A scalable bounding method for multi-stage stochastic integer programs

Burhaneddin Sandıkçı
Booth School of Business, University of Chicago, Chicago, IL 60637, burhan@chicagobooth.edu

Osman Y. Özaltın
Edward P. Fitts Department of Industrial and Systems Engineering, North Carolina State University, 111 Lampe Dr., Raleigh, NC 27695, oyozalti@ncsu.edu

May 30, 2014

Many dynamic decision problems involving uncertainty can be appropriately modeled as multi-stage stochastic programs. However, most practical instances are so large and/or complex that it is impossible to solve them on a single computer, especially due to memory limitations. Extending the work of Sandıkçı et al. (2013) on two-stage stochastic mixed-integer-programs (SMIPs), this paper develops a bounding method for general multi-stage SMIPs that is based on scenario decomposition. This method is broadly applicable, as it does not assume any problem structure including convexity. Moreover, it naturally fits into a distributed computing environment, which can address truly large-scale instances. Extensive computational experiments with large-scale instances (with up to 40 million scenarios, 5.2×10^8 binary variables and 3.3×10^8 constraints) demonstrate that the proposed method scales nicely with problem size and has immense potential to obtain high quality solutions to practical instances within a reasonable time frame.

Key words: Stochastic programming, Multi-stage, Mixed-integer variables, Bounding, Parallel computing

1. Introduction

Many dynamic decision problems involving uncertainty can be appropriately modeled as stochastic programs (SP). Examples include problems in energy (Wallace and Fleten 2003), finance (Infanger 2006, Mulvey and Shetty 2004), manufacturing (Ahmed and Sahinidis 2003), telecommunication (Gaivoronski 2005), transportation and logistics (Powell and Topaloglu 2003), and health care (Erdogan and Denton 2013, Özaltın et al. 2011). Despite their appeal as a flexible modeling framework, SPs suffer exponential growth in their size with the number of decision stages. Moreover, there is a lack of generally applicable efficient solution algorithms for SPs with integer decision variables. Therefore, practical applications have generally been restricted to two-stage linear programs, truncating or merging the problem data, and therefore, losing information that is otherwise available. Two-stage programs, however, lack an ability to dynamically respond to information that gradually becomes available to the decision maker. Furthermore, the linearity assumption hinders the applicability of SPs to a broad set of real life problems.

Developing solution algorithms for multi-stage stochastic mixed-integer programs (SMIPs) is an active research area. Significant progress has been made in developing tailored algorithms to solve special classes of SMIPs, particularly for two-stage models (for reviews, see Birge and Louveaux 2011, Römisch and Schultz 2001, Sen 2005). Decomposition based methods are commonly used for multi-stage models. Originally proposed for stochastic linear programs, progressive hedging is successfully applied as a heuristic to solve multi-stage SMIPs (Watson and Woodruff 2011). As for exact algorithms, Lulli and Sen (2004) propose a branch-and-price (BP) approach, which relaxes the nonanticipativity constraints to decompose the problem into scenario subproblems, and then applies column generation. Carøe and Schultz (1999) propose a dual decomposition (DD) approach, which dualizes the nonanticipativity constraints, and then uses the bounds obtained from the Lagrangian dual problems within a branch-and-bound framework. Lubin et al. (2013) show an effective equivalence between the Lagrangian dual problem solved by DD and the restricted master problem solved by BP, and present numerical results from a parallel implementation that exploits special structure. In a related recent study, Ahmed (2013) proposes a scenario decomposition algorithm for two-stage SPs with binary first-stage variables. He reports encouraging results from a parallel implementation achieving almost perfect speedup.

Exact solution algorithms generally require discrete random variables and suffer from long running times due to slow convergence. To alleviate some of these difficulties, stochastic algorithms that return statistical bounds have been developed. Examples include sample average approximation (SAA) and stochastic decomposition (SD). The main idea of the SAA method is to approximate the expected recourse function by a sample average function, which is calculated from optimizing several SAA-subproblems, each of which is defined by a finite number of realizations sampled from the given distribution of the random variables in the full problem (see, for example, Kall and Mayer 2011 for more details on the SAA method). Using the SAA method, one obtains statistical lower and upper bounds for the optimal objective value and candidate solutions for the full problem. Increasing the sample size used in the SAA-subproblems monotonically improves (in expectation) the lower bound for minimization problems (Mak et al. 1999, Norkin et al. 1998). Norkin et al. (1998) use these bounds in a branch-and-bound algorithm. In practice, however, using crude Monte-Carlo sampling in constructing SAA-subproblems has slow convergence, hence variance reduction techniques are considered as a remedy. Another major difficulty in using the SAA method is determining the size of a sample to obtain good approximate solutions, for which there is no generally accepted approach. Originally proposed for two-stage linear programs with fixed recourse (Higle and Sen 1991), the SD method constructs a lower bounding approximation of the sample average function by sequentially sampling a scenario and solving the associated recourse problem. Extension of this framework to multi-stage linear programs is offered in (Sen and Zhou 2014).

Various bounding methods are also proposed to approximate SPs. A common approach is to replace the true distribution of the random parameters by finitely supported approximate distributions. The most celebrated examples of this approach are Jensen's inequality (Jensen 1906) for lower bounding and Edmundson-Madansky inequality (Edmundson 1956, Madansky 1960) for upper bounding, which have been extended in many dimensions (Birge and Wets 1987, 1989, Dokov and Morton 2005, Edirisinghe 1996, Edirisinghe and Ziemba 1994, Frauendorfer 1988, 1996, Kuhn 2004). An alternative bounding approach is based on aggregation (Birge 1985, Kuhn 2008, Wright 1994). These bounding approaches can also be incorporated into exact solution algorithms. In doing so, the support of the scenario set is typically partitioned into a number of subsets. Bounds are then applied in each of those subsets, and the partition is refined sequentially to achieve tighter bounds (Birge and Louveaux 2011).

The aforementioned bounds typically rely on some form of convexity assumption in the problem structure, and therefore, do not generally apply to SMIPs. Our primary objective in this paper is to present a bounding method for general multi-stage SMIPs that can efficiently compute approximate solutions in a timely fashion. In particular, we propose a scenario decomposition method that returns lower and upper bounds for the optimal objective value of the full problem. This paper extends our previous work on two-stage SMIPs (Sandıkçı et al. 2013). As the proposed methodology does not rely on convexity assumptions, it is much broadly applicable than most approaches in the literature. Moreover, an inherent feature of this approach is its natural fit into a distributed computing environment, which makes it amenable to solving truly large-scale instances. Using a modern computing cluster, we illustrate the performance of this method with a rich set of results on problems from the literature. The size of the instances we solve are truly large compared to what is reported previously. We observe immense potential to obtain high quality solutions to large-scale multi-stage SMIPs within a reasonable time frame.

The rest of this paper is organized as follows. We review multi-stage programs in §2. We present our decomposition approach in §3, followed in §4 by the bounds based on this decomposition. We present extensive numerical results in §5, and conclude in §6 with a summary and future research directions.

2. Multi-stage stochastic programs

We consider finite-horizon sequential decision making problems under uncertainty. Decisions are made at discrete stages, indexed by $t = 1, 2, \ldots, T$, using only the available information by that stage.

A finite number of random events $\omega_t \in \Omega_t$ may be realized in stage t. The decision process starts with an initial decision x_1 in stage 1. In each subsequent stage $t = 2, \dots, T$, first an outcome of the random event ω_t is observed and then a recourse action x_t is taken. The decision-maker accrues a contribution $f_t(x_t, \omega_t)$ for taking action x_t in stage t, which is also a function of the random event ω_t. Furthermore, the set of permissible decisions $\mathcal{X}_t \subseteq \mathbb{R}^{n_t}$, where n_t is a positive integer, in stage t may depend on the history of decisions $x_{1:t-1} := (x_1, \dots, x_{t-1})$ as well as the observed outcomes of the random events $\omega_{2:t} := (\omega_2, \dots, \omega_t)$.

We associate a random vector $\tilde{\xi}_t \in \mathbb{R}^{r_t}$ with each outcome $\omega_t \in \Omega_t$, and drop the explicit dependence on ω_t and hereafter write, for example, $f_t(x_t, \tilde{\xi}_t)$ for the objective function contribution in stage t. The overall objective is to minimize the total expected contributions, so a nested formulation is

$$\min_{x_1 \in \mathcal{X}_1} f_1(x_1) + \mathbb{E}_{\tilde{\xi}_2}\left[\min_{x_2 \in \mathcal{X}_2} f_2(x_2, \tilde{\xi}_2) + \cdots + \mathbb{E}_{\tilde{\xi}_T|\xi_{2:T-1}}\left[\min_{x_T \in \mathcal{X}_T} f_T(x_T, \tilde{\xi}_T)\right]\right], \tag{1}$$

where $\mathbb{E}_{\tilde{\xi}_{t+1}|\xi_{2:t}}[\cdot]$ is the expectation with respect to $\tilde{\xi}_{t+1}$ conditioned on the realization $\xi_{2:t} := \{\xi_2, \dots, \xi_t\}$, and $\mathcal{X}_t = \left\{x_t \in \mathbb{R}^{n_t}_+ : g_t(x_{1:t}, \tilde{\xi}_{2:t}) = h_t(\tilde{\xi}_{2:t})\right\}$ is the set of permissible decisions in stage $t = 1, \dots, T$, where $g_t(\cdot) : \mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_t} \times \mathbb{R}^{r_2} \times \cdots \times \mathbb{R}^{r_t} \to \mathbb{R}^{m_t}$ and $h_t(\cdot) : \mathbb{R}^{r_2} \times \cdots \times \mathbb{R}^{r_t} \to \mathbb{R}^{m_t}$ for positive integers n_t, r_t, and m_t for all t. Note that some or all components of the decision vector x_t can be restricted to integers.

We assume that, for all stages t, the random vector $\tilde{\xi}_t$ is discrete with finite number of realizations. Thus, the stochastic process $\tilde{\xi} = \{\tilde{\xi}_2, \tilde{\xi}_3, \dots, \tilde{\xi}_T\}$ has a finite support $\Xi = \{\xi^1, \xi^2, \dots, \xi^N\}$, where $\xi^s = (\xi^s_2, \xi^s_3, \dots, \xi^s_T)$ for $s \in \mathcal{S} := \{1, 2, \dots, N\}$ is referred to as a *scenario* and is associated with a probability mass p^s. As a result of this assumption, the uncertainty in the decision process can be represented by a scenario tree as illustrated in Figure 1. Each layer of the tree corresponds to a stage t of the decision process. The single (root) node, indexed by 0, in stage 1 is due to the fact that the first decision x_1 is made prior to observing the outcomes of any random events. Let $\mathcal{N}_t$ denote the set of nodes in stage t, $\mathcal{N} := \bigcup_{t=1}^T \mathcal{N}_t$ denote the set of all nodes in the tree, $\mathcal{A}_{m,n}$ denote the set of nodes on the path from node $m \in \mathcal{N}$ to $n \in \mathcal{N}$, $a(n)$ denote the parent node and $\mathcal{C}_n$ denote the set of children nodes of node $n \in \mathcal{N}$ (clearly, $a(0) = \emptyset$, $\mathcal{C}_n = \emptyset$ for $n \in \mathcal{N}_T$). Then, any node $n \in \mathcal{N}_t$ for $t = 2, \dots, T$ represents a particular realization of $\tilde{\xi}_{2:t}$, which can be interpreted as the state of the system in stage t. The (conditional) probability π_n of a node $n \in \mathcal{N}_t$ is $P\{\tilde{\xi}_t | \tilde{\xi}_{2:t-1}\}$, so $\sum_{n \in \mathcal{C}_m} \pi_n = 1$ for any node $m \in \mathcal{N}$. The unique path from the root node to a terminal node $n \in \mathcal{N}_T$ corresponds to a scenario, and thus the number of terminal nodes $|\mathcal{N}_T|$ equals the number of scenarios N. The probability associated with taking a path $s \in \mathcal{S}$ from the root node to a terminal node is simply p^s, which equals the product of the conditional probabilities of all nodes that belong to path s. Let $\mathcal{S}_n$ be the index set of scenarios passing through node n. Then the marginal probability of node $n \in \mathcal{N}$ is given by $q_n = \sum_{s \in \mathcal{S}_n} p^s$.

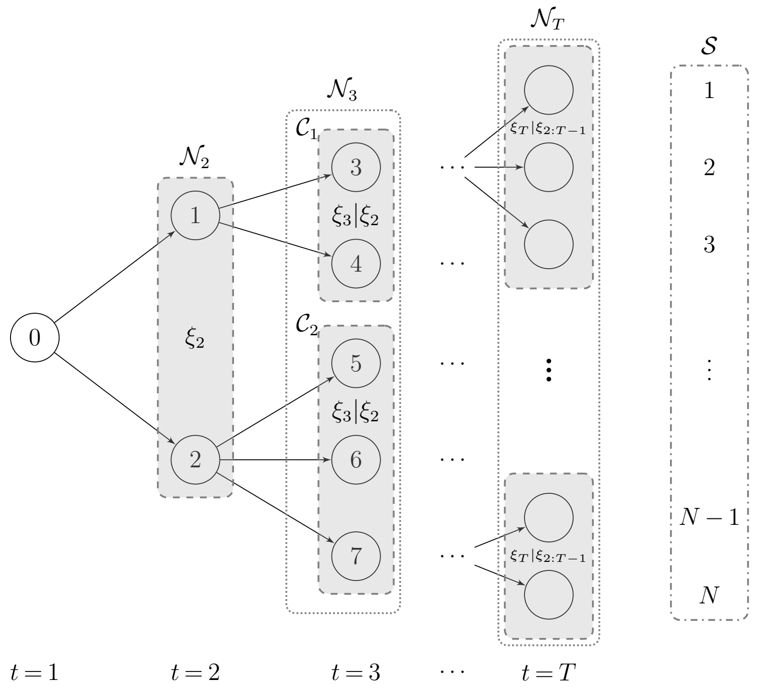


Figure 1 Scenario tree representation of uncertainty.

When there is a finite number of scenarios, problem (1) can be equivalently formulated as a large-scale deterministic program by duplicating the decision vector x_t in stage t for each scenario $s \in \mathcal{S}$. Letting x_t^s denote the decisions made in stage t under scenario s, the extensive form is written as:

$$z^* = \min \sum_{s\in\mathcal{S}} p^s \cdot \sum_{t=1}^{T} f_t\left(x_t^s, \xi_t^s\right) \tag{2a}$$

$$\text{s.t. } g_t\left(x_{1:t}^s, \xi_{2:t}^s\right) = h_t(\xi_{2:t}^s) \quad t=1,\ldots,T,\ s\in\mathcal{S}, \tag{2b}$$

$$x_t^{s_1} = x_t^{s_2} \qquad t=1,\ldots,T,\ s_1,s_2\in\mathcal{S} \text{ s.t. } \xi_{2:t}^{s_1} = \xi_{2:t}^{s_2}, \tag{2c}$$

$$x_t^s \in \mathbb{R}^{n_t} \qquad t=1,\ldots,T,\ s\in\mathcal{S}. \tag{2d}$$

Constraints (2c) are typically referred to as the nonanticipativity constraints, which ensure that, for any stage t, a subset of the duplicated set of decision vectors $\{x_t^1, x_t^2, \ldots, x_t^N\}$ that share a common history must be equal to each other. That is, if two scenarios $s_1, s_2 \in \mathcal{S}$ are indistinguishable from each other up to stage t, then the associated decision vectors $x_t^{s_1}$ and $x_t^{s_2}$ must not differ. We call this formulation the *explicit* formulation since the nonanticipativity conditions are modeled explicitly with constraints (2c).

Alternatively, the nonanticipativity conditions can be modeled implicitly by redefining the decision vector x_t in stage t for each node $n \in \mathcal{N}_t$ of the scenario tree. Let x^n denote the decisions made at node

$n \in \mathcal{N}_t$ in stage t (i.e., $x^n = x_t^s$ under all scenarios $s \in \mathcal{S}_n$); and for $s \in \mathcal{S}_n$ define $f^n(x^n) \equiv f_t(x_t^s, \xi_t^s)$, $g^n(\{x^i\}_{i\in\mathcal{A}_{0,n}}) \equiv g_t(x_{1:t}^s, \xi_{2:t}^s)$, and $h^n \equiv h_t(\xi_{2:t}^s)$. Then the *implicit* formulation can be written as:

$$z^* = \min \quad \sum_{n\in\mathcal{N}} q_n f^n(x^n) \tag{3a}$$

$$\text{s.t.} \quad g^n\left(\{x^i\}_{i\in\mathcal{A}_{0,n}}\right) = h^n \quad n \in \mathcal{N}, \tag{3b}$$

$$x^n \in \mathbb{R}^{n_t} \quad n \in \mathcal{N}_t, \quad t = 1,\ldots,T. \tag{3c}$$

While the implicit formulation is more compact than the explicit formulation, the special constraint structure of the explicit formulation may be exploited in developing efficient solution algorithms for specific instances. Nevertheless, in most applications, both formulations result in very large-scale mathematical programs and so neither formulation is amenable to direct solutions with commercial solvers. The solution method proposed in this paper applies to either formulation with no additional difficulty. For notational convenience, we use the implicit formulation to illustrate our method.

3. A group subproblem

Our goal in this section is to construct a deterministic mathematical program similar to problem (3), but smaller in size so that it can be solved with less computational effort (e.g., by an off-the-shelf commercial solver). One way to construct such a program is to write problem (3) for a subset $\Gamma \subseteq \mathcal{S}$ of scenarios, which we refer to as a *block* henceforth. Deriving valid bounds for z^* from such restricted problems requires adjusting the probability of each scenario $s \in \Gamma$ in a structured manner.

Let $\mathcal{P}(\mathcal{S})$ be the power set of $\mathcal{S}$ excluding the empty set. A *blockset* $\mathcal{G}$ is a set of blocks such that $\mathcal{G} \subseteq \mathcal{P}(\mathcal{S})$ and $\bigcup_{\Gamma\in\mathcal{G}} \Gamma = \mathcal{S}$. For any blockset $\mathcal{G}$, let $m_s(\mathcal{G})$ be the multiplicity of scenario $s \in \mathcal{S}$ in $\mathcal{G}$. That is,

$$m_s(\mathcal{G}) = \sum_{\Gamma\in\mathcal{G}} \mathbf{1}_\Gamma(s), \tag{4}$$

where $\mathbf{1}_\Gamma(s) = 1$ if $s \in \Gamma$, and 0 otherwise. And, for $\Gamma \in \mathcal{G}$ and $s \in \Gamma$, define the adjusted probability of scenario s as

$$\hat{p}^s := \frac{p^s/m_s}{\rho(\Gamma)}, \tag{5}$$

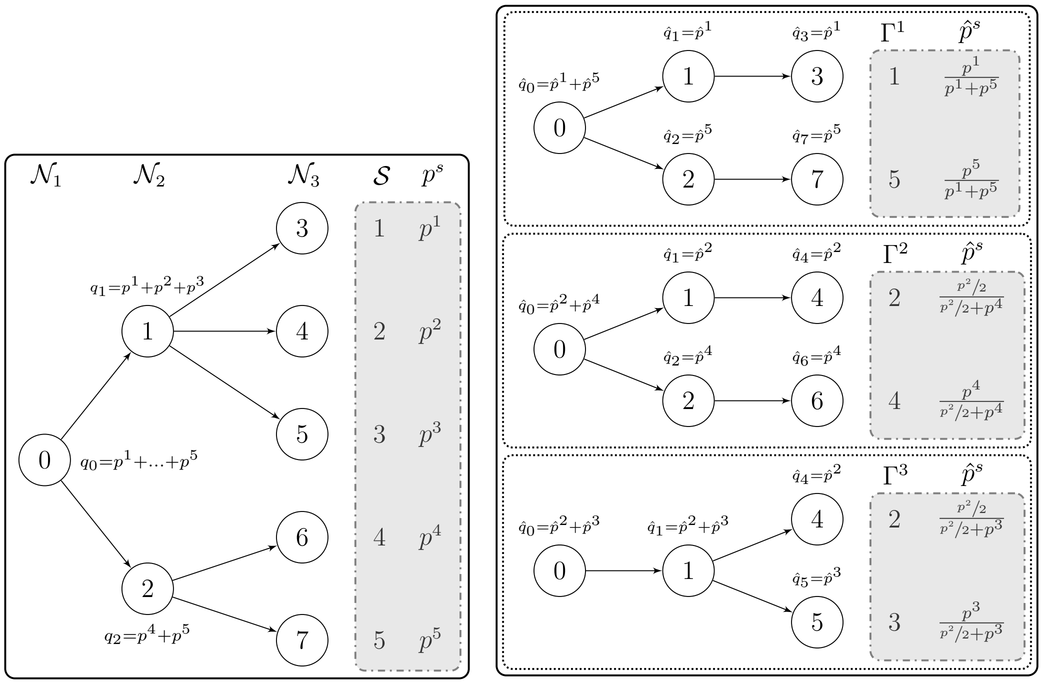
where $\rho(\Gamma) := \sum_{s\in\Gamma} p^s/m_s$. Also let $\mathcal{N}^\Gamma \subseteq \mathcal{N}$ be the set of nodes of the original scenario tree that are associated with the scenarios in $\Gamma \in \mathcal{G}$, and $\mathcal{N}_t^\Gamma$ denote those nodes of $\mathcal{N}^\Gamma$ at stage $t = 1,\ldots,T$. And define the adjusted probability of node $n \in \mathcal{N}^\Gamma$ as $\hat{q}_n := \sum_{s\in\Gamma_n} \hat{p}^s$, where Γ_n is defined (similar to $\mathcal{S}_n$) as the set of scenarios in Γ that pass through node n.

Definition 1. The *group subproblem* for a block of scenario indices $\Gamma \subseteq \mathcal{S}$ is

$$(GR(\Gamma)) \qquad z(\Gamma) = \min \quad \sum_{n\in\mathcal{N}^\Gamma} \hat{q}_n f^n(x^n)$$

$$\text{s.t.} \quad g^n\left(\{x^i\}_{i\in\mathcal{A}_{0,n}}\right) = h^n \quad n \in \mathcal{N}^\Gamma,$$

$$x^n \in \mathbb{R}^{n_t} \quad n \in \mathcal{N}_t^\Gamma, \quad t = 1,\ldots,T.$$



(a) Scenario tree for the original problem **(b)** Scenario trees for group subproblems

Figure 2 An example group subproblem decomposition.

This definition of a group subproblem extends the one in Sandıkçı et al. (2013), which uses a *reference scenario* to temper the effect of extreme scenarios so that each group subproblem solution is not too biased. Unlike Sandıkçı et al. (2013), our definition no longer requires a reference scenario, the choice of which is not obvious in general. However, we could still include a reference scenario even as an artificial scenario (i.e., not in the sample set) in our group subproblem with no added difficulty. This could serve again to balance the members of a group to ensure some level of consistency. As another advancement over Sandıkçı et al. (2013), we allow the multiplicity m_s of scenario $s \in \mathcal{S}$ to be larger than 1, which may return tighter bounds. In both definitions, the adjusted probabilities of the scenarios in Γ add up to 1.0, which is required for a well-defined expected value.

Also note that for any block $\Gamma \subseteq \mathcal{S}$, a similar group subproblem based on the explicit formulation can be obtained by replacing $\mathcal{S}$ with Γ and p^s with $\hat{p}^s$ for $s \in \Gamma$ in problem (2).

Figure 2 illustrates group subproblems and our notation. The original stochastic program depicted in Figure 2(a) has $T = 3$ stages and $N = 5$ scenarios indexed as $\mathcal{S} = \{1,2,3,4,5\}$. Figure 2(b) depicts the scenario trees of group subproblems for $\mathcal{G} = \{\Gamma^1, \Gamma^2, \Gamma^3\}$, where $\Gamma^1 = \{1,5\}$, $\Gamma^2 = \{2,4\}$, and $\Gamma^3 = \{2,3\}$, and thus $m_1 = m_3 = m_4 = m_5 = 1$ and $m_2 = 2$. Note that $\Gamma^1 \cup \Gamma^2 \cup \Gamma^3 = \mathcal{S}$ as required by our decomposition, and that each group subproblem has the same number of stages as the original problem, but defined over a reduced number of scenarios.

4. Bounds from the group subproblems

We now illustrate how one can generate lower and upper bounds for z^* from a blockset $\mathcal{G}$. A unique feature of these bounds is that they can solve the group subproblems $GR(\Gamma)$ for $\Gamma \in \mathcal{G}$ independently in parallel, which makes this method attractive to potentially address large-scale instances. The user has the flexibility of adjusting the block size, i.e., number of scenarios in the group subproblems, according to the available computational resources. Moreover, the proposed bounds are broadly applicable, because they do not assume convexity.

4.1. A lower bound

Our main result in this section (Proposition 1) proves a lower bound for the optimal objective value z^* of the full problem (3) for any given blockset $\mathcal{G}$. Let

$$\mathcal{L}(\mathcal{G}) := \sum_{\Gamma \in \mathcal{G}} \rho(\Gamma) \cdot z(\Gamma)$$

denote the weighted average of the optimal objective values of group subproblems defined by $\Gamma \in \mathcal{G}$.

Lemma 1. $\sum_{\Gamma \in \mathcal{G}} \rho(\Gamma) = 1$.

Proof of Lemma 1. $\sum_{\Gamma \in \mathcal{G}} \rho(\Gamma) = \sum_{\Gamma \in \mathcal{G}} \sum_{s \in \Gamma} \frac{p^s}{m_s} = \sum_{s \in \mathcal{S}} m_s \frac{p^s}{m_s} = 1.$ □

Lemma 2. *For any function $h(\cdot) : \mathcal{S} \to \mathbb{R}$, $\sum_{\Gamma \in \mathcal{G}} \sum_{s \in \Gamma} \frac{p^s}{m_s} \cdot h(\{s\}) = \sum_{s \in \mathcal{S}} p^s \cdot h(\{s\})$.*

Proof of Lemma 2 $\sum_{\Gamma \in \mathcal{G}} \sum_{s \in \Gamma} \frac{p^s}{m_s} \cdot h(\{s\}) = \sum_{s \in \mathcal{S}} m_s \frac{p^s}{m_s} \cdot h(\{s\}) = \sum_{s \in \mathcal{S}} p^s \cdot h(\{s\}).$ □

Proposition 1. *$\mathcal{L}(\mathcal{G}) \le z^*$ for any blockset $\mathcal{G}$.*

Proof of Proposition 1 Let $(\tilde{x}^n)_{n \in \mathcal{N}}$ be an optimal solution to the full problem (3). Let $\mathcal{G}$ denote an arbitrary blockset. Substituting this solution into the objective function of the problem $GR(\Gamma)$ for $\Gamma \in \mathcal{G}$, we find

$$z(\Gamma) \le \sum_{n \in \mathcal{N}^\Gamma} \hat{q}_n f^n(\tilde{x}^n) = \sum_{n \in \mathcal{N}^\Gamma} \sum_{s \in \Gamma_n} \hat{p}^s f^n(\tilde{x}^n) = \sum_{n \in \mathcal{N}^\Gamma} \sum_{s \in \Gamma_n} \frac{p^s/m_s}{\rho(\Gamma)} f^n(\tilde{x}^n)$$

Multiplying this inequality with $\rho(\Gamma)$ and summing for $\Gamma \in \mathcal{G}$, we find

$$\mathcal{L}(\mathcal{G}) = \sum_{\Gamma \in \mathcal{G}} \rho(\Gamma) \cdot z(\Gamma) \le \sum_{\Gamma \in \mathcal{G}} \sum_{n \in \mathcal{N}^\Gamma} \sum_{s \in \Gamma_n} \frac{p^s}{m_s} f^n(\tilde{x}^n) \tag{6a}$$

$$= \sum_{n \in \mathcal{N}} \sum_{s \in \mathcal{S}_n} p^s f^n(\tilde{x}^n), \tag{6b}$$

$$= \sum_{n \in \mathcal{N}} q_n f^n(\tilde{x}^n), \tag{6c}$$

$$= z^*, \tag{6d}$$

where equality (6b) follows because $\bigcup_{\Gamma \in \mathcal{G}} \Gamma = \mathcal{S}$. □

The quality of the lower bound $\mathcal{L}(\mathcal{G})$ clearly depends on the choice of the blockset $\mathcal{G}$. Thus, we are interested in finding a set $\mathcal{G}^* \subseteq \mathcal{P}(\mathcal{S})$ that returns the tightest lower bound $\mathcal{L}(\mathcal{G}^*)$ for z^*, which can be specified as:

$$\max_{\mathcal{G} \subseteq \mathcal{P}(\mathcal{S})} \mathcal{L}(\mathcal{G}). \tag{7}$$

Without any restrictions on how $\mathcal{G}$ can be formed, an optimal solution to this problem is $\mathcal{G}^* = \mathcal{S}$, but evaluating $\mathcal{L}(\mathcal{G}^*)$ is as difficult as solving the original problem (3). Therefore, we restrict the size of each block $\Gamma \in \mathcal{G}$ by $b > 0$:

$$\max_{\mathcal{G} \subseteq \mathcal{P}(\mathcal{S})} \quad \mathcal{L}(\mathcal{G}) \tag{8a}$$

$$\text{s.t.} \quad |\Gamma| \le b \quad \text{for } \Gamma \in \mathcal{G}. \tag{8b}$$

The parameter b can only be an integer in $\{1, 2, \ldots, N\}$, otherwise causing infeasibility, and must be chosen considering the available computational resources. A useful question is to characterize if, for a given b, constraints (8b) can be non-binding at optimality. Proposition 2 answers this question negatively. That is, solving a group subproblem for any $\Gamma \subseteq \mathcal{S}$ dominates solving multiple subproblems defined by any combination of subsets of Γ.

Proposition 2. *For $\Gamma \subseteq \mathcal{S}$ and any blockset $\mathcal{G} \subseteq \mathcal{P}(\Gamma)$,*

$$\rho(\Gamma)\, z(\Gamma) \ge \sum_{\Gamma^i \in \mathcal{G}} \rho(\Gamma^i) z(\Gamma^i).$$

Proof of Proposition 2 Similar to the proof of Proposition 1. □

This result implies that an optimal solution $\mathcal{G}^*$ to problem (8) is composed of blocks of size b each, except may be for one block of size $N \pmod{b}$ if N is not divisible by b. As a result, increasing b yields a tighter lower bound at the expense of solving larger group subproblems $GR(\Gamma)$ for $\Gamma \in \mathcal{G}^*$.

Computing $\mathcal{L}(\mathcal{G})$ introduces flexibility in constructing group subproblems as it allows the scenarios to be repeated for an unequal number of times in $\mathcal{G}$, which may be a preferred strategy if some scenarios have more influence on the optimal objective value z^* than others. However, Proposition 3 establishes that it is needless to *uniformly* increase the multiplicity of all scenarios $s \in \mathcal{S}$ from $m_s = 1$. That is, if $m_s = m_{s'}$ is required for all $s, s' \in \mathcal{S}$ when constructing a blockset $\mathcal{G}$, then a best lower bounding blockset $\mathcal{G}^*$ must be a *partition* of $\mathcal{S}$, implying $m_s(\mathcal{G}^*) = 1$ for all $s \in \mathcal{S}$. On the other hand, it can be shown that the best bound corresponding to a partition of $\mathcal{S}$ may be further improved by *selectively* increasing the multiplicity of some scenarios, while it may be worsened if the multiplicities of a *random* set of scenarios are increased.

Proposition 3. *Consider problem (8) with the additional constraint $m_s = m$ for all $s \in \mathcal{S}$, for a given positive integer m. Let $\mathcal{G}_m^*$ be an optimal solution to this modified problem. Then,*

$$\mathcal{L}(\mathcal{G}_1^*) \ge \mathcal{L}(\mathcal{G}_m^*) \quad \text{for } m = 1, 2, \ldots, \sum_{k=1}^{b} \binom{N-1}{k-1}.$$

Proof of Proposition 3 For notational clarity, assume N is perfectly divisible by b – the proof when N is not divisible by b proceeds similarly. Let $\mathcal{P}_b(\mathcal{S}) := \{\Gamma \in \mathcal{P}(\mathcal{S}) : |\Gamma| = b\}$, $\mathbb{G}_m = \{\mathcal{G} \subseteq \mathcal{P}_b(\mathcal{S}) : |\mathcal{G}| = m\}$, and $\mathcal{G}_m \in \mathbb{G}_m$. Since N is divisible by b, Proposition 2 implies that $\mathcal{P}(\mathcal{S})$ can be replaced by $\mathcal{P}_b(\mathcal{S})$ and "$\leq$" can be replaced by "$=$" with no loss of optimality in the modified problem (8), so we only need to consider $m = 1, \ldots, \binom{N-1}{b-1}$. Observe that, given a block $\Gamma \subset \mathcal{S}$, the values of $z(\Gamma)$ for $\Gamma \in \mathcal{G}_1$ and $\Gamma \in \mathcal{G}_m$ $(m \geq 2)$ for some $\mathcal{G}_1$ and $\mathcal{G}_m$ are the same, because the adjusted probability $\hat{p}^s$ for all $s \in \Gamma$ are the same in either case, and therefore identical group subproblems $GR(\Gamma)$ are solved. Furthermore, the coefficient $\rho(\Gamma)$ of $z(\Gamma)$ when computing $\mathcal{L}(\mathcal{G}_m)$ is $\frac{1}{m}$ times that used when computing $\mathcal{L}(\mathcal{G}_1)$.

By definition,

$$\mathcal{L}(\mathcal{G}_1^*) = \sum_{\Gamma \in \mathcal{G}_1^*} \rho(\Gamma) z(\Gamma) \geq \sum_{\Gamma \in \mathcal{G}_1} \rho(\Gamma) z(\Gamma) \quad \text{for } \mathcal{G}_1 \in \mathbb{G}_1. \tag{9}$$

Observe that there exists $\mathcal{G}_1^1, \mathcal{G}_1^2, \ldots, \mathcal{G}_1^m \in \mathbb{G}_1$ such that $\mathcal{G}_m = \bigcup_{i=1}^m \mathcal{G}_1^i$ for any $\mathcal{G}_m \in \mathbb{G}_m$. Therefore, summing those m inequalities in (9) after multiplying them by $\frac{1}{m}$, and repeating this for all $\mathcal{G}_m \in \mathbb{G}_m$, we obtain

$$\begin{aligned}
\mathcal{L}(\mathcal{G}_1^*) &\geq \frac{1}{m}\left(\sum_{\Gamma \in \mathcal{G}_1^1} \rho(\Gamma) z(\Gamma) + \cdots + \sum_{\Gamma \in \mathcal{G}_1^m} \rho(\Gamma) z(\Gamma) \right) && \text{(10a)}\\
&= \mathcal{L}(\mathcal{G}_m) \qquad \text{for } \mathcal{G}_m \in \mathbb{G}_m, && \text{(10b)}
\end{aligned}$$

which completes the proof. $\square$

Proposition 3 also has significant implications for comparing the $\mathcal{L}(\mathcal{G}^*)$ bound, for a given block size b, to the $EGSO(b)$ bound derived in Sandıkçı et al. (2013). Note that $EGSO(b)$ requires solving group subproblems for all blocks $\Gamma \in \mathcal{P}_b(\mathcal{S})$, where $|\mathcal{P}_b(\mathcal{S})| = \binom{N}{b}$, and therefore $m_s = m = \binom{N-1}{b-1}$ for all $s \in \mathcal{S}$, and $EGSO(b) = \mathcal{L}(\mathcal{G}_m^*) \leq \mathcal{L}(\mathcal{G}_1^*) \leq \mathcal{L}(\mathcal{G}^*)$. Furthermore, $\mathcal{L}(\mathcal{G}_1)$ for any $\mathcal{G}_1$ offers significant computational savings over $EGSO(b)$. In particular, $EGSO(b)$ requires solving $\binom{N}{b}$ group subproblems, whereas $\mathcal{L}(\mathcal{G}_1)$ requires solving only $\lceil \frac{N}{b} \rceil$ group subproblems that are of identical size to those solved for $EGSO(b)$.

4.2. An upper bound

One needs an upper bound to evaluate the quality of the lower bounds proposed in §4.1. In practical applications, problem specific features may indeed warrant special solution approaches or even heuristics to find feasible solutions. However, we can also use the solution of a group subproblem to generate a feasible solution to the full problem. The main advantage of this approach is its generality in that it does not rely on any specific problem structure.

We generalize the approach first proposed in Birge (1982), and then improved in Sandıkçı et al. (2013). The main idea of our proposed upper bound is to use the solution from a given group

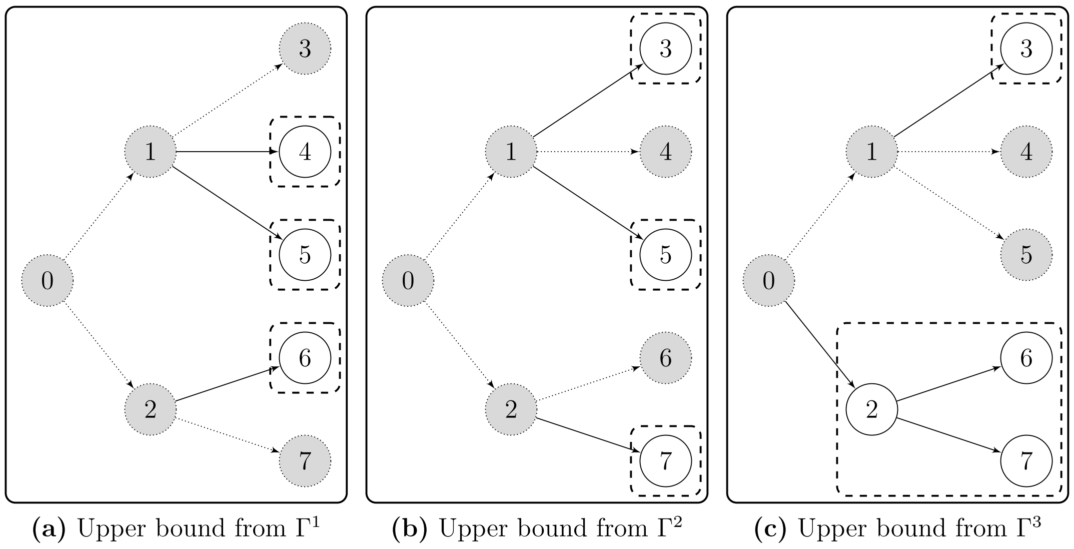

0
1
2
3
4
5
6
7
(a) Upper bound from Γ^1
(b) Upper bound from Γ^2
(c) Upper bound from Γ^3

Figure 3 Creating upper bounds for the example in Figure 2.

subproblem in the full problem. Let $\hat{x}(\Gamma)$ denote an optimal solution of the group subproblem $GR(\Gamma)$ for some $\Gamma \in \mathcal{G}$. If $\hat{x}(\Gamma)$ has a feasible completion when substituted into the full problem, then we conclude with a feasible solution for the full problem, and therefore the objective value associated with such a solution is an upper bound $U(\Gamma)$ for z^*. If, however, $\hat{x}(\Gamma)$ does not have a feasible completion in the full problem, then no upper bound can be obtained from using block Γ. We ultimately pick the best upper bound: $\mathcal{U}(\mathcal{G}) = \min_{\Gamma \in \mathcal{G}} U(\Gamma)$.

When $\hat{x}(\Gamma)$ is substituted, the resulting problem becomes smaller in size and possibly separates into multiple even further smaller problems, which can all be solved in parallel. To illustrate, consider the example in Figure 2. After substituting $\hat{x}(\Gamma^1)$ into the full problem in Figure 2(a), the residual problem is composed only of those decision variables and constraints that are associated with nodes 4, 5, and 6 as depicted in Figure 3(a), which is indeed separable into three distinct problems. If all three subproblems of the residual problem has a feasible solution, then we conclude with an upper bound $U(\Gamma^1)$. Figures 3(b) and (c) illustrate two alternative upper bounds that can be obtained by using $\hat{x}(\Gamma^2)$ and $\hat{x}(\Gamma^3)$, respectively.

The residual problem after substituting $\hat{x}(\Gamma)$ for some $\Gamma \subseteq \mathcal{S}$ may still be too large to be solved with the available computing resources. In this case, it can be recursively split into smaller subproblems. This aspect of our method provides immense flexibility in practice. However, the solution of each subproblem must adhere to the nonanticipativity constraints, and therefore the resulting bound may be weaker compared to the bound that could be obtained by solving the residual problem as a whole.

Figure 4 illustrates the recursive nature of our upper bounding method for Γ^3. Upon substituting $\hat{x}(\Gamma^3)$, the residual problem is composed of two separable pieces: one that corresponds to node 3 and another that corresponds to nodes 2, 6, and 7 as depicted in Figure 3(c). The problem of node 3

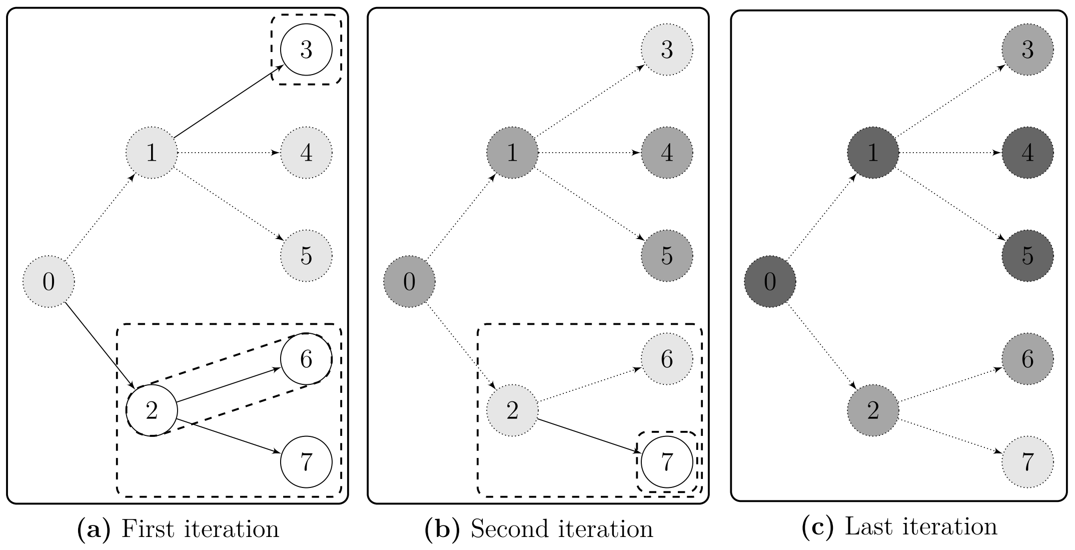


Figure 4 An alternative upper bound from Γ^3 obtained via recursively solving the residual problems.

may be solvable within the limits of computational resources. But, for the sake of argument, let's assume we can only solve problems with at most two nodes, and therefore we cannot directly solve the problem composed of nodes 2, 6, and 7. In Figure 4(a), we illustrate the first iteration of the recursion by solving the problem corresponding to nodes 2 and 6 only. Upon solving this subtree, the residual problem is now composed only of node 7 as depicted in Figure 4(b), which is solvable with the available computational resources.

5. Computational study

We test the numerical performance of the proposed method using two problems from the literature, which are described in §5.1. We discuss instance generation in §5.2, and illustrate the parallel performance of our distributed algorithm in §5.3. In all of our computations, we take $m_s = 1$ for all $s \in \mathcal{S}$, so that the blockset $\mathcal{G}$ forms a partition of $\mathcal{S}$. In §5.4, we illustrate the impact of choosing a blockset $\mathcal{G}$ on the performance of our bounds. We point out some practical considerations in §5.5, and present results for large-scale instances in §5.6.

All computations are done using the `sandyb` partition within the Midway high performance computing cluster at the University of Chicago Research Computing Center. This partition has 272 compute nodes that are linked with Infiniband interconnect. Each node has two eight-core 2.6GHz Intel 'Sandy Bridge' processors and 32GB of main memory. We use C++ to implement the proposed bounding approach, the callable library of CPLEX 12.5 in single thread mode (otherwise at default settings) to solve the subproblems, openMPI 1.6 for parallelization, and RandomLib 1.8 for random variate generation.

5.1. Test Problems

Our first test set is the multi-stage single-item stochastic lot-sizing problem studied in Guan et al. (2006, 2009). The objective of this problem is to determine a minimum cost production and inventory holding schedule for a product so as to satisfy its stochastic demand over a finite discrete planning horizon. A multi-stage SMIP formulation is:

$$\begin{aligned}
\text{[MLS]} \qquad & \min \sum_{n\in\mathcal{N}} q_n(\alpha_n x_n + \beta_n y_n + h_n s_n) && && \text{(11a)}\\
& \text{s.t. } s_{a(n)} + x_n = d_n + s_n, && n\in\mathcal{N}, && \text{(11b)}\\
& x_n \le M_n y_n, && n\in\mathcal{N}, && \text{(11c)}\\
& x_n, s_n \ge 0, \quad y_n \in \{0,1\}, && n\in\mathcal{N}, && \text{(11d)}\\
& s_{a(0)} = 0, && && \text{(11e)}
\end{aligned}$$

where x_n, s_n, and y_n denote the production level, inventory level, and setup indicator variables, and the parameters α_n, β_n, h_n, d_n, and M_n denote production cost, setup cost, inventory holding cost, demand, and production capacity, respectively, for scenario tree node $n\in\mathcal{N}$. Objective function (11a) minimizes the total expected inventory holding, production, and setup costs. Constraints (11b) enforce inventory balance conditions, (11c) enforce the production capacity limits, and (11e) enforces no initial inventory.

Our second test set is a multi-stage dynamic capacity acquisition problem, which seeks to determine a minimum cost schedule of acquiring capacity for a set $\mathcal{R}=\{1,\dots,I\}$ of resources to satisfy uncertain processing requirements for a set $\mathcal{T}=\{1,\dots,J\}$ of tasks. We extend the two-stage model of Ahmed and Garcia (2003) to a multi-stage setting to allow revising the capacity decisions as uncertainty unfolds gradually. A multi-stage SMIP formulation is:

$$\begin{aligned}
\text{[MCAP]}\ & \min \sum_{n\in\mathcal{N}} q_n \left(\sum_{i\in\mathcal{R}} \left[f_n^i u_n^i + g_n^i x_n^i \right] + \sum_{j\in\mathcal{T}} \left[s_n^j z_n^j + \sum_{i\in\mathcal{R}} c_n^{ij} y_n^{ij} \right] \right) && && \text{(12a)}\\
& \text{s.t. } x_n^i \le \kappa_n^i u_n^i, && n\in\mathcal{N}, i\in\mathcal{R}, && \text{(12b)}\\
& \sum_{j\in\mathcal{T}} d_n^j y_n^{ij} \le \sum_{n'\in\mathcal{A}_{0,n}} x_{n'}^i, && n\in\mathcal{N}, i\in\mathcal{R}, && \text{(12c)}\\
& \sum_{i\in\mathcal{R}} y_n^{ij} + z_n^j = 1, && n\in\mathcal{N}, j\in\mathcal{T}, && \text{(12d)}\\
& x_n^i \ge 0, \quad u_n^i, z_n^j, y_n^{ij} \in \{0,1\}, && n\in\mathcal{N}, i\in\mathcal{R}, j\in\mathcal{T}, && \text{(12e)}
\end{aligned}$$

where u_n^i, y_n^{ij}, and z_n^i are indicator variables for expanding capacity of resource i, assigning resource i to task j, and not completing task j, respectively, and x_n^i is the amount of capacity expansion for resource i at scenario tree node $n\in\mathcal{N}$, for $i\in\mathcal{R}$ and $j\in\mathcal{T}$. The parameters f_n^i and g_n^i, respectively, denote the fixed and variable costs of expanding capacity for resource i, c_n^{ij} denotes the cost of

Table 1 Scenario tree sizes for the test instances

Branches (r)	Stages (T)	Scenarios (N)	Nodes ($\|\mathcal{N}\|$)
100	4	1,000,000	1,010,101
50	5	6,250,000	6,377,551
5	11	9,765,625	12,207,031
10	8	10,000,000	11,111,111
3	16	14,348,907	21,523,360
2	25	16,777,216	33,554,431
7	10	40,353,607	47,079,208

assigning resource i to task j, s_n^j denotes the cost of not completing task j, κ_n^i denotes the upper limit on the capacity of resource i, and d_n^j denotes the processing requirement of task j at scenario tree node $n \in \mathcal{N}$, for $i \in \mathcal{R}$ and $j \in \mathcal{T}$. Objective function (12a) minimizes the total expected cost of capacity acquisition and resource assignment to tasks. Constraints (12b) enforce the capacity acquisition limit for each resource, (12c) limit the total assigned capacity of a resource by the acquired capacity of that resource, and (12d) identify if task j is assigned to any resource or not.

5.2. Instance generation

For each problem class, we randomly generate multiple instances using different initial seeds and symmetric scenario trees. Two parameters that are common to all instances are the number of stages T, and the number of scenario tree branches r emanating from each non-terminal node. The pair (r, T) determines the size of the scenario tree. In particular, any instance with T stages and r branches per scenario tree node is composed of r^{T-1} scenarios and $(r^T - 1)/(r - 1)$ scenario tree nodes. Table 1 displays various scenario trees that we use in generating instances from each problem class.

For any given instance, we generate the data of a particular node in its scenario tree on the fly when that node's data is needed, and then discard this data from the memory as soon as it is processed. At times, this approach may require re-generating the same data more than once. We ensure re-generating the exact same data for a node when it is needed more than once by re-seeding our random number generator with a function of the unique node index. We choose this implementation to avoid running out of the allocated 2GB computer memory per processor at the expense of slightly increased running times.

We populate data for `MLS` instances as in Guan et al. (2009), which solves instances with up to 128 scenarios to optimality using a tailored cutting-plane algorithm. In particular, d_n is uniformly distributed between 0 and 100, denoted $d_n \sim U[0, 100]$, $M_n \sim U[40T, 60T]$, $h_n \sim U[0, 10]$, $\alpha_n \sim U[0.8, 1.2](\alpha/h)\mathbb{E}[h_n]$, and $\beta_n \sim U[0.8, 1.2](\beta/h)\mathbb{E}[h_n]$, for $n \in \mathcal{N}$, where the production-to-holding-cost ratio $\alpha/h = 4$, and the setup-to-holding-cost ratio $\beta/h = 400$. We label the `MLS` instance with T stages and r branches as `MLS-r-T`.

We populate data for `MCAP` instances similar to the two-stage instances of Ahmed and Garcia (2003), which solves instances up to 3 resources, 4 tasks, and 500 scenarios to optimality using a decomposition based branch-and-bound algorithm. In particular, we set $\kappa_n^i = 1$, $f_n^i \sim U[25,50]$, $g_n^i \sim U[5,10]$, $s_n^j \sim U[500,1000]$, $c_n^{ij} \sim U[5,10]$, and $d_n^j \sim U[0.5,1.5]$ for all $i \in \mathcal{R}$, $j \in \mathcal{T}$, $n \in \mathcal{N}$. We label the `MCAP` instance with $|\mathcal{R}|$ resources, $|\mathcal{T}|$ tasks, T stages, and r branches as `MCAP-r-T-`$|\mathcal{R}|$-$|\mathcal{T}|$.

5.3. Parallel computing performance

One of the major bottlenecks in solving large-scale SMIPs is the difficulty involved in generating and solving the extensive form, which require a huge amount of computing power (i.e., memory and time). One way to overcome this difficulty is to use the modern parallel computing technology.

Parallel implementations of various solution algorithms for SPs have been reported in the literature. The focus in these implementations has been SPs *without* integer decision variables. Nielsen and Zenios (1993, 1996) formulate an asset allocation problem as an SP with network recourse, and solve the resulting model by a parallel algorithm that relies on this structure. Parallel version of the nested decomposition algorithm is applied to multi-stage linear programs (Birge et al. 1996), and to multi-stage convex programs with a special nonlinear structure (Birge and Rosa 1996). Ruszczyński (1993) presents a parallel version of the regularized decomposition method for multi-stage SPs. The communication overhead between the processors in these early implementations forms a major bottleneck in obtaining good parallel performance. Using an interior point implementation, Fragniére et al. (2000) solve a two-stage linear financial model with 1M(illion) scenarios, whose extensive form formulation has 2.6M variables and 1.1M constraints. However, the parallel efficiency they report decreases with the number of processors used – 65% for 10 processors. Gondzio and Kouwenberg (2001) solve a multi-stage version of a similar linear financial model with up to 4.8M scenarios, 24.9M continuous variables, and 12.5M constraints. Based on scenario tree decomposition, Blomvall (2003) describes a parallel primal interior point method for multi-stage convex nonlinear programs. He reports phenomenal parallel performance solving very large instances of a financial asset management problem with up to 8.4M scenarios, 50.3M continuous variables, and 92.3M linear constraints. Gondzio and Grothey (2006) solve a simpler version of a similar financial model with up to 16M scenarios, resulting in 1,011M variables and 353M constraints, using a structure-exploiting parallel primal-dual interior-point method. Linderoth and Wright (2003) describe parallel implementations of the multicut L-shaped algorithm and an ℓ_∞ trust regions algorithm for two-stage stochastic linear programs with fixed recourse. Unlike previous implementations, however, they use a geographically dispersed heterogeneous computational grid. They solve phenomenally large instances with up to 10^{81} scenarios by constructing sampled problems with 10^7 scenarios, whose extensive form contains

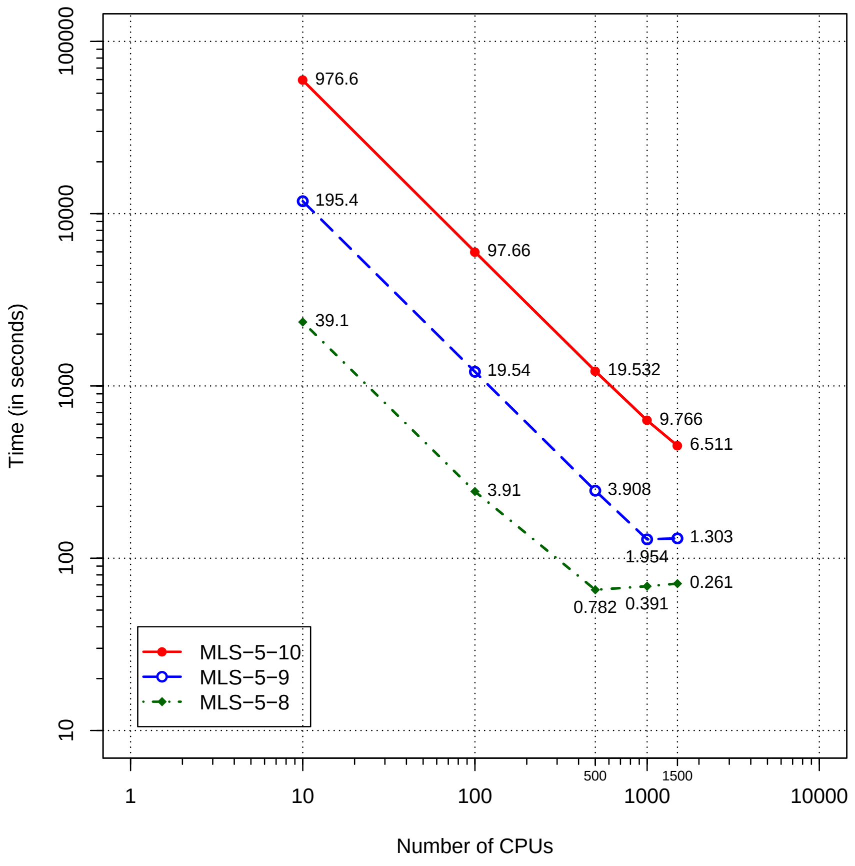


Figure 5 Parallel computing performance

1.26×10^{10} variables, with a parallel efficiency of 67%. For other instances solved, they report parallel efficiency in the range 21%-98%.

In general, the opportunity for parallelism grows, and therefore, parallel performance improves with the problem size. The theoretical limit for the best performance is a *linear* speedup, indicating an improvement in runtime by a factor of k when the number of processors used is multiplied by k. We expect improved parallel performance when (i) the number of subproblems per processor is large enough, and (ii) each subproblem takes a nontrivial amount of time. The implication of (i) is that the time a processor is busy solving a subproblem with a long runtime would be offset with subproblems that have shorter runtimes. We have observed that roughly 30 subproblems per processor achieves close to linear speedup. Condition (ii) ensures that parallelization overhead does not dominate the performance results.

The `log-log` plots in Figure 5 display parallel performance, averaged across 5 independent runs to account for load variability at `sandyb`, for three `MLS-5-T` instances, where $T = 8, 9, 10$. For each instance, we form group subproblems with $b = 200$ scenarios. The numbers around each marker in Figure 5 display the average number of subproblems solved by each processor.

We observe almost perfect speedup in Figure 5. For example, an increase in the number of processors from 10 to 100 is accompanied by about a 10-fold decrease in the runtimes for each instance. However, as the number of processors is further increased, we observe less than perfect speedup and sometimes an increase in the runtimes. For example, increasing the number of processors from 500 to 1,500 for `MLS-5-8` lengthens its overall runtime, which is explained by a severe violation of conditions (i) and (ii) as indicated by the numbers around markers. Also observe in Figure 5 that our approach scales nicely with problem size.

5.4. Impact of choosing a blockset $\mathcal{G}$

The number of partitions of an N-set is commonly known as the Bell number, which increases exponentially with N. When each subset of the partition is restricted to have exactly b elements, the number of partitions of an N-set, for N divisible by b, is given by

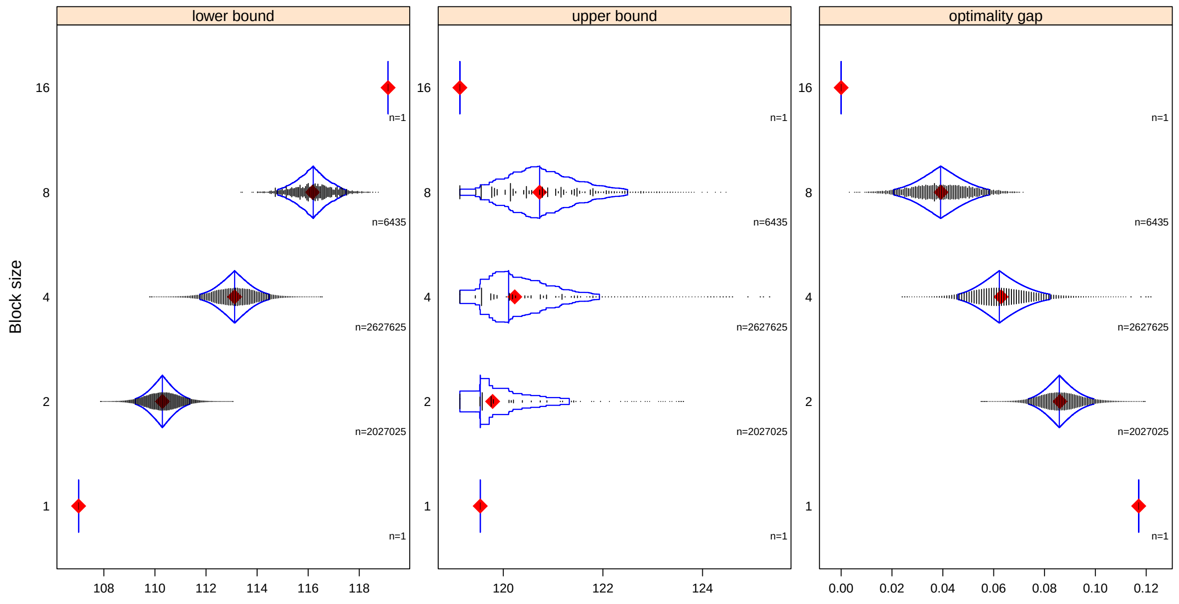
$$P(N,b) = \frac{1}{(\frac{N}{b})!}\binom{N}{b,b,\cdots,b} = \frac{N!}{(\frac{N}{b})!b!^{(\frac{N}{b})}},$$

which still presents a formidable number even for small values of N and b (e.g., $P(32,4) = 6\times 10^{19}$). Therefore, solving problem (8) for an optimal partition is a difficult combinatorial problem.

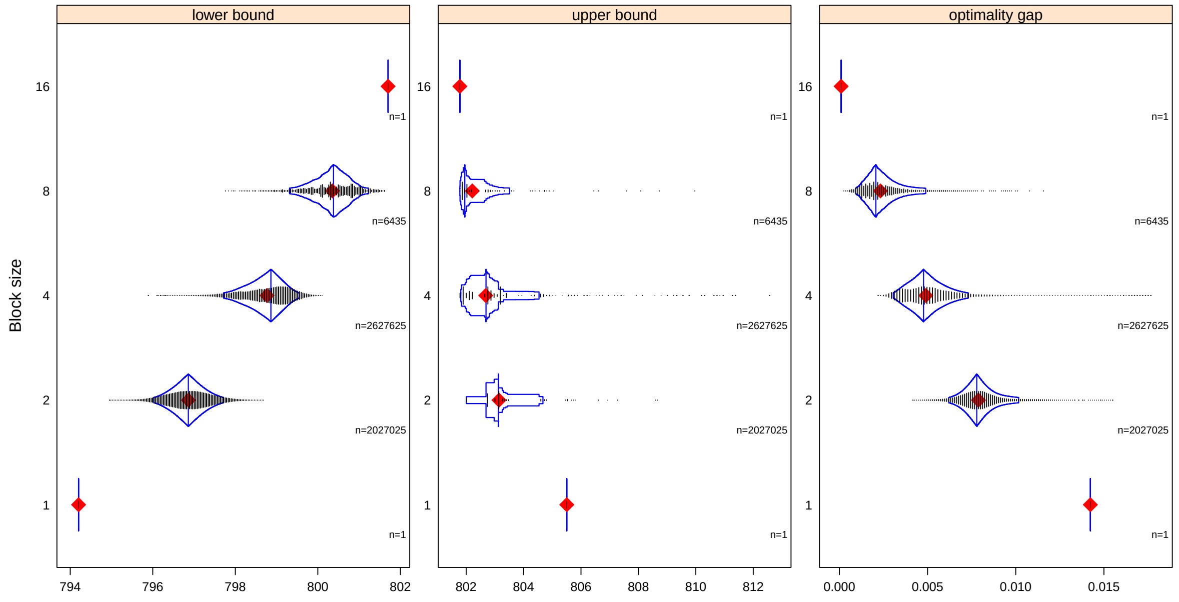
To evaluate the impact of various partitions, we consider instances with $N = 16$ scenarios. For each instance, we enumerate all possible partitions that could be formed with block size $b = 1,2,4,8,16$. Figure 6 presents the distributions of $\mathcal{L}(\mathcal{G})$, $\mathcal{U}(\mathcal{G})$, and the resulting optimality gap (measured as $(\mathcal{U}(\mathcal{G}) - \mathcal{L}(\mathcal{G}))/\mathcal{U}(\mathcal{G})$) across all possible partitions for each block size b. These results are presented as box-percentile plots. The vertical lines at the maximum height of each "box" indicates the median of the observed distribution of values. The mean of the distribution is plotted in diamond shape. The lower and upper endpoints of each box denote, respectively, the 5th and 95th percentiles of the observed distribution. In each panel of Figure 6, we also overlay onto each box-percentile plot the data intensity displayed as vertical line segments of varying height, which indicate the relative frequency of the observed value across all partitions.

Increasing the block size b leads to smaller optimality gaps, which can be attributed entirely to tighter lower bounds. As illustrated by the 'lower bound' panels in Figure 6, lower bounds for all instances strictly improve, on average, with block size. Upper bounds, on the other hand, do not follow any consistent pattern, except when $b = N$, in which case $\mathcal{U}(\mathcal{G}) = z^*$.

Note that *arbitrarily* forming a partition with increased block size does not guarantee any improvement in the lower bound. Such arbitrary partitions can indeed deteriorate the lower bound despite an increase in block size. An improved lower bound from an increased block size can only be guaranteed if each block of a partition is contained in one of the blocks of the new partition. Observe in the



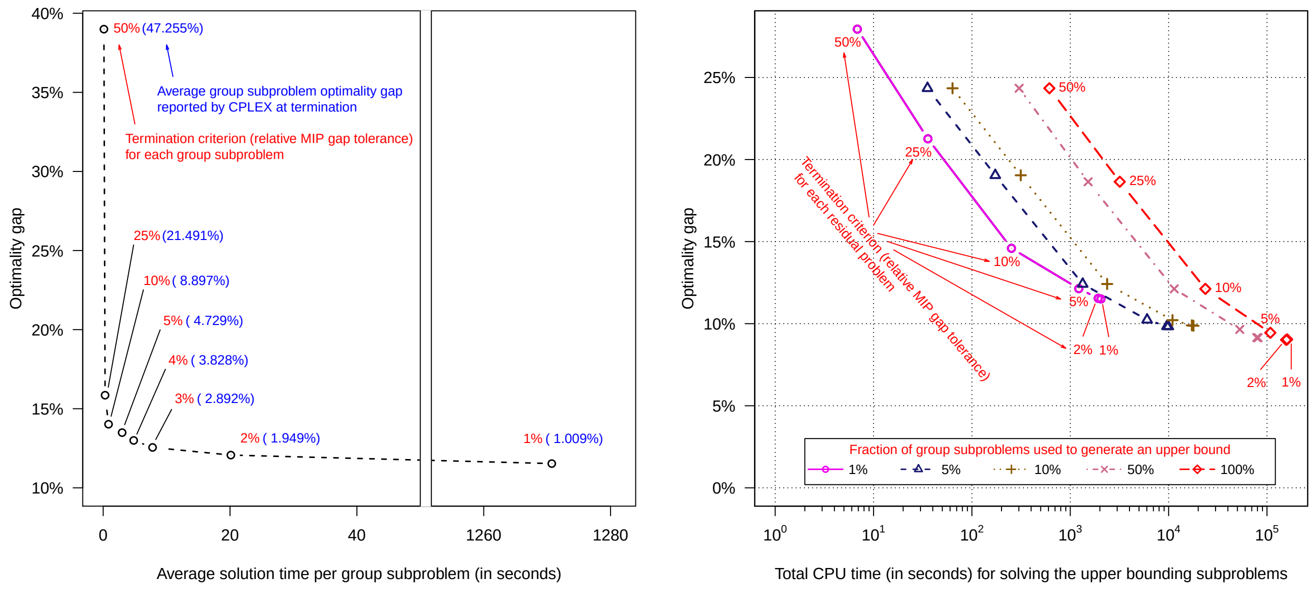
(a) Instance: `MLS-2-5`



(b) Instance: `MCAP-2-5-3-5`

Figure 6 Box-percentile plots for all partitions

'lower bound' panels in Figure 6 that, for each instance, there are significant overlaps between the box-percentile plots when $b = 2, 4, 8$, indicating a possible worsening of the lower bound despite an increase in b. The only exceptions to this phenomenon are when $b = 1$ and 16. Every block of the partition formed when $b = 1$ is necessarily contained by exactly one block of any partition formed with $b \geq 2$, and therefore, it is not surprising that the lower bound is the worst when $b = 1$. Conversely,



(a) Terminating lower bounding subproblems (b) Terminating upper bounding subproblems

Figure 7 Impact of suboptimally terminating subproblems on optimality gap

the only block of the partition formed when $b = 16$ necessarily contains any block of any partition formed with $b < 16$, and therefore, it is expected that the lower bound is the best when $b = 16$ (also see Proposition 2).

5.5. Practical considerations

As confirmed by the results in §5.4, using a larger block size b generally leads to a smaller optimality gap, but there are tradeoffs. Forming a partition $\mathcal{G}$ with a larger b requires solving a smaller number of relatively harder subproblems. In particular, while the number of subproblems to solve decreases linearly with b, the size of each subproblem increases linearly with b, but the solution time of an optimization problem typically increases exponentially with its size. An additional consideration concerning b is its impact on the memory requirements for solving the subproblems. In practice, some preliminary work is required for choosing an appropriate block size for a particular instance.

For any given partition $\mathcal{G}$, one need not solve $U(\Gamma)$ for all $\Gamma \in \mathcal{G}$. In practice, computing $U(\cdot)$ for a small subset of $\mathcal{G}$ may indeed be preferred as this strategy would tradeoff a little bit of upper bound quality with significant runtime savings. Moreover, one can suboptimally terminate a residual problem for any $\hat{x}(\Gamma)$ anytime after a feasible solution is found, and use the incumbent solution for $U(\Gamma)$. Similarly, in computing $\mathcal{L}(\mathcal{G})$, one can suboptimally terminate a group subproblem $GR(\Gamma)$, and use its best known lower bound for $z(\Gamma)$. Such early termination strategies can provide significant runtime savings without severely hurting the quality of the bounds, as illustrated in Figure 7.

The data in Figure 7 comes from ten `MLS-2-15` instances generated with different initial seeds. We construct a partition $\mathcal{G}$ with $b = 200$. In Figure 7(a), we investigate the impact of early termination for lower bounding problems. In particular, we compute the upper bounds for a fixed fraction (5%) of the blocks, and allow each upper bounding residual problem to run for a maximum of 5 minutes. We then re-solve group subproblem $GR(\Gamma)$, for $\Gamma \in \mathcal{G}$, by varying the termination threshold (relative MIP gap tolerance) from 50% to 1%, with one-hour time limit. In Figure 7(b), we investigate the impact of early termination for upper bounding problems. In particular, we solve each group subproblem $GR(\Gamma)$ with a relative MIP gap tolerance of 2%, and allow a maximum runtime of one hour. We then solve a varying fraction (1% to 100%) of blocks in $\mathcal{G}$ for upper bounds, with different termination thresholds (relative MIP gap tolerance) from 50% to 1%.

As predicted, reducing the relative MIP gap tolerance improves the quality of our bounds at the expense of longer running times. It is, however, important to note that the marginal improvement in quality of the bounds is decreasing, while the marginal change in running time is increasing at a much faster rate. For example, the quality of the bounds practically do not change when the termination threshold is reduced from 2% to 1% in Figure 7(a), however the average running time per group subproblem increases by a factor of 60. Also observe that the (almost) parallel lines in Figure 7(b) imply that using a higher fraction of blocks for upper bounding makes no practical difference in the quality of the upper bounds, but causes an exponential increase in runtime.

5.6. Experiments with large-scale instances

In this section, we test the performance of the proposed bounding scheme on large-scale `MLS` and `MCAP` instances. In our computations, we use 2GB RAM per core and 1,440 cores, one reserved for a master processor distributing jobs and the rest composed of worker processors. For a given partition $\mathcal{G}$, we solve group subproblems $GR(\Gamma)$, for $\Gamma \in \mathcal{G}$, in parallel, each with 2% relative MIP gap tolerance and 5-minute time limit.

We perform initial experiments with large instances for upper bounding as described in §4.2. We generate upper bounds $U(\Gamma)$ for a small fraction (1%) of blocks $\Gamma \in \mathcal{G}$, up to 1,000 blocks. For each block Γ, we use a single processor to solve the residual problem with 5% relative MIP gap tolerance and one-hour time limit. For the size of problems considered in this section, completing $\hat{x}(\Gamma)$ to a feasible solution for the full problem by solving the residual problem as an MIP takes unreasonably long runtimes. Therefore, we substitute this generic completion scheme with heuristics tailored for each problem.

Table 2 reports sizes, solution times, and optimality gaps for 7 `MLS` instances. We formulate group subproblems with $b = 200$ scenarios for each instance. For upper bounding, we employ the following heuristic: at a scenario tree node, production takes place only if the beginning inventory is not

Table 2 Problem sizes for `MLS` instances, solution times, and optimality gaps

Scenario tree size (r, T)	Extensive Form: Constraints	Extensive Form: Variables (% binary)	Wall time (seconds)	Gap (%)
$(100, 4)$	2,020,202	3,030,303 (33.3)	53	8.20
$(50, 5)$	12,755,102	19,132,653 (33.3)	233	9.41
$(10, 8)$	22,222,222	33,333,333 (33.3)	496	12.26
$(5, 11)$	24,414,062	36,621,093 (33.3)	600	13.14
$(3, 16)$	43,046,720	64,570,080 (33.3)	1,294	14.77
$(2, 25)$	67,108,862	100,663,293 (33.3)	1,735	15.88
$(7, 10)$	94,158,416	141,237,624 (33.3)	8,439	13.12

enough to satisfy current demand, where the amount of production equals that node's requirement (i.e. demand minus beginning inventory), plus the expected demand over a look-ahead period. After some experimentation, we set the look-ahead period to 2 stages.

Observe that the optimality gap monotonically increases with the number of time stages T, while this pattern is not strictly true with respect to the problem size (e.g., the optimality gap for `MLS-7-10` is smaller than that for `MLS-5-11`). The deterioration in optimality gap as T increases can be explained by a degradation in the performance of our heuristic, which makes myopic production decisions.

Table 3 reports sizes, solution times, and optimality gaps for 10 `MCAP` instances. We formulate group subproblems with $b = 50$ scenarios for each instance. For upper bounding, we employ the following heuristic. For a given capacity level at scenario tree node n, we first sort resources in increasing order of f_n^i, $i \in \mathcal{R}$. Then, for resource i, we sort all unassigned tasks $j \in \mathcal{T}$ in decreasing order of $s_n^j - c_n^{ij}$, and assign tasks from the beginning of this sorted list to resource i until no other task can be assigned. We use this assignment rule in making capacity expansion decisions at each node n. In particular, starting from the beginning of the sorted resource list, we temporarily expand the capacity of resource i by κ_n^i, and re-assign tasks. If this expansion is cost-saving, then the increase in the capacity of resource i is made permanent.

In the first four instances, the size of the scenario tree is constant, but the size of each node varies with the number of resources $|\mathcal{R}|$ and the number of tasks $|\mathcal{T}|$. Not surprisingly, the optimality gap increases with the size of the node, partly because the performance of the assignment rule degrades with the increased number of available combinations for larger values of $|\mathcal{R}|$ and $|\mathcal{T}|$. In the last seven instances, the node size is fixed, but the size of the scenario tree varies as in Table 2. Unlike the `MLS` instances, however, observe that the optimality gap is insensitive to the number of time stages.

Figure 8 illustrates the distribution of computational resources. The left panel of Figure 8 shows that the time for upper bounding heuristics (i.e., lightly shaded bars) increases sublinearly with problem size. The lower bounding group subproblems of `MLS` instances with $b = 200$ are relatively more demanding than those of `MCAP` instances with 2 resources, 3 tasks, and $b = 50$, although the

Table 3 Problem sizes for `MCAP` instances, solution times, and optimality gaps

Scenario tree	Resources	Tasks	Extensive Form		Wall time	Gap
size (r, T)	$(\|\mathcal{R}\|)$	$(\|\mathcal{T}\|)$	Constraints	Variables (% binary)	(seconds)	(%)
$(100, 4)$	5	20	30,303,030	131,313,130 (96.2)	4,024	10.92
$(100, 4)$	5	10	20,202,020	70,707,070 (92.9)	1,692	7.16
$(100, 4)$	3	5	11,111,111	26,262,626 (88.5)	591	3.96
$(100, 4)$	2	3	7,070,707	13,131,313 (84.6)	313	3.59
$(50, 5)$	2	3	44,642,857	82,908,163 (84.6)	1,918	3.33
$(10, 8)$	2	3	77,777,777	144,444,443 (84.6)	3,395	2.75
$(5, 11)$	2	3	85,449,217	158,691,403 (84.6)	3,785	2.73
$(3, 16)$	2	3	150,663,520	279,803,680 (84.6)	6,846	2.37
$(2, 25)$	2	3	234,881,017	436,207,603 (84.6)	11,056	2.79
$(7, 10)$	2	3	329,554,456	612,029,704 (84.6)	35,184	2.71

subproblem sizes are comparable. This fact manifests itself as the number of scenarios are increased by about 40 times from $(r, T) = (100, 4)$ to $(7, 10)$. In particular, while the lower bounding times remain insignificant for `MCAP` instances from `100-4-2-3` to `7-10-2-3`, they become proportionally longer for `MLS` instances from `100-4` to `7-10`. Observe, however, that as the number of resources and/or tasks for the `MCAP` instances increase the lower bounding group subproblems become much larger in size and, therefore, computationally more demanding (see the change in lower bounding times from `MCAP-100-4-2-3` to `MCAP-100-4-5-20`).

The right panel of Figure 8 displays boxplots for the number of jobs per processor for each instance. The number of jobs (i.e., group subproblems) and the time spent per job are reflected by the variation

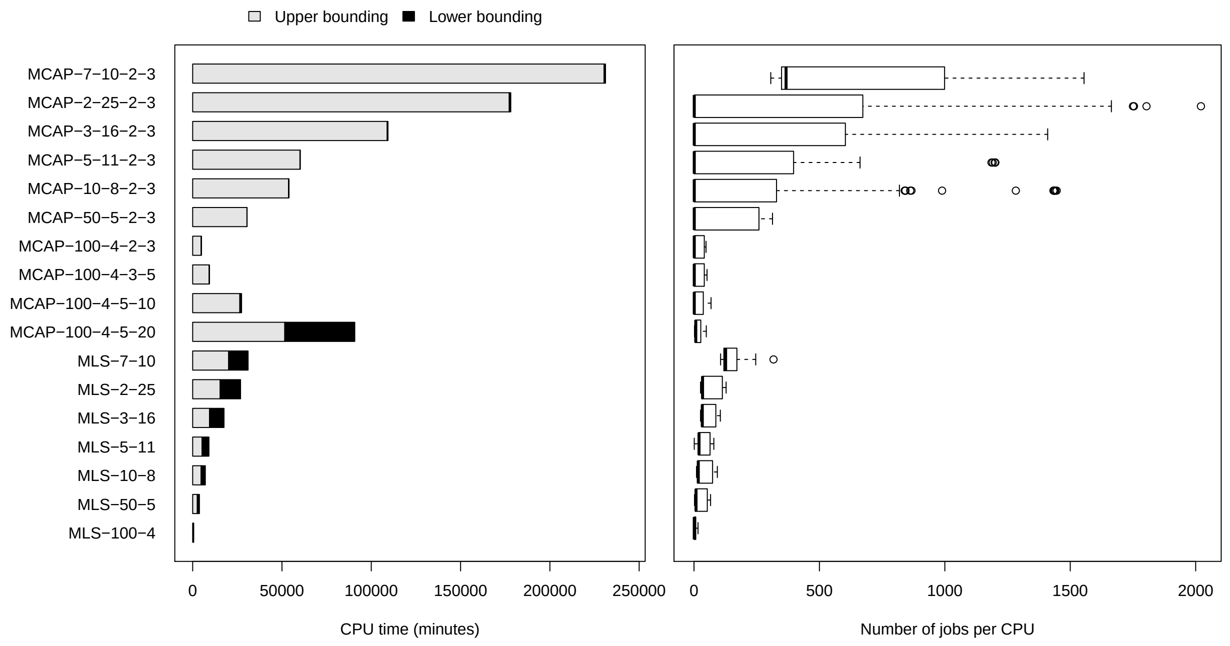


Figure 8 Illustrating the distribution of computational resources across tasks

in boxplots. The variation increases when there are relatively longer jobs. As an example, for `MCAP` instances from `50-5-2-3` to `2-25-2-3` more than half of the CPUs process a single long job composed of lower and upper bounding before other CPUs process many short jobs composed of only lower bounding, and therefore variation increases. However, for `MCAP-7-10-2-3` the number of short jobs is large enough such that a significant fraction of them are left to be processed when all long jobs are finished, and therefore, the median jumps up to around 500.

5.7. Performance comparison to CPLEX

This section compares the performance of the proposed bounding approach to CPLEX 12.5. In these comparisons, we consider `MLS` and `MCAP` instances that are significantly smaller in size than those solved in §5.6, so that their extensive form can be loaded to CPLEX. For example, the largest `MCAP` instance in §5.6 has about 1,500 times as many scenarios and about 1,000 times as many decision variables as the largest `MCAP` instance in this section.

In order to make fair comparisons, we execute our approach in serial mode on a single CPU, and subsequently execute CPLEX in single thread mode with a maximum time limit of 1 hour (otherwise at default settings) for the same instance on the same CPU. While executing our bounding approach, we employ the same parameter settings as those used in §5.6, except for setting the blocksize b to 50 and computing an upper bound for only randomly selected 10 blocks, because of significantly smaller instances.

For each instance, we record the total runtime and the gap obtained by our bounding approach. We then execute CPLEX on the same instance for a duration that is equal to the runtime of our bounding approach, and record the optimality gap reported by CPLEX. These results are summarized in Tables 4 and 5, respectively, for `MLS` and `MCAP` instances. The median gap obtained with our approach across all `MLS` and `MCAP` instances is 11.9% and 2.8%, respectively, whereas the median CPLEX gap is 85.1% and 80.4%, respectively.

The significantly smaller gaps returned by our bounding approach are also obtained fairly fast – median time is 170 seconds and 68 seconds, respectively, for `MLS` and `MCAP`. In order to investigate how long it would take CPLEX to achieve as tight of an optimality gap as our bounding approach, we re-run CPLEX for each instance with extra time up to a global time limit of 1 hour per instance. The results of these experiments are summarized in the last three columns of Tables 4 and 5.

Table 4 displays that CPLEX is able to obtain optimality gaps as tight as our approach only in 2 of the 12 `MLS` instances (namely, `MLS-3-10` and `MLS-2-15`), but it takes about 20 times longer runtime. On the other hand, for 3 instances (namely, `MLS-10-5`, `MLS-18-5`, and `MLS-50-4`), CPLEX terminates due to exhausting the available memory without being able to close the gap to the level obtained by our approach. In the remaining 7 instances, CPLEX exceeds the 1 hour time limit, yet

Table 4 Performance comparison to CPLEX for small `MLS` instances

	Extensive Form		Our approach		CPLEX Gap[1,‡]		CPLEX with extra time		
Instance	Constraints	Variables*	Time†	Gap‡	(Looseness[2])		Time†	(Delay[3])	Final gap‡
MLS-10-5	22,222	33,333	18	10.7	25.7	(2.4x)	366	(20.7x)	19.0¶
MLS-3-10	59,048	88,572	91	13.3	91.9	(6.9x)	2,194	(24.1x)	13.1
MLS-2-15	65,534	98,301	155	14.6	95.7	(6.5x)	3,056	(19.8x)	13.2
MLS-8-6	74,898	112,347	71	11.0	82.8	(7.5x)	>3,600	(50.4x)	18.9
MLS-6-7	111,974	167,961	121	11.9	86.5	(7.3x)	>3,600	(29.8x)	24.0
MLS-9-6	132,860	199,290	122	12.0	83.2	(7.0x)	>3,600	(29.6x)	28.7
MLS-3-11	177,146	265,719	269	13.8	92.4	(6.7x)	>3,600	(13.4x)	50.0
MLS-5-8	195,312	292,968	210	12.4	89.2	(7.2x)	>3,600	(17.2x)	86.6
MLS-10-6	222,222	333,333	186	11.3	83.8	(7.4x)	>3,600	(19.4x)	29.5
MLS-18-5	222,302	333,453	190	9.7	33.8	(3.5x)	3,264	(17.2x)	26.1¶
MLS-50-4	255,102	382,653	209	9.5	31.0	(3.3x)	541	(2.6x)	27.6¶
MLS-2-17	262,142	393,213	588	15.2	96.6	(6.4x)	>3,600	(6.1x)	45.6
		Median	170	11.9	85.1	(7.1x)	>3,600	(21.2x)	26.8

[1] The optimality gap reported by CPLEX when it is allowed to run as long as our approach.
[2] The degree of looseness as measured by the ratio of CPLEX gap to the gap obtained by our approach.
[3] The amount of extra time CPLEX took to reduce its gap to the level of the gap obtained by our approach.
* $1/3$ of the variables are binary for each `MLS` instance.
† All times are reported as seconds.
‡ All gaps are reported as percentages.
¶ CPLEX terminated due to exceeding memory limit before closing the gap.

Table 5 Performance comparison to CPLEX for small `MCAP` instances

	Extensive Form		Our approach		CPLEX Gap[1,‡]		CPLEX with extra time		
Instance	Constraints	Variables*	Time†	Gap‡	(Looseness[2])		Time†	(Delay[3])	Final gap‡
MCAP-5-5-2-3	5,467	10,153	2	2.8	0.7	(0.2x)	1	(0.4x)	0.0
MCAP-6-5-2-3	10,885	20,215	5	2.8	0.7	(0.2x)	3	(0.7x)	0.0
MCAP-5-6-2-3	27,342	50,778	12	2.9	36.3	(12.5x)	73	(6.2x)	0.9
MCAP-6-6-2-3	65,317	121,303	29	2.8	76.9	(27.8x)	414	(14.5x)	1.1
MCAP-6-6-3-5	102,641	242,606	68	3.1	81.5	(26.6x)	2,119	(30.9x)	0.6
MCAP-5-7-2-3	136,717	253,903	64	2.9	80.4	(28.1x)	1,344	(21.1x)	1.3
MCAP-4-8-2-3	152,915	283,985	74	2.8	83.1	(29.4x)	1,093	(14.8x)	1.1
MCAP-30-4-2-3	195,517	363,103	91	3.4	69.6	(20.6x)	>3,600	(39.7x)	7.9
MCAP-3-10-2-3	206,668	383,812	100	2.5	85.8	(34.0x)	860	(8.6x)	1.3
MCAP-2-15-2-3	229,369	425,971	120	2.5	90.4	(36.7x)	1,486	(12.4x)	1.8
MCAP-6-6-5-10	186,620	653,170	361	7.3	86.7	(11.9x)	>3,600	(10.0x)	86.3
		Median	68	2.8	80.4	(28.5x)	>1,093	(16.0x)	1.1

[1] The optimality gap reported by CPLEX when it is allowed to run as long as our approach.
[2] The degree of looseness as measured by the ratio of CPLEX gap to the gap obtained by our approach.
[3] The amount of extra time CPLEX took to reduce its gap to the level of the gap obtained by our approach.
* More than 85% of the variables are binary for each `MCAP` instance.
† All times are reported as seconds.
‡ All gaps are reported as percentages.

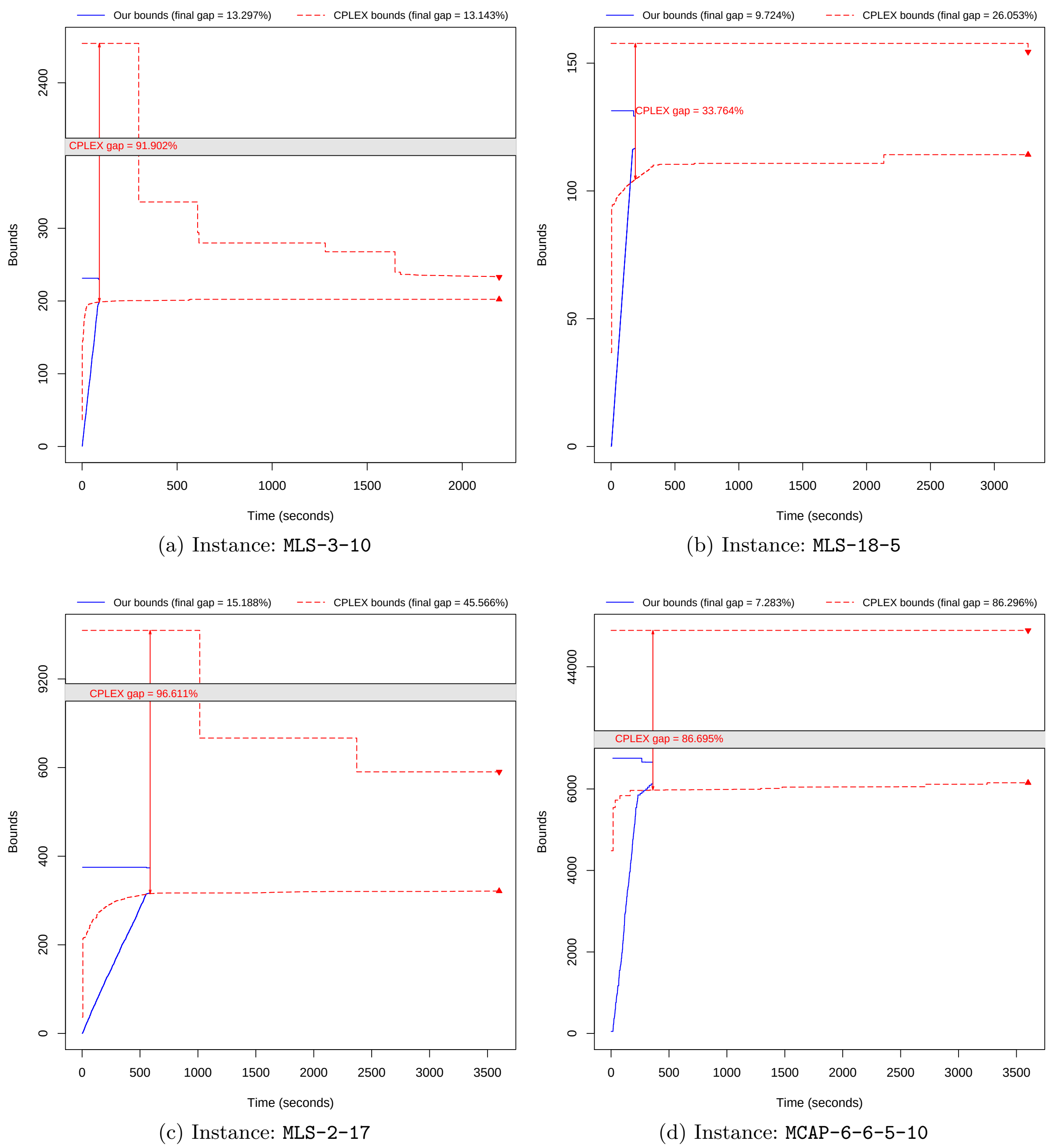
still leaving a large optimality gap – on average, 40% CPLEX gap versus the 12% gap (despite an average runtime of 225 seconds) from our approach.

Table 5 displays that, when extra time is allowed, CPLEX performs much better for the `MCAP` instances than for the `MLS` instances. In particular, it does not exhaust the available memory in any `MCAP` instance, and exceeds the 1 hour time limit only in 2 of the 11 instances (namely, `MCAP-30-4-2-3` and `MCAP-6-6-5-10`). For the 2 smallest instances (namely, `MCAP-5-5-2-3` and `MCAP-6-5-2-3`), which are indeed so small that they can be solved to optimality in less than 3 seconds, CPLEX performs better than our approach. For the remaining 7 instances, CPLEX ultimately obtains a smaller gap than our approach at the expense of, on average, about 16 times longer runtime – 1,056 seconds versus 67 seconds.

Figure 9 plots detailed progression of the lower and upper bounds, hence the optimality gap, for our bounding approach and for CPLEX on 4 instances from Tables 4 and 5. All panels in Figure 9 display that our approach, which exploits scenario information, produces much favorable upper bounds than those obtained by CPLEX, and that CPLEX spends most of its time enhancing the upper bounds, sometimes with no success – see Figures 9b and 9d. Figure 9b is an instance for which CPLEX starts with a relatively good upper bound, but spends over 50 minutes without making much improvement on this initial upper bound and terminates due to exhausting the available memory before reducing the optimality gap. Similarly, for the instance displayed in Figure 9d, CPLEX spends the entire hour without improving the initial upper bound and terminates with a large optimality gap – 86%. For the instances displayed in Figures 9a and 9c, CPLEX starts with a relatively poor upper bound, but makes several improvements along the way until it terminates either by reducing the gap to a level that is below the one found by our approach (Figure 9a) or by running out of time (Figure 9c).

In terms of lower bounds, however, we observe that the bounds from our approach are initially inferior to those obtained by CPLEX via the advanced cutting plane methods. However, our lower bounds increase almost linearly while CPLEX's lower bounds remains almost the same after a brief period, and therefore our lower bounds catch up with those of CPLEX in a short amount of time. The reason for almost linear increase in our lower bounds is mostly because of the identically-sized group subproblems, each of which has the same constraint structure with changes in their coefficients and possibly in right hand-side values.

All of these results suggest that even in serial mode the proposed bounding approach is competitive with CPLEX. Furthermore, the inherent parallelizability of the proposed approach suggests an immense potential for obtaining high quality solutions to practical large-scale SMIP instances within a reasonable time frame.



(a) Instance: `MLS-3-10` (b) Instance: `MLS-18-5`

(c) Instance: `MLS-2-17` (d) Instance: `MCAP-6-6-5-10`

Figure 9 Comparing the performance of our approach to CPLEX

6. Concluding remarks

Multistage SMIPs are known for their computational difficulty, but this paper gives evidence that there is hope to obtain high quality solutions to practical instances in a reasonable time. We develop a bounding framework for general multi-stage SMIPs based on scenario decomposition, extending our previous work on two-stage SMIPs (Sandıkçı et al. 2013). This framework does not make convexity assumptions, rather it builds directly on the power of parallelism and makes use of out of the box modern mathematical programming solvers. As a result, this approach stands out as a good candidate for use in practical implementations of SP to various problems. However, some of the major concerns with respect to this framework remain topics for further research. Of particular importance is using the bounds obtained from this framework in an exact solution algorithm. Furthermore, we have demonstrated the importance of choosing a blockset $\mathcal{G}$ on the quality of the bounds, but the investigation of choosing a good blockset $\mathcal{G}$ is open. Finally, applying this framework to nonlinear SMIPs is of future interest.

Acknowledgments

Sandıkçı acknowledges the financial support from the University of Chicago Booth School of Business. This work was completed in part with resources provided by the University of Chicago Research Computing Center (RCC). The authors are particularly thankful for the technical assistance of the RCC staff Robin Weiss. We also thank John R. Birge, Kipp Martin, and Miles Lubin for valuable discussions and helpful comments.

References

Ahmed, S. 2013. A scenario decomposition algorithm for 0âĂŞ1 stochastic programs. *Operations Research Letters* **41**(6) 565–569.

Ahmed, S., R. Garcia. 2003. Dynamic capacity acquisition and assignment under uncertainty. *Annals of Operations Research* **124**(1–4) 267–283.

Ahmed, S., N. V. Sahinidis. 2003. An approximation scheme for stochastic integer programs arising in capacity expansion. *Operations Research* **51**(3) 461–471.

Birge, J. R. 1982. The value of the stochastic solution in stochastic linear programs with fixed recourse. *Mathematical Programming* **24**(1) 314–325.

Birge, J. R. 1985. Aggregation bounds in stochastic linear programming. *Mathematical Programming* **31**(1) 25–41.

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

Birge, J. R., F. V. Louveaux. 2011. *Introduction to Stochastic Programming*. 2nd ed. Springer, New York.

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

Birge, J. R., R. J. B. Wets. 1987. Computing bounds for stochastic programming problems by means of a generalized moment problem. *Mathematics of Operations Research* **12**(1) 149–162.

Birge, J. R., R. J. B. Wets. 1989. Sublinear upper bounds for stochastic programs with recourse. *Mathematical Programming* **43**(1–3) 131–149.

Blomvall, J. 2003. A multistage stochastic programming algorithm suitable for parallel computing. *Parallel Computing* **29**(4) 431–445.

Carøe, C. C., R. Schultz. 1999. Dual decomposition in stochastic integer programming. *Operations Research Letters* **24**(1–2) 37–45.

Dokov, S. P., D. P. Morton. 2005. Second-order lower bounds on the expectation of a convex function. *Mathematics of Operations Research* **30**(3) 662–677.

Edirisinghe, N. C. P. 1996. New second-order bounds on the expectation of saddle functions with applications to stochastic linear programming. *Operations Research* **44**(6) 909–922.

Edirisinghe, N. C. P., W. T. Ziemba. 1994. Bounding the expectation of a saddle function with application to stochastic programming. *Mathematics of Operations Research* **19**(2) 314–340.

Edmundson, H. P. 1956. Bounds on the expectation of a convex function of a random variable. Tech. rep., RAND Corporation, Santa Monica, CA.

Erdogan, S. A., B. T. Denton. 2013. Dynamic appointment scheduling with uncertain demand. *INFORMS Journal on Computing* **25**(1) 116–132.

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

Frauendorfer, K. 1988. Solving SLP recourse problems with binary multivariate distributions – The dependent case. *Mathematics of Operations Research* **13**(3) 377–394.

Frauendorfer, K. 1996. Barycentric scenario trees in convex multistage stochastic programming. *Mathematical Programming* **75**(2) 277–293.

Gaivoronski, A. A. 2005. Stochastic optimization problems in telecommunications. S. W. Wallece, W. T. Ziemba, eds., *Applications of Stochastic Programming*. SIAM, Philadelphia, PA, 669–701.

Gondzio, J., A. Grothey. 2006. Solving nonlinear financial planning problems with 10^9 decision variables on massively parallel architectures. M. Constantino, C.A. Brebbia, eds., *Computational Finance and its Applications II*. WIT Press, Southampton, UK, 95–108.

Gondzio, J., R. Kouwenberg. 2001. High-performance computing for asset-liability management. *Operations Research* **49**(6) 879–891.

Guan, Y., S. Ahmed, G. L. Nemhauser. 2009. Cutting planes for multistage stochastic integer programs. *Operations Research* **57**(2) 287–298.

Guan, Y., S. Ahmed, G. L. Nemhauser, A. J. Miller. 2006. A branch-and-cut algorithm for the stochastic uncapacitated lot-sizing problem. *Mathematical Programming* **105**(1) 55–84.

Higle, J. L., S. Sen. 1991. Stochastic decomposition: An algorithm for two-stage linear programs with recourse. *Mathematics of Operations Research* **16**(3) 650–669.

Infanger, G. 2006. Dynamic asset allocation strategies using a stochastic dynamic programming approach. S. A. Zenios, W. T. Ziemba, eds., *Handbook of Asset and Liability Management*. Springer, Berlin, 200–251.

Jensen, J. L. 1906. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta Mathematica* **30**(1) 175–193.

Kall, P., J. Mayer. 2011. *Stochastic Linear Programming: Models, Theory, and Computation*. Springer, New York, NY.

Kuhn, D. 2004. *Generalized Bounds for Convex Multistage Stochastic Programs*. Springer, Berlin Heidelberg, Germany.

Kuhn, D. 2008. Aggregation and discretization in multistage stochastic programming. *Mathematical Programming* **113**(1) 61–94.

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

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

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

Madansky, A. 1960. Inequalities for stochastic linear programming problems. *Management Science* **6**(2) 197–204.

Mak, W. K., D. P. Morton, R. K. Wood. 1999. Monte Carlo bounding techniques for determining solution quality in stochastic programs. *Operations Research Letters* **24**(1–2) 47–56.

Mulvey, J. M., B. Shetty. 2004. Financial planning via multi-stage stochastic optimization. *Computers and Operations Research* **31**(1) 1–20.

Nielsen, S. S., S. A. Zenios. 1993. A massively parallel algorithm for nonlinear stochastic network problems. *Operations Research* **41**(2) 319–337.

Nielsen, S. S., S. A. Zenios. 1996. Solving multistage stochastic network programs on massively parallel computers. *Mathematical Programming* **73**(3) 227–250.

Norkin, V. I., G. C. Pflug, A. Ruszczyński. 1998. A branch and bound method for stochastic global optimization. *Mathematical Programming* **83**(1–3) 425–450.

Özaltın, O. Y., O. A. Prokopyev, A. J. Schaefer, M. S. Roberts. 2011. Optimizing the societal benefits of the annual influenza vaccine: A stochastic programming approach. *Operations Research* **59**(5) 1131–1143.

Powell, W. B., H. Topaloglu. 2003. Stochastic programming in transportation and logistics. A. Ruszczyński, A. Shapiro, eds., *Handbooks in Operations Research & Management Sciences*, vol. 10 (Stochastic Programming). Elsevier, Amsterdam, The Netherlands, 555–635.

Römisch, W., R. Schultz. 2001. Multistage stochastic integer programs: An introduction. M. Grötschel, S. O. Krumke, J. Rambau, eds., *Online Optimization of Large Scale Systems*. Springer, Berlin, 581–600.

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

Sandıkçı, B., N. Kong, A. J. Schaefer. 2013. A hierarchy of bounds for stochastic mixed-integer programs. *Mathematical Programming* **138**(1) 253–272.

Sen, S. 2005. Algorithms for stochastic mixed-integer programming models. G. L. Nemhauser K. Aardal, R. Weismantel, eds., *Handbooks in Operations Research & Management Sciences*, vol. 12 (Discrete Optimization). Elsevier, Amsterdam, The Netherlands, 515–558.

Sen, S., Z. Zhou. 2014. Multistage stochastic decomposition: A bridge between stochastic programming and approximate dynamic programming. *SIAM Journal on Optimization* **24**(1) 127–153.

Wallace, S. W., S. E. Fleten. 2003. Stochastic programming models in energy. A. Ruszczyński, A. Shapiro, eds., *Handbooks in Operations Research & Management Sciences*, vol. 10 (Stochastic Programming). Elsevier, Amsterdam, The Netherlands, 637–677.

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

Wright, S. E. 1994. Primal-dual aggregation and disaggregation for stochastic linear programs. *Mathematics of Operations Research* **19**(4) 893–908.