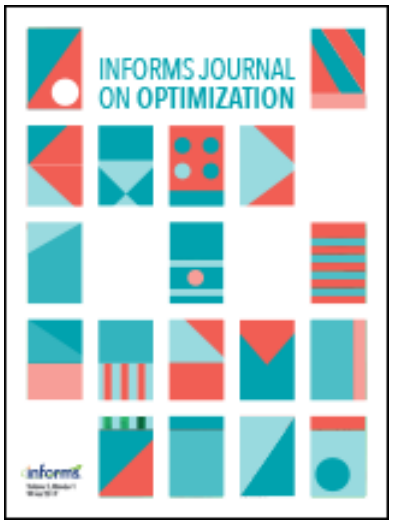


INFORMS Journal on Optimization

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Parallelizing Subgradient Methods for the Lagrangian Dual in Stochastic Mixed-Integer Programming

Cong Han Lim, Jeffrey T. Linderoth, James R. Luedtke, Stephen J. Wright

To cite this article:
Cong Han Lim, Jeffrey T. Linderoth, James R. Luedtke, Stephen J. Wright (2021) Parallelizing Subgradient Methods for the Lagrangian Dual in Stochastic Mixed-Integer Programming. INFORMS Journal on Optimization 3(1):1-22. https://doi.org/10.1287/ijoo.2019.0029

http://pubsonline.informs.org/journal/ijoo

INFORMS JOURNAL ON OPTIMIZATION
Vol. 3, No. 1, Winter 2021, pp. 1–22
ISSN 2575-1484 (print), ISSN 2575-1492 (online)

Parallelizing Subgradient Methods for the Lagrangian Dual in Stochastic Mixed-Integer Programming

Cong Han Lim,[a] Jeffrey T. Linderoth,[b] James R. Luedtke,[b] Stephen J. Wright[c]

[a] Wisconsin Institute for Discovery, University of Wisconsin–Madison, Madison, Wisconsin 53706; [b] Department of Industrial and Systems Engineering, University of Wisconsin–Madison, Madison, Wisconsin 53706; [c] Department of Computer Sciences, University of Wisconsin–Madison, Madison, Wisconsin 53706

Contact: clim9@wisc.edu, https://orcid.org/0000-0002-2033-4927 (CHL); linderoth@wisc.edu, https://orcid.org/0000-0003-4442-3059 (JTL); jim.luedtke@wisc.edu, https://orcid.org/0000-0001-9265-7728 (JRL); swright@cs.wisc.edu, https://orcid.org/0000-0001-6815-7379 (SJW)

Received: October 14, 2018
Revised: June 7, 2019; November 27, 2019
Accepted: January 10, 2020
Published Online in Articles in Advance: January 26, 2021

https://doi.org/10.1287/ijoo.2019.0029

Abstract. The dual decomposition of stochastic mixed-integer programs can be solved by the projected subgradient algorithm. We show how to make this algorithm more amenable to parallelization in a master-worker model by describing two approaches, which can be combined in a natural way. The first approach partitions the scenarios into batches and makes separate use of subgradient information for each batch. The second approach drops the requirement that evaluation of function and subgradient information is synchronized across the scenarios. We provide convergence analysis of both methods. We also evaluate their performance on two families of problems from SIPLIB on a single server with 32 single-core worker processes, demonstrating that when the number of workers is high relative to the number of scenarios, these two approaches (and their synthesis) can significantly reduce running time.

Funding: Financial support from the National Science Foundation Division of Civil, Mechanical and Manufacturing Innovation [Grant 1634597] is gratefully acknowledged.

Keywords: stochastic mixed-integer programming • subgradient methods • parallel optimization

1. Introduction

We study subgradient approaches for solving the Lagrangian dual of stochastic mixed-integer programs (SMIPs) that are amenable to parallel implementation. SMIPs can be used to model multistage problems with uncertainty, where information is revealed stage-wise and the decisions available in each stage depend on those made in prior stages. A two-stage SMIP can be formulated as:

$$\phi^{\mathrm{SMIP}} := \min_{\mathbf{x},\mathbf{y}_1,\mathbf{y}_2,\ldots,\mathbf{y}_N} \left\{ \mathbf{c}^\top \mathbf{x} + \frac{1}{N} \sum_{s\in S} \mathbf{q}_s^\top \mathbf{y}_s \;:\; (\mathbf{x},\mathbf{y}_s) \in K_s,\ s \in S \right\}, \tag{1a}$$

$$\text{where } K_s := \left\{ (\mathbf{x},\mathbf{y}_s) \;:\; W_s \mathbf{y}_s = \boldsymbol{h}_s - T_s \mathbf{x},\ \mathbf{x} \in X,\ \mathbf{y}_s \in Y \right\}, \tag{1b}$$

where $\mathbf{c} \in \mathbb{R}^n$, X is a closed mixed-integer set defined by linear inequalities and integrality restrictions on some of the variables, Y is a closed mixed-integer set, and the random outcomes are represented with a finite set of equally likely scenarios $(\mathbf{q}_s, \boldsymbol{h}_s, T_s, W_s)$, for $s \in S := \{1,\ldots,N\}$. We assume for simplicity that all scenarios are equally likely, but all our techniques can be generalized easily to the case of nonuniform probabilities. In this setup, the first-stage decision $\mathbf{x}$ must be fixed before knowing the scenario s, and there is a separate set of decisions $\mathbf{y}_s$ for each scenario $s \in S$, indicating that these decisions can be made after observing the scenario. The approaches we study in this paper can be extended beyond the two-stage setting considered here, to multiple stages, but we limit our exposition to two-stage problems for clarity.

SMIPs can be solved directly via the formulation (1), which is a large mixed-integer program whose size scales linearly with the number of scenarios. The size of this formulation makes it computationally intractable for many interesting cases, so we focus instead on methods that work with decompositions of the problem. Many methods, ranging from the classic Benders decomposition to branch-and-bound techniques, make certain assumptions on the structure of the underlying SMIP. We focus on *dual decomposition* (Carøe 1998, Carøe and Schultz 1999), which applies to all multistage SMIPs.

In dual decomposition, N copies of the first-stage decision variables are introduced, along with a "master copy" $\mathbf{z}$, yielding the following equivalent expression for ϕ^{SMIP}:

$$\min_{\mathbf{x}_1,\ldots,\mathbf{x}_N;\mathbf{y}_1,\ldots,\mathbf{y}_N;\mathbf{z}} \left\{ \frac{1}{N} \sum_{s\in S} \left(\mathbf{c}^\top \mathbf{x}_s + \mathbf{q}_s^\top \mathbf{y}_s\right) \,:\, \left(\mathbf{x}_s, \mathbf{y}_s\right) \in K_s;\ \mathbf{x}_s = \mathbf{z},\ s \in S \right\}. \tag{2}$$

The constraints $\mathbf{x}_s = \mathbf{z}$, $s \in S$ enforce *nonanticipativity*—that is, the same first-stage decisions must be made for all second-stage scenarios. Introducing multiplier vectors $\boldsymbol{\lambda} := [\boldsymbol{\lambda}_s]_{s\in S}$ for the nonanticipativity constraints, we obtain the Lagrangian dual function for (2):

$$\min_{\mathbf{x}_1,\ldots,\mathbf{x}_N;\mathbf{y}_1,\ldots,\mathbf{y}_N;\mathbf{z}} \left\{ \frac{1}{N} \sum_{s\in S} \left(\mathbf{c}^\top \mathbf{x}_s + \mathbf{q}_s^\top \mathbf{y}_s + \boldsymbol{\lambda}_s^\top (\mathbf{x}_s - \mathbf{z})\right) \,:\, \left(\mathbf{x}_s, \mathbf{y}_s\right) \in K_s,\ s \in S \right\}. \tag{3}$$

The Lagrangian dual problem for (2) is to find the $\boldsymbol{\lambda}$ that maximizes (3). Note that because $\mathbf{z}$ is unrestricted, this function takes a value greater than $-\infty$ only when $\sum_{s\in S} \boldsymbol{\lambda}_s = 0$. When this requirement holds, the $\mathbf{z}$ variables vanish from (3), and the minimization in (3) becomes separable over the different scenarios. We can write

$$\mathcal{L}(\boldsymbol{\lambda}) := \sum_{s\in S} \mathcal{L}_s(\boldsymbol{\lambda}_s), \tag{4}$$

where

$$\mathcal{L}_s(\boldsymbol{\lambda}_s) := \min_{\mathbf{x},\mathbf{y}} \left\{ \mathbf{c}^\top \mathbf{x} + \mathbf{q}_s^\top \mathbf{y} + \boldsymbol{\lambda}_s^\top \mathbf{x} \,:\, \left(\mathbf{x}, \mathbf{y}\right) \in K_s \right\} \text{ for all } s \in S. \tag{5}$$

(Note that $\mathcal{L}$ and $\mathcal{L}_s$, $s \in S$ are all concave functions based on their definitions.) The Lagrangian dual problem is

$$\phi^{\text{LD}} := \max_{\boldsymbol{\lambda}\in C} \frac{1}{N} \mathcal{L}(\boldsymbol{\lambda}), \quad \text{where } C := \left\{ \boldsymbol{\lambda} \,:\, \sum_{s\in S} \boldsymbol{\lambda}_s = 0 \right\}. \tag{6}$$

For any feasible choice of multipliers $\boldsymbol{\lambda}$, we have $(1/N)\mathcal{L}(\boldsymbol{\lambda}) \le \phi^{\text{SMIP}}$ because $\sum_{s\in S} \boldsymbol{\lambda}_s^\top \mathbf{z} = 0$ for any vector $\mathbf{z}$. Thus, the optimal value ϕ^{LD} of the Lagrangian dual provides a lower bound on ϕ^{SMIP}.

The lower bound ϕ^{LD} is at least as good as the linear programming (LP)-relaxation bound of the extensive form (1) (see, for example, Conforti et al. 2014). Empirical (for example, Rahmanai et al. 2018 and Bodur et al. 2016) and theoretical (Dey et al. 2018) evidence indicates that it is often much better. Thus, the ability to compute high-quality Lagrangian bounds efficiently is useful for exact solution approaches—for example, those that use a branch-and-bound framework (Carøe and Schultz 1999, Lubin et al. 2013). Additionally, the solutions to the scenario subproblems (5) can provide useful information for finding high-quality primal feasible solutions. Another advantage of dual decomposition is that it extends readily to multistage stochastic programs. The only change is that the nonanticipativity constraints must be redefined to reflect the structure of the scenario tree representing the evolution of the uncertain parameters.

Our focus in this paper is to effectively make use of parallel computing to solve the dual problem (6) efficiently. This problem is nonsmooth with many (possibly very many) variables. On the other hand, it is a concave maximization problem, and its objective is separable; the only coupling between variables $\boldsymbol{\lambda}_s$, $s \in S$ is via the constraint requiring these vectors to sum to zero. Any constrained nonsmooth convex optimization method can be applied to solve the dual problem (6), but we focus here on the traditional subgradient method because it is amenable to analysis and easy to implement.

Subgradients of the Lagrangian dual can be obtained by solving N scenario-wise subproblems. Traditional subgradient methods (Shor 1985, Bertsekas 1999, Ruszczyński 2006) require the full subgradient to be evaluated (across all scenarios) before any progress can be made, thus requiring all N scenario-wise subproblems to be solved at each iteration. Scenarios requiring larger computation time can cause delays in the execution, as well as inefficient usage of parallel computing resources. We describe two variants of the traditional subgradient method (Shor 1985, Bertsekas 1999, Ruszczyński 2006) that alleviate this issue. The first variant uses stochastic subgradients of $\mathcal{L}(\boldsymbol{\lambda})$ constructed from batches of scenarios. Because the batches arise from a partition of the full scenario set S, we term this method the *partitioned subgradient method*. The second variant, an *asynchronous subgradient method*, does not necessarily wait for all scenarios to complete before taking a step, using the current returned solution for completed scenarios and the most recent solution for the

remaining scenarios to construct a "noisy" subgradient. The two variants can be combined into an *asynchronous partitioned subgradient* algorithm. All these approaches maintain feasibility of iterates λ throughout, allowing us to continually update a lower bound to monitor the algorithm's progress.

We focus on improving parallel variants of the projected subgradient algorithm, which performs a step along the gradient direction followed by a projection onto the feasible set, for several reasons. First, although more advanced subgradient methods tend to converge faster in experiments on standard SMIP instances from SIPLIB, they are based on construction of a master problem that approximates the expected recourse function more closely as the algorithm advances. The per-iteration cost of the algorithm can increase significantly with the complexity of the master problem, an operation that can become a bottleneck in a large-scale implementation. Second, parallel and more generally distributed versions of the subgradient methods have been studied in several other contexts; it is interesting to consider it in the context of dual decomposition as well. Third, many variants of the subgradient method (especially the stochastic gradient method) have been developed in recent years, and our techniques may be applicable to these variants, too.

In the remainder of this introduction, we review related work on SMIPs and distributed optimization. Section 2 recaps the application of the standard projected subgradient method to solve (6) and highlights the possible disadvantage of requiring full subgradients at every iteration. Sections 3 and 4 develop our partitioned and asynchronous variants (respectively) and give convergence results. Computational results on canonical instances from the SIPLIB library are shown in Section 5, and Section 6 gives some concluding remarks. The appendix contains proofs for some results in Sections 3 and 4, together with a discussion on how partitioning can be applied to other subgradient methods.

1.1. Related Work

In the context of SMIP, alternatives to the subgradient algorithm for solving the Lagrangian problem include coordinate descent (Aravena and Papavasiliou 2015), column generation (Lulli and Sen 2004), cutting plane (Lubin et al. 2013, Kim and Zavala 2017), and bundle methods (Lubin et al. 2013, Kim and Zavala 2017, Kim et al. 2017). The approach of Aravena and Papavasiliou (2015) is the most similar in flavor to our work. They work with a slightly different Lagrangian relaxation that avoids elimination of the term involving z, instead defining a smoothed subproblem that incorporates z. They then apply an asynchronous block-coordinate descent approach in which each block corresponds to a single scenario. The cutting-plane approach (which is an application of the classic Kelley's method (Kelley 1960) to the present setting) uses subgradients to construct cutting planes for the objective function. The model of the objective function constructed in this way can be formulated as a linear program and solved exactly at each iteration. Bundle methods are a refinement of this process that solve regularized or restricted problems over the cutting-plane objective. Such methods avoid the oscillating iterates often observed in the cutting-planes method and tend to converge faster in practice. Lubin et al. (2013) apply the proximal bundle method; Kim and Zavala (2017) use an interior-point method to generate the next iterate; and Kim et al. (2017) use an asynchronous trust-region method.

Our partitioned subgradient approach repartitions the scenario set uniformly at random at each iteration of the algorithm for the purposes of improving parallelization. The concept of partitioning has been used in the context of SMIPs in a completely different manner. Instead of completely decomposing the problem so that each scenario can be treated as a separate subproblem, scenarios can be grouped together to form larger subproblems (Boland et al. 2016, Maggioni and Pflug 2016, Maggioni et al. 2016, Ryan et al. 2016, Sandikçi and Ozaltin 2017). This approach can potentially yield better relaxations at the cost of having more expensive subproblems.

As we were finishing this work, we found the paper of Necoara et al. (2017), which performs batch updates similar to those in our partitioned approach. There are several key differences with our work. First, they adopt a randomized block-coordinate descent perspective rather than our stochastic gradient viewpoint. Second, they consider a more general distributed optimization model based on a graph that determines how information can be shared, whereas we assume no such structure. In effect, we work with a completely connected graph. Third, whereas their batches are sampled independently at every step, we generate many batches at a time (with nonoverlapping scenarios) by partitioning the entire group of scenarios. Finally, our convergence analysis focuses on the case of nonsmooth objective function, whereas they study the smooth case with and without strong convexity.

2. Subgradient Method for the Lagrangian Dual

This section describes the basic projected subgradient method applied to the problem and its convergence properties. Section 2.1 demonstrates how subgradients can be computed, and Section 2.2 presents the projected subgradient method.

2.1. Subgradient Computation

The following statement characterizes the subdifferential $\partial\mathcal{L}_s(\boldsymbol{\lambda}_s)$: If for some $\boldsymbol{\lambda}_s$ in the domain of $\mathcal{L}_s$, $(\mathbf{x}_s, \mathbf{y}_s)$ is a vector pair that achieves the minimum in (5), then $\mathbf{x}_s \in \partial\mathcal{L}_s(\boldsymbol{\lambda}_s)$, that is,

$$\mathcal{L}_s(\boldsymbol{\mu}_s) \leq \mathcal{L}_s(\boldsymbol{\lambda}_s) + \langle \mathbf{x}_s, \boldsymbol{\mu}_s - \boldsymbol{\lambda}_s \rangle, \tag{7}$$

for all $\boldsymbol{\mu}_s$ in the domain of $\mathcal{L}_s$.

By making use of the indicator function $\delta_0 : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$:

$$\delta_0(\mathbf{x}) := \begin{cases} 0, & \text{if } \mathbf{x} = 0, \\ \infty, & \text{otherwise,} \end{cases}$$

we restate (6) as follows:

$$\phi^{LD} := \frac{1}{N} \max_{\boldsymbol{\lambda} \in \mathbb{R}^{n \times N}} \mathcal{L}_0(\boldsymbol{\lambda}), \quad \text{where } \mathcal{L}_0(\boldsymbol{\lambda}) := \mathcal{L}(\boldsymbol{\lambda}) - \delta_0\left(\sum_{s \in S} \boldsymbol{\lambda}_s\right). \tag{8}$$

The domain of $\mathcal{L}_0$ (containing those values of $\boldsymbol{\lambda}$ for which $\mathcal{L}_0(\boldsymbol{\lambda}) > -\infty$) is a subset of C defined in (6). Note that $\mathcal{L}_0$ is a concave function with a closed convex domain, because each $\mathcal{L}_s$ is concave with closed convex domain, and the set C is a hyperplane.

Lemma 1. *Fix a vector $\boldsymbol{\lambda} = [\boldsymbol{\lambda}_s]_{s \in S}$ in the domain of $\mathcal{L}_0$. For all $s \in S$, let $(\mathbf{x}_s, \mathbf{y}_s)$ be a solution of (5) at these values of $\boldsymbol{\lambda}_s$. Then, $[\mathbf{x}_s - z]_{s \in S}$ is a subgradient of $\mathcal{L}_0$ at $\boldsymbol{\lambda}$ for all $z \in \mathbb{R}^n$.*

Proof. We need to show that

$$\mathcal{L}_0(\boldsymbol{\mu}) \leq \mathcal{L}_0(\boldsymbol{\lambda}) + \sum_{s \in S} \langle \mathbf{x}_s - z, \boldsymbol{\mu}_s - \boldsymbol{\lambda}_s \rangle, \quad \text{for any } \boldsymbol{\mu} = [\boldsymbol{\mu}_s]_{s \in S}.$$

In the case in which $\sum_{s \in S} \boldsymbol{\mu}_s \neq 0$, we have $\mathcal{L}_0(\boldsymbol{\mu}) = -\infty$, so the required inequality is satisfied trivially. In the other case, we have $\sum_{s \in S} \boldsymbol{\mu}_s = \sum_{s \in S} \boldsymbol{\lambda}_s = 0$, so using the fact that $\mathbf{x}_s$ is a subgradient of $\mathcal{L}_s$ at $\boldsymbol{\lambda}_s$, we have that

$$\mathcal{L}_0(\boldsymbol{\mu}) = \mathcal{L}(\boldsymbol{\mu}) \leq \mathcal{L}(\boldsymbol{\lambda}) + \sum_{s \in S} \langle \mathbf{x}_s, \boldsymbol{\mu}_s - \boldsymbol{\lambda}_s \rangle = \mathcal{L}_0(\boldsymbol{\lambda}) + \sum_{s \in S} \langle \mathbf{x}_s - z, \boldsymbol{\mu}_s - \boldsymbol{\lambda}_s \rangle,$$

as required. □

Lemma 1 gives us a useful way to compute subgradients to be used as step directions in a subgradient method. We have freedom in the choice of z; it makes sense to choose z in such a way that any step along the subgradient direction maintains feasibility of $\boldsymbol{\lambda}$ with respect to the set C defined in (6). Specifically, we should choose

$$z = \frac{1}{N} \sum_{s \in S} \mathbf{x}_s,$$

where the $\mathbf{x}_s$ are the solutions to (5), because we then have $\sum_{s \in S}(\mathbf{x}_s - z) = 0$.

2.2. Projected Subgradient Method

The projected subgradient method for the Lagrangian dual, shown in Algorithm 1, generates a sequence of iterates $(\boldsymbol{\lambda}^k)_{k \in \mathbb{N}}$ satisfying $\boldsymbol{\lambda}^k \in C$—that is, $\sum_{s \in S} \boldsymbol{\lambda}_s^k = 0$. At iteration k, we solve (5) for all $s \in S$ to obtain a subgradient $\mathbf{x}_s^k$. (We assume knowledge of just one of the solutions $(\mathbf{x}_s^k, \mathbf{y}_s^k)$ of (5) for each $s \in S$.) We assemble a subgradient of $\partial\mathcal{L}_0(\boldsymbol{\lambda}^k)$ as follows (see Lemma 1):

$$\mathbf{g}^k := \left[\mathbf{x}_s^k - z^k\right]_{s \in S}, \quad \text{where } z^k := \frac{1}{N} \sum_{s \in S} \mathbf{x}_s^k. \tag{9}$$

By this definition of z^k, $\mathbf{g}^k$ satisfies $\sum_{s \in S} \mathbf{g}_s^k = 0$, so that for any $\alpha \in \mathbb{R}$, we have

$$\boldsymbol{\lambda}^k \in C \implies \boldsymbol{\lambda}^k + \alpha \mathbf{g}^k = \boldsymbol{\lambda}^k + \alpha\left[\mathbf{x}_s^k - z^k\right]_{s \in S} \in C. \tag{10}$$

The subgradient method in general has no good practical stopping criterion (Lemarechal 1978). One can decide to terminate the algorithm when a time limit or desired objective has been reached. Alternatively, one can set an iteration limit based on the convergence guarantees, such as the one in Corollary 1. In the context of dual decomposition for SMIPs, another method that has been used in practice is to compute an upper bound by evaluating candidate first-stage solutions, and one can terminate the algorithm once the gap between the lower and upper bounds is sufficiently small (for example, Aravena and Papavasiliou 2015).

Algorithm 1 (Subgradient Algorithm for (8))

Input: starting point $\lambda^1 = [\lambda_s^1]_{s\in S}$ with $\lambda^1 \in C$
Output: Lower bound value $LB_{\max}$

$k \leftarrow 1$
$LB_{\max} \leftarrow -\infty$;
while termination criteria not met **do**
  for all $s \in S$ **do**
    Evaluate $\mathcal{L}_s(\lambda_s^k)$ and $\mathbf{x}_s^k \in \partial\mathcal{L}_s(\lambda_s^k)$;
  end for
  $LB_{\max} \leftarrow \max(LB_{\max}, (1/N)\sum_{s\in S}\mathcal{L}_s(\lambda_s^k))$;
  Set $z^k = (1/N)\sum_{s\in S}\mathbf{x}_s^k$;
  for all $s \in S$ **do**
    Set $\lambda_s^{k+1} = \lambda_s^k + \alpha_k(\mathbf{x}_s^k - z^k)$;
  end for
  $k \leftarrow k+1$;
end while

The basic convergence result for the subgradient algorithm (Ermoliev 1966) is well known, but we state it below for completeness. Convergence is proved for the function values at *weighted averages* of the iterates, not the iterates themselves. We assume that a solution of (8) exists, denoted by $\lambda^* = [\lambda_s^*]_{s\in S}$, and obtain the following results.

Theorem 1. *Let M be a constant such that $\|\mathbf{x}_s\|_2 \le M$ for all $\mathbf{x}_s \in \partial\mathcal{L}_s(\lambda_s)$, for all $[\lambda_s]_{s\in S}$ with $\sum_{s\in S}\lambda_s = 0$. Let $(\lambda^k)_{k\in\mathbb{N}}$ be generated by Algorithm 1, with $\alpha_k > 0$ for all k. Then, for all $L \ge 1$, we have*

$$\sum_{k=1}^{L}\alpha_k\left[\mathcal{L}(\lambda^*) - \mathcal{L}(\lambda^k)\right] \le \frac{1}{2}\|\lambda^1 - \lambda^*\|_2^2 + \frac{1}{2}NM^2\sum_{k=1}^{L}\alpha_k^2.$$

Proof. We have

$$\frac{1}{2}\|\lambda^{k+1} - \lambda^*\|_2^2 = \frac{1}{2}\|\lambda^k + \alpha_k\mathbf{g}^k - \lambda^*\|_2^2 = \frac{1}{2}\|\lambda^k - \lambda^*\|_2^2 + \alpha_k\langle\mathbf{g}^k, \lambda^k - \lambda^*\rangle + \frac{1}{2}\alpha_k^2\|\mathbf{g}_k\|_2^2. \tag{11}$$

For the second term, we have from (9) and Lemma 1 that

$$\langle\mathbf{g}^k, \lambda^k - \lambda^*\rangle \le \mathcal{L}_0(\lambda^k) - \mathcal{L}_0(\lambda^*) = \mathcal{L}(\lambda^k) - \mathcal{L}(\lambda^*), \tag{12}$$

because $\lambda^k \in C$ and $\lambda^* \in C$. For the third term in (11), we have from (9) that

$$\begin{aligned}\|\mathbf{g}^k\|^2 &= \left\|[\mathbf{x}_s^k - z^k]_{s\in S}\right\|_2^2 \\ &= \sum_{s\in S}\|\mathbf{x}_s^k\|_2^2 - 2(z^k)^\top\sum_{s\in S}\mathbf{x}_s^k + N\|z^k\|_2^2 \\ &= \sum_{s\in S}\|\mathbf{x}_s^k\|_2^2 - N\|z^k\|_2^2 \\ &\le \sum_{s\in S}\|\mathbf{x}_s^k\|_2^2 \le NM^2.\end{aligned} \tag{13}$$

By substituting these bounds into (11) and rearranging, we obtain

$$\alpha_k\left[\mathcal{L}(\lambda^*) - \mathcal{L}(\lambda^k)\right] \le \frac{1}{2}\|\lambda^k - \lambda^*\|_2^2 - \frac{1}{2}\|\lambda^{k+1} - \lambda^*\|_2^2 + \frac{1}{2}\alpha_k^2NM^2.$$

We obtain the desired result by summing both sides over $k = 1, 2, \ldots, L$ and using the fact that $\|\boldsymbol{\lambda}^{L+1} - \boldsymbol{\lambda}^*\|_2 \geq 0$. □

The following corollary, concerning convergence of the objective values of the averaged iterates, follows from concavity of $\mathscr{L}$ by a standard argument.

Corollary 1. *Suppose the assumptions of Theorem 1 hold. Define*

$$\bar{\boldsymbol{\lambda}}^L := \frac{\sum_{k=1}^L \alpha_k \boldsymbol{\lambda}^k}{\sum_{k=1}^L \alpha_k}.$$

Then,

$$\mathscr{L}(\boldsymbol{\lambda}^*) - \mathscr{L}(\bar{\boldsymbol{\lambda}}^L) \leq \frac{\|\boldsymbol{\lambda}^1 - \boldsymbol{\lambda}^*\|^2 + NM^2 \sum_{k=1}^L \alpha_k^2}{2 \sum_{k=1}^L \alpha_k}.$$

Various strategies for choosing step lengths α_k give rise to various convergence rate guarantees. If we have

$$\sum_{k=1}^{\infty} \alpha_k^2 < \infty, \quad \sum_{k=1}^{\infty} \alpha_k = \infty, \tag{14}$$

then Corollary 1 ensures convergence: $\mathscr{L}(\bar{\boldsymbol{\lambda}}^L) \to \mathscr{L}(\boldsymbol{\lambda}^*)$ as $L \to \infty$. Particular choices of α_k lead to guarantees on worst-case convergence rates. For example, the common choice of $\alpha_k = \theta/\sqrt{k}$ for some constant $\theta > 0$ leads to $\mathscr{L}(\boldsymbol{\lambda}^*) - \mathscr{L}(\bar{\boldsymbol{\lambda}}^L) \leq O((\log L)/\sqrt{L})$.

Algorithm 1 can be parallelized in a straightforward way. Each iteration requires evaluation of a complete subgradient, constructed from the solutions $\mathbf{x}_s^k$ to the subproblems for each of the N scenarios $s \in S$ (i.e., the loop in lines 4–6). The subproblems can be solved in parallel, but before z^k can be calculated (step 8), every scenario needs to be solved. This synchronization requirement can be a bottleneck in parallel execution of this algorithm. Because the subproblems are mixed-integer programs (MIPs), their solution time could vary significantly, not just from scenario to scenario, but also according to the values of $\boldsymbol{\lambda}^k$. If significant time is spent waiting for a small fraction of subproblems to be processed, parallel computing resources may not be utilized to their full potential.

We describe two variants of the subgradient method in the next two sections that can alleviate this bottleneck. The first variant uses stochastic subgradients of $\mathscr{L}(\boldsymbol{\lambda})$ constructed from batches of scenarios. Because the batches arise from a partition of the full scenario set S, we term this method the *partitioned subgradient method*. The second variant, an *asynchronous subgradient method*, does not necessarily wait for all scenarios to complete before taking a step, using the current returned solution for completed scenarios and the most recent solution for the remaining scenarios to construct a "noisy" subgradient. The two variants can be combined into an *asynchronous partitioned subgradient* algorithm. All these approaches maintain feasibility of iterates $\boldsymbol{\lambda}$ throughout, allowing us to continually update a lower bound to monitor the algorithm's progress.

3. Partitioned Stochastic Subgradient

We now propose a method in which a stochastic estimate of the subgradient is used in place of the subgradient itself. We describe first a serial variant, then a parallel implementation that alleviates to some extent the bottleneck issues associated with the full subgradient method of Section 2.

3.1. A Serial Variant

A random vector $\tilde{\mathbf{g}}^k$ taking values in $\mathbb{R}^{n \times N}$ is said to be a *stochastic subgradient* of $\mathscr{L}_0$ at $\boldsymbol{\lambda}^k$ if

$$\mathbb{E}[\tilde{\mathbf{g}}^k] \in \partial \mathscr{L}_0(\boldsymbol{\lambda}^k),$$

where the expectation is taken over all the random quantities on which $\tilde{\mathbf{g}}^k$ depends, conditional on $\boldsymbol{\lambda}^k$. We obtain such a vector by combining the subgradients from only a subset of scenarios instead of from all of them as in (9), leading to a similar subgradient analysis (in expectation) to that of Algorithm 1.

At iteration k, we pick a partition of N into batches of equal size $K \geq 2$ uniformly at random.[1] (We assume for simplicity that N is a multiple of K.) This can be done by randomly reordering the indices and assigning the indices in positions $\mathbf{c}K + 1, \mathbf{c}K + 2, \ldots, (\mathbf{c} + 1)K$ to the same batch for each nonnegative integer $\mathbf{c}$. We use $\mathscr{P}^k$ to denote this partition:

$$\mathscr{P}^k := \left\{T_1^k, T_2^k, \ldots, T_{N/K}^k\right\}, \tag{15}$$

where

$$T_1^k \cup T_2^k \cup \ldots \cup T_{N/K}^k = \{1, 2, \ldots, N\}, \quad \text{where } |T_i^k| = K, \text{ for all } i. \tag{16}$$

For any $s \in S$ and iteration k, we use the following term to denote which batch scenario s belonged to in iteration k:

$$i_k(s) := i \text{ where } s \in T_i^k. \tag{17}$$

A subgradient step like that of Algorithm 1 is computed for each T_i^k, for $i = 1, 2, \ldots, N/K$, separately, as follows:

$$\hat{\mathbf{g}}^k := \left[\hat{\mathbf{g}}_s^k\right]_{s\in S}, \quad \text{where } \hat{\mathbf{g}}_s^k = \mathbf{x}_s^k - z^k_{T^k_{i_k(s)}} \quad \text{for all } s \in S, \tag{18}$$

where for any $T \in \mathcal{P}^k$, we define

$$z_T^k := \frac{1}{K}\sum_{j\in T} \mathbf{x}_j^k. \tag{19}$$

(By setting $T = S$, we have from (9) that $z_S^k = z^k$.)

The following result proven in Appendix B establishes that a scaling of $\hat{\mathbf{x}}^k$ is a stochastic subgradient of $\mathcal{L}_0$ at $\boldsymbol{\lambda}^k$. In fact, in expectation, it is a multiple of $\mathbf{g}^k$ defined in (9).

Proposition 1. *For the random vector $\hat{\mathbf{g}}^k$ defined by (18), we have*

$$\frac{K(N-1)}{N(K-1)}\mathbb{E}_{\mathcal{P}^k}\hat{\mathbf{g}}^k = \mathbf{g}^k = \left[\mathbf{x}_s^k - z^k\right]_{s\in S} \in \partial\mathcal{L}_0(\boldsymbol{\lambda}^k),$$

where the expectation is with respect to the random partition $\mathcal{P}^k$ defined by (15).

Algorithm 2 (Partitioned Stochastic Subgradient Algorithm for (8))

Input: starting point $\boldsymbol{\lambda}^1 = [\boldsymbol{\lambda}_s^1]_{s\in S}$ with $\boldsymbol{\lambda}^1 \in C$
Output: Lower bound value $LB_{\max}$

$k \leftarrow 1$;
$LB_{\max} \leftarrow -\infty$;
while termination criteria not met **do**
  for all $s \in S$ **do**
    Evaluate $\mathcal{L}_s(\boldsymbol{\lambda}_s^k)$ and $\mathbf{x}_s^k \in \partial\mathcal{L}_s(\boldsymbol{\lambda}_s^k)$;
  end for
  $LB_{\max} \leftarrow \max(LB_{\max}, (1/N)\sum_{s\in S}\mathcal{L}_s(\boldsymbol{\lambda}_s^k))$;
  Define $\hat{\mathbf{g}}^k$ as in (18) with respect to some uniform random partition $\mathcal{P}^k$;
  for all $s \in S$ **do**
    Set $\boldsymbol{\lambda}_s^{k+1} = \boldsymbol{\lambda}_s^k + \alpha_k \frac{K(N-1)}{N(K-1)}\hat{\mathbf{g}}_s^k$;
  end for
  $k \leftarrow k + 1$
end while

A partitioned stochastic subgradient algorithm is shown as Algorithm 2. To prove a convergence result (stochastic counterparts of Theorem 1 and Corollary 1), we need a bound on $\mathbb{E}_{\mathcal{P}^k}\|\hat{\mathbf{g}}^k\|^2$, given by the following result, whose proof appears in Appendix B.

Lemma 2. *Let M be a constant such that $\|\mathbf{x}_s\|_2 \le M$ for all $\mathbf{x}_s \in \partial\mathcal{L}_s(\boldsymbol{\lambda}_s)$, for all $[\boldsymbol{\lambda}_s]_{s\in S}$ with $\sum_{s\in S}\boldsymbol{\lambda}_s = 0$. Then, for the random partition $\mathcal{P}^k$ and stochastic subgradient $\hat{\mathbf{g}}^k$ defined in (15) and (18), respectively, we have*

$$E_{\mathcal{P}^k}\|\hat{\mathbf{g}}^k\|^2 \le \frac{N(K-1)}{K(N-1)}NM^2.$$

A convergence result for Algorithm 2 (following Theorem 1) is stated next. The proof appears in Appendix B.

Theorem 2. *Let M be a constant such that $\|\mathbf{x}_s\|_2 \le M$ for all $\mathbf{x}_s \in \partial\mathcal{L}_s(\boldsymbol{\lambda}_s)$, for all $[\boldsymbol{\lambda}_s]_{s\in S}$ with $\sum_{s\in S}\boldsymbol{\lambda}_s = 0$. Let $(\boldsymbol{\lambda}^k)_{k\in\mathbb{N}}$ be generated by Algorithm 2, with $\alpha_k > 0$ for all k. Then, for all $L \ge 1$, we have*

$$\sum_{k=1}^{L}\alpha_k\mathbb{E}\left[\mathcal{L}(\boldsymbol{\lambda}^*) - \mathcal{L}(\boldsymbol{\lambda}^k)\right] \le \frac{1}{2}\|\boldsymbol{\lambda}^1 - \boldsymbol{\lambda}^*\|_2^2 + \frac{1}{2}\frac{K(N-1)}{N(K-1)}NM^2\sum_{k=1}^{L}\alpha_k^2,$$

where the expectation is taken over the random partitions $\mathcal{P}_1, \mathcal{P}_2, \ldots$.

The following corollary is immediate.

Corollary 2. *Suppose the assumptions of Theorem 2 hold. Define*

$$\bar{\lambda}^L := \frac{\sum_{k=1}^{L} \alpha_k \lambda^k}{\sum_{k=1}^{L} \alpha_k}.$$

Then,

$$\mathbb{E}\left(\mathcal{L}(\lambda^*) - \mathcal{L}(\bar{\lambda}^L)\right) \leq \frac{\|\lambda^1 - \lambda^*\|^2 + \frac{K(N-1)}{N(K-1)} NM^2 \sum_{k=1}^{L} \alpha_k^2}{2 \sum_{k=1}^{L} \alpha_k},$$

where the expectation is taken over the random partitions $\mathcal{P}_1, \mathcal{P}_2, \ldots$.

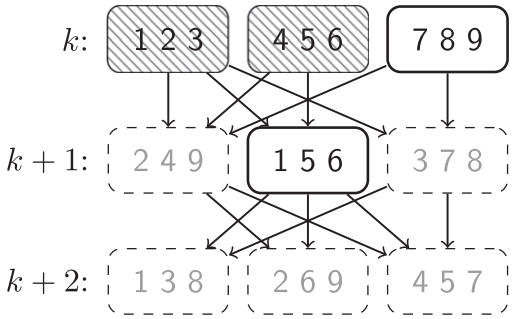
Note that the bound in Corollary 2 is slightly worse than the corresponding bound for the full-subgradient algorithm in Corollary 1, because of the presence of the factor $\frac{K(N-1)}{N(K-1)}$ in the numerator. Because $K \leq N$, this factor lies in the interval $[1, 2)$. (It is close to two when K takes its minimal value of two.) However, as we discuss next, the partitioned approach of Algorithm 2 is potentially more amenable to parallel implementation than the full-subgradient approach of Algorithm 1.

3.2. Parallel Partitioned Subgradient Implementation

Algorithm 2 still requires computation of $x_s^k \in \partial \mathcal{L}_s(\lambda_s^k)$ for every scenario s at every iteration k (line 6), but it can potentially limit the impact of synchronization by allowing computations for a future iteration to be done before all scenarios from iteration k are solved. In particular, for a given batch T_i^k, as soon as $x_s^k \in \partial \mathcal{L}_s(\lambda_s^k)$ has been found for all $s \in T_i^k$, it is possible to compute $\hat{g}_s^k$ and, hence, λ_s^{k+1} for all $s \in T_i^k$. As a result, the subproblems for iteration $k+1$ for scenario $s \in T_i^k$ can be started. This algorithmic feature contrasts with the projected subgradient method, which does not start processing subproblems corresponding to later iterations if there is some unfinished subproblem in the current iteration.

Figure 1 gives a concrete example of how the partitioning process improves parallelism. Each row corresponds to a partition/iteration, each box represents a batch with three scenarios, and the arrows pointing to a batch show which batches need to be finished before that batch can be processed. The hatched boxes represent completed batches (that is, λ_s^k, $\mathcal{L}_s(\lambda_s^k)$, and $x_s^k \in \partial \mathcal{L}_s(\lambda_s^k)$ have been computed for all s in the batch). The boxes with thick outlines represent batches where, for every s in the batch, λ_s^k has been computed, and not all $\mathcal{L}_s(\lambda^k)$ and $x_s^k \in \partial \mathcal{L}_s(\lambda_s^k)$ have been computed. The boxes with dashed outlines are batches where not every λ_s^k has been computed. Even if the subproblems corresponding to scenarios 7–9 at iteration k are still being solved, the subproblems for scenarios 1–6 at iteration $k+1$ can be solved because their corresponding batches in iteration k are complete, and we know λ_s^{k+1} for $s \in \{1, \ldots, 6\}$. Further, once the subproblems for scenarios 1, 5, and 6 at iteration $k+1$ have completed processing, we can compute λ_1^{k+2}, λ_5^{k+2}, and λ_6^{k+2} and start processing their subproblems for iteration $k+2$, even if the subproblems involving scenarios 7–9 from iteration k are still being solved.

Figure 1. Example of Dependency Graph Between Batches Across Different Partitions



We describe the algorithm formally in Algorithm 3. To parallelize the computation, we use the master-worker distributed computing framework, which is a centrally coordinated framework in which a master node runs the main algorithm while assigning subtasks or jobs to worker processors. Each job consists of computing the objective $\mathscr{L}_s(\lambda_s^k)$ and subgradient $\mathbf{x}_s^k \in \partial\mathscr{L}_s(\lambda_s^k)$ associated with a specific scenario s at iteration k. The master adds these jobs to a queue , and workers are assigned scenario–iteration pairs (s,k) one at a time from .

An interesting feature of this implementation is that, once $\mathscr{L}_{s'}(\lambda_{s'}^1)$ has been computed for all $s' \in S$ (the initial jobs added to the queue have been completed), the algorithm maintains an updated lower bound LB each time a batch completes (line 23). In particular, after $\mathscr{L}_{s'}(\lambda_{s'}^1)$ has been computed for all $s' \in S$, the value LB updated in line 24 equals the value of $(1/N)\mathscr{L}(\hat{\lambda})$, where $\hat{\lambda}_s$ refers to the most recent λ_s^k used in the computation of LB. Indeed, in line 24, $\hat{\lambda}_{s'}$ is implicitly updated from $\lambda_{s'}^{k-1}$ to $\lambda_{s'}^k$ for $s' \in T_i^{k-1}$, and, hence, the impact of this update on $\sum_{s' \in S} \hat{\lambda}_{s'}$ is the term

$$\sum_{s' \in T_i^{k-1}} \left(\lambda_{s'}^k - \lambda_{s'}^{k-1}\right) = 0,$$

due to the computation of $\lambda_{s'}^k$ in line 15. Thus, once $\hat{\lambda}_{s'}$ has been initialized to $\lambda_{s'}^1$ for all $s' \in S$, the algorithm maintains $\hat{\lambda} \in C$ thereafter, and, hence, $LB = (1/N)\mathscr{L}(\hat{\lambda})$ is a lower bound on ϕ^{LD}.

Note that when $N = K$, Algorithm 3 is simply a master-worker implementation of the standard subgradient method.

Algorithm 3 (Parallel Partitioned Subgradient Algorithm for (8))

Input: starting point $\lambda^1 = [\lambda_s^1]_{s \in S}$ with $\lambda^1 \in C$
Output: Lower bound value $LB_{\max}$

1: $Q \leftarrow \{(s,1) : s \in S\}$;
2: $LB \leftarrow 0$; $LB_{\max} \leftarrow -\infty$;
3: **while** termination criteria not met **do**
4: **while** worker available and $Q \neq \emptyset$ **do**
5: $(s,k) \leftarrow \mathrm{pop}(Q)$;
6: Assign a worker the job to compute $\mathscr{L}_s(\lambda_s^k)$ and $\mathbf{x}_s^k \in \partial\mathscr{L}_s(\lambda_s^k)$;
7: **end while**
8: **while** there is some finished job (s,k) not yet processed by master **do**
9: $(\mathscr{L}_s(\lambda_s^k), \mathbf{x}_s^k)$ at master $\leftarrow$ output from job (s,k);
10: Mark job (s,k) as processed;
11: $i \leftarrow i_k(s)$; {see definition (17)}
12: **if** all jobs (s',k) for $s' \in T_i^k$ have finished processing **then**
13: $z \leftarrow (1/K) \sum_{s' \in T_i^k} \mathbf{x}_{s'}^k$;
14: **for** all $s' \in T_i^k$ **do**
15: $\lambda_{s'}^{k+1} \leftarrow \lambda_{s'}^k + \alpha_k \frac{K(N-1)}{N(K-1)}(\mathbf{x}_{s'}^k - z)$;
16: **end for**
17: $Q \leftarrow Q \cup \{(s',k+1) : s' \in T_i^k\}$;
18: **end if**
19: **if** $k = 1$ **then**
20: $LB \leftarrow LB + (1/N)(\mathscr{L}_s(\lambda_s^k))$;
21: **else**
22: $j \leftarrow i_{k-1}(s)$; {see definition (17)}
23: **if** all jobs (s',k) for $s' \in T_j^{k-1}$ have finished processing **then**
24: $LB \leftarrow LB + (1/N) \sum_{s' \in T_j^{k-1}} (\mathscr{L}_{s'}(\lambda_{s'}^k) - \mathscr{L}_{s'}(\lambda_{s'}^{k-1}))$;
25: **end if**
26: **end if**
27: **if** all jobs $(s',1)$ for all scenarios $s' \in S$ have finished processing **then**
28: $LB_{\max} \leftarrow \max(LB_{\max}, LB)$;
29: **end if**
30: **end while**
31: **end while**

3.3. Batch Size and Parallel Performance

The choice of K involves balancing the trade-off between the improved parallelism afforded by a small value of K versus the deterioration in convergence rate due to the $\frac{K(N-1)}{N(K-1)}$ factor in Corollary 2. Note that once K is larger than N/K, it is technically possible for a *single* hard scenario to hold up progress on all subsequent partitions if the batch the scenario is contained in has at least one scenario in every batch in the following partition. In our experiments, we see that setting K to slightly larger than N/K can still offer computational benefits in some cases, especially when we incorporate asynchronous updates (see Section 4.1).

In the pathological case where there is a single scenario that is consistently extremely difficult to solve relative to all other scenarios, then even a small value of K may not be enough to help significantly. That single scenario will eventually hold up progress on all other scenarios, even if a smaller K value may allow for more iterations before this starts happening.

4. An Asynchronous Subgradient Scheme

In the master-worker implementation of the projected subgradient method, when the job queue is empty, there are no more tasks that can be assigned to idle workers. To prevent this from occurring, when the queue is sufficiently small, we can take measures to refill it. We do this by forcing a step with an estimate of a full subgradient that uses the *most recent* value $\mathbf{x}_s$ that has been computed for each scenario s. Algorithm 4 describes this procedure in detail. We use the notation τ_s to denote the iteration that gave rise to $\mathbf{x}_s$. (If the latest $\mathbf{x}_s$ at the master comes from $\boldsymbol{\lambda}^{k'}$, then $\tau_s = k'$.)

Note that, in order to update our lower bound, we have to wait for *all* scenarios from a particular iteration k to complete. As a result, we do not terminate jobs that are currently being processed until all the jobs corresponding to a more recent iteration are complete.

Algorithm 4 (Asynchronous Subgradient Algorithm for (8))

Input: starting point $\boldsymbol{\lambda}^1 = [\boldsymbol{\lambda}_s^1]_{s\in S}$ with $\boldsymbol{\lambda}^1 \in C$, $Q_{\text{threshold}}$;
Output: Lower bound value $LB_{\max}$

$Q \leftarrow \{(s,1) : s \in S\}$;
$k \leftarrow 1$;
$\tau_s^0 \leftarrow 0$;
$LB_{\max} \leftarrow -\infty$;
while termination criteria not met **do**
  while $|Q| > Q_{\text{threshold}}$ or ($k = 1$ and not all $(s,1)$ jobs have been processed) **do**
    if worker available **then**
      $(s,\ell) \leftarrow \text{pop}(Q)$;
      Assign a worker the job to compute $\mathcal{L}_s(\boldsymbol{\lambda}_s^\ell)$ and $\mathbf{x}_s^\ell \in \partial\mathcal{L}_s(\boldsymbol{\lambda}_s^\ell)$;
    end if
    while there is some finished job (s,ℓ) not yet processed by master **do**
      $(\mathcal{L}_s(\boldsymbol{\lambda}^\ell), x_s^\ell)$ at master $\leftarrow$ output from job (s,ℓ);
      Mark job (s,ℓ) as processed;
      if (s',ℓ) is done for all $s' \in S$ **then**
        $LB_{\max} \leftarrow \max(LB_{\max}, (1/N)\sum_{s'\in S}\mathcal{L}_{s'}(\boldsymbol{\lambda}_{s'}^\ell))$;
        Terminate all running jobs (s',j) for which $j < \ell$ for all $s' \in S$;
      end if
      if $\tau_s^{k-1} < \ell$ **then**
        $\hat{\mathbf{x}}_s^k \leftarrow \mathbf{x}_s^\ell$;
        $\tau_s^k \leftarrow \ell$;
      end if
    end while
  end while
  $\hat{z}^k \leftarrow (1/N)\sum_{s\in S}\hat{\mathbf{x}}_s^k$;
  for all $s \in S$ **do**
    $\boldsymbol{\lambda}_s^{k+1} \leftarrow \boldsymbol{\lambda}_s^k + \alpha_k(\hat{\mathbf{x}}_s^k - \hat{z}^k)$;
  end for
  $Q \leftarrow \{(s,k+1) : s \in S\}$;
  $k \leftarrow k+1$;
end while

We introduce notation τ_s^k to denote the iteration $j \le k$ from which the version of the subgradient $\hat{\mathbf{x}}_s$ stored on the master at line 24, corresponding to the kth iteration of the main loop in Algorithm 4 was derived. This means that $\hat{\mathbf{x}}_s^k = \mathbf{x}_s^{\tau_s^k}$. Define $\hat{\boldsymbol{\lambda}}_s^k := \boldsymbol{\lambda}_s^{\tau_s^k}$, and let $\hat{\mathbf{x}}^k := [\hat{\mathbf{x}}_s^k]_{s\in S}$, $\hat{\boldsymbol{\lambda}}^k := [\hat{\boldsymbol{\lambda}}_s^k]_{s\in S}$. Note that

$$\hat{\mathbf{x}}^k \in \partial \mathscr{L}\big(\hat{\boldsymbol{\lambda}}^k\big). \tag{20}$$

The subgradient step in Algorithm 4 can now be written as follows:

$$\boldsymbol{\lambda}_s^{k+1} = \boldsymbol{\lambda}_s^k + \alpha_k\big(\hat{\mathbf{x}}_s^k - \hat{\mathbf{z}}^k\big), \quad \text{for all } s \in S, \tag{21}$$

where $\hat{\mathbf{z}}^k = \frac{1}{N}\sum_{s\in S}\hat{\mathbf{x}}_s^k$.

We now state a result about the convergence of a weighted average of the iterates $(\boldsymbol{\lambda}^k)_{k\in\mathbb{N}}$ to an optimal value.

Theorem 3. *Let M be a constant such that $\|\mathbf{x}_s^k\|_2 \le M$ for all $\mathbf{x}_s^k \in \partial\mathscr{L}_s(\boldsymbol{\lambda}_s^k)$, all $\boldsymbol{\lambda}_s^k$, all $s \in S$, and all $k \in \mathbb{N}$. Let $(\boldsymbol{\lambda}^k)_{k\in\mathbb{N}}$ be generated by Algorithm 4, with $\alpha_k > 0$ for all k. For any $L = 1, 2, \ldots$, define*

$$\bar{\boldsymbol{\lambda}}^L := \frac{\sum_{k=1}^L \alpha_k \hat{\boldsymbol{\lambda}}^k}{\sum_{k=1}^L \alpha_k}, \quad \tilde{\boldsymbol{\lambda}}^L := \frac{\sum_{k=1}^L \alpha_k \boldsymbol{\lambda}^k}{\sum_{k=1}^L \alpha_k}.$$

Then,

$$\mathscr{L}\big(\boldsymbol{\lambda}^*\big) - \mathscr{L}\big(\bar{\boldsymbol{\lambda}}^L\big) \le \frac{\|\boldsymbol{\lambda}^1 - \boldsymbol{\lambda}^*\|^2 + M^2\Big(N\sum_{k=1}^L \alpha_k^2 + 2\sum_{k=1}^L \alpha_k \sum_{s\in S}\sum_{i=\tau_s^k}^{k-1}\alpha_i\Big)}{2\sum_{k=1}^L \alpha_k}.$$

Furthermore, we have

$$\big(\bar{\boldsymbol{\lambda}}^L, C\big) \le \|\bar{\boldsymbol{\lambda}}^L - \tilde{\boldsymbol{\lambda}}^L\| \le \frac{2M\sum_{k=1}^L \alpha_k \sum_{s\in S}\sum_{i=\tau_s^k}^{k-1}\alpha_i}{\sum_{k=1}^L \alpha_k},$$

where (w, C) denotes the Euclidean distance between w and the set C, defined in (6).

For α_k satisfying the usual conditions (14), the right-hand sides of the inequalities in Theorem 3 approach zero, provided that the "ages" of the updates are bounded—that is, there is a positive integer D such that

$$k - \tau_s^k \le D, \quad \text{for all } k \in \mathbb{N} \text{ and all } s \in S. \tag{22}$$

(We expect D to be modest unless there are a small fraction of scenario evaluations that require much longer to process than the times for all other scenarios combined.) For each $s \in S$, using the elementary inequality $2ab \le a^2 + b^2$ for any $a, b \in \mathbb{R}$, we have

$$\alpha_k \sum_{i=\tau_s^k}^{k-1}\alpha_i \le \sum_{i=\max(k-D,1)}^{k-1}\alpha_k\alpha_i \le \sum_{i=\max(k-D,1)}^{k-1}\frac{1}{2}\big(\alpha_k^2 + \alpha_i^2\big) \le \frac{1}{2}\left(D\alpha_k^2 + \sum_{i=\max(k-D,1)}^{k-1}\alpha_i^2\right).$$

Hence, we have

$$\sum_{k=1}^L \alpha_k \sum_{i=\tau_s^k}^{k-1}\alpha_i \le \frac{1}{2}\left(\sum_{k=1}^L D\alpha_k^2 + \sum_{k=1}^L \sum_{i=\max(k-D,1)}^{k-1}\alpha_i^2\right) \le D\sum_{k=1}^L \alpha_k^2,$$

and it follows that

$$\operatorname{dist}\big(\bar{\boldsymbol{\lambda}}^L, C\big) \le \frac{2M\sum_{k=1}^L \alpha_k \sum_{s\in S}\sum_{i=\tau_s^k}^{k-1}\alpha_i}{\sum_{k=1}^L \alpha_k} \le \frac{2MND\sum_{k=1}^L \alpha_k^2}{\sum_{k=1}^L \alpha_k},$$

so that $(\bar{\boldsymbol{\lambda}}^L, C) \to 0$ because $\sum_{k=1}^\infty \alpha_k^2$ is bounded and $\sum_{k=1}^\infty \alpha_k$ is not. For the second bound in Theorem 3, similar logic yields $\mathscr{L}(\boldsymbol{\lambda}^*) - \mathscr{L}(\hat{\boldsymbol{\lambda}}^L) \to 0$ under the same assumptions.

4.1. An Asynchronous Partitioned Method

The partitioned method is likely to be blocked much less often than the projected subgradient method, but blocking is still a possibility. To reduce the possibility of blocking further, we consider an asynchronous variant of the partitioned method: When the input queue is empty and all batches are waiting on some

scenarios, we simply take a batch that was created at the earliest iteration among the open batches, use the most recent subgradient information for the scenarios in that batch, and take a step. The description of the algorithm and the analysis is quite similar to what we had in the preceding two sections. The choice of the batch size K for the asynchronous partitioned method follows the same intuition as in Section 3.3.

5. Computational Results

In this section, we report results from a numerical comparison between our methods and the standard full subgradient approach, reporting on the number of MIP solves and the wall-clock time required to reach certain levels of accuracy.

5.1. Computing Setup and Implementation Details

The experiments were run on a dedicated server with two 2.2-Ghz Intel E5-4640 Xeon processors (40 cores total) with 256 GB of RAM. We implemented the subgradient algorithms in Python (Version 3.6.4) in NumPY (Version 1.13). Gurobi (Version 7.5.2) was used for solving the mixed-integer subproblems associated with each scenario. We used 32 workers with a single core each to process the subproblems.

We implemented four variants of the subgradient method in our experiments. Our baseline method was the standard projected subgradient method, Algorithm 1, with the subproblems for all scenarios solved in parallel over all processors. We also implemented the parallel partitioned subgradient method (Algorithm 3), the asynchronous projected full subgradient method (Algorithm 4), and an algorithm that combines partitioning with asynchronicity (discussed in Section 4.1). We denote these methods by FSG, PSG, ASG, and APSG, respectively.

The parallel versions of the subgradient algorithms required two modifications to a standard serial version. We first created a data structure to maintain the list of batches created and the dependencies, as illustrated in Figure 1. Secondly, we organized each algorithm around issuing scenario subproblems to solve and process the solutions as each result comes in.

To evaluate the scenario subproblems in parallel, we used a master-worker parallel computing framework provided by the Python package Dask.distributed. We launched a scheduler on a process then separately created 32 single-core workers to connect to the scheduler. Each subgradient algorithm issues tasks (single-scenario subproblems) to the scheduler as needed, which subsequently assigns these tasks to workers. Whenever a task is finished by a worker, the main routine is notified, and the results are processed.

We briefly discuss how the number of workers W affects the parallel performance. If W is many factors smaller than the number of scenarios N, then in the projected subgradient method, the fraction of time that workers spend idling is significantly smaller, which, in turn, reduces the potential benefit of using the partitioned or asynchronous methods. On the other hand, if W is very close to N, the fraction of time that workers spend idling in the projected subgradient method can take up a large portion of the running time. We also note that there is no benefit in having $W > N$ in the projected subgradient and partitioned subgradient methods because only N subproblems can be solved simultaneously in those methods. On the other hand, the asynchronous methods can have the same scenario (with different $\boldsymbol{\lambda}_s$ values) being evaluated by multiple workers at the same time.

5.2. Methods Evaluated

The partitioning methods use a batch parameter K, and we use PSG-K and APSG-K to denote this parameter when needed. We set the size of each batch within the partition K to be five or 10 for the 50-scenario instances and 10 or 20 for the 200-scenario instances. For these values, the ratio $\frac{K(N-1)}{N(K-1)}$ that appears in the convergence results of Section 3 takes on the values shown in Table 1.

For the asynchronous methods ASG and APSG, we ensured saturation of the processors by forcing a step to be taken with the most recent information available, whenever there were five or fewer scenarios remaining in the job queue for which processing had not yet been started. This corresponds to setting $Q_{\text{threshold}} = 5$ in Algorithm 4.

We set the step lengths as $\alpha_k := \theta/k$, where θ is a constant chosen via grid search for FSG to maximize the lower bound within a chosen time limit then used for all the methods. Initial experiments with constant step lengths or of the form $\alpha_k = \theta/\sqrt{k}$ did not perform as well as the θ/k step length, so we did not use those.

5.3. Problem Instances

We used the canonical SSLP (Ntaimo and Sen 2005) and DCAP (Ahmed and Garcia 2003) problem families from SIPLIB (Ahmed et al. 2015). SSLP (stochastic server location) has binary first-stage variables and mixed second stage, whereas DCAP (dynamic capacity acquisition) has mixed first stage and binary second stage. We

Table 1. Scenarios, Partition Sizes, and Convergence Ratios

N (scenarios)	K (partition size)	$\frac{K(N-1)}{N(K-1)}$ (convergence ratio)
50	5	1.225
50	10	1.089
200	10	1.106
200	20	1.047

performed experiments on both 50-scenario and 200-scenario versions of these problems. Existing instances of these problems have a relatively small number of first-stage variables and can be solved fairly quickly, so we created new instances using the same generation process as the original problems, but with significantly more variables in both the first and second stages.[2] In Table 2, we describe the numbers of binary variables (#$\mathbb{B}$), continuous variables (#$\mathbb{R}$), and constraints (#cons) in each stage of the problem, and in the extensive form for the 200-scenario instance. The two numbers in the SSLP problem names correspond to (i) the number of server locations and (ii) the number of customers. The three numbers in DCAP names correspond to the numbers of (i) resources, (ii) tasks, and (iii) time periods.

As discussed in Section 1, the variability in the time taken to solve scenario-wise subproblems can have a significant effect on overall running time. We report statistics on the average solve times in Table 3. For each problem and each scenario, we averaged the time it takes to solve a subproblem associated with that particular scenario as a measure of how difficult that scenario is to process. We then summarized the distribution over all the scenarios by reporting the quartiles. We will again refer to these statistics when discussing the experimental results in Section 5.4.3. The scenarios in SSLP instances have comparatively high solve times compared with the DCAP instances, whereas the variability in SSLP times is much smaller. Specifically, the ratio between the median and the maximum is always within a factor of two to three for SSLP, whereas this ratio can reach 100 for DCAP. We also provide an aggregate measure of solve time variability for each instance via the coefficient of variation (CV) for all the solve times over all scenarios for each instance, obtained by dividing the standard deviation by the mean. As the number of scenarios increases from 50 to 200, the tail ends of the distribution can become far more extreme for DCAP instances (especially for DCAP 6-6-5), whereas for SSLP, the CV remains relatively stable.

5.4. Comparison of Subgradient Algorithms

Dual decomposition is used to obtain high-quality lower bounds quickly, so it is natural to compare either how much time each method requires to reach a target value, or else the objective value of each method after a given amount of computation time. We used the former in our experiments. If the method is being used to obtain "good enough" lower bounds, we can set a termination criterion based on the objective value relative to some upper bound that is updated on the fly by evaluating candidate first-stage solutions (see, for example, Aravena and Papavasiliou 2015). To make this termination criterion consistent between the different methods in our experiments, we simply picked threshold values and checked how long it took each method to reach this value.

We evaluated the relative performance of the various subgradient methods by comparing the wall-clock time and number of MIP solves required for each method to reach specific optimality thresholds. To define these thresholds, we took the tightest lower bound $f^* = \mathscr{L}(\boldsymbol{\lambda}^*)$ obtained over all runs of all methods and

Table 2. Properties of Our Stochastic Programming Test Problems

	First stage			Second stage			Extensive form		
Problem	#$\mathbb{B}$	#$\mathbb{R}$	#cons	#$\mathbb{B}$	#$\mathbb{R}$	#cons	#$\mathbb{B}$	#$\mathbb{R}$	#cons
SSLP-20-100	20	0	1	2,000	20	120	4e6	4,000	24,001
SSLP-30-100	30	0	1	3,000	30	130	6e6	6,000	26,001
SSLP-60-60	60	0	1	3,600	60	120	7.2e6	12,000	24,001
SSLP-90-45	90	0	1	4,050	90	135	8.1e6	18,000	27,001
DCAP-4-6-8	32	32	32	192	48	80	38,432	9,632	16,032
DCAP-5-7-5	25	25	25	175	35	60	35,025	7,025	12,025
DCAP-6-6-5	30	30	30	180	30	60	36,030	6,030	6,030
DCAP-7-4-7	49	49	49	196	28	77	38,490	9,649	16,049

Table 3. Statistics of Scenario Solve Times and Coefficients of Variation

		Mean solve time quartiles (seconds)					
Problem	Scenarios	Minimum	25%	Median	75%	Maximum	Coefficients of variation
SSLP-20-100	50	0.57	1.23	1.51	2.14	3.62	0.95
	200	0.50	1.29	1.64	2.24	4.85	0.99
SSLP-30-100	50	0.84	5.07	6.47	8.06	10.90	0.65
	200	1.24	4.57	6.52	9.06	15.26	0.63
SSLP-60-60	50	1.32	8.54	10.41	14.27	20.21	0.46
	200	1.17	8.87	11.63	14.94	22.80	0.42
SSLP-90-45	50	1.21	6.50	8.82	11.55	23.43	0.57
	200	1.19	6.64	8.97	12.50	25.32	0.52
DCAP-4-6-8	50	0.05	0.11	0.18	0.27	1.18	1.00
	200	0.03	0.08	0.14	0.23	2.43	1.79
DCAP-5-7-5	50	0.04	0.06	0.10	0.17	2.42	2.28
	200	0.04	0.07	0.12	0.21	6.78	2.69
DCAP-6-6-5	50	0.05	0.10	0.15	0.37	1.58	1.42
	200	0.03	0.08	0.13	0.23	17.63	6.34
DCAP-7-4-7	50	0.07	0.19	0.26	0.39	1.59	0.99
	200	0.06	0.21	0.30	0.40	1.59	0.81

compared it with the initial lower bound of $f_0 = \mathcal{L}(0)$. The thresholds were then set to be $f^* + c(f_0 - f^*)$ for $c \in \{0.1, 0.05, 0.02\}$. These values correspond to 10%, 5%, and 2% gaps from the best Lagrangian dual value ever observed for the instance.

We ran each subgradient method on each instance five times and computed the wall-clock time taken and of single-scenario MIP subproblems (MIPs for short) required for each method to reach each optimality threshold. Results are shown in Tables 4, 5, 6, and 7 for SSLP and in Tables 8, 9, 10, and 11 for DCAP. The "Base (s)" column reports the average time (in seconds) required for the parallel implementation of FSG to attain the threshold value. The columns for PSG, ASG, and APSG show the time for each of these methods as a fraction of the base time. (A number less than one means that the algorithm is faster than FSG.) The two numbers in the column represent the timings of the fastest run/slowest run. If a run does not reach the threshold value, then the ratio for the slowest run is shown as infinity. If all runs do not reach the value, then the minimum ratio is also infinity.

5.4.1. SSLP Results. On the SSLP instances, PSG performed better than FSG in terms of wall-clock time for most of the runs, indicating that PSG makes more efficient use of the parallel computing resources.

The ASG method in most cases performed worse than FSG, possibly because asynchronicity introduces a significant amount of noise, especially when the number of scenarios is small, making the asynchronous step relying proportionally more often on outdated values of $\mathbf{x}_s$. On the other hand, asynchronicity provides better

Table 4. Running Time Ratios for SSLP Instances: 50 Scenarios

			Minimum/maximum ratio				
Instance	% Gap	Base (s)	PSG ($K = 5$)	PSG ($K = 10$)	ASG	APSG ($K = 5$)	APSG ($K = 10$)
20-100	10	137	0.81/0.86	0.73/0.88	1.16/1.29	1.11/1.22	1.02/1.28
	5	292	0.77/0.83	0.73/0.79	1.22/1.32	1.03/1.19	0.99/1.09
	2	723	0.76/0.80	0.70/0.84	1.88/2.66	1.10/1.18	1.10/1.25
30-100	10	274	0.88/1.01	0.94/1.06	1.18/1.26	1.25/1.44	1.21/1.43
	5	470	0.80/0.94	0.88/0.96	1.10/1.17	1.14/1.22	1.13/1.22
	2	753	0.82/0.93	0.85/0.94	1.24/1.34	1.16/1.30	1.21/1.32
60-60	10	822	0.80/0.91	0.86/0.91	1.10/1.15	0.89/0.94	1.15/1.30
	5	1,712	0.71/0.79	0.81/0.84	1.00/1.12	0.79/0.87	1.14/1.24
	2	2,907	0.75/0.85	0.78/0.88	1.21/∞	0.83/1.00	1.26/1.37
90-45	10	505	0.85/0.92	0.85/0.98	1.07/1.21	0.96/1.15	1.18/1.34
	5	1,087	0.81/0.90	0.81/0.84	0.95/1.07	0.86/0.94	1.07/1.17
	2	2,143	0.83/0.91	0.82/0.85	1.09/1.21	0.88/0.98	1.12/1.24

Table 5. Running Time Ratios for SSLP Instances: 200 Scenarios

			Minimum/maximum ratio				
Instance	% Gap	Base (s)	PSG ($K = 10$)	PSG ($K = 20$)	ASG	APSG ($K = 10$)	APSG ($K = 20$)
20-100	10	657	0.93/0.99	0.90/0.96	1.09/1.18	0.95/1.01	0.94/0.97
	5	1,376	0.90/0.96	0.88/0.92	1.08/1.15	0.92/0.96	0.90/0.96
	2	2,912	0.85/0.90	0.83/0.91	1.12/1.21	0.86/0.91	0.85/0.91
30-100	10	1,047	0.84/0.89	0.92/0.96	1.01/1.15	0.84/0.89	0.95/0.99
	5	1,611	0.85/0.89	0.90/0.96	0.99/1.12	0.84/0.87	0.95/0.98
	2	2,329	0.86/0.89	0.96/1.00	1.05/1.13	0.85/0.91	0.97/1.03
60-60	10	2,338	0.84/0.92	0.97/1.01	1.08/1.18	0.86/0.93	0.98/1.03
	5	4,739	0.85/0.92	0.98/1.02	1.12/1.20	0.87/0.92	0.98/1.04
	2	9,166	0.82/0.89	0.96/0.99	1.18/1.30	0.84/0.91	0.98/1.03
90-45	10	2,354	0.91/0.96	0.96/1.04	1.12/1.19	0.92/0.97	0.97/1.02
	5	4,755	0.87/0.92	0.96/0.98	1.11/1.14	0.89/0.93	0.97/0.99
	2	9,131	0.88/0.90	0.93/0.95	1.08/1.17	0.86/0.90	0.94/1.02

CPU utilization: By comparing the difference between the number of MIPs and the timing, we see that within a fixed amount of wall-clock time, ASG is able to solve many more MIPs than FSG, and slightly more than PSG. As for APSG, it performs somewhere in between PSG and ASG.

The differences between the methods are less pronounced when there are more scenarios, possibly because the relative amount of time in which workers are idle is significantly reduced for FSG in this situation (as discussed in Sections 2.2 and 3.3).

We now consider how varying K affects the performance of the partitioning methods. PSG-5 performs slightly better than PSG-10 in most cases for 50 scenarios. Likewise, PSG-10 is better than PSG-20 for the 200-scenario instances. This demonstrates that the additional parallelism gained from a finer partitioning can potentially be worth the slight increase in number of MIPs needed to attain the approximate solution. This also holds when we consider APSG, reinforcing the idea that partitioning can provide better parallel performance (in terms of convergence time) than asynchronicity for these instances. Note that the ratio of number of MIP instances required to reach each threshold for PSG is close to the ratio in the bound provided in Corollary 2 (see Table 1 for the numbers).

5.4.2. DCAP Results. For the DCAP instances, we note from Tables 10 and 11 that, in terms of MIPs that need to be solved, the ratio of PSG to FSG significantly exceeds what we would expect from Corollary 2 and Table 1, with the difference being most pronounced on DCAP-5-7-5 with 50 and 200 scenarios and DCAP-6-6-5 with 200 scenarios. This surprising observation may indicate that FSG is performing better than the bound in Corollary 1 would indicate.

Partitioning alone does not help in most cases. PSG is faster than FSG for higher precision for DCAP-4-6-8 and DCAP-6-6-5 at 50 scenarios for both PSG-5 and PSG-10 and for DCAP-5-7-5 at 200 scenarios for PSG-20,

Table 6. MIP Ratios for SSLP Instances: 50 Scenarios

		Minimum/maximum ratio				
Instance	% Gap	PSG ($K = 5$)	PSG ($K = 10$)	ASG	APSG ($K = 5$)	APSG ($K = 10$)
20-100	10	1.14/1.29	1.02/1.12	1.76/1.88	1.68/1.74	1.59/1.95
	5	1.15/1.24	1.01/1.11	1.92/2.21	1.62/1.81	1.62/1.79
	2	1.16/1.21	0.98/1.18	3.33/5.15	1.78/1.88	1.84/2.10
30-100	10	1.22/1.40	1.11/1.21	2.13/2.24	1.82/2.08	1.82/2.03
	5	1.13/1.29	1.09/1.12	2.11/2.25	1.62/1.79	1.74/1.83
	2	1.15/1.28	1.03/1.12	2.22/2.48	1.69/1.88	1.79/1.93
60-60	10	1.27/1.37	1.07/1.12	2.26/2.35	1.87/1.94	1.76/1.98
	5	1.14/1.25	1.03/1.07	2.06/2.31	1.68/1.85	1.77/1.91
	2	1.19/1.31	1.01/1.13	2.46/∞	1.74/2.07	1.94/2.11
90-45	10	1.23/1.31	1.07/1.18	2.00/2.20	1.80/2.13	1.80/2.00
	5	1.17/1.31	1.00/1.10	1.90/2.10	1.72/1.88	1.71/1.89
	2	1.13/1.28	1.04/1.09	2.26/2.47	1.79/1.99	1.84/2.04

Table 7. MIP Ratios for SSLP Instances: 200 Scenarios

		Minimum/maximum ratio				
Instance	% Gap	PSG ($K = 10$)	PSG ($K = 20$)	ASG	APSG ($K = 10$)	APSG ($K = 20$)
20-100	10	1.05/1.13	1.00/1.05	1.29/1.33	1.07/1.13	1.03/1.08
	5	1.06/1.12	1.01/1.05	1.31/1.36	1.07/1.12	1.03/1.11
	2	1.01/1.08	0.97/1.07	1.37/1.47	1.02/1.07	1.00/1.06
30-100	10	1.05/1.11	0.99/1.03	1.33/1.39	1.06/1.10	1.03/1.08
	5	1.08/1.14	0.98/1.03	1.33/1.41	1.08/1.10	1.06/1.07
	2	1.09/1.15	1.02/1.07	1.38/1.41	1.09/1.14	1.06/1.12
60-60	10	1.07/1.16	1.03/1.06	1.41/1.43	1.11/1.15	1.07/1.11
	5	1.11/1.16	1.05/1.10	1.45/1.49	1.12/1.17	1.08/1.14
	2	1.09/1.13	1.04/1.07	1.54/1.64	1.10/1.15	1.08/1.14
90-45	10	1.11/1.15	1.03/1.10	1.41/1.41	1.13/1.16	1.05/1.12
	5	1.09/1.14	1.06/1.07	1.41/1.43	1.11/1.16	1.08/1.11
	2	1.11/1.12	1.03/1.04	1.38/1.49	1.08/1.12	1.07/1.14

but otherwise performs the same or worse. We observe an interesting phenomenon when comparing the number of MIPs solved versus the running time. For PSG, the variability in the timings can be significantly larger than the variability in the number of MIPs solved, so much so that in certain outlying instances (for example, DCAP-7-4-7 with 50 scenarios for PSG-5 and DCAP-6-6-5 with 200 scenarios for both PSG-10 and PSG-20), the running-time ratio can be much larger than the MIP ratio. In these circumstances, the average per-MIP time for PSG is higher than for FSG, suggesting that the order in which subproblems are solved can significantly alter the difficulty of the subproblems, making the performance of partitioning schemes much less predictable.

Unlike the case for SSLP, a larger batch size K seems to help in general, even for in the majority of cases.

Although Tables 8 and 9 show that parallel FSG is generally hard to beat, the DCAP instances reveal potential for large benefits from asynchronicity and from combining asynchronicity with partitioning. In particular, APSG and ASG often significantly outperformed FSG on instances DCAP-5-7-5 (50 and 200 scenarios) and DCAP-6-6-5 (200 scenarios). The only instances where neither asynchronicity nor partitioning helped were DCAP-4-6-8 and DCAP-7-4-7 with 200 scenarios.

We see that the APSG methods often slightly outperforms ASG, showing that even if partitioning alone (as in PSG) does not improve performance, there may still be benefits to be gained by using partitioning in conjunction with asynchronicity.

5.4.3. Summary of Experiments. The results reported in Tables 4, 5, 8, and 9 show that it is usually possible to reduce wall-clock times relative to a basic parallel implementation of Algorithm 1 by using some combination of partitioning and asynchronicity. It is not obvious to predict in advance, however, which particular

Table 8. Running Time Ratios for DCAP Instances: 50 Scenarios

			Minimum/maximum ratio				
Instance	% Gap	Base (s)	PSG ($K = 5$)	PSG ($K = 10$)	ASG	APSG ($K = 5$)	APSG ($K = 10$)
4-6-8	10	159	0.91/1.15	0.78/0.94	1.58/1.70	1.54/2.22	1.21/1.49
	5	244	0.89/1.07	0.83/0.89	1.43/1.61	1.46/2.44	1.08/1.40
	2	500	0.82/0.90	0.79/1.02	1.28/1.49	1.25/2.09	1.15/1.35
5-7-5	10	565	1.12/1.56	0.89/0.97	0.68/0.78	0.66/0.76	0.62/0.66
	5	1,103	1.09/1.44	0.90/1.12	0.57/0.65	0.58/0.63	0.52/0.57
	2	2,074	1.03/1.30	0.89/1.08	0.53/0.64	0.57/0.64	0.51/0.57
6-6-5	10	212	0.86/1.17	0.80/1.07	1.09/1.21	1.08/1.33	0.97/1.08
	5	379	0.90/1.11	0.84/0.93	1.06/1.17	0.98/1.09	0.89/0.93
	2	788	0.90/1.03	0.94/1.00	1.05/1.12	0.92/0.99	0.87/0.97
7-4-7	10	325	1.10/1.72	0.97/1.01	1.58/1.87	1.45/1.81	1.46/1.53
	5	569	1.02/2.18	1.00/1.04	1.58/1.74	1.42/1.71	1.45/1.64
	2	1,100	0.96/1.77	0.90/0.99	1.66/1.90	1.36/1.62	1.17/1.46

Table 9. Running Time Ratios for DCAP Instances: 200 Scenarios

			Minimum/maximum ratio				
Instance	% Gap	Base (s)	PSG ($K = 10$)	PSG ($K = 20$)	ASG	APSG ($K = 10$)	APSG ($K = 20$)
4-6-8	10	599	1.16/1.65	1.02/1.13	1.54/1.77	1.09/1.28	1.05/1.16
	5	905	1.28/1.60	1.06/1.18	1.64/1.99	1.18/1.38	1.16/1.33
	2	1,800	1.20/1.41	1.06/1.19	1.54/1.83	1.14/1.29	1.15/1.34
5-7-5	10	1,318	0.92/1.07	0.87/0.98	0.96/1.04	0.79/0.88	0.79/0.83
	5	2,442	0.96/1.15	0.91/0.94	0.87/0.95	0.78/0.84	0.74/0.79
	2	4,748	0.95/1.25	0.89/0.96	0.84/0.90	0.76/0.82	0.67/0.73
6-6-5	10	908	1.11/1.20	1.07/1.18	1.29/1.38	1.10/1.19	1.07/1.11
	5	1,904	1.14/2.17	1.25/2.14	1.08/1.20	0.92/1.01	0.90/0.94
	2	8,484	∞/∞	2.03/∞	0.44/0.48	0.40/0.43	0.44/0.47
7-4-7	10	1,010	1.08/1.19	1.07/1.13	1.36/1.49	1.10/1.19	1.14/1.19
	5	1,672	1.11/1.22	1.05/1.13	1.36/1.52	1.14/1.20	1.11/1.20
	2	3,442	1.13/1.24	1.04/1.10	1.46/1.61	1.13/1.25	1.07/1.20

combination and which choices of parameters will work best. The distribution of scenario solve times has some predictive value with regard to the benefits of asynchronicity, as we explain below. Additionally, the effect of partitioning can be predicted well from the convergence results on some families of instances, such as SSLP.

To optimize overall performance on a particular problem and computational platform, each of the proposed approaches (including possibly multiple instances of the same approach with different choices of partition parameter K) can be run for a fixed amount of wall-clock time, and the one that makes the best progress could be adopted to complete the solution of the optimization problem. A more sophisticated and systematic approach would be to treat the meta-problem of algorithm selection as a multiarmed bandit problem (Bubeck and Cesa-Bianchi 2002), deploying a limited computational budget to search for the best combination of algorithmic strategies and parameter K and then selecting the best found approach to continue maximizing the Lagrangian dual.

A potential explanation for when asynchronicity helps comes from variability in the scenario solve times; see Table 3. Among DCAP instances, this variability is correlated with the benefit obtained from using the asynchronous methods ASG and APSG. For 50 scenarios, the highest coefficients of variation are for DCAP-5-7-5 and DCAP-6-6-5, where the asynchronous methods perform much better than in the other two DCAP instances. Similar observations can be made for the 200-scenario instances, except that DCAP-7-4-7 performs better than DCAP-4-6-8, despite having a lower CV. For SSLP, Table 3 shows relatively low CV, and asynchronicity alone always hurts the performance on these instances. We conclude that the CV (and its standing relative to other instances in the same family) provides a reasonable first indicator of whether asynchronicity may help.

Table 10. MIP Ratios for DCAP Instances: 50 Scenarios

		Minimum/maximum ratio				
Instance	% Gap	PSG ($K = 5$)	PSG ($K = 10$)	ASG	APSG ($K = 5$)	APSG ($K = 10$)
4-6-8	10	1.38/1.56	1.21/1.26	2.46/2.66	2.42/3.39	2.07/2.55
	5	1.32/1.46	1.14/1.22	2.23/2.54	2.26/3.67	1.78/2.25
	2	1.15/1.23	0.97/1.20	1.94/2.36	1.91/3.11	1.84/2.11
5-7-5	10	1.43/1.52	1.24/1.30	2.12/2.38	2.03/2.30	2.02/2.11
	5	1.42/1.56	1.23/1.29	2.04/2.28	2.02/2.22	1.91/2.06
	2	1.45/1.58	1.28/1.31	1.90/2.25	1.96/2.20	1.89/2.08
6-6-5	10	1.29/1.35	1.14/1.21	1.87/2.03	1.78/2.15	1.72/1.88
	5	1.28/1.36	1.08/1.13	2.03/2.17	1.82/1.96	1.65/1.77
	2	1.24/1.31	1.12/1.21	2.17/2.34	1.86/1.98	1.76/1.96
7-4-7	10	1.30/1.37	1.12/1.17	2.03/2.14	1.77/2.02	1.77/1.87
	5	1.23/1.31	1.18/1.23	2.10/2.23	1.81/2.01	1.81/2.09
	2	1.21/1.31	1.10/1.17	2.38/2.64	1.85/2.05	1.66/2.09

Table 11. MIP Ratios for DCAP Instances: 200 Scenarios

		Minimum/maximum ratio				
Instance	% Gap	PSG ($K = 10$)	PSG ($K = 20$)	ASG	APSG ($K = 10$)	APSG ($K = 20$)
4-6-8	10	1.32/1.48	1.22/1.30	1.86/2.09	1.32/1.45	1.32/1.45
	5	1.34/1.54	1.23/1.37	1.90/2.28	1.38/1.51	1.39/1.59
	2	1.30/1.44	1.20/1.35	1.76/2.07	1.29/1.40	1.33/1.53
5-7-5	10	1.26/1.32	1.22/1.26	1.50/1.66	1.27/1.32	1.27/1.35
	5	1.32/1.41	1.31/1.32	1.52/1.67	1.38/1.42	1.35/1.44
	2	1.32/1.49	1.30/1.34	1.58/1.77	1.39/1.49	1.31/1.43
6-6-5	10	1.26/1.31	1.21/1.26	1.54/1.63	1.27/1.34	1.26/1.30
	5	1.28/1.34	1.23/1.29	1.59/1.67	1.32/1.34	1.31/1.35
	2	∞/∞	$1.21/\infty$	1.50/1.60	1.32/1.35	1.30/1.36
7-4-7	10	1.15/1.19	1.14/1.16	1.47/1.52	1.16/1.22	1.17/1.26
	5	1.16/1.22	1.12/1.18	1.51/1.58	1.18/1.21	1.17/1.29
	2	1.17/1.22	1.11/1.17	1.60/1.72	1.17/1.22	1.15/1.29

We leave it to future work to determine which statistical measure better captures the potential benefit of asynchronicity across a variety of instances. Given the above correlation between the benefits of asynchronicity and the variability in subproblem solve time, we believe that the asynchronous algorithms will see a larger benefit in heterogeneous computing environments with a high degree of variability in worker performance, such as over a large-scale cloud computing platform.

6. Conclusions

The standard projected subgradient algorithm for solving the Lagrangian dual of an SMIP can be implemented in parallel by distributing the MIP subproblem for different scenarios among the available processors. This naive approach can be effective, but bottlenecks can occur that result in underutilization of the processors, particularly when there is wide variation among the time required to solve the MIPs for different scenarios.

We have proposed and analyzed two modifications to the projected subgradient algorithm that potentially make better use of parallel computing resources. The first modification (partitioned subgradient) breaks the subgradient step into a sequence of steps based on smaller groups of scenarios, thus limiting the need to wait for all scenario subproblems to complete before making progress. The second modification (asynchronous) allows the use of some old subproblem information to take steps, rather than waiting for all scenarios to be evaluated at the most recent iterate. The two approaches can be combined in a straightforward way.

Our convergence-rate analysis provides insight into the price paid by these methods (compared with the standard subgradient method) in terms of the number of subproblems that must be solved to achieve a given accuracy. We can expect these modifications to outperform the projected subgradient method when the gains from better utilization of the parallel resources outweigh the degradation in convergence rate.

Our computational experiments indicate that, for one problem class, the partitioned subgradient method provides significant and consistent reduction in wall-clock time, whereas for a second problem class, the asynchronous variations provide significant wall-clock time reduction when the time required for solving the MIP subproblems varies widely between scenarios. Although no one approach to modifying the subgradient method for parallel execution is best in all circumstances, we have described a small suite of algorithmic variants for making the best use of parallel resources. Our partitioned methods work best when the number of workers is large relative to the number of scenarios. This can potentially make a large difference in large-scale distributed applications, and we leave this evaluation to future work. Techniques from machine learning can be applied to determine which of these variants is most appropriate on a given instance by performing partial computations with each.

Appendix A. Applying the Partitioning Technique to Other Subgradient Methods

Convergence of the batching approach described in the subsections above relies on the fact that the update directions $\hat{\mathbf{g}}_s^k$ (18) are stochastic subgradients (Proposition 1). It is, therefore, natural to ask if one can use $\hat{\mathbf{g}}_s^k$ in other algorithms that rely on stochastic subgradients and if the resulting per-step updates in these methods are still efficiently parallelizable. We briefly show here that this is true for the specific cases of the dual averaging method, and also for multiple averaging extensions of dual averaging. We leave a detailed evaluation of these methods to future work.

The *dual averaging algorithm* was introduced by Nesterov (2009) in response to the counterintuitive notion that in the traditional subgradient method, subgradients calculated more recently are weighted less than older subgradients in

calculating iterates. The approach has been extended to regularized optimization problems (Xiao 2010) and distributed variants thereof (Duchi et al. 2012). In our notation, each iteration of the (stochastic variant of) dual averaging is performed as follows:

1. Compute stochastic subgradient $\hat{\mathbf{g}}^k$.
2. $G^k \leftarrow G^{k-1} + \alpha_k \hat{\mathbf{g}}^k$, for some $\alpha_k > 0$;
3. $\boldsymbol{\lambda}^k \leftarrow \arg\min_{\lambda \in C}\{-\boldsymbol{\lambda}^\top G^k + \beta_k d(\boldsymbol{\lambda})\}$, for some $\beta_k > 0$.

Here, α_k, β_k are parameters that are chosen a priori to guarantee convergence, and d is a strongly convex *prox-function*. Observe that if $C = \mathbb{R}^n$ and $d(\boldsymbol{\lambda}) := (\rho/2)\|\boldsymbol{\lambda}\|_2^2$ for some constant $\rho > 0$, then $\boldsymbol{\lambda}^k = G^k/(\rho\beta_k)$.

We can use the stochastic subgradients $\hat{\mathbf{g}}^k$ from (18) in dual averaging in an efficient parallel manner. The definition of $\hat{\mathbf{g}}^k$ ensures that $\sum_{s\in S} \hat{\mathbf{g}}_s^k = 0$, which, in turn, implies that $\sum_{s\in S} G_s^k = 0$. Hence, setting $d(\boldsymbol{\lambda}) = (\rho/2)\|\boldsymbol{\lambda}\|_2^2$ provides a cheap and efficient way to satisfy the constraint $\sum_{s\in S} \boldsymbol{\lambda}_s^k = 0$. Furthermore, because each $\boldsymbol{\lambda}_s^k$ does not depend on any $\hat{\mathbf{g}}_{s'}^k$, where s and s' are not in the same batch, we can parallelize the algorithm in a fashion similar to Algorithm 3. Because $\sum_{s\in S} \boldsymbol{\lambda}_s^k = 0$, the lower bound from iteration k can be computed once all scenarios in an iteration k have been evaluated.

A similar principle applies for the multiple averaging methods introduced in Nesterov and Shikhman (2015), which guarantee that the $\boldsymbol{\lambda}^k$ values (and not just their average) converge to the optimal solution. One variant, double averaging, defines $\boldsymbol{\lambda}^{k+1}$ to be a convex combination of $\boldsymbol{\lambda}^k$ and $\arg\min_{\lambda\in C}\{-\boldsymbol{\lambda}^\top G^k + \beta_k d(\boldsymbol{\lambda})\}$, while triple averaging forms a convex combination of these two vectors along with the initial iterate $\boldsymbol{\lambda}^0$. Using similar arguments, we can see that if we use $d(\boldsymbol{\lambda}) = (\rho/2)\|\boldsymbol{\lambda}\|_2^2$, we can perform the updates in parallel in the manner described above.

Appendix B. Proofs from Section 3

Proof of Proposition 1. *We derive the expectation by a sequence of equalities; the less obvious ones are explained below. For a fixed $s \in S$, we have*

$$\mathbb{E}_{\mathcal{P}^k}\hat{\mathbf{g}}_s^k = \sum_{i=1}^{N/K} \mathbb{P}\big(s \in T_i^k\big)\mathbb{E}_{\mathcal{P}^k}\Big[\hat{\mathbf{g}}_s^k \,|\, s \in T_i^k\Big], \tag{B.1(a)}$$

$$= \frac{K}{N}\sum_{i=1}^{N/K} \mathbb{E}_{\mathcal{P}^k}\Big[\hat{\mathbf{x}}_s^k \,|\, s \in T_i^k\Big], \tag{B.1(b)}$$

$$= \frac{K}{N}\sum_{i=1}^{N/K} \mathbb{E}_{\mathcal{P}^k}\left[\mathbf{x}_s^k - \frac{1}{K}\sum_{j\in T_i^k}\mathbf{x}_j^k \,\middle|\, s \in T_i^k\right], \tag{B.1(c)}$$

$$= \frac{K}{N}\sum_{i=1}^{N/K} \mathbb{E}_{\mathcal{P}^k}\left[\mathbf{x}_s^k\left(1-\frac{1}{K}\right) - \frac{1}{K}\sum_{j\in T_i^k\setminus s}\mathbf{x}_j^k \,\middle|\, s \in T_i^k\right], \tag{B.1(d)}$$

$$= \frac{K}{N}\frac{N}{K}\left[\mathbf{x}_s^k\left(1-\frac{1}{K}\right) - \frac{1}{K}\frac{K-1}{N-1}\sum_{j\in S\setminus s}\mathbf{x}_j^k\right], \tag{B.1(e)}$$

$$= \left[\mathbf{x}_s^k\left(1-\frac{1}{K}\right) - \frac{1}{K}\frac{K-1}{N-1}\Big[N\mathbf{z}^k - \mathbf{x}_s^k\Big]\right], \tag{B.1(f)}$$

$$= \frac{N(K-1)}{K(N-1)}\Big[\mathbf{x}_s^k - \mathbf{z}^k\Big]. \tag{B.1(g)}$$

Here, (B.1(b)) follows from $\mathbb{P}(s \in T_i^k) = K/N$ for all i, because all N/K partitions have the same number of scenarios. To obtain (B.1(e)), we note that each of the summations over $T_i^k \setminus s$ contains exactly $K-1$ terms, and that because the partitioning is done independently and uniformly at each iteration, all terms $\mathbf{x}_j^k$ for $j \in S \setminus s$ are equally represented when we take the expectation over the partition. We use the definition of $\mathbf{z}^k$ in (9) for (B.1(f)), while (B.1(g)) is obtained from arithmetic manipulation. □

Proof of Lemma 2. *We first show that*

$$\big(\hat{\mathbf{g}}^k\big)^\top\big(\hat{\mathbf{g}}^k - \mathbf{g}^k\big) = 0. \tag{B.2}$$

The argument is as follows:

$$\begin{aligned}
\big(\hat{\mathbf{g}}^k\big)^\top\big(\hat{\mathbf{g}}^k - \mathbf{g}^k\big) &= \sum_{i=1}^{N/K}\sum_{s\in T_i^k}\Big(\mathbf{x}_s^k - \mathbf{z}_{T_i^k}^k\Big)^\top\Big(\Big(\mathbf{x}_s^k - \mathbf{z}_{T_i^k}^k\Big) - \big(\mathbf{x}_s^k - \mathbf{z}^k\big)\Big)\\
&= \sum_{i=1}^{N/K}\sum_{s\in T_i^k}\Big(\mathbf{x}_s^k - \mathbf{z}_{T_i^k}^k\Big)^\top\Big(\mathbf{z}^k - \mathbf{z}_{T_i^k}^k\Big)\\
&= \sum_{i=1}^{N/K}\left(\sum_{s\in T_i^k}\mathbf{x}_s^k - K\mathbf{z}_{T_i^k}^k\right)^\top\Big(\mathbf{z}^k - \mathbf{z}_{T_i^k}^k\Big) = 0,
\end{aligned}$$

where third equality follows because $|T_i^k| = K$ and the last equality follows from the definition of $z^k_{T_i^k}$. Thus, using (B.2) yields

$$\mathbb{E}_{\mathcal{P}^k}\|\hat{\mathbf{g}}^k\|^2 = \mathbb{E}_{\mathcal{P}^k}(\hat{\mathbf{g}}^k)^\top \mathbf{g}^k = \frac{N(K-1)}{K(N-1)}\|\mathbf{g}^k\|^2 \le \frac{N(K-1)}{K(N-1)}NM^2,$$

where the second equality follows from Proposition 1 and we used the bound (13) for the final inequality. □

Proof of Theorem 2. *Defining* $\boldsymbol{\xi}^k = [\boldsymbol{\xi}^k_s]_{s\in S}$ *by*

$$\boldsymbol{\xi}^k_s := \frac{K(N-1)}{N(K-1)}\hat{\mathbf{g}}^k_s - \mathbf{g}^k_s, \tag{B.3}$$

we have from Proposition 1 that $\mathbb{E}_{\mathcal{P}^k}\boldsymbol{\xi}^k = 0$. Expanding the iteration formula in Algorithm 2, and using (B.3), we have

$$\begin{aligned}
\frac{1}{2}\|\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^*\|^2 &= \frac{1}{2}\left\|\boldsymbol{\lambda}^k + \alpha_k\frac{K(N-1)}{N(K-1)}\hat{\mathbf{g}}^k - \boldsymbol{\lambda}^*\right\|^2 \\
&= \frac{1}{2}\|\boldsymbol{\lambda}^k - \boldsymbol{\lambda}^*\|^2 + \alpha_k\langle \mathbf{g}^k, \boldsymbol{\lambda}^k - \boldsymbol{\lambda}^*\rangle + \frac{1}{2}\alpha_k^2\left(\frac{K(N-1)}{N(K-1)}\right)^2\|\hat{\mathbf{g}}^k\|^2 + \alpha_k\langle \boldsymbol{\xi}^k, \boldsymbol{\lambda}^k - \boldsymbol{\lambda}^*\rangle \\
&= \frac{1}{2}\|\boldsymbol{\lambda}^k - \boldsymbol{\lambda}^*\|^2 + \alpha_k\big(\mathscr{L}(\boldsymbol{\lambda}^k) - \mathscr{L}(\boldsymbol{\lambda}^*)\big) + \frac{1}{2}\alpha_k^2\left(\frac{K(N-1)}{N(K-1)}\right)^2\|\hat{\mathbf{g}}^k\|^2 + \alpha_k\langle \boldsymbol{\xi}^k, \boldsymbol{\lambda}^k - \boldsymbol{\lambda}^*\rangle,
\end{aligned}$$

where the last step follows from (12). By taking expectations of both sides over $\mathcal{P}^k$, using Lemma 2 and $\mathbb{E}_{\mathcal{P}^k}\boldsymbol{\xi}^k = 0$, and noting that $\boldsymbol{\lambda}^k$ does not depend on $\mathcal{P}^k$, we have

$$\frac{1}{2}\mathbb{E}_{\mathcal{P}^k}\|\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^*\|^2 \le \frac{1}{2}\|\boldsymbol{\lambda}^k - \boldsymbol{\lambda}^*\|^2 + \alpha_k\big(\mathscr{L}(\boldsymbol{\lambda}^k) - \mathscr{L}(\boldsymbol{\lambda}^*)\big) + \frac{1}{2}\alpha_k^2\frac{K(N-1)}{N(K-1)}NM^2.$$

By taking expectations over the partitions $\mathcal{P}_l$ at *all* iterations and rearranging, we obtain

$$\alpha_k\mathbb{E}\big(\mathscr{L}(\boldsymbol{\lambda}^*) - \mathscr{L}(\boldsymbol{\lambda}^k)\big) \le \frac{1}{2}\mathbb{E}\|\boldsymbol{\lambda}^k - \boldsymbol{\lambda}^*\|^2 - \frac{1}{2}\mathbb{E}\|\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^*\|^2 + \frac{1}{2}\alpha_k^2\frac{K(N-1)}{N(K-1)}NM^2.$$

We obtain the result by summing both sides of this expression over $k = 1, 2, \ldots, L$, using the fact that $\boldsymbol{\lambda}^1$ does not depend on any of the partitions (so the expectation can be omitted for this term), and using $\mathbb{E}\|\boldsymbol{\lambda}^{L+1} - \boldsymbol{\lambda}^*\|^2 \ge 0$. □

Appendix C. Proofs from Section 4

Proof of Theorem 3. *Using the usual expansion, and using the facts that* $\boldsymbol{\lambda}^k \in C$, $\boldsymbol{\lambda}^* \in C$, *the projection operation* $P_C(\cdot)$ *is a contraction, and* $\boldsymbol{\lambda}^k + \alpha_k(\hat{\mathbf{x}}^k - \hat{z}^k) = P_C(\boldsymbol{\lambda}^k + \alpha_k\hat{\mathbf{x}}^k)$, *we obtain*

$$\begin{aligned}
\frac{1}{2}\|\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^*\|^2 &= \frac{1}{2}\left\|\left[\lambda^k_s + \alpha_k\big(\hat{x}^k_s - \hat{z}^k\big) - \lambda^*_s\right]_{s\in S}\right\|^2 \\
&= \frac{1}{2}\left\|P_C\big(\boldsymbol{\lambda}^k + \alpha_k\hat{\mathbf{x}}^k\big) - \boldsymbol{\lambda}^*\right\|^2 \\
&\le \frac{1}{2}\left\|\boldsymbol{\lambda}^k + \alpha_k\hat{\mathbf{x}}^k - \boldsymbol{\lambda}^*\right\|^2 \\
&= \frac{1}{2}\|\boldsymbol{\lambda}^k - \boldsymbol{\lambda}^*\|^2 + \frac{1}{2}\alpha_k^2\|\hat{\mathbf{x}}^k\|^2 + \alpha_k\big\langle\hat{\mathbf{x}}^k, \boldsymbol{\lambda}^k - \boldsymbol{\lambda}^*\big\rangle \\
&\le \frac{1}{2}\|\boldsymbol{\lambda}^k - \boldsymbol{\lambda}^*\|^2 + \frac{1}{2}\alpha_k^2\|\hat{\mathbf{x}}^k\|^2 + \alpha_k\big\langle\hat{\mathbf{x}}^k, \hat{\boldsymbol{\lambda}}^k - \boldsymbol{\lambda}^*\big\rangle + \alpha_k\big\langle\hat{\mathbf{x}}^k, \boldsymbol{\lambda}^k - \hat{\boldsymbol{\lambda}}^k\big\rangle.
\end{aligned}$$

By rearranging this inequality, we obtain

$$\alpha_k\big\langle\hat{\mathbf{x}}^k, \boldsymbol{\lambda}^* - \hat{\boldsymbol{\lambda}}^k\big\rangle \le \frac{1}{2}\big\|\boldsymbol{\lambda}^k - \boldsymbol{\lambda}^*\big\|^2 - \frac{1}{2}\big\|\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^*\big\|^2 + \frac{1}{2}\alpha_k^2\big\|\hat{\mathbf{x}}^k\big\|^2 + \alpha_k\big\langle\hat{\mathbf{x}}^k, \boldsymbol{\lambda}^k - \hat{\boldsymbol{\lambda}}^k\big\rangle. \tag{B.4}$$

From (20), we have

$$\mathscr{L}(\boldsymbol{\lambda}^*) - \mathscr{L}(\hat{\boldsymbol{\lambda}}^k) \le \big\langle\hat{\mathbf{x}}^k, \boldsymbol{\lambda}^* - \hat{\boldsymbol{\lambda}}^k\big\rangle.$$

By summing (B.4) from $k = 1$ to $k = L$, and using $\|\lambda^{L+1} - \lambda^*\| \geq 0$, we have

$$\sum_{k=1}^{L} \alpha_k \left[\mathcal{L}(\lambda^*) - \mathcal{L}(\hat{\lambda}^k)\right] \leq \frac{1}{2}\|\lambda^1 - \lambda^*\|^2 + \frac{1}{2}\sum_{k=1}^{L} \alpha_k^2 \|\hat{x}^k\|^2 + \sum_{k=1}^{L} \alpha_k \langle \hat{x}^k, \lambda^k - \hat{\lambda}^k \rangle.$$

As before, we use concavity of $\mathcal{L}$ and the definition of $\bar{\lambda}^L$ to replace the left-hand side, obtaining

$$\begin{aligned} \mathcal{L}(\lambda^*) - \mathcal{L}(\bar{\lambda}^L) &\leq \frac{\|\lambda^1 - \lambda^*\|^2 + \sum_{k=1}^{L} \alpha_k^2 \|\hat{x}^k\|^2 + \sum_{k=1}^{L} \alpha_k \langle \hat{x}^k, \lambda^k - \hat{\lambda}^k \rangle}{2\sum_{k=1}^{L} \alpha_k}. \\ &\leq \frac{\|\lambda^1 - \lambda^*\|^2 + NM^2 \sum_{k=1}^{L} \alpha_k^2}{2\sum_{k=1}^{L} \alpha_k} + \frac{\sum_{k=1}^{L} \alpha_k \langle \hat{x}^k, \lambda^k - \hat{\lambda}^k \rangle}{2\sum_{k=1}^{L} \alpha_k}, \end{aligned} \tag{B.5}$$

where in the second inequality, we use the bound $\|\hat{x}^k\|^2 = \sum_{s \in S} \|\hat{x}_s^k\|^2 \leq NM^2$. To bound the numerator in the final term, note first that

$$\lambda_s^k - \hat{\lambda}_s^k = \lambda_s^k - \lambda_s^{\tau_s^k} = \sum_{i=\tau_s^k}^{k-1} \alpha_i (\hat{x}_s - \hat{z}^i).$$

Thus, by the assumed bound on $\|x_s\|$ and the definition of z^i, we have $\|\hat{x}_s^i\| \leq M$ and $\|\hat{z}^i\| \leq M$, so

$$\begin{aligned} \left|\langle \hat{x}_s^k, \lambda_s^k - \hat{\lambda}_s^k \rangle\right| &\leq \|\hat{x}_s^k\| \left\| \sum_{i=\tau_s^k}^{k-1} \alpha_i (\hat{x}_s^i - \hat{z}^i) \right\| \\ &\leq M \sum_{i=\tau_s^k}^{k-1} \alpha_i (\|\hat{x}_s^i\| + \|\hat{z}^i\|) \\ &\leq 2M^2 \sum_{i=\tau_s^k}^{k-1} \alpha_i. \end{aligned}$$

Substitution into (B.5) completes the proof of the first claim.

We now prove the second claim. Because $\lambda^k \in C$ for all $k = 1, 2, \ldots, L$, we have that $\tilde{\lambda}^L \in C$ because $\tilde{\lambda}_L$ is their weighted average, so

$$\text{dist}\,(\bar{\lambda}^L, C) \leq \|\bar{\lambda}^L - \tilde{\lambda}^L\| = \frac{\left\|\sum_{k=1}^{L} \alpha_k (\hat{\lambda}^k - \lambda^k)\right\|}{\sum_{k=1}^{L} \alpha_k} \leq \frac{\sum_{k=1}^{L} \alpha_k \left\|\hat{\lambda}^k - \lambda^k\right\|}{\sum_{k=1}^{L} \alpha_k}. \tag{B.6}$$

As in the analysis above, we have

$$\left\|\lambda_s^k - \hat{\lambda}_s^k\right\| = \left\| \sum_{i=\tau_s^k}^{k-1} \alpha_i (\hat{x}_s^i - \hat{z}^i) \right\| \leq \sum_{i=\tau_s^k}^{k-1} \alpha_i \|\hat{x}_s^i - \hat{z}^i\| \leq 2M \sum_{i=\tau_s^k}^{k-1} \alpha_i.$$

By substituting into (B.6), we obtain the desired inequality. □

Endnotes

[1] The algorithm can, in theory, be adjusted to handle batches of different sizes.

[2] These instances are available at https://limconghan.github.io/smip/.

References

Ahmed S, Garcia R (2003) Dynamic capacity acquisition and assignment under uncertainty. *Ann. Oper. Res.* 124(1–4):267–283.

Ahmed S, Garcia R, Kong N, Ntaimo L, Qiu F, Sen S (2015) SIPLIB: A stochastic integer programming test problem library. Accessed November 17, 2019, https://www2.isye.gatech.edu/~sahmed/siplib/.

Aravena I, Papavasiliou A (2015) A distributed asynchronous algorithm for the two-stage stochastic unit commitment problem. *2015 IEEE Power Energy Society General Meeting* (IEEE, New York), 1–5.

Bertsekas DP (1999) *Nonlinear Programming*, 2nd ed. (Athena Scientific, Belmont, MA).

Bodur M, Dash S, Günlük O, Luedtke J (2016) Strengthened Benders cuts for stochastic integer programs with continuous recourse. *INFORMS J. Comput.* 29:77–91.

Boland N, Bakir I, Dandurand B, Erera A (2016) Scenario set partition dual bounds for multistage stochastic programming: A hierarchy of bounds and a partition sampling approach. Preprint, submitted January 28, http://www.optimization-online.org/DB_HTML/2016/01/5311.html.

Bubeck S, Cesa-Bianchi N (2002) Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations Trends Machine Learn.* 5(1):1–122.

Carøe CC (1998) Decomposition in stochastic integer programming. Unpublished doctoral thesis, Department of Operations Research, University of Copenhagen, Copenhagen, Denmark.

Carøe CC, Schultz R (1999) Dual decomposition in stochastic integer programming. *Oper. Res. Lett.* 24(1–2):37–45.

Conforti M, Cornuéjols G, Zambelli G (2014) *Integer Programming*, *Graduate Texts in Mathematics*, vol. 271 (Springer, Cham, Switzerland).

Dey SS, Molinaro M, Wang Q (2018) Analysis of sparse cutting planes for sparse MILPS with applications to stochastic MILPS. *Math. Oper. Res.* 43(1):304–332.

Duchi JC, Agarwal A, Wainwright MJ (2012) Dual averaging for distributed optimization: Convergence analysis and network scaling. *IEEE Trans. Automatic Control* 57(3):592–606.

Ermoliev YM (1966) Methods of solution of nonlinear extremal problems. *Cybernetics* 2(4):1–14.

Kelley JJ (1960) The cutting-plane method for solving convex programs. *J. Soc. Indust. Appl. Math.* 8(4):703–712.

Kim K, Zavala VM (2017) Algorithmic innovations and software for the dual decomposition method applied to stochastic mixed-integer programs. *Math. Program. Comput.* 10:225–266.

Kim K, Petra CG, Zavala VM (2017) An asynchronous bundle-trust-region method for dual decomposition of stochastic mixed-integer programming. Technical Report ANL/MCS-8046-0917, Argonne National Laboratory, Lemont, IL.

Lemarechal C (1978) Nonsmooth optimization and descent methods. IIASA Research Report RR-78-004, International Institute for Applied Systems Analysis, Laxenburg, Austria.

Lubin M, Martin K, Petra CG, Sandikci B (2013) On parallelizing dual decomposition in stochastic integer programming. *Oper. Res. Lett.* 41(3):252–258.

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

Maggioni F, Pflug GC (2016) Bounds and approximations for multistage stochastic programs. *SIAM J. Optim.* 26(1):831–855.

Maggioni F, Allevi E, Bertocchi M (2016) Monotonic bounds in multistage mixed-integer stochastic programming. *Comput. Management Sci.* 13(3):423–457.

Necoara I, Nesterov Y, Glineur F (2017) Random block coordinate descent methods for linearly constrained optimization over networks. *J. Optim. Theory Appl.* 173(1):227–254.

Nesterov Y (2009) Primal-dual subgradient methods for convex problems. *Math. Programming* 120(1):221–259.

Nesterov Y, Shikhman V (2015) Quasi-monotone subgradient methods for nonsmooth convex minimization. *J. Optim. Theory Appl.* 165(3):917–940.

Ntaimo L, Sen S (2005) The million-variable 'march' for stochastic combinatorial optimization. *J. Global Optim.* 32(3):385–400.

Rahmanai R, Ahmed S, Crainic T, Gendreau M, Rei W (2018) The Benders dual decomposition method. Technical Report CIRRELT-2018-03, Centre Interuniversitaire de Recherche sur les Réseaux d'Entreprise, la Logistique et le Transport, Montreal.

Ruszczyński A (2006) *Nonlinear Optimization* (Princeton University Press, Princeton, NJ).

Ryan K, Ahmed S, Dey SS, Rajan D (2016) Optimization driven scenario grouping. Preprint, submitted March 10, http://www.optimization-online.org/DB_HTML/2016/03/5366.html.

Sandikçi B, Ozaltin OY (2017) A scalable bounding method for multistage stochastic programs. *SIAM J. Optim.* 27(3):1772–1800.

Shor N (1985) *Minimization Methods for Non-Differentiable Functions* (Springer-Verlag, Berlin).

Xiao L (2010) Dual averaging methods for regularized stochastic learning and online optimization. *J. Machine Learn. Res.* 11(88):2543–2596.