$

where we compute the expectation of the random $f(x,\xi)$ based on an unknown probability vector $q = (q_1, \ldots, q_K)^\top$. Here q belongs to an ambiguity set $\mathcal{Q}_\rho(p) \subseteq \mathcal{M}$, determined by the coherent risk measure ρ and nominal probabilities $p_1, \ldots, p_K$. We later describe a few explicit forms of $\mathcal{Q}_\rho(p)$ when examining specific coherent risk measures in our computations.

According to (3), the value of $\rho(f(x,\xi))$ is equivalent to the worst-case expectation of the random cost $f(x,\xi)$ over the probability distributions $q \in \mathcal{Q}_\rho(p)$. Therefore, we present an equivalent minimax reformulation of the original problem (1):

$$\texttt{MIMA}: \quad \min_{x \in X} \max_{q \in \mathcal{Q}_\rho(p)} \left\{ \sum_{k=1}^{K} q_k f_k(x) \right\} \tag{4}$$

where we denote $f_k(x) \equiv f(x, \xi^k)$ for notation brevity. Throughout this paper, we assume that $\min_{x \in X} \{f_k(x)\}$ is bounded for all $k = 1, \ldots, K$. The goal is to optimize MIMA via scenario decomposition approaches and efficient parallel implementation schemes.

2.1 The Generic Scenario Decomposition Scheme

We solve MIMA using an iterative method that approaches the optimal objective through updating a lower bound ℓ and an upper bound u. The upper bounds are obtained through objective values of feasible x-solutions. The lower bounds are obtained from solving Lagrangian relaxations of MIMA.

Following the standard scenario decomposition approach, we first replace the binary decision variable x with scenario-based copies $x^1, \ldots, x^K$ and reformulate MIMA as

$$\begin{aligned} \min_{x^1,\ldots,x^K \in X} \max_{q \in \mathcal{Q}_\rho(p)} \quad & \sum_{k=1}^{K} q_k f_k(x^k) \\ \text{s.t.} \quad & \sum_{k=1}^{K} \alpha_k x^k = x^1 \end{aligned} \tag{5}$$

where $\alpha_1, \ldots, \alpha_K$ are positive scalars that sum to 1, and (5) are the nonanticipativity constraints (NAC) ensuring that $x^1, \ldots, x^K$ take the same values. Note that the above formulation of the nonanticipativity constraints makes use of the fact that the variables are 0-1. A general formulation would be of the form $x^k = x^1, \ \forall k = 2, \ldots, K$ which, however, would involve $(K-1) \times d$ constraints as compared to the d constraints in (5).

To decompose the problem, we relax (5) and penalize the violation (if any) by a Lagrangian dual multiplier $\lambda \in \mathbb{R}^d$. The resulting Lagrangian relaxation is given by

$$\min_{x^1,\ldots,x^K \in X} \max_{q \in \mathcal{Q}_\rho(p)} \left\{ \sum_{k=1}^{K} \Big((\alpha_k - \delta_k) \lambda^\top x^k + q_k f_k(x^k) \Big) \right\} \tag{6}$$

where for notational convenience, we let $\delta_1 = 1$ and $\delta_k = 0$ for $k = 2, \ldots, K$. To recover scenario subproblems, we consider a lower approximation of the Lagrangian model (6) by interchanging the min and max operators:

$$\begin{aligned} g(\lambda) &:= \max_{q \in \mathcal{Q}_\rho(p)} \min_{x^1,\ldots,x^K \in X} \left\{ \sum_{k=1}^{K} \Big((\alpha_k - \delta_k) \lambda^\top x^k + q_k f_k(x^k) \Big) \right\} \\ &= \max_{q \in \mathcal{Q}_\rho(p)} \left\{ \sum_{k=1}^{K} \min_{x^k \in X} \Big((\alpha_k - \delta_k) \lambda^\top x^k + q_k f_k(x^k) \Big) \right\}. \end{aligned} \tag{7}$$

For any $\lambda \in \mathbb{R}^d$, $g(\lambda)$ represents a global lower bound for MIMA. The overall scenario decomposition algorithm optimizes (7) for given λ to update the lower bound ℓ and also obtains candidate feasible solutions of x from scenario subproblems of (7) to update the upper bound u. At each iteration, we eliminate binary solutions of x that have been evaluated and terminate the algorithm when the gap between u and ℓ is closed.

Next we discuss three scenario decomposition algorithms that differ in how to solve $g(\lambda)$ or its variations to update ℓ. We refer to them as DD1, DD2 and DD3, respectively.

2.2 DD1 by Optimizing $g(0)$

In the first approach, we simply set $\lambda = 0$, and use $g(0)$ to update the lower bound. We essentially relax the NAC (5), and update the lower bound ℓ with

$$\begin{aligned} g(0) &= \max_{q \in \mathcal{Q}_\rho(p)} \left\{ \sum_{k=1}^{K} \min_{x^k \in X} \left\{ q_k f_k(x^k) \right\} \right\} \\ &= \max_{q \in \mathcal{Q}_\rho(p)} \left\{ \sum_{k=1}^{K} q_k \min_{x^k \in X} \left\{ f_k(x^k) \right\} \right\}, \end{aligned} \tag{8}$$

where we are able to pull $q_1, \ldots, q_K$ out of the K inner minimization subproblems, and thus parallelize the computation of each subproblem.

Without going through Lagrangian relaxation steps, we can verify (8) being a valid lower bound to `MIMA` by following an alternative derivation below:

$$\begin{aligned} \min_{x \in X} \rho(f(x,\xi)) &= \min_{x \in X} \max_{q \in \mathcal{Q}_\rho(p)} \mathbb{E}_q[f(x,\xi)] \\ &\geq \max_{q \in \mathcal{Q}_\rho(p)} \min_{x \in X} \mathbb{E}_q[f(x,\xi)] \\ &\geq \max_{q \in \mathcal{Q}_\rho(p)} \mathbb{E}_q \left[\min_{x \in X} \{ f(x,\xi) \} \right] \\ &= \rho \left(\min_{x \in X} f(x,\xi) \right) = (8). \end{aligned}$$

We compute the value of $g(0)$ through (8) in two steps as follows.

- *Step (i)*: for each $k = 1, \ldots, K$, optimize scenario subproblem

$$\textbf{Scen}(k): \quad \beta_k = \min_x \{ f_k(x) : x \in X \}; \tag{9}$$

- *Step (ii)*: letting $\beta = (\beta_1, \ldots, \beta_K)$, optimize a maximization problem over variable q as

$$\textbf{Cont}(\beta): \quad \max_q \left\{ \sum_{k=1}^{K} \beta_k q_k : q \in \mathcal{Q}_\rho(p) \right\}. \tag{10}$$

Note that (8) is the dual representation of a coherent risk $\rho\left(\min_{x \in X} f(x,\xi)\right)$ according to (3). Here the random value $\min_{x \in X} f(x,\xi)$ has finite realizations $\beta_1, \ldots, \beta_K$ computed through Step (i), with the corresponding probabilities $p_1, \ldots, p_K$. Therefore, Step (ii) can be done by optimizing the risk function $\rho\left(\min_{x \in X} f(x,\xi)\right)$ directly if possible, rather than using the dual form in (10).

Algorithm 1 provides the details of the DD1 algorithm, which iterates until ℓ and u are sufficiently close. (We denote ϵ as a gap tolerance value.) In each iteration, we solve scenario subproblems $\textbf{Scen}(k),\ \forall k = 1, \ldots, K$, and compute $g(0)$ to update the lower bound ℓ (steps 4–8).

Each subproblem has the same feasible region as MIMA. Thus, we collect subproblem solutions $\hat{x}_k,\ k = 1, \ldots, K$ into the set S, which are feasible to MIMA. We then use the candidate solutions in set S to update the upper bound u (steps 9–11). This is done by evaluating

$$\textbf{Eval}(\hat{x}) := \rho(f(\hat{x}, \xi)) \tag{11}$$

for given solution $\hat{x} \in S$ (i.e., step (10)). We cut off all the evaluated solutions by adding the corresponding no-good cut

$$\sum_{i=1,\ldots,d:\hat{x}_i=0} x_i + \sum_{i=1,\ldots,d:\hat{x}_i=1} (1 - x_i) \geq 1 \tag{12}$$

to every scenario subproblem **Scen**(k), for $k = 1, \ldots, K$, in future iterations (see step 12).

Algorithm 1 The DD1 scenario decomposition algorithm for solving MIMA

1: $u \leftarrow +\infty$, $\ell \leftarrow -\infty$
2: **repeat**
3: $\quad S \leftarrow \emptyset$
4: $\quad$ **for** $k = 1, \ldots, K$ **do**
5: $\quad\quad (\beta_k, \hat{x}^k) \leftarrow$ **Scen**(k)
6: $\quad\quad S \leftarrow S \cup \{\hat{x}^k\}$
7: $\quad$ **end for**
8: $\quad \ell \leftarrow \max\{\ell,$ **Cont**$(\beta)\}$
9: $\quad$ **for** $\hat{x} \in S$ **do**
10: $\quad\quad u \leftarrow \min\{u,$ **Eval**$(\hat{x})\}$
11: $\quad$ **end for**
12: $\quad X \leftarrow X \setminus S$.
13: **until** $u - \ell < \epsilon$

2.3 DD2 with a Cutting-Plane Approach

Note that in DD1 we do not update the Lagrange multipliers λ. Here we consider an algorithm that combines an inner loop of deriving the maximum lower bound $g(\lambda)$ in variable λ and an outer loop of bounding the objective value and cutting off suboptimal solutions. Following (7), we formulate a master problem as $\max_\lambda \{g(\lambda)\}$ over variables $\phi \in \mathbb{R}$, $\lambda \in \mathbb{R}^d$ and $q \in [0, 1]^K$ as:

$$\max_{\phi,\lambda,q} \quad \phi \tag{13}$$

$$\text{s.t.} \quad \phi \leq \sum\nolimits_{k=1}^{K} \min_{x^k \in X} \left\{ (\alpha_k - \delta_k)\lambda^\top x^k + q_k f_k(x^k) \right\} \tag{14}$$

$$q \in \mathcal{Q}_\rho(p), \tag{15}$$

where constraints (14) are linear in ϕ, q, and λ. We relax (14) and enforce them by generating cuts iteratively. Specifically, given a solution $(\hat{\phi}, \hat{\lambda}, \hat{q})$ to a relaxed master problem, for each $k =$

$1, \ldots, K$, we compute a scenario subproblem given by

$$\beta_k^{\text{DD2}} = \min_x \left\{ (\alpha_k - \delta_k)\hat{\lambda}^\top x + \hat{q}_k f_k(x) : x \in X \right\}. \tag{16}$$

If $\hat{\phi} > \sum_{k=1}^K \beta_k^{\text{DD2}}$, then we add an optimality cut

$$\phi \leq \sum\nolimits_{k=1}^K \left((\alpha_k - \delta_k)\lambda^\top \hat{x}^k + q_k f_k(\hat{x}^k) \right) \tag{17}$$

where $\hat{x}^k$ represents an optimal solution to the subproblem (16), and re-iterate to solve the master problem (13)–(15). We terminate the algorithm when $\hat{\phi} \leq \sum_{k=1}^K \beta_k^{\text{DD2}}$ which implies that constraints (14) are satisfied and $\hat{\phi}$ equals to $\max_\lambda g(\lambda)$. We then update the lower bound ℓ with $\hat{\phi}$. The following relation holds throughout the cutting-plane iterations

$$\sum_{k=1}^K \beta_k^{\text{DD2}} \leq \max_\lambda \{g(\lambda)\} \leq \hat{\phi}.$$

Therefore, even before attaining $\max_\lambda \{g(\lambda)\}$, we can update the lower bound ℓ with $\sum_{k=1}^K \beta_k^{\text{DD2}}$ in every iteration, which may close the gap between ℓ and u earlier.

Remark 1 *For any $\hat{\lambda}$ and nonnegative $\hat{q}$, the subproblem* (16) *must be bounded, because* ***Scen***(k) *is bounded by assumption and $x^k \in X \subseteq \{0,1\}^d$.*

With the above new lower-bound calculation, the new decomposition algorithm is deduced from replacing steps 4–8 in Algorithm 1 by Algorithm 2. To avoid the cutting-plane method taking excessively long time to converge, we can set an upper limit to the number of cutting-plane iterations.

Algorithm 2 Lower bound computation subroutine in DD2.

$\hat{\lambda} \leftarrow 0$, $\hat{q} \leftarrow p$, $\hat{\phi} \leftarrow +\infty$
repeat
  for $k = 1, \ldots, K$ **do**
    solve (16) to attain the optimal objective value β_k^{DD2} and the optimal solution $\hat{x}^k$
    $S \leftarrow S \cup \{\hat{x}^k\}$
  end for
  $\ell \leftarrow \max\{\ell, \sum_{k=1}^K \beta_k^{\text{DD2}}\}$
  add cut (17) to the master problem (13)–(15)
  solve the master problem and attain an optimal solution $(\hat{\phi}, \hat{\lambda}, \hat{q})$
until $\hat{\phi} \leq \sum_{k=1}^K \beta_k^{\text{DD2}}$

2.4 DD3 as a Subgradient-based Algorithm

Given nonnegative weights $\alpha_k,\ k = 1, \ldots, K$ with $\sum_{k=1}^{K} \alpha_k = 1$, unlike in DD1 and DD2, here we consider NACs:

$$\sum_{k=1}^{K} \alpha_k x^k = x^i \quad \forall i = 1, \ldots, K, \tag{18}$$

which are K copies of (5) with the right-hand sides varying from x^1 to x^K. We associate each of these constraints with multiplier $q_i \lambda^i$ where q_i is the probability mass value of a "dual" distribution from the ambiguity set $\mathcal{Q}_\rho(p)$ and λ^i is a d-dimensional variable, for all $i = 1, \ldots, K$. Following similar steps for obtaining $g(\lambda)$ in (7), we formulate a relaxation of MIMA with NAC (18) that replace (5) by

$$\begin{aligned} g(\lambda^1, \ldots, \lambda^K) &:= \max_{q \in \mathcal{Q}_\rho(p)} \min_{x^1, \ldots, x^K \in X} \left\{ \sum_{i=1}^{K} q_i \left((\lambda^i)^\top \left(\sum_{k=1}^{K} q_k x^k - x^i \right) + f_i(x^i) \right) \right\} \\ &= \max_{q \in \mathcal{Q}_\rho(p)} \min_{x^1, \ldots, x^K \in X} \left\{ \sum_{k=1}^{K} q_k \left(f_k(x^k) - \left(\lambda^k\right)^\top x^k \right) + \left(\sum_{k=1}^{K} q_k \lambda^k \right)^\top \left(\sum_{k=1}^{K} q_k x^k \right) \right\}. \end{aligned} \tag{19}$$

Now consider a polyhedral set $A(\lambda^1, \ldots, \lambda^K) = \left\{ q : \sum_{k=1}^{K} \lambda^k q_k = 0 \right\}$. A lower approximation of $g(\lambda^1, \ldots, \lambda^K)$ that has a decomposable inner minimization problem is given by

$$\underline{g}(\lambda^1, \ldots, \lambda^K) := \max_{q \in \mathcal{Q}_\rho(p) \cap A(\lambda^1, \ldots, \lambda^K)} \min_{x^1, \ldots, x^K \in X} \left\{ \sum_{k=1}^{K} q_k \left(f_k(x^k) - \left(\lambda^k\right)^\top x^k \right) \right\} \tag{20}$$

$$= \max_{q \in \mathcal{Q}_\rho(p) \cap A(\lambda^1, \ldots, \lambda^K)} \left\{ \sum_{k=1}^{K} \left(q_k \min_{x^k \in X} \left\{ f_k(x^k) - \left(\lambda^k\right)^\top x^k \right\} \right) \right\}. \tag{21}$$

We compute $\underline{g}(\lambda^1, \ldots, \lambda^K)$ by following the two steps below.

- *Step (i):* for each $k = 1, \ldots, K$, solve a scenario subproblem:

$$\beta_k^{\text{DD3}} = \min_{x \in X} \left\{ f_k(x) - (\lambda^k)^\top x \right\} \tag{22}$$

  which is bounded following Remark 1.

- *Step (ii):* solve a maximization problem over the continuous variable q:

$$\max_{q} \left\{ \sum_{k=1}^{K} \beta_k^{\text{DD3}} q_k : q \in \mathcal{Q}_\rho(p) \cap A(\lambda^1, \ldots, \lambda^K) \right\}. \tag{23}$$

The value of $\underline{g}(\lambda^1, \ldots, \lambda^K)$ for any given multipliers $\lambda^1, \ldots, \lambda^K$, will provide a valid lower bound. We strive to obtain the best lower bound by maximizing $\underline{g}(\lambda^1, \ldots, \lambda^K)$ via a subgradient-based

Algorithm 3 Lower bound computation subroutine in DD3.

$\lambda^k \leftarrow 0,\ \forall k = 1, \ldots, K$
repeat
  for $k = 1, \ldots, K$ **do**
    solve scenario subproblem (22) to attain the optimal objective value β_k^{DD3} and the optimal solution $\hat{x}^k$.
    $S \leftarrow S \cup \{\hat{x}^k\}$
  end for
  solve (23) to attain an optimal solution $\hat{q}$ and the optimal objective value $\sum_{k=1}^{K} \beta_k^{\text{DD3}} \hat{q}_k$.
  $\ell \leftarrow \max\{\ell, \sum_{k=1}^{K} \beta_k^{\text{DD3}} \hat{q}_k\}$
  for $k = 1, \ldots, K$ **do**
    update λ^k by using the subgradient $-\hat{q}_k \hat{x}^k$.
  end for
until achieving the general stop criteria for the subgradient method

algorithm. We repeat the steps of computing $\underline{g}(\lambda^1, \ldots, \lambda^K)$, and then updating λ^k for $k = 1, \ldots, K$ by using the subgradient $-\hat{q}_k \hat{x}^k$, where $\hat{x}^k$ is an optimal solution to the k^{th} subproblem (22). The DD3 algorithm is deduced from replacing steps 4–8 in Algorithm 1 by Algorithm 3. Note that the three approaches differ only in the steps of updating the lower bound ℓ.

Remark 2 *Suppose that the probability distribution p of the uncertainty ξ is not explicitly known. Instead, an ambiguity set $\mathcal{P} \subseteq \mathcal{M}$ of p, consisting of all possible distributions, is available. Let $\overline{\rho}\left(f(x, \xi)\right)$ be the worst-case risk outcome for any $p \in \mathcal{P}$. We define set*

$$\mathcal{D}_\rho(\mathcal{P}) := \left\{q : q \in \mathcal{Q}_\rho(p), \forall p \in \mathcal{P}\right\}, \tag{24}$$

and consider a distributionally robust risk-averse program:

$$\min_{x \in X} \overline{\rho}\left(f(x, \xi)\right) = \min_{x \in X} \max_{q \in \mathcal{D}_\rho(\mathcal{P})} \left\{\sum_{k=1}^{K} q_k f_k(x)\right\}. \tag{25}$$

Comparing (25) *with the formulation of* MIMA *in* (4)*, we have $\mathcal{Q}_\rho(p)$ in the latter replaced by $\mathcal{D}_\rho(\mathcal{P})$ in the former, under the ambiguity of p. Therefore, we can easily adapt the aforementioned DD1, DD2, and DD3 algorithms to the distributionally robust risk-averse program* (25)*.*

3 Parallel Implementation Schemes

In this section, we explore parallel computing schemes for implementing the proposed dual decomposition methods in a distributed framework. DD1 has a simpler iterative structure than DD2 and DD3. (The latter two run two loops and take potentially multiple iterations for updating the Lagrange multipliers and the lower bound in each round.) As a result, DD1 is easier to be parallelized, and the parallel schemes can scale better. In Section 4.2, we show that DD1 outperforms

the other two algorithms even when the steps in each algorithm are implemented in serial. Thus, we focus on improving parallel computing and propose three schemes for parallelizing DD1. (We present parallel schemes for DD2 or DD3 in Appendix B.)

3.1 An Overview of Parallel Algorithms

In a serial implementation of DD1/DD2/DD3, subproblems like **Scen**$(\cdot)$ and **Eval**$(\cdot)$ are solved one by one. If there are multiple processes, one can spread the subproblems, and place a barrier to synchronize all the processes which then exchange results for updating bounds and cuts before re-iteration. We refer to this scheme as *Basic Parallel (BP)* and present the details in Section 3.2. In this case, waiting caused by barriers and high intensity of communication may compromise parallel efficiency. In Section 3.3, we introduce a master-worker scheme that dedicates one process to consolidate information. In Section 3.4, we introduce another master-worker scheme that avoids barriers. These two schemes are referred to as *Master-Worker with Barriers (MWB)* and *Master-Worker without Barriers (MWN)*, respectively. In particular, BP and MWB are "push" systems where we pre-assign computing jobs to processes. Specifically, given tasks $1, \ldots, J$ (with no prior knowledge about their time complexity), we assign them to N processes in a round-robin manner, such that the n^{th} process receives a subset

$$\Omega_n^{J,N} := \{j \in \{1, \ldots, J\} : (j-1) \bmod N = (n-1)\} \tag{26}$$

of tasks. In contrast, MWN is a "pull" system, where jobs are kept in a queue, and each is waiting to be solved whenever a process becomes available.

3.2 Basic Parallel (BP)

We present BP in Algorithm 4, which assigns the K scenarios across the N processes such that Process n (named "Proc^n") receives a subset $\Omega_n^{K,N}$ of scenarios, for $n = 1, \ldots, N$. Each process solves subproblem **Scen**(k) for every assigned scenario k, and evaluates its optimal solution. The evaluation result is used to update the local upper bound u_n in Proc^n. We then let all the processes share their results through some collective operation, which, in general, communicates faster than point-to-point sending/receiving messages, but implies a barrier such that all the processes must reach the point before they can begin communication (Pacheco, 1997; Gropp et al., 1996). In practice, barriers cause waiting.

This scheme has another weakness as evaluating repeated solutions from different scenario subproblems. In each process, we can do a local check between steps 5 and 6, to avoid evaluating some $\hat{x}^k$ that we have encountered earlier in the loop. However, we cannot avoid re-evaluation if the same solution occurs in different processes, which is very likely.

3.3 Master-worker Parallel with Barriers (MWB)

Let Proc^N be the master (process) consolidating solutions and updating the bounds. The rest $N-1$ processes are workers sharing the computation of **Scen**$(\cdot)$ and **Eval**$(\cdot)$.

Algorithm 4 The BP Scheme at Proc^n ($n \in \{1, \ldots, N\}$)

1: $u \leftarrow +\infty$, $\ell \leftarrow -\infty$
2: **repeat**
3: Initialize the local upper bound $u_n \leftarrow +\infty$.
4: **for** $k \in \Omega_n^{K,N}$ **do**
5: $(\beta_k, \hat{x}^k) \leftarrow$ **Scen**(k)
6: $u_n \leftarrow \min\{u_n,$ **Eval**$(\hat{x}^k)\}$
7: $S \leftarrow S \cup \{\hat{x}^k\}$
8: **end for**
9: pass $\{(\beta_k, \hat{x}^k) : k \in \Omega_n^{K,N}\}$ and u_n to all the other processes
10: $\ell \leftarrow \max\{\ell,$ **Cont**$(\beta)\}$
11: $u \leftarrow \min\{u, u_1, \ldots, u_N\}$
12: $X \leftarrow X \setminus S$.
13: **until** $u - \ell < \epsilon$

Every Proc^n ($n \neq N$) solves subproblem **Scen**(k) for all $k \in \Omega_n^{K,N-1}$, and sends the results to the master. The master stores optimal value β_k and collects non-repeated $\hat{x}^k$ in a solution list $\mathcal{S}$. (Note that $\mathcal{S}$ is different from the solution set S - the latter only collects solutions but they are not necessarily ordered, while the former arranges the candidate solutions in a certain order.) Once it has attained the results from all K scenarios, it broadcasts the list $\mathcal{S}$. Every worker receives the whole list (for adding no-good cuts), and shares the evaluation of all solutions in the list $\mathcal{S}$. Specifically, let $\mathcal{S}_i$ represent the i^{th} solution in $\mathcal{S}$. Every Proc^n ($n \neq N$) evaluates $\mathcal{S}_i$ for all $i \in \Omega_n^{|\mathcal{S}|,N-1}$, and sends the results to the master. The master, after broadcasting $\mathcal{S}$, updates the bounds and forces all processes to terminate once it detects a sufficiently small gap. We present the algorithmic steps of a worker process in Algorithm 5 and of the master in Algorithm 6. (Here u' denotes a temporary, valid lower bound obtained at each worker, and $\mathcal{S} + \hat{x}^k$ means appending solution $\hat{x}^k$ to the tail of the list $\mathcal{S}$.

With the master collecting and broadcasting solutions, evaluating duplicated solutions can be avoided. In terms of communication, we use asynchronous "send/receive" signals, so that the process that sends the message does not wait for the reception of the message but proceed with the succeeding steps, which allows for more parallelism. However, this requires buffers to store the data in transit. In each iteration of MWB, data to be transmitted contains at most K d-dimensional binary vectors and K real numbers, which is a fairly modest amount. In step 10 of Algorithm 6 (or step 6 of Algorithm 5), to send the solution list $\mathcal{S}$ from the master to the worker, we use broadcast, which, again, is a collective operation implying a barrier across all the processes.

3.4 Master-worker Parallel without Barriers (MWN)

In BP and MWB, all the processes are going through iterations synchronously due to the barriers implied by collective communication steps. Here we design a new scheme that avoids these

Algorithm 5 The MWB Scheme at Proc^n ($n \in \{1, \ldots, N-1\}$) (worker)

loop
  for $k \in \Omega_n^{K,N-1}$ **do**
    $(\beta_k, \hat{x}^k) \leftarrow$ **Scen**(k)
    send $(\beta_k, \hat{x}^k)$ to Proc^N
  end for
  gather $\mathcal{S}$ from Proc^N
  for $i \in \Omega_n^{|\mathcal{S}|,N-1}$ **do**
    $u' \leftarrow$ **Eval**$(\mathcal{S}_i)$
    send u' to Proc^N
  end for
  $X \leftarrow X \setminus \mathcal{S}$
end loop

Algorithm 6 The MWB Scheme at Proc^N (master)

repeat
  $u \leftarrow +\infty$, $\ell \leftarrow -\infty$
  $\mathcal{S} \leftarrow \emptyset$
  for K times **do**
    receive $(\beta_k, \hat{x}^k)$ from Proc^n
    if $\hat{x}^k \notin \mathcal{S}$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} + \hat{x}^k$
    end if
  end for
  broadcast $\mathcal{S}$ to Proc^1, ..., Proc^{N-1}
  $\ell \leftarrow \max\{\ell, \textbf{Cont}(\beta)\}$
  for $|\mathcal{S}|$ times **do**
    receive u' from Proc^n
    $u \leftarrow \min\{u, u'\}$
  end for
until $u - \ell < \epsilon$
terminate all processes

barriers. At the same time, we make it a pull system for better load balancing.

Formally, the master keeps a queue of idle workers as $\mathtt{Q}_{\text{proc}}$, and a queue of subproblems to solve as $\mathtt{Q}_{\text{job}}$. As long as both queues are non-empty, it repeatedly pops out from each a worker and a job, then assigns the job to the worker. It then waits to receive any result from the workers. Once heard from some worker, it adds the worker to $\mathtt{Q}_{\text{proc}}$, process the received results, and if necessary creates new jobs. These steps are summarized in Algorithm 7. In particular, steps 1, 5 and 9 are detailed in Algorithms 10, 9 and 8, respectively.

Algorithm 7 The MWN Scheme at Proc^N (worker)

initialization
repeat
  while $\mathtt{Q}_{\text{proc}} \neq \emptyset$ and $\mathtt{Q}_{\text{job}} \neq \emptyset$ **do**
    $n \leftarrow \text{pop}(\mathtt{Q}_{\text{proc}})$, $j \leftarrow \text{pop}(\mathtt{Q}_{\text{job}})$
    assign j to Proc^n
  end while
  receive r from Proc^n
  $\mathtt{Q}_{\text{proc}} \leftarrow \mathtt{Q}_{\text{proc}} + n$
  process r (and create jobs to $\mathtt{Q}_{\text{job}}$)
until $u - \ell < \epsilon$
terminate all processes

The master still keeps the bounds and the list $\mathcal{S}$ of solutions. On the reception of a result of **Eval**$(\cdot)$, e.g., u', the master uses it to update the upper bound. In the other case when receiving a result of **Scen**(k), i.e., $(\beta_k, \hat{x}^k)$, the master resolves **Cont**(β) which takes in the new value of β_k, to update the lower bound. If $\hat{x}^k$ is new, it is appended to $\mathcal{S}$ and a job for evaluating its objective value is created. In addition, every reception of **Scen**(k)'s result triggers to create a new job for solving **Scen**(k), to roll on the iterations.

Algorithm 8 Master result processing subroutine (step 9 of Algorithm 7)

switch r**:**
  case u' **:**
    $u \leftarrow \min\{u, u'\}$
  case $(\beta_k, \hat{x}^k)$ **:**
    $\ell \leftarrow \max\{\ell, \textbf{Cont}(\beta)\}$
    if $\hat{x}^k \notin \mathcal{S}$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} + \hat{x}^k$
      $\mathtt{Q}_{\text{job}} \leftarrow \mathtt{Q}_{\text{job}} + \textbf{Eval}(\hat{x}^k)$
    end if
    $\mathtt{Q}_{\text{job}} \leftarrow \mathtt{Q}_{\text{job}} + \textbf{Scen}(k)$
end switch

To avoid barriers, the system has no centralized step for informing the workers of all the explored solutions, but point-to-point communication between the master and any individual worker for a single job or the result of it. In order for the workers to know what cuts to add to the subproblems, we let the master keep, for each worker n, an index $\tau(n) \in \{0, \ldots, |\mathcal{S}|\}$ pointing to the end of a sublist of $\mathcal{S}$ which the worker has known and generated cuts. For example, if $\tau(1) = 3$ and $\tau(2) = 5$, it means that Proc^1 and Proc^2 have got cuts to exclude solutions $\mathcal{S}_1, \ldots, \mathcal{S}_3$, and $\mathcal{S}_1, \ldots, \mathcal{S}_5$, respectively. Once it is time for Proc^n to update cuts, the master sends $\mathcal{S}_{\tau(n)+1}, \ldots, \mathcal{S}_{|\mathcal{S}|}$,

then accordingly increase $\tau(n)$ to $|\mathcal{S}|$. Note that additional cuts affect **Scen**$(\cdot)$ but not **Eval**$(\cdot)$. The master therefore sends these solutions along with the jobs of **Scen**$(\cdot)$. Algorithm 9 presents the job assignment subroutine with these details. Algorithm 10 presents the initialization steps.

Algorithm 9 Master job assignment subroutine (step 5 of Algorithm 7)

switch j**:**
  case Eval$(\hat{x})$ **:**
    send $(\textbf{Eval}(\hat{x}))$ to Proc^n
  case Scen(k) **:**
    $S \leftarrow \{\mathcal{S}_i : i = \tau(n) + 1, \ldots, |\mathcal{S}|\}$
    $\tau(n) \leftarrow |\mathcal{S}|$
    send $(S, \textbf{Scen}(k))$ to Proc^n
end switch

Algorithm 10 Master initialization subroutine (step 1 of Algorithm 7)

$u \leftarrow +\infty$, $\ell \leftarrow -\infty$
$\mathcal{S} \leftarrow \emptyset$
$\tau(1), \ldots, \tau(N-1) \leftarrow 0$
$\mathtt{Q}_{\text{proc}} \leftarrow \langle 1, \ldots, N-1 \rangle$
$\mathtt{Q}_{\text{job}} \leftarrow \langle \textbf{Scen}(1), \ldots, \textbf{Scen}(K) \rangle$

The logic at the workers is straightforward. They receive tasks from the master, work on them and send back the results. Algorithm 11 presents the steps.

Algorithm 11 The MWN Scheme at Proc^n $(n \in \{1, \ldots, N-1\})$

loop
  receive j from Proc^N
  switch j**:**
    case $(\textbf{Eval}(\hat{x}))$ **:**
      $u' \leftarrow \textbf{Eval}(\hat{x})$
      send u' to Proc^N
    case $(S, \textbf{Scen}(k))$ **:**
      $X \leftarrow X \setminus S$
      $(\beta_k, \hat{x}^k) \leftarrow \textbf{Scen}(k)$
  end switch
end loop

A major difference of MWN from the previous schemes is that, the workers are asynchronous in adding cuts. Given the cuts that Proc^n has generated, we denote the feasible region as X_n. Then

each **Scen**(k) that Proc^n solves, precisely, is:

$$\beta'_k(n_k) = \min\{f_k(x) : x \in X_{n_k}\}$$

where $n_k \in \{1, \ldots, N-1\}$ specifies the worker, to which the scenario subproblem **Scen**(k) is assigned. A lower bound obtained in this case is

$$\tilde{\ell} = \textbf{Cont}((\beta'_1(n_1), \ldots, \beta'_K(n_K))).$$

To justify that the algorithm is valid with this lower bound, we show in the following that $\tilde{\ell}$ will not cross with the upper bound u, when u is not yet optimal. Define $\overline{X} = X_{n_1} \cap \cdots \cap X_{n_K}$, and for $k = 1, \ldots, K$, consider

$$\overline{\beta}_k = \min\{f_k(x) : x \in \overline{X}\}.$$

By definition, $\overline{\beta}_k \geq \beta'_k(n_k)$ for $k = 1, \ldots, K$. When u is suboptimal, any optimal solution x^* must not have been explored yet (i.e., $x^* \notin \mathcal{S}$), and thus $x^* \in X_{n_k}$ for $k = 1, \ldots, K$ which implies that $x^* \in \overline{X}$. Therefore, we also have $\overline{\beta}_k \leq f_k(x^*)$ for $k = 1, \ldots, K$. Therefore,

$$u > \textbf{Cont}((f_1(x^*), \ldots, f_K(x^*))) \geq \textbf{Cont}((\overline{\beta}_1, \ldots, \overline{\beta}_K)) \geq \textbf{Cont}((\beta'_1(n_1), \ldots, \beta'_K(n_K))) = \tilde{\ell}.$$

4 Computational Results

We implement the proposed parallelization schemes by OpenMPI 1.6 (Gabriel et al., 2004), and perform the computation on the Flux HPC cluster at the University of Michigan. We use up to 21 compute nodes of the cluster, and each compute node has twelve 2.67 GHz Intel Xeon X5650 processors and 48GB RAM. All involved optimization models (including subproblems as a result of decomposition) are solved by CPLEX 12.6 via ILOG Concert Technology. We set a runtime limit of six hours for computing each instance.

4.1 Instances and Experimental Setup

We extract two sets of instances from the test problem library SIPLIB (Ahmed et al., 2015a):

- Stochastic server location problem (SSLP) instances from a telecommunication application studied by Ntaimo and Sen (2005): The first stage decides where to place servers out of n locations, and the second stage satisfies uncertain demand from m clients. The cost function is given by

  $$f(x, \xi) = \gamma^\top x + \min_y\{\theta_1^\top y + \theta_2^\top z : W_1 y + W_2 z \geq r(\xi) - Tx,\ y \in \{0,1\}^{n \times m},\ z \in \mathbb{R}^n_+\}$$

  and the feasible region is $x \in X = \{0,1\}^n$. We use four sets of instances with $n = 10$ and $m = 50$. They contain 50, 100, 500 and 1000 scenarios, respectively.

- Stochastic multiple 0-1 knapsack problem (SMKP) instances studied by Angulo et al. (2014): The first and the second stages are multiple 0-1 knapsack problems with n and m entities, respectively. The cost function reads as:

$$f(x,\xi) = \gamma^\top x + \min_y \{\theta(\xi)^\top y : Wy \geq r - Tx,\ y \in \{0,1\}^m\}$$

and the feasible region is $X = \{x \in \{0,1\}^n : Ax \geq w\}$ with linear constraints $Ax \geq w$. The original dataset contains 30 instances each of which contains 20 scenarios. To examine how problem size affects the algorithm performance, we pick the first instance, then modify it to attain instances with 40, 80 and 160 scenarios. To modify a K-scenario dataset to K'-scenario, we first include the original K scenarios. Then repeat for $K' - K$ times, generate random numbers $\sigma_1, \ldots, \sigma_K$, then create a scenario k such that $\theta(\xi^k)$ is a weighed average of the original $\theta(\xi^1), \ldots, \theta(\xi^K)$ with weighing coefficients given by normalized $\sigma_1, \ldots, \sigma_K$, i.e.,

$$\theta(\xi^k) = \frac{\sum_{k'=1}^K \sigma_{k'} \theta(\xi^{k'})}{\sum_{k'=1}^K \sigma_{k'}}.$$

Table 1 presents the size and the performance of LP relaxations of these instances. Specifically, #*_var* and #*_constr* respectively give the numbers of binary variables and constraints in the corresponding problems. SMKP has pure binary problems in both stages. SSLP has binary first-stage, and mixed-binary second-stage problem, which involves 10 continuous variables in each scenario. Both sets contain four instances with varying number of scenarios. In every instance, all the scenarios are equal likely, i.e., $p_1 = \cdots = p_K = 1/K$. Regarding the number of variables and constraints, SSLP has the second-stage much bigger than the first-stage, while SMKP is the opposite. The last section of the table presents the computational time (in seconds) of the LP relaxation (see *total_time*) and the gaps between the optimal objective values of the LP relaxation and the original binary program (see *tightness*). There is a notable trend that the more scenarios, the longer the computation and the weaker the LP relaxation.

Table 1: Sizes of different instances and performance of their LP relaxations

		SSLP				SMKP			
		_50	_100	_500	_1000	_20	_40	_80	_160
1st-stage	#_var	10	10	10	10	240	240	240	240
	#_constr	1	1	1	1	50	50	50	50
2nd-stage	#_scen (K)	50	100	500	1000	20	40	80	160
	#_var (per scen)	500	500	500	500	120	120	120	120
	#_constr (per scen)	60	60	60	60	5	5	5	5
LP relaxation	total_time	0.48	2.35	59.29	45.19	0.043	0.071	0.069	0.229
	tightness	24.8%	25.8%	24.3%	27.9%	0.36%	2.99%	3.15%	5.53%

For the coherent risk measure, we choose conditional value-at-risk (CVaR), i.e., $\rho(\cdot) = \text{CVaR}_\alpha(\cdot)$, and set the reliability parameter α to 0.9. The uncertainty set in the dual representation of CVaR

contains only linear constraints (see Shapiro and Ahmed, 2004), given by

$$\mathcal{Q}_\rho(p) = \left\{ (q_1, \ldots, q_K) : \sum_{k=1}^{K} q_k = 1,\ 0 \le q_k \le p_k/(1-\alpha),\ \forall k = 1, \ldots, K \right\}.$$

In this case, `MIMA` becomes a minimax linear integer program, and the maximization problems **Cont**(β), (13)–(15), and (23), which emerge from the scenario decomposition algorithms, are all linear programs.

To benchmark the performance of the scenario decomposition algorithms, we solve the following equivalent reformulation of CVaR (Rockafellar and Uryasev, 2000, 2002):

$$\text{CVaR}_\alpha(f(x,\xi)) = \min_{\eta} \left\{ \eta + \frac{1}{1-\alpha} \sum_{k=1}^{K} p_k \left[f_k(x) - \eta \right]^+ : \eta \in \mathbb{R} \right\},$$

and solve the considered risk-averse problem an extensive form mixed-integer program:

$$\min_{x,\eta,v_1,\ldots,v_K} \left\{ \eta + \frac{1}{1-\alpha} \sum_{k=1}^{K} p_k v_k : v_k \ge f_k(x) - \eta,\ v_k \ge 0,\ \forall k = 1, \ldots, K,\ x \in X \right\}. \tag{27}$$

4.2 Results of Serial Implementation

The *default* serial scheme is directly calling an off-the-shelf solver to optimize model the extensive form model (27). Table 2 compares the serial computational time (in second) of different approaches implemented in one process. If an instance is not solved to optimality within the runtime limit of six hours, we present the optimality gaps in parentheses. We refer to Table 8 in Appendix A for the detailed results of the time spent on solving subproblems, evaluating solutions, and the number of iterations in the default and DD1 algorithms.

Table 2: Serial solution time (in second) for solving SSLP and SMKP instances

Scheme	SSLP				SMKP			
	_50	_100	_500	_1000	_20	_40	_80	_160
default	195	201	(100%)	(100%)	300	(0.09%)	(0.11%)	(0.16%)
DD1	248	502	4663	12750	2692	9866	11249	18774
DD2	1276	2570	(10%)	(16%)	(0.02%)	(0.01%)	(0.02%)	(0.02%)
DD3	415	602	7231	(9%)	3496	9080	(0.11%)	(0.11%)

Compared with default, the scenario decomposition methods are slower when the number of scenarios is relatively small (e.g., SSLP_50, SSLP_100 and SMKP_20). However, as the number of scenarios increases, they perform better in time efficiency: (i) the optimality gaps on SSLP_500 and SSLP_1000 are, respectively, 10% and 16% under DD2, which are much better than the two 100%s under the default scheme; (ii) DD3 yields a better optimality gap of 9% on SSLP_1000, and can solve SSLP_500 in two hours; (iii) DD1 is the most efficient, solving the two instances in 4663

seconds and 12750 seconds, respectively. We observe a similar trend of computational efficacy on solving modest and large instances of SMKP (_40, _60, and _80) as

$$\text{default} < \text{DD2} < \text{DD3} < \text{DD1}.$$

Next we focus on implementing the three parallelization schemes on the DD1 algorithm, and demonstrate the improved performance of the algorithm.

4.3 Results of Parallel Implementations of DD1

We capture the parallel time as the time that elapsed from the start of the parallel program to the end. For each run we compute *speedup* as the ratio of the serial time to the parallel time. We test all the three schemes introduced in Section 3 on both sets of SSLP and SMKP instances, and show how the results of speedup change as we vary the number of processes in Figure 1.

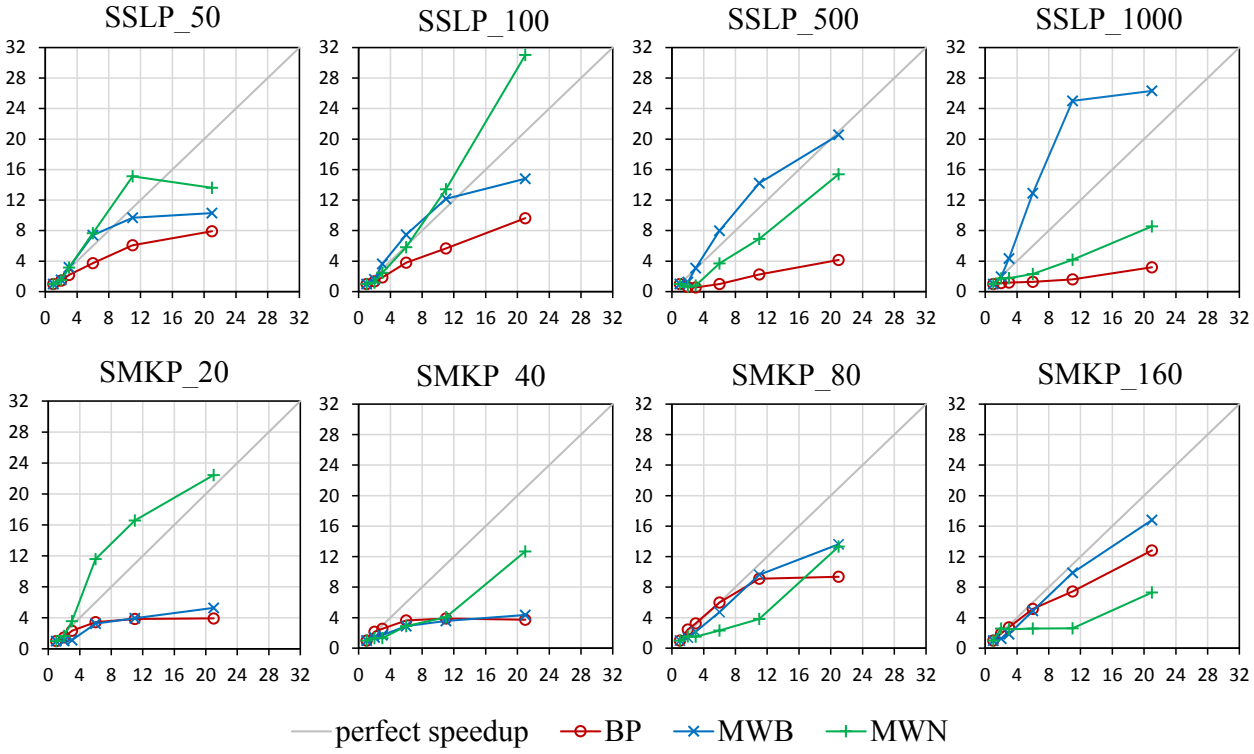

Figure 1: Speedup versus number of processes for implementing DD1

In Figure 1, each subfigure presents the results on one particular instance. The vertical and horizontal axes represent "speedup" and "number of processes", respectively. We have them scaled equally, so the main diagonal (in grey) indicates perfect parallelism where speedup is equal to the number of processes. Each background grid cell has a height of 5 (unit-less) and a width of 5 processes. We use 2, 3, 6, 11 and 21 processes. The red, the blue and the green curves are results of BP, MWB and MWN, respectively.

We observe several cases of super-linear speedup, i.e., points located above the diagonal line. This is due to the way we parallelize, and the total workload may abate from the serial implementation. Consider the following simplified example. The serial program has spent 2 hours going through some iterations, then comes to an iteration with three outstanding jobs (e.g., **Scen**(k) or **Eval**$(\hat{x})$), that respectively take 1, 8 and 1 hours. The result of the third job leads to convergence. The serial total time is thus $2 + (1 + 8 + 1) = 12$ hours. However, if we have three processes each taking care of one job. Then, the process responsible for the third job will detect convergence right after 1 hour, and will right away terminate all the other processes. With two extra processes, the iterations that previously took 2 hours will now take shorter, too. Therefore, the parallel total time is shorter than $2 + 1 = 3$ hours. In this case, we have achieved a speedup ≥ 4 with only three processes.

We dive into Figure 1 for deducing other interesting results of using different parallel schemes. First, comparing the red (BP) and the blue (MWB) curves, for every SMKP instance, we observe a crossover of the two lines: the red line always starts climbing sharper, but flatten out earlier and crosses with the blue one. Recall that BP has one more process sharing the computation than MWB. This is an important advantage when N is small, which explains the red line climbing faster in the beginning. This advantage, however, fades out as we increase N. In contrast, the advantage of MWB that it avoids re-evaluating duplicate solutions stands out. This can be verified by comparing the number of solutions BP and MWB evaluate, reported in Table 3.

Table 3: Number of evaluated x-solutions in the implementation of DD1 algorithm

		SSLP				SMKP			
scheme	num of processes (N)	_50	_100	_500	_1000	_20	_40	_80	_160
BP	2	139	183	259	172	9	14	22	34
	3	155	214	489	221	11	16	24	42
	6	170	247	701	324	15	21	30	52
	11	184	266	892	882	17	27	37	62
	21	189	283	1095	1151	20	34	48	74
MWB	2	117	136	240	187	8	12	15	23
	3	117	136	240	187	8	12	15	23
	6	117	136	240	188	8	12	15	23
	11	117	134	240	188	8	12	15	23
	21	117	136	240	189	8	12	15	23

In Table 3, for MWB, less time is spent on evaluation and is thus faster, which is reflected as the blue curve climbing higher when N is large. On SSLP instances, BP scales worse than MWB, and the discrepancy gets larger as N increases. This is because the more processes, the more time BP wastes on evaluating duplicated solutions. This can be seen from the BP section in Table 3 where the number or evaluated solutions increases along with the increase of N in all cases.

As we examine horizontally, MWB and MWN are complementary. When the number of sce-

narios is still small (e.g., SSLP_50, SSLP_100, SMKP_20, SMKP_40), MWB suffers from load imbalance, but MWN achieves good speedup due to the deployed pull mechanism. As the number of scenarios increases, there are more subproblems to solve and more solutions to evaluate. MWN has to communicate more to dispatch jobs, which compromises the speed. In contrast, more scenarios means more jobs MWB can shuffle around between barriers, and thus load imbalance gets alleviated. This explains why MWB being faster than MWN on SSLP_500, SSLP_100, SMKP_80 and SMKP_160.

In addition to speedup, we also analyze communication time, which is the total time elapsed from the completion of the predecessor of a communication step (e.g., send, receive, broadcast, etc.) to the start of its successor. Therefore, it consists of the time on transmitting data (the net communication time) and the time on waiting to receive or broadcast data (the idle time). In practice, compared to the latter, the former is almost negligible due to the modest quantity of data being transmitted. We therefore can view the communication time as the idle time. We capture the communication time of each process as $\text{comm}(n)$. Figure 2 plots the total communication time (in second)

$$\sum_{n=1}^{N} \text{comm}(n),$$

against the number of processes. Table 4 presents the percentage of communication time con-

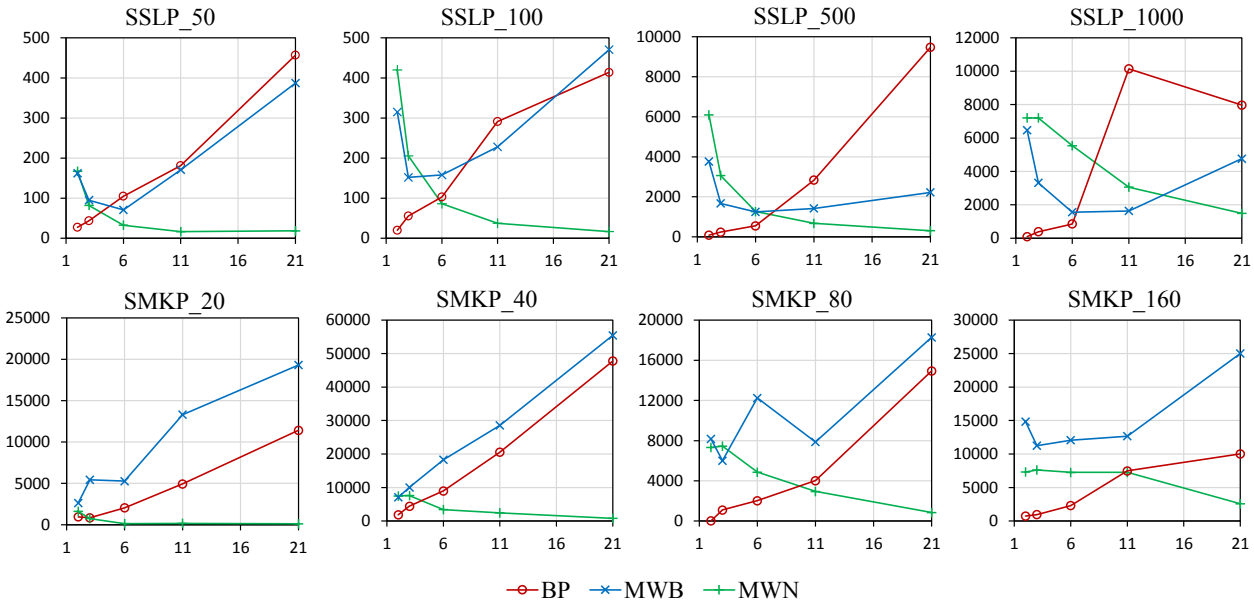

Figure 2: Communication time versus number of processes (N) of parallelizing DD1

tributed by the master, i.e.,

$$\frac{\text{comm}(N)}{\sum_{n=1}^{N} \text{comm}(n)}.$$

We make the following observations.

Table 4: The percentage of communication time contributed by the master

		SSLP				SMKP			
scheme	N	_50	_100	_500	_1000	_20	_40	_80	_160
MWB	2	100%	100%	100%	100%	100%	100%	100%	100%
	3	82%	91%	91%	89%	83%	50%	50%	85%
	6	47%	43%	47%	63%	26%	20%	20%	29%
	11	15%	18%	23%	31%	11%	10%	10%	11%
	21	6%	7%	10%	10%	5%	5%	5%	5%
MWN	2	100%	100%	100%	100%	100%	100%	100%	100%
	3	100%	100%	100%	100%	100%	100%	100%	100%
	6	100%	100%	100%	100%	100%	100%	100%	100%
	11	99%	99%	100%	100%	100%	100%	100%	100%
	21	98%	97%	100%	100%	95%	99%	99%	100%

- In Figure 2, the curves for BP, MWB and MWN follow increasing, increasing and decreasing trends, respectively.
- In BP, there are only collective communication steps which imply barriers. Except for the small amount of time on transmitting data, the communication time is mainly from waiting at barriers, which increases as the number of processes N increases.
- In MWN, the master is devoted to consolidating results from the workers. The more workers we have, the shorter the master waits for results. We also see from Table 4 that nearly all the communication time is contributed by the master (which implies that the workers are well utilized for computation thanks to the pull mechanism). These explain why MWN curves decrease in Figure 2.
- Although MWB has a similar master-worker structure, it contains a broadcasting step involving all the processes and barrier waiting. As N increases, the communication time of the master becomes shorter, however, the communication time of the workers increases in a bigger magnitude. This offsets the drop of communication-time percentage for the master in Table 4, and also leads to the shape of MWB curves in Figure 2.

Lastly, we compare the runtime (in second) between serial and parallel implementation schemes of DD1 in Table 5, in which the first two rows recall the serial computational time of the default and DD1 algorithms presented in Table 2. For each instance, we indicate in bold the shortest serial time and the shortest parallel time. We observe in all instances that the fastest parallel scheme outperforms the fastest serial scheme. Among the parallel schemes, MWN performs better for small instances while MWB performs better for the largest scale instances.

Table 5: Comparing serial and parallel computation time (in second) of DD1 algorithm

Scheme		N	SSLP				SMKP			
			_50	_100	_500	_1000	_20	_40	_80	_160
Serial	default	1	**195**	**201**	(100%)	(100%)	**300**	(0.09%)	(0.11%)	(0.16%)
	DD1	1	248	502	**4663**	**12750**	2692	9866	**11249**	**18774**
Parallel	BP	2	170	377	7633	11412	1808	4560	4630	9779
		3	113	271	8514	10781	1162	3897	3462	6877
		6	66	132	4694	9844	782	2711	1879	3659
		11	41	89	2074	7870	696	2539	1233	2521
		21	31	52	1122	3991	685	2648	1202	1464
	MWB	2	162	315	3758	6462	2641	7046	8162	14830
		3	78	138	1514	2938	2400	5834	5251	10297
		6	33	67	584	987	825	3418	2375	3840
		11	26	41	327	510	684	2773	1169	1903
		21	24	34	**227**	**484**	509	2269	**826**	**1117**
	MWN	2	166	418	7200	7304	1625	7481	7322	7312
		3	79	203	5783	7224	755	7582	7449	7615
		6	32	86	1260	5540	232	3411	4860	7263
		11	16	37	674	3058	162	2444	2943	7258
		21	**18**	**16**	303	1489	**120**	**777**	844	2572

4.4 Results of 0-1 Stochastic Program with Mean-risk Measure

To investigate whether or not the selection of risk measure can affect the computational difficulty of 0-1 risk-averse programs, we consider a mean-risk variant of the above CVaR-based model, in which we minimize a weighted sum of a coherent risk and an expectation:

$$\min \{w \cdot \rho(f(x,\xi)) + (1-w) \cdot \mathbb{E}[f(x,\xi)] : x \in X\}$$

where w is a given weight parameter between 0 and 1. The extended formulation (used to be `MIMA` given the generic coherent risk measure ρ) now becomes:

$$\texttt{MIMA}': \quad \min_{x\in X} \max_{q\in\mathcal{Q}_\rho(p)} \left\{ \sum_{k=1}^{K} \left(wq_k + (1-w)p_k\right) f_k(x) \right\}$$

to which all the proposed algorithms (i.e., DD1, DD2, DD3, and the parallel computing schemes) can easily adapt.

Table 6 compares the runtime (in second) of the risk-averse problem (i.e., $w = 1$) with its expectation-based counterpart (i.e., $w = 0$), under MWB. We see that they are very close on most of the instances. We therefore can conclude that the proposed algorithm can handle a risk-averse problem equally well with any coherent risk measure, as a transitional expectation-based stochastic program.

Table 6: Results of mean-risk model variants and its expectation counterpart under MWB

		SSLP_50		SSLP_100		SSLP_500		SSLP_1000	
	N	$w=1$	$w=0$	$w=1$	$w=0$	$w=1$	$w=0$	$w=1$	$w=0$
time	2	6	6	13	13	162	162	315	344
	3	3	3	6	6	78	76	138	138
	6	1	1	3	3	33	33	67	65
	11	1	1	2	2	26	25	41	48
	21	1	1	2	2	**24**	29	**34**	34
optimal obj		-253	-365	-248	-355	-246	-350	-250	-352

		SMKP_20		SMKP_40		SMKP_80		SMKP_160	
	N	$w=1$	$w=0$	$w=1$	$w=0$	$w=1$	$w=0$	$w=1$	$w=0$
time	2	3758	3814	6462	11930	2641	3651	7046	8706
	3	1514	1525	2938	5995	2400	3448	5834	7257
	6	584	583	987	2110	825	983	3418	4082
	11	327	332	510	964	684	846	2773	3656
	21	227	186	484	864	509	602	2269	3071
optimal obj		9357	9043	9550	9187	9475	9194	9572	9251

4.5 Results of Distributionally Robust Risk-averse Programs

As discussed in Remark 2, the distributionally robust risk-averse problem can be viewed as a special case of the considered problem as long as we replace $\mathcal{Q}_\rho(p)$ with $\mathcal{D}_\rho(\mathcal{P})$. Here we test a distributionally robust risk-averse problem with

$$\mathcal{P} = \left\{ p : \sum_{k=1}^{K} p_k = 1,\ (1-\upsilon)/K \le p_k \le (1+\upsilon)/K,\ \forall k \right\}$$

where υ is a constant between 0 and 1. We vary $\upsilon = 0.3$ and 0.6 on three instances (i.e., SSLP_100, SSLP_1000, SMKP_80), use MWB to solve each case and present the results in Table 7. We also show the results of the corresponding run given singleton-$\mathcal{P}$ (i.e., $\upsilon = 0.0$) as benchmarks.

In Table 7, we observe that the change in runtime depending on υ is very small. This is reasonable because the structure of $\mathcal{P}$ only affects the computation of **Cont**(β) in step 11 of Algorithm 6, which in this case is a simple LP with negligible solution time.

5 Conclusions

In this paper, we reformulate a class of risk-averse 0-1 stochastic programs into minimax programs by utilizing dual representations of coherent risk measures. We then develop three scenario-decomposition-based algorithms (DD1, DD2, and DD3) based on different Lagrangian relaxation

Table 7: Computational results of distributionally robust risk-averse problem under MWB

		SSLP_100			SSLP_1000			SMKP_80		
	N	$\upsilon = 0$	$\upsilon = 0.3$	$\upsilon = 0.6$	$\upsilon = 0$	$\upsilon = 0.3$	$\upsilon = 0.6$	$\upsilon = 0$	$\upsilon = 0.3$	$\upsilon = 0.6$
time	2	315	311	311	6462	8016	7210	8162	8326	8565
	3	138	137	137	2938	2950	2866	5251	5444	5122
	6	67	70	70	987	992	977	2375	2475	2474
	11	41	42	42	510	641	640	1169	1245	1243
	21	34	34	34	484	660	681	826	796	799
optimal obj		-248	-245	-243	-250	-244	-239	9475	9484	9493

schemes. In DD1, the Lagrange multipliers are simply set to zero, and the gap between the upper and lower bounds are closed solely by evaluating and cutting off feasible solutions. DD2 and DD3 seek to accelerate the convergence by adding an inner loop for strengthening the lower bounds through a cutting-plane method and a subgradient method to update the multiplers, respectively. Our computation results suggest the following:

1. All the scenario decomposition algorithms significantly outperform the method of directly solving the risk-averse 0-1 program (e.g., the default scheme in Section 4.2), especially when the number of scenarios is large.

2. The speeds of the three approaches follow:

$$\text{DD2} < \text{DD3} < \text{DD1},$$

   which implies that adjusting the dual multiplier is not quite effective in improving the lower bounds, and motivates the parallelization of DD1.

To further reduce the runtime, we introduce three parallel schemes for DD1. In BP, there is no hierarchy among processes. All of them are sharing subproblem computation and passing results through collective communication steps. In MWB, with the objective of avoiding reevaluating duplicate solutions and abating workload, we dedicate one process, termed as the master, to collecting solutions and removing duplicates. This, however, means that there is one less process sharing the computation. Similar to BP, there is extra waiting at the barriers implied by the collective communication operations for passing cuts. In MWN, to avoid barriers we use an asynchronous cut-adding strategy that only requires point-to-point communication between the master and individual workers. Moreover, we deploy a "pull" mechanism in assigning subproblems to processes which much better balances the load than the "push" mechanism in the previous two schemes. Our computational results suggest the following:

1. Parallel implementation of DD1 can lead to super-linear speedup due to the early detection of convergence achieved from spreading out the computation of subproblems.

2. When the number of processes is small, BP performs well. However, as we increase the number of processes, the performance of BP is deteriorated by waiting caused by barriers.

3. MWB and MWN are complementary to each other. When the number of scenarios is small, MWB suffers from load imbalance, and MWN achieves good speedup due to the deployed pull mechanism. As the number of scenarios increases, the communication for job dispatching in MWN starts to compromise the performance, while MWB has load imbalance alleviated and thus starts to perform well.

4. The choice of coherent risk measure in a risk-averse 0-1 stochastic program does not generate significant impact on the computation time.

5. The same scenario decomposition algorithm and parallelization methods can be applied to solve distributionally robust risk-averse 0-1 stochastic programs, with similar computational efficiency.

Acknowledgment

Shabbir Ahmed is supported in part by the Office of Naval Research under grant number 127307. Yan Deng and Siqian Shen have been supported in part by the National Science Foundation under grant CMMI-1433066. Yan Deng is also grateful for the support from Michigan Institute for Computational Discovery and Engineering Fellowship.

References

Ahmed, S. (2013). A scenario decomposition algorithm for 0-1 stochastic programs. *Operations Research Letters*, 41(6):565–569.

Ahmed, S., Garcia, R., Kong, N., Ntaimo, L., Parija, G., Qiu, F., and Sen., S. (2015a). SIPLIB: a stochastic integer programming test library. `http://www2.isye.gatech.edu/~sahmed/siplib/`.

Ahmed, S., Luedtke, J., Song, Y., and Xie, W. (2015b). Nonanticipative duality, relaxations, and formulations for chance-constrained stochastic programs. Available at Optimization-Online `http://www.optimization-online.org/DB_HTML/2014/07/4447.html`.

Angulo, G., Ahmed, S., and Dey, S. S. (2014). Improving the integer L-shaped method. Available at Optimization-Online `http://www.optimization-online.org/DB_HTML/2014/04/4332.html`.

Artzner, P., Delbaen, F., Eber, J.-M., and Heath, D. (1999). Coherent measures of risk. *Mathematical Finance*, 9(3):203–228.

Birge, J. R., Donohue, C. J., Holmes, D. F., and Svintsitski, O. G. (1996). A parallel implementation of the nested decomposition algorithm for multistage stochastic linear programs. *Mathematical Programming*, 75(2):327–352.

Birge, J. R. and Louveaux, F. V. (2011). *Introduction to Stochastic Programming*. Springer, New York, NY.

Birge, J. R. and Rosa, C. H. (1996). Parallel decomposition of large-scale stochastic nonlinear programs. *Annals of Operations Research*, 64(1):39–65.

Carøe, C. C. and Schultz, R. (1999). Dual decomposition in stochastic integer programming. *Operations Research Letters*, 24:37–45.

Collado, R. A., Papp, D., and Ruszczyński, A. (2012). Scenario decomposition of risk-averse multistage stochastic programming problems. *Annals of Operations Research*, 200(1):147–170.

Crainic, T. G., Fu, X., Gendreau, M., Rei, W., and Wallace, S. W. (2011). Progressive hedging-based metaheuristics for stochastic network design. *Networks*, 58(2):114–124.

Crainic, T. G., Hewitt, M., and Rei, W. (2014). Scenario grouping in a progressive hedging-based metaheuristic for stochastic network design. *Computers & Operations Research*, 43:90–99.

Dentcheva, D. and Römisch, W. (2004). Duality gaps in nonconvex stochastic optimization. *Mathematical Programming*, 101(3):515–535.

Fragniere, E., Gondzio, J., and Vial, J.-P. (2000). Building and solving large-scale stochastic programs on an affordable distributed computing system. *Annals of Operations Research*, 99(1-4):167–187.

Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., et al. (2004). Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 97–104. Springer.

Gropp, W., Lusk, E., Doss, N., and Skjellum, A. (1996). A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828.

Linderoth, J. and Wright, S. (2003). Decomposition algorithms for stochastic programming on a computational grid. *Computational Optimization and Applications*, 24(2-3):207–250.

Linderoth, J. and Wright, S. J. (2005). Computational grids for stochastic programming. *Applications of stochastic programming*, 5:61–77.

Lubin, M., Martin, K., Petra, C. G., and Sandıkçı, B. (2013). On parallelizing dual decomposition in stochastic integer programming. *Operations Research Letters*, 41(3):252–258.

Miller, N. and Ruszczyński, A. (2011). Risk-averse two-stage stochastic linear programming: Modeling and decomposition. *Operations Research*, 59(1):125–132.

Mulvey, J. M. and Ruszczyński, A. (1995). A new scenario decomposition method for large-scale stochastic optimization. *Operations Research*, 43(3):477–490.

Nielsen, S. S. and Zenios, S. A. (1997). Scalable parallel benders decomposition for stochastic linear programming. *Parallel Computing*, 23(8):1069–1088.

Ntaimo, L. and Sen, S. (2005). The million-variable "march" for stochastic combinatorial optimization. *Journal of Global Optimization*, 32:385–400.

Pacheco, P. S. (1997). *Parallel programming with MPI*. Morgan Kaufmann.

Rockafellar, R. T. and Uryasev, S. (2000). Optimization of conditional Value-at-Risk. *Journal of Risk*, 2(3):21–42.

Rockafellar, R. T. and Uryasev, S. (2002). Conditional Value-at-Risk for general loss distributions. *Journal of Banking and Finance*, 26(7):1443–1471.

Rockafellar, R. T. and Wets, R.-B. (1976). Nonanticipativity and l^1-martingales in stochastic optimization problems. In *Stochastic Systems: Modeling, Identification and Optimization II*, pages 170–187. Springer.

Ruszczyński, A. (1993). Parallel decomposition of multistage stochastic programming problems. *Mathematical programming*, 58(1-3):201–228.

Ruszczyński, A. (2013). Advances in risk-averse optimization. In *INFORMS Tutorials in Operations Research*. INFORMS.

Ryan, K., Rajan, D., and Ahmed, S. (2015). Scenario decomposition for 0-1 stochastic programs: Improvements and asynchronous implementation. Available at Optimization-Online `http://www.optimization-online.org/DB_FILE/2015/11/5201.pdf`.

Shapiro, A. (2012). Minimax and risk averse multistage stochastic programming. *European Journal of Operational Research*, 219(3):719–726.

Shapiro, A. and Ahmed, S. (2004). On a class of minimax stochastic programs. *SIAM Journal on Optimization*, 14(4):1237–1249.

Shapiro, A., Dentcheva, D., and Ruszczyński, A. (2009). *Lectures on Stochastic Programming: Modeling and Theory*, volume 9. SIAM, Philadelphia, PA.

Watson, J.-P., Wets, R. J., and Woodruff, D. L. (2010). Scalable heuristics for a class of chance-constrained stochastic programs. *INFORMS Journal on Computing*, 22(4):543–554.

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

APPENDIX

A Time spent on each step of the default and DD1 implementations

For the total solution time given in Table 2, we break it down and record the time of solving scenario subproblems (i.e., **Scen**$(\cdot)$ in DD1), and the time of evaluating solutions (i.e., **Eval**$(\cdot)$), respectively presented in the columns *scen_time* and *eval_time* in Table 8. We also report in column #*_iter* the number of iterations required for the upper and lower bounds to converge.

Table 8: Time (in second) and iteration counts of default and DD1 algorithms

		SSLP				SMKP			
		_50	_100	_500	_1000	_20	_40	_80	_160
default	total_time	195	201	(100%)	(100%)	299	(0.09%)	(0.11%)	(0.16%)
DD1	total_time	248	502	4663	12750	2692	9866	11249	18774
	scen_time	68	98	110	4857	2688	9853	11218	18704
	eval_time	180	404	4552	7893	3	13	31	70
	#_iter	4	3	2	2	2	1	1	2

B Parallel Algorithm for DD2 and DD3

Unlike DD1, both DD2 and DD3 contain an inner loop that improves ℓ by adjusting objective-function parameters as opposed to shrinking the feasible region. Let Λ_k represent the concatenation of parameters that affect the scenario-k subproblem and vary in the inner loop. Let $h(\cdot)$ represent the function that maps $(\beta_k^{\text{DD}i}, \hat{x}^k, \Lambda_k)_{k=1}^K$ to a tentative lower bound, and $r\big((\beta_k^{\text{DD}i}, \hat{x}^k, \Lambda_k)_{k=1}^K\big) \geq$

0 represent the stop condition of the inner loop. We redescribe the double-loop structure as a single loop, and present a general parallel scheme for DD2 and DD3 in Algorithm 12.

Algorithm 12 The Parallel Scheme for DD2 and DD3 at Proc^n $(n \in \{1, \ldots, N\})$

1: $u \leftarrow +\infty$, $\ell \leftarrow -\infty$
2: initialize $\Lambda_1, \ldots, \Lambda_K$
3: **repeat**
4: $\quad u_n \leftarrow +\infty$
5: $\quad$ **for** $k \in \Omega_n^{K,N}$ **do**
6: $\quad\quad (\beta_k^{\mathrm{DD}i}, \hat{x}^k) \leftarrow$ a scenario-k-based subproblem parameterized by Λ_k
7: $\quad\quad u_n \leftarrow \min\{u_n, \textbf{Eval}(\hat{x}^k)\}$
8: $\quad$ **end for**
9: $\quad$ pass $\{(\beta_k^{\mathrm{DD}i}, \hat{x}^k) : k \in \Omega_n^{K,N}\}$ and u_n to all the other processes
10: $\quad \ell \leftarrow \max\{\ell, h\big((\beta_k^{\mathrm{DD}i}, \hat{x}^k, \Lambda_k)_{k=1}^K\big)\}$
11: $\quad u \leftarrow \min\{u, u_1, \ldots, u_N\}$
12: $\quad$ **if** $r\big((\beta_k^{\mathrm{DD}i}, \hat{x}^k, \Lambda_k)_{k=1}^K\big) < 0$ **then**
13: $\quad\quad$ update $\Lambda_1, \ldots, \Lambda_K$
14: $\quad$ **else**
15: $\quad\quad X \leftarrow X \setminus S$
16: $\quad\quad$ reset $\Lambda_1, \ldots, \Lambda_K$
17: $\quad$ **end if**
18: **until** $u - \ell < \epsilon$

This algorithm is similar to BP. The major difference is in the end of each iteration there are two options. If $r\big((\beta_k^{\mathrm{DD}i}, \hat{x}^k, \Lambda_k)_{k=1}^K\big) < 0$, we update $\Lambda_1, \ldots, \Lambda_K$ (i.e., step 13) which corresponds to proceeding to the next iteration of the inner loop in the double-loop algorithm. Otherwise, we shrink the feasible region and reset $\Lambda_1, \ldots, \Lambda_K$ (i.e., steps 15,16) which corresponds to closing the inner loop and proceeding to the next iteration of the outer loop.