SIAM J. OPTIM.
Vol. 29, No. 1, pp. 318–342

AN ASYNCHRONOUS BUNDLE-TRUST-REGION METHOD FOR DUAL DECOMPOSITION OF STOCHASTIC MIXED-INTEGER PROGRAMMING*

KIBAEK KIM†, COSMIN G. PETRA‡, AND VICTOR M. ZAVALA§

Abstract. We present an asynchronous bundle-trust-region algorithm within the context of Lagrangian dual decomposition for stochastic mixed-integer programs. The approach solves the Lagrangian master problem by using a bundle method with a trust-region constraint. This scheme enables asynchronous computations and can thus help mitigate severe load imbalance issues (associated with the solution of scenario subproblems) and improve parallel efficiency. We provide a convergence analysis and an implementation of the proposed scheme. We also present extensive numerical results on eighty instances of a large-scale stochastic unit commitment problem, and demonstrate that the proposed approach provides significant reductions in solution time and achieves strong scaling.

Key words. stochastic integer programming, bundle methods, asynchronous, parallel computing

AMS subject classifications. 49M27, 65K05, 68W10, 90C10, 90C15

DOI. 10.1137/17M1148189

1. Introduction.

We consider the Lagrangian dual decomposition (DD) of two-stage stochastic mixed-integer programs (SMIPs) of the form

$$\min_{x_j, y_j} \quad \sum_{j=1}^{N} p_j \left(c^T x_j + q_j^T y_j \right) \tag{1a}$$

$$\text{s.t.} \quad \sum_{j=1}^{N} H_j x_j = 0, \tag{1b}$$

$$(x_j, y_j) \in G_j, \quad j \in J. \tag{1c}$$

Here, $x_j \in \mathbb{R}^{n_1}$ and $y_j \in \mathbb{R}^{n_2}$ are decision variables associated to scenario $j \in J := \{1, \ldots, N\}$. The matrices H_j are such that (1b) represent the *nonanticipativity* constraints $x_1 = x_2 = \cdots = x_N$. We define the feasible sets

$$G_j := \{(x, y) : Ax \geq b,\ T_j x + W_j y \geq h_j,\ x \in X,\ y \in Y\}, \quad j \in J,$$

and assume that these sets are bounded and nonempty. We also define mixed-integer sets for x_j and y_j of the form $X := \mathbb{R}^{n_1 - p_1} \times \mathbb{Z}^{p_1}$ and $Y := \mathbb{R}^{n_2 - p_2} \times \mathbb{Z}^{p_2}$, respectively.

*Received by the editors September 25, 2017; accepted for publication (in revised form) October 8, 2018; published electronically January 24, 2019.

http://www.siam.org/journals/siopt/29-1/M114818.html

Funding: This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357. The work of the second author was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. The work of the third author was supported by the U.S. Department of Energy under grant DE-SC0014114.

†Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL 60439 (kimk@anl.gov).

‡Lawrence Livermore National Laboratory, Livermore, CA 94550 (petra1@llnl.gov).

§Department of Chemical and Biological Engineering, University of Wisconsin-Madison, Madison, WI 53706 (victor.zavala@wisc.edu).

The DD of (1) is obtained by applying a Lagrangian relaxation of the nonanticipativity constraints (1b). The Lagrangian dual function is given by

$$D(\lambda) := \min_{x_j, y_j} \left\{ \sum_{j=1}^{N} L_j(x_j, y_j, \lambda) : (x_j, y_j) \in G_j,\ j \in J \right\}, \tag{2}$$

where $L_j(x_j, y_j, \lambda) := p_j \left(c^T x_j + q_j^T y_j\right) + \lambda^T H_j x_j$ and λ are the dual variables of the nonanticipativity constraints. The evaluation $D(\lambda)$ involves the solution of $|J|$ decoupled scenario subproblems. Consequently, the Lagrangian dual problem can be written as

$$z_{LD} := \max_{\lambda} \sum_{j=1}^{N} D_j(\lambda), \tag{3}$$

where

$$D_j(\lambda) := \min_{x_j, y_j} \{L_j(x_j, y_j, \lambda) : (x_j, y_j) \in G_j\}. \tag{4}$$

We use λ^* to denote an optimal dual variable (i.e., satisfying $D(\lambda^*) = z_{LD}$).

We recall that the function $D_j(\lambda)$ is a piecewise concave function of λ. Moreover, $D_j(\cdot)$ is composed of a finite number of segments, which is no more than the number of extreme points of $conv(G_j)$ (the convex hull of G_j).

DD methods for SMIP problems have been widely studied in the literature (e.g., [5, 1, 14]). These methods offer the ability to address integer recourse variables (i.e., integer variables in the second stage and beyond) and the opportunity to evaluate the dual function by solving parallel subproblems. Because each scenario subproblem of the decomposition method is a mixed-integer program, however, parallel computations may suffer from significant load imbalance and decreased parallel efficiency.

A few studies reported in the literature have applied asynchronous schemes for DD. Ryan, Rajan, and Ahmed [23] extended a (synchronous) scenario decomposition method proposed in [1] to conduct asynchronous computation of scenario subproblems. This method eliminates discrete feasible solutions sequentially instead of finding the best dual variable. Aravena and Papavasiliou [2] use an incremental subgradient search that updates dual variables based only on a subset of the scenario subproblem functions. A limitation of this approach is that subgradient methods rely on an appropriate choice of the step size, which is not straightforward to determine in practice.

Incremental methods that enable asynchronous computations have been widely studied in the context of large-scale convex optimization. The proposed methods are based mainly on subgradient-based search strategies [19, 15]. An asynchronous subgradient variant was proposed and analyzed in [20]. Bertsekas further extended the incremental subgradient method to a proximal variant [3] and an augmented Lagrangian variant [4]. Gaudioso, Giallombardo, and Miglionico [10] studied the case of minimizing a function defined as the pointwise maximum over a number of convex functions, where only a subset of component functions are evaluated. As pointed out in [7], the method in [10] cannot be applied to sums of functions (as needed in the context of DD). Emiel and Sagastizábal [7] consider the case of minimizing the sum of convex functions with the focus on inexact function evaluations [16]. In a recent work [24], an incremental bundle method was proposed to use lower and upper

models for updating the stability center. This work also considers inexact function evaluations for the bundle model. Numerical performance for the schemes proposed in [10, 7, 24] was limited. Furthermore, none of these works address issues associated with parallel implementation and performance issues (such as load imbalancing).

An asynchronous implementation of a proximal bundle method is provided in [8], where bundle solutions are *implicitly* regularized in the objective function. Fischer and Helmberg [8] present a proximal bundle framework that can asynchronously choose and solve the subspace of a general convex function. Unfortunately, this framework suffers from important scalability limitations. In particular, the method stores and frequently accesses global data which must be distributed for large-scale problems. At every iteration, a significant amount of communication overhead would occur. As a result, computational performance and efficiency of this method remain unanswered [8].

Trust-region constraints provide an *explicit* regularization mechanism to develop bundle methods, as compared with the *implicit* regularization provided by proximal bundle methods. Linderoth and Wright [17] applied a parallel bundle-trust-region (BTR) method and its asynchronous variant to the L-shaped method (also known as Benders decomposition) for small- and medium-sized stochastic linear programming problems [17]. A generalization of bundle methods with a unified form of penalty-like and trust-region-like stabilizing terms has been considered and analyzed in [9]. In this work, we further develop these ideas to develop an incremental bundle method in the context of Lagrangian dual decomposition for solving large-scale SMIP problems. We prove that our incremental bundle method is convergent for any work-allocation policy (static against dynamic in subsection 3.3), any trial-point selection policy (first-in-first-out against last-in-first-out in subsection 3.1), any trust region norm (including 2-norm and ∞-norms), and any bundle management step. We also perform extensive numerical experiments to demonstrate that the asynchronous implementation achieves significant improvements in parallel efficiency and solution time over a standard synchronous implementation.

The remainder of this paper is organized as follows. In section 2 we present the BTR method for Lagrangian DD and associated convergence analysis. Section 3 presents an asynchronous variant of the method and associated convergence analysis. In section 4 we present the implementation of the methods in the open-source software package `DSP` [14] and numerical experiments. In section 5, we summarize the paper and discuss possible directions for future research.

2. A bundle-trust-region algorithm. We propose a BTR algorithm to solve the Lagrangian dual problem (3). The proposed BTR method iteratively approximates the Lagrangian dual function $D(\lambda)$ by adding a set of cutting planes (cuts). We let $k \in \mathbb{Z}_+$ and $l \in \mathbb{Z}_+$ be the indices for major and minor iterations, respectively. Every major iteration updates the best lower bound, whereas every minor iterate updates the approximation of the Lagrangian dual function. We define the model function:

$$m_{k,l}(\lambda) := \max \sum_{j \in J} \theta_j \tag{5a}$$

$$\text{s.t. } \theta_j \leq D_j(\lambda^i) + (H_j x_j^i)^T(\lambda - \lambda^i),\ i \in \mathcal{B}^{k,l},\ j \in J. \tag{5b}$$

We recall that $H_j x_j^i \in \partial D_j(\lambda^i)$ for $j \in J$ and $\mathcal{B}^{k,l}$ is a set of cut indices at iteration (k,l). We also recall that the model function $m_{k,l}(\cdot)$ outer-approximates $D(\cdot)$ (i.e., $m_{k,l}(\lambda) \geq D(\lambda)$ for all $\lambda \in \mathbb{R}^{Nn_1}$ and (k,l)).

At iteration (k,l), the master problem is given by

$$\max_{\lambda\in\mathbb{R}^{Nn_1}} \quad m_{k,l}(\lambda) \tag{6a}$$

$$\text{s.t.} \quad \|\lambda-\lambda^k\|\le\Delta_{k,l}, \tag{6b}$$

where λ^k is the trust region (TR) center and $\Delta_{k,l}>0$ is the TR size. The master problem (6) finds a new trial point $\lambda^{k,l}$ to evaluate $D(\cdot)$. We define the *predicted increase* of $D(\cdot)$ as

$$v_{k,l} := m_{k,l}(\lambda^{k,l}) - D(\lambda^k). \tag{7}$$

The TR center is updated as $\lambda^{k+1}\leftarrow\lambda^{k,l}$ if the sufficient decrease condition

$$D(\lambda^{k,l}) \ge D(\lambda^k) + \xi v_{k,l} \tag{8}$$

is satisfied, where $\xi\in(0,1/2)$. We call this type of iteration a *serious step*. In this case, the master problem (6) is resolved with the updated TR center. Otherwise, a new set of cuts (5b) is added to improve the model $m_{k,l}(\lambda)$. We call this type of iteration a *null step*. The method terminates whenever

$$v_{k,l} \le \epsilon\cdot(1+|D(\lambda^k)|) \tag{9}$$

holds for some $\epsilon\in\mathbb{R}_+$.

The TR size $\Delta_{k,l}$ needs to be carefully updated to accelerate performance. For example, if the TR size is too large, a number of null steps must be taken before each serious step is taken. On the other hand, if the TR size is too small, the algorithm takes serious steps at almost all iterations with only marginal improvements in the lower bound. We thus devise tests for detecting whether the TR size is too large or too small. To do so, we define the model approximation error at iterate (k,l) as

$$\delta^{k,l} := \sum_{j\in J}\delta_j^{k,l}, \tag{10}$$

where

$$\delta_j^{k,l} := D_j(\lambda^{k,l}) + (H_j x_j^{k,l})^T(\lambda^k-\lambda^{k,l}) - D_j(\lambda^k). \tag{11}$$

By construction, $\delta_j^{k,l}\ge 0$ holds. We define the maximum model variation as

$$V_k := \max_{\lambda}\left\{D(\lambda) : \|\lambda-\lambda^k\|\le 1\right\} - D(\lambda^k). \tag{12}$$

We deem the TR size to be too large if either $D(\lambda^k)-D(\lambda^{k,l})$ or the approximation error $\delta^{k,l}$ are much larger than V_k. By construction, we have that

$$V_k \le \max_{\lambda}\left\{m_{k,l}(\lambda) : \|\lambda-\lambda^k\|\le 1\right\} - D(\lambda^k) \tag{13}$$

$$\le \frac{v_{k,l}}{\min\{1,\|\lambda^{k,l}-\lambda^k\|\}}. \tag{14}$$

Consequently, it is sufficient to test whether

$$\max\left\{D(\lambda^k)-D(\lambda^{k,l}),\delta^{k,l}\right\} > \frac{v_{k,l}}{\min\{1,\|\lambda^{k,l}-\lambda^k\|\}} \tag{15}$$

holds in order to determine whether the TR size is too large. We now let

$$\rho := \min\{1, \|\lambda^{k,l} - \lambda^k\|\} \frac{\max\left\{D(\lambda^k) - D(\lambda^{k,l}), \delta^{k,l}\right\}}{v_{k,l}}, \tag{16}$$

and let τ^k count the iterations in which the TR size is not reduced. We then update the TR size as follows:

1. If $\rho > 0$, then $\tau^{k+1} := \tau^k + 1$.
2. If $\rho > \bar{\rho}$ or ($\rho \in (0, \bar{\rho})$ and $\tau \geq \bar{\rho}$), then set $\tau^{k+1} := 0$, and

$$\Delta_{k,l+1} \leftarrow \max\left\{\frac{\Delta_{k,l}}{\min\{\rho, \underline{\rho}\}}, \underline{\Delta}\right\}. \tag{17}$$

We deem the TR to be too small if a larger (i.e., better) Lagrangian function value is found, and if the solution is bounded by the TR constraint. That is, whenever

$$D(\lambda^{k+1}) \geq D(\lambda^k) + \frac{1}{2} v_{k,l} \quad \text{and} \quad \|\lambda - \lambda^k\| \leq \Delta_{k,l} \tag{18}$$

hold, we increase the TR size as

$$\Delta_{k,l+1} \leftarrow \min\{2\Delta_{k,l}, \bar{\Delta}\}. \tag{19}$$

Algorithm 1 BTR method.

1: Initialize $\lambda^0 \in \mathbb{R}^{Nn_1}, \Delta_{0,0} \in [\underline{\Delta}, \bar{\Delta}], \xi \in (0, 1/2), \epsilon \geq 0, \mathcal{B}^{0,0} \leftarrow \{0\}, k \leftarrow 0$, and $l \leftarrow 0$.
2: Solve the Lagrangian subproblem (4) to find $D_j(\lambda^0)$ and x_j^0 for all $j \in J$.
3: Initialize the model function $m_{0,0}$.
4: **loop**
5: Solve the master (6) to $\lambda^{k,l}$.
6: Solve the Lagrangian subproblem (4) to find $D_j(\lambda^{k,l})$ and $x_j^{k,l}$ for all $j \in J$.
7: **if** $m_{k,l}(\lambda^{k,l}) - D(\lambda^k) \leq \epsilon(1 + |D(\lambda^k)|)$ **then** ▷ Termination test
8: Stop
9: **end if**
10: **if** $D(\lambda^{k,l}) \geq D(\lambda^k) + \xi[m_{k,l}(\lambda^{k,l}) - D(\lambda^k)]$ **then** ▷ Serious step
11: $\lambda^{k+1} \leftarrow \lambda^{k,l}$.
12: Choose $\Delta_{k+1,0} \in [\Delta_{k,l}, \bar{\Delta}]$.
13: $m_{k+1,0} \leftarrow m_{k,l}$.
14: $k \leftarrow k + 1$ and $l \leftarrow 0$.
15: **else** ▷ Null step
16: Choose $\Delta_{k,l+1} \in [\underline{\Delta}, \Delta_{k,l}]$.
17: Update $m_{k,l+1}$ by adding cuts (5b).
18: $l \leftarrow l + 1$.
19: **end if**
20: **end loop**

2.1. Algorithmic steps. We now summarize the steps of Algorithm 1. The BTR algorithm is initialized using the dual λ^0 and parameters $\Delta_{k,l}, \xi$, and ϵ (line 1 of Algorithm 1). For a given λ^0, the Lagrangian dual function is evaluated by solving the subproblem (4) for each scenario $j \in J$ (line 2 of Algorithm 1). The model

function $m_{0,0}$ is initialized by adding cuts (5b) generated at x_j^0 (line 3 of Algorithm 1). The algorithm continues by finding a new dual value $\lambda^{k,l}$ (line 5 of Algorithm 1), solving the Lagrangian subproblems (line 6 of Algorithm 1), updating the TR (lines 11, 12, and 16 of Algorithm 1), and updating the master problem (lines 13 and 17 of Algorithm 1) until the termination criterion is met (line 8 of Algorithm 1).

The BTR algorithm solves the Lagrangian dual problem and thus only provides a lower bound for the original SMIP (1). An upper bound for SMIP (1) can be obtained by evaluating first-stage variable solutions x_j^k obtained for each Lagrangian subproblem $D_j(\lambda^k)$ at iteration k. We note that at most $|J|$ first-stage solutions can be obtained in each iteration. The evaluation of first-stage solution x^k solves the second-stage recourse function $\mathcal{Q}(x^k) := \sum_{j=1}^N p_j Q_j(x^k)$, where

$$Q_j(x) := \min_{y \in Y} \left\{ q_j^T y : W_j y \geq h_j - T_j x \right\}. \tag{20}$$

We assume that $Q_j(x) = \infty$ if there does not exist recourse $y \in Y$ such that $W_j y \geq h_j - T_j x$ (i.e., the candidate x is infeasible). We recall that obtaining an optimal upper bound requires an exhaustive branch-and-bound scheme [5, 12].

2.2. Convergence analysis. We now present a convergence analysis for Algorithm 1. We assume that $\epsilon = 0$ throughout this section. We first show that only a finite number of cuts (5b) are available to construct the model function $m_{k,l}(\cdot)$.

LEMMA 2.1. *Algorithm* 1 *can generate only a finite number of cuts* (5b).

Proof. The subgradients $(H_j x_j^k)$ at major iteration k are obtained at the vertices (x_j, y_j) of the polyhedra $conv(G_j)$, $j \in J$. Such vertices are finitely many in $conv(G_j)$ for all $j \in J$. ∎

We now show that the algorithm does not perform an infinite number of null steps.

LEMMA 2.2. *Suppose that Algorithm* 1 *takes a null step at iteration* $(k, 0)$. *There exists a finite* $l > 0$ *such that the algorithm either takes a serious step or terminates at iteration* l.

Proof. To establish a contradiction, suppose that the algorithm takes infinitely many null steps at major iteration k. Then, we have for $l \geq 0$,

$$m_{k,l}(\lambda^{k,l}) - D(\lambda^k) > \epsilon(1 + |D(\lambda^k)|) = 0 \tag{21}$$

and

$$D(\lambda^{k,l}) < D(\lambda^k) + \xi \left[m_{k,l}(\lambda^{k,l}) - D(\lambda^k) \right]. \tag{22}$$

Let $\theta_j^{k,l}$ be the solution of the model $m_{k,l}(\lambda^{k,l})$. There exists some $j \in J$ such that linear inequalities generated at iteration (k, l),

$$\theta_j \leq D_j(\lambda^{k,l}) + (H_j x_j^{k,l})^T (\lambda - \lambda^{k,l}), \tag{23}$$

are violated at $(\theta^{k,l}, \lambda^{k,l})$; otherwise,

$$\begin{aligned} \sum_{j \in J} \theta_j^{k,l} - D(\lambda^{k,l}) &= m_{k,l}(\lambda^{k,l}) - D(\lambda^{k,l}) \\ &> m_{k,l}(\lambda^{k,l}) - D(\lambda^k) - \xi[m_{k,l}(\lambda^{k,l}) - D(\lambda^k)] \end{aligned}$$

$$> 0.$$

Here, the first and second inequalities hold due to (22) and (21), respectively. By Lemma 2.1, we have that either (21) or (22) will be violated after a finite number of null steps, contradicting the assumption. ∎

We adapt an approach that obtains the lower bound of the predicted increase, as shown in [17].

LEMMA 2.3. *For $k \geq 0$ and $l \geq 0$, we have that*

$$m_{k,l}(\lambda^{k,l}) - m_{k,l}(\lambda^k) \geq \min\left\{\frac{\Delta_{k,l}}{\|\lambda^* - \lambda^k\|}, 1\right\}\left[D(\lambda^*) - D(\lambda^k)\right]. \tag{24}$$

Proof. We need only show that (24) holds for the ∞-norm because $\|\cdot\|_\infty \leq \|\cdot\|_p \leq \|\cdot\|_r$ for $p > r > 0$. Suppose that we obtain an optimal step size $\alpha_{k,l}$ of the form

$$\alpha_{k,l} := \arg\max_{\alpha\in[0,1]} \left\{m_{k,l}(\lambda^k + \alpha[\lambda^* - \lambda^k]) : \|\alpha[\lambda^* - \lambda^k]\| \leq \Delta_{k,l}\right\}.$$

We have

$$\begin{aligned} m_{k,l}(\lambda^{k,l}) &\geq m_{k,l}(\lambda^k + \alpha_{k,l}[\lambda^* - \lambda^k]) \\ &\geq D(\lambda^k + \alpha_{k,l}[\lambda^* - \lambda^k]) \\ &\geq D(\lambda^k) + \alpha_{k,l}[D(\lambda^*) - D(\lambda^k)], \end{aligned}$$

where the first inequality holds because $\lambda^{k,l}$ is the maximizer of $m_{k,l}$, the second inequality holds because $m_{k,l}$ is an outer approximation of $D(\cdot)$, and the last inequality holds because of concavity of $D(\cdot)$. Since $m_{k,l}(\lambda^k) = D(\lambda^k)$, we have that

$$m_{k,l}(\lambda^{k,l}) - m_{k,l}(\lambda^k) \geq m_{k,l}(\lambda^{k,l}) - D(\lambda^k) \geq \alpha_{k,l}[D(\lambda^*) - D(\lambda^k)]. \tag{25}$$

Moreover, since $\|\alpha[\lambda^* - \lambda^k]\| \leq \Delta_{k,l}$, the optimal step size is given by

$$\alpha_{k,l} = \min\left\{\frac{\Delta_{k,l}}{\|\lambda^* - \lambda^k\|}, 1\right\}. \tag{26}$$

Therefore, (24) is obtained from (25) and (26). ∎

We now show that Algorithm 1 finds an optimal Lagrangian dual bound.

THEOREM 2.4. *Algorithm 1 delivers a dual iterate sequence $\{\lambda_k\}$ satisfying $\lim_{k\to\infty} D(\lambda^k) \to D(\lambda^*)$.*

Proof. Let l_k be the iteration index in which λ^{k,l_k} takes a serious step and thus $\lambda^{k+1} = \lambda^{k,l_k}$. Since

$$m_{k,l_k}(\lambda^{k,l_k}) - D(\lambda^k) \geq \epsilon(1 + |D(\lambda^k)|) > 0 \tag{27}$$

and

$$D(\lambda^{k,l_k}) - D(\lambda^k) \geq \xi[m_{k,l_k}(\lambda^{k,l_k}) - D(\lambda^k)] > 0, \tag{28}$$

we have that $D(\lambda^{k+1}) - D(\lambda^k) > 0$. Since $\{D(\lambda^k)\}$ is an increasing sequence bounded above by z_{LD}, we have that

$$\lim_{k\to\infty} D(\lambda^{k+1}) - D(\lambda^k) = 0. \tag{29}$$

Moreover, from Lemma 2.3 we have that

$$m_{k,l_k}(\lambda^{k,l_k}) - m_{k,l_k}(\lambda^k) = D(\lambda^{k+1}) - D(\lambda^k) \geq \min\left\{\frac{\Delta_{k,l_k}}{\|\lambda^* - \lambda^k\|}, 1\right\}\left[D(\lambda^*) - D(\lambda^k)\right].$$

From (29) and $\Delta_{k,l} > 0$, we have that $\lim_{k\to\infty} D(\lambda^*) - D(\lambda^k) = 0$. ∎

Theorem 2.4 implies finite convergence to an ϵ-optimum for $\epsilon > 0$. In other words, there exists $K \in \mathbb{Z}_+$ for any $\epsilon > 0$ such that $D(\lambda^*) - D(\lambda^k) < \epsilon$ for $k \geq K$. Also note that Theorem 2.4 is valid for any $\underline{\Delta} > 0$. The convergence analysis remains valid independent of the choice of the TR norm, as shown in the proof of Lemma 2.3. The convergence result is also independent of the bundle management strategy at serious steps. For instance, Theorem 2.4 holds even if all the cuts are removed at every serious step. However, a suitable bundle management strategy may improve the efficiency of Algorithm 1.

3. An asynchronous variant. In this section we present an asynchronous implementation variant of the BTR algorithm described in Algorithm 1. To achieve asynchronicity, we only use a subset of scenario indices to update the master problem and the TR. Let $J^i \subseteq J$ be the subset of scenario indices such that bundle information $i \in \mathcal{B}^{k,l}$ is added to the model. We define the model function

$$\tilde{m}_{k,l}(\lambda) := \max \sum_{j\in J} \theta_j \tag{30a}$$

$$\text{s.t. } \theta_j \leq D_j(\lambda^i) + (H_j x_j^i)^T(\lambda - \lambda^i)\ \forall i \in \mathcal{B}^{k,l},\ j \in J^i, \tag{30b}$$

where the cuts (30b) are generated only for a subset of scenarios J^i. Note that $\tilde{m}_{k,l}(\lambda) \geq D(\lambda)$ and $m_{k,l}(\lambda) \geq D(\lambda)$ for any given $\lambda \in \mathbb{R}^{Nn_1}$ and (k,l).

At iteration (k,l), the master problem of the asynchronous variant is given by

$$\max_{\lambda\in\mathbb{R}^{Nn_1}} \left\{\tilde{m}_{k,l}(\lambda) : \|\lambda - \lambda^k\| \leq \Delta_{k,l}\right\}. \tag{31}$$

While the master problem (31) is a natural extension of (6), it is not straightforward to guarantee convergence under this setting. In particular, existing algorithms assume full scenario synchronization before updating the TR (i.e., by checking (8), (16), and (18)), and terminating the algorithm (9). We now describe necessary modifications to ensure convergence.

We consider a set of processes that consist of a master process and multiple worker processes. The master process is responsible for solving the master problem (31) to find a new trial point $\lambda^{k,l}$ for the Lagrangian dual function $D(\cdot)$. We let Π denote the set of worker processes. Each worker process $\pi \in \Pi$ is responsible for solving Lagrangian subproblems for a subset $J_\pi \subseteq J$ of scenarios to evaluate the trial point. We let $\Pi_{k,l} \subseteq \Pi$ denote a subset of idle worker processes ready for subproblem solutions at iteration (k,l). We define $\underline{\Pi} \in (0, |\Pi|]$ as the minimum number of worker processes that are ready for subproblem solutions. We let $\Lambda_{k,l}$ denote a queue for trial points λ^q and status s^q, where q index iterations (k,l), the status of evaluation of trial point λ^q is encoded in s^q_π for $\pi \in \Pi$, and $s^q := (s^q_\pi\ \forall \pi \in \Pi)$. Each queue element is initialized by $s^q = \texttt{ready}$, and $s^q_\pi = \texttt{assigned}$ is encoded when λ^q is under evaluation at process π. Once λ_q is evaluated at process π, the status is set to $s^q_\pi = \texttt{evaluated}$. We let $\bar{\Lambda}$ be the maximum number of elements in the queue $\Lambda_{k,l}$ at any iteration (k,l).

We denote the predicted increase at iteration (k,l) by

$$\tilde{v}_{k,l} := \tilde{m}_{k,l}(\lambda^{k,l}) - D(\lambda^k). \tag{32}$$

The serious steps and null steps are taken only when a trial point is evaluated by all worker processes. Otherwise, we update the model function $\tilde{m}_{k,l+1}(\cdot)$ by adding cuts (30b). In particular, if there exists

$$\hat{\lambda}^{k,l} \in \arg\max\{D(\lambda) : (\lambda, s) \in \Lambda_{k,l},\ s_\pi = \texttt{evaluated}\ \forall \pi \in \Pi\} \tag{33}$$

and

$$D(\hat{\lambda}^{k,l}) \geq D(\lambda^k) + \xi \tilde{v}_{k,l}, \tag{34}$$

we update the TR center $\lambda^{k+1} \leftarrow \hat{\lambda}^{k,l}$. Otherwise, a null step is taken to update the model function $\tilde{m}_{k,l+1}(\cdot)$. We terminate the algorithm if

$$\tilde{v}_{k,l} \leq \epsilon(1 + |D(\lambda^k)|). \tag{35}$$

We devise tests for adjusting the TR size $\Delta_{k,l}$ while still retaining convergence. For a small TR size, we can use the criterion (18) and update (19). In the asynchronous variant, not all scenario information is available at most null steps. To handle this case, we deem the TR to be too large if

$$\tilde{\delta}^{k,l} := \frac{|J| \sum_{\pi \in \Pi_{k,l}} \sum_{j \in J_\pi} \delta_j^{k,l}}{\sum_{\pi \in \Pi_{k,l}} |J_\pi|} > \frac{\tilde{v}_{k,l}}{\min\{1, \|\lambda^{k,l} - \lambda^k\|\}}. \tag{36}$$

We define

$$\tilde{\rho} := \min\left\{1, \|\lambda^{k,l} - \lambda^k\|\right\} \frac{\tilde{\delta}^{k,l}}{v_{k,l}} \tag{37}$$

and decrease the TR size as follows:

1. If $\tilde{\rho} > 0$, then $\tau^{k+1} \leftarrow \tau^k + 1$.
2. If $\tilde{\rho} > \overline{\rho}$ or $(\rho \in (0, \overline{\rho})$ and $\tau \geq \overline{\rho})$, then set $\tau^{k+1} \leftarrow 0$, and

$$\Delta_{k,l+1} \leftarrow \max\left\{\frac{\Delta_{k,l}}{\min\{\rho, \underline{\rho}\}}, \Delta^{min}\right\}. \tag{38}$$

3.1. Algorithmic steps. The steps of the asynchronous BTR algorithm are summarized in Algorithms 2 and 3. The steps in Algorithm 2 are taken in the master process and those in Algorithm 3 are taken in the worker processes. Algorithm 2 is initialized by setting parameters (line 1), collecting the initial bundle information (line 2), and creating the model function $\tilde{m}_{0,0}$ (line 3). The main steps at each iteration (k,l) are described in the loop (lines 4–45). A new trial point $\lambda^{k,l}$ is obtained by solving the master (31) (line 5). If the termination criterion is satisfied by $\tilde{m}_{k,l}(\lambda^{k,l})$, the algorithm terminates (lines 6–8). Otherwise, the trial point is queued with the initial status $s_\pi^{k,l} \leftarrow \texttt{ready}$ for all $\pi \in \Pi$ if the queue $\Lambda_{k,l}$ is not full (lines 9–12). For each available worker process $\pi \in \Pi_{k,l}$, the master process chooses a trial point λ^q in queue $\Lambda_{k,l}$ for the Lagrangian subproblem solutions (line 15), if it exists, and sends it to the worker process (line 16). Here, we may consider two policies for choosing the trial points: first-in-first-out (FIFO) and last-in-first-out (LIFO). The convergence

results in section 3.2 are not affected by the choice of policy. If there exists no trial point to evaluate in the queue, the current trial point is sent to the worker process (line 19).

We note that the master process does not wait for all the worker processes to complete the evaluations. The master process receives the subproblem solutions from at least $\underline{\Pi}$ worker processes π and encodes $s_\pi^q \leftarrow$ `evaluated` for each λ^q and $\pi \in \Pi_{k,l}$ if the trial point is from the queue (lines 23–27). The parameter $\underline{\Pi}$ controls the frequency of each master problem solve. It is possible that $|\Pi_{k,l}| > \underline{\Pi}$ if the previous iteration takes time for completing the evaluations in a large enough number of worker processes. For the trial points evaluated by all worker processes, the master process chooses the best trial point for the serious step test (8) (line 30). If there exists a trial point evaluated by all worker processes, the TR size may be updated (lines 33 and 38). If the serious step test is satisfied, the master process updates the TR center (line 35). If there exists no trial point evaluated by all worker processes, the model function $\tilde{m}_{k,l+1}$ is updated by adding cuts (line 42). Each worker process π repeats the steps in lines 1–5 until receiving the termination signal from the master process. In the loop, the worker process receives a new trial point λ from the master process (line 2), evaluates it (line 3), and sends the information $D_j(\lambda), x_j$ to the master process (line 4).

3.2. Convergence analysis. We analyze the convergence of Algorithm 2. We note that for $\underline{\Pi} = |\Pi|$, Algorithm 2 is equivalent to Algorithm 1. Therefore, we assume that $\underline{\Pi} \in (0, |\Pi|)$, and we assume that $\epsilon = 0$ throughout this section. At every major iteration $k \geq 0$, we have that $\tilde{m}_{k,l+1}(\lambda) \leq \tilde{m}_{k,l}(\lambda)$ for all $l \geq 0$.

We first show that Algorithm 2 takes only a finite number of null steps.

LEMMA 3.1. *Suppose that Algorithm* 2 *takes a null step at iteration* $(k, l_{(1)})$. *There exist a finite number* $i > 1$ *such that the algorithm either makes a serious step or terminates at iteration* $(k, l_{(i)})$.

Proof. Suppose for contradiction that the algorithm takes null steps at iteration $(k, l_{(i)})$ for all $i \geq 1$. Then, we have for $i \geq 1$ that

$$\tilde{m}_{k,l_{(i)}}(\lambda^{k,l_{(i)}}) - D(\lambda^k) > \epsilon(1 + |D(\lambda^k)|) = 0 \tag{39}$$

and

$$D(\hat{\lambda}^{k,l_{(i)}}) < D(\lambda^k) + \xi \left[\tilde{m}_{k,l_{(i)}}(\lambda^{k,l_{(i)}}) - D(\lambda^k)\right]. \tag{40}$$

Let (k_i, l_i) be the iteration at which $\hat{\lambda}^{k,l_{(i)}}$ was obtained by the master (31) so that $\lambda^{k_i,l_i} = \hat{\lambda}^{k,l_{(i)}}$ for $i \geq 1$. For the case when $k_i < k$, the algorithm can find $\hat{\lambda}^{k,l_{(I)}}$ such that $k_I = k$ and $l_I = l_{(I)}$ for $i < I < \infty$. Therefore, we need only consider the case when $k_i = k$ and $l_i \leq l_{(i)}$. In such a case, there exists some $j \in J$ such that the linear inequalities generated at λ^{k_i,l_i},

$$\theta_j \leq D(\lambda^{k_i,l_i}) + (H_j x_j^{k_i,l_i})^T(\lambda - \lambda^{k_i,l_i}), \tag{41}$$

are violated at $(\theta^{k_i,l_i}, \lambda^{k_i,l_i})$ because

$$\begin{aligned}\sum_{j\in J} \theta_j^{k_i,l_i} - D(\lambda^{k_i,l_i}) &= \tilde{m}_{k_i,l_i}(\lambda^{k_i,l_i}) - D(\lambda^{k_i,l_i}) \\ &> \tilde{m}_{k_i,l_i}(\lambda^{k_i,l_i}) - D(\lambda^k) - \xi[\tilde{m}_{k,l_{(i)}}(\lambda^{k,l_{(i)}}) - D(\lambda^k)]\end{aligned}$$

Algorithm 2 Asynchronous BTR algorithm.

Initialize $\lambda^0 \in \mathbb{R}^{Nn_1}, \Delta_{0,0} \in (0, \Delta^{max}], \xi \in (0, 0.5), \epsilon \geq 0, \bar{\Lambda} > 0, \underline{\Pi} \in (0, |\Pi|], \Lambda_{0,0} \leftarrow \emptyset, \Pi_{0,0} \leftarrow \Pi, \mathcal{B}^{0,0} \leftarrow \{0\}, k \leftarrow 0$, and $l \leftarrow 0$.
Solve the Lagrangian subproblem (4) to find $D_j(\lambda^0)$ and x_j^0 for all $j \in J$.
Initialize the model function $\tilde{m}_{0,0}$.
loop
  Solve the master (31) to find $\lambda^{k,l}$.
  if $\tilde{m}_{k,l}(\lambda^{k,l}) - D(\lambda^k) \leq \epsilon(1 + |D(\lambda^k)|)$ **then** ▷ Termination test
    Stop
  end if
  if $|\Lambda_{k,l}| < \bar{\Lambda}$ **then** ▷ Queue new dual variable
    $s_\pi^{k,l} \leftarrow$ `ready` for all $\pi \in \Pi$.
    $\Lambda_{k,l} \leftarrow \Lambda_{k,l} \cup \{(\lambda^{k,l}, s^{k,l})\}$.
  end if
  for $\pi \in \Pi_{k,l}$ **do** ▷ Send dual variables to processes
    if $\exists (\lambda^q, s^q) \in \Lambda_{k,l}$ such that $s_\pi^q =$ `ready` **then**
      Choose an element $(\lambda^q, s^q) \in \Lambda_{k,l}$ such that $s_\pi^q =$ `ready`.
      SEND λ^q to process π.
      $s_\pi^q \leftarrow$ `assigned`.
    else
      SEND $\lambda^{k,l}$ to process π.
    end if
    $\Pi_{k,l} \leftarrow \Pi_{k,l} \backslash \{\pi\}$.
  end for
  repeat ▷ Receive subproblem solutions from processes
    RECEIVE $D_j(\lambda^q)$ and x_j^q for $j \in J_\pi$ from any process $\pi \in \Pi$.
    $s_\pi^q \leftarrow$ `evaluated` if $(\lambda^q, \cdot) \in \Lambda_{k,l}$.
    $\Pi_{k,l} \leftarrow \Pi_{k,l} \cup \{\pi\}$.
  until $|\Pi_{k,l}| \geq \underline{\Pi}$
  `serious` $\leftarrow$ `false`.
  if $\exists (\lambda, s) \in \Lambda_{k,l}$ such that $s_\pi =$ `evaluated` $\forall \pi \in \Pi$ **then**
    $\hat{\lambda}^{k,l} \leftarrow \arg\max\{D(\lambda) : (\lambda, s) \in \Lambda_{k,l},\ s_\pi =$ `evaluated` $\forall \pi \in \Pi\}$
    $\Lambda_{k,l} \leftarrow \Lambda_{k,l} \backslash \{(\hat{\lambda}^{k,l},$ `evaluated`$)\}$
    if $D(\hat{\lambda}^{k,l}) \geq D(\lambda^k) + \xi[\tilde{m}_{k,l}(\lambda^{k,l}) - D(\lambda^k)]$ **then** ▷ Serious step
      Choose $\Delta_{k+1,0} \in [\Delta_{k,l}, \Delta^{max}]$.
      Choose $\Lambda_{k+1,0} \subseteq \Lambda_{k,l}$.
      Set $\lambda^{k+1} \leftarrow \hat{\lambda}^{k,l}, \tilde{m}_{k+1,0} \leftarrow \tilde{m}_{k,l}, \Pi_{k+1,0} \leftarrow \Pi_{k,l}, k \leftarrow k+1$, and $l \leftarrow 0$.
      `serious` $\leftarrow$ `true`.
    else ▷ Null step
      Choose $\Delta_{k,l+1} \in (0, \Delta_{k,l}]$.
    end if
  end if
  if `serious` = `false` **then** ▷ Model update
    Update the model function $\tilde{m}_{k,l+1}$ by adding cuts (30b).
    Set $\Pi_{k,l+1} \leftarrow \Pi_{k,l}, \Lambda_{k,l+1} \leftarrow \Lambda_{k,l}, l \leftarrow l+1$.
  end if
end loop

Algorithm 3 Asynchronous BTR algorithm—Worker (π).

1: **repeat**
2: RECEIVE new trial point λ from the master process.
3: Solve the Lagrangian subproblem (4) to find $D_j(\lambda)$ and x_j for all $j \in J_\pi$.
4: SEND $D_j(\lambda), x_j$ for $j \in J_\pi$ to the master process.
5: **until** the master process terminates.

$$\begin{aligned}
&\geq \tilde{m}_{k_i,l_i}(\lambda^{k,l_{(i)}}) - D(\lambda^k) - \xi[\tilde{m}_{k,l_{(i)}}(\lambda^{k,l_{(i)}}) - D(\lambda^k)] \\
&\geq \tilde{m}_{k,l_{(i)}}(\lambda^{k,l_{(i)}}) - D(\lambda^k) - \xi[\tilde{m}_{k,l_{(i)}}(\lambda^{k,l_{(i)}}) - D(\lambda^k)] \\
&> 0.
\end{aligned}$$

Here, the second inequality holds because λ^{k_i,l_i} is a maximizer of $\tilde{m}_{k_i,l_i}$ and the third inequality holds because $\tilde{m}_{k,l}$ is nonincreasing within a given major iteration k. From Lemma 2.1, the algorithm violates either (39) or (40). ∎

Analogous to Lemma 2.3, we derive a lower bound for the predicted increase in the lower bound.

LEMMA 3.2. *For $k \geq 0$ and $l \geq 0$, we have that*

$$\tilde{m}_{k,l}(\lambda^{k,l}) - \tilde{m}_{k,l}(\lambda^k) \geq \min\left\{\frac{\Delta_{k,l}}{\|\lambda^* - \lambda^k\|}, 1\right\}\left[D(\lambda^*) - D(\lambda^k)\right]. \tag{42}$$

Proof. The proof follows the steps in the proof of Lemma 2.3. ∎

We now show that Algorithm 2 finds the Lagrangian dual bound in the limit.

THEOREM 3.3. *Algorithm 2 delivers a sequence of dual iterates $\{\lambda^k\}$ satisfying $\lim_{k\to\infty} D(\lambda^k) \to D(\lambda^*)$.*

Proof. Let l_k be such that $\hat{\lambda}^{k,l_k}$ takes a serious step and thus $\lambda^{k+1} = \hat{\lambda}^{k,l}$. Since

$$\tilde{m}_{k,l_k}(\lambda^{k,l_k}) - D(\lambda^k) \geq \epsilon(1 + |D(\lambda^k)|) > 0 \tag{43}$$

and

$$D(\hat{\lambda}^{k,l_k}) - D(\lambda^k) \geq \xi[\tilde{m}_{k,l_k}(\lambda^{k,l_k}) - D(\lambda^k)] > 0, \tag{44}$$

we have $D(\lambda^{k+1}) - D(\lambda^k) > 0$. Since $\{D(\lambda^k)\}$ is an increasing sequence that is bounded above by z_{LD}, we have that

$$\lim_{k\to\infty} D(\lambda^{k+1}) - D(\lambda^k) = 0. \tag{45}$$

Moreover, by Lemma 3.2 we have

$$\begin{aligned}
\tilde{m}_{k,l_k}(\hat{\lambda}^{k,l_k}) - \tilde{m}_{k,l_k}(\lambda^k) &= D(\lambda^{k+1}) - D(\lambda^k) \\
&\geq \min\left\{\frac{\Delta_{k,l_k}}{\|\lambda^* - \lambda^k\|}, 1\right\}\left[D(\lambda^*) - D(\lambda^k)\right].
\end{aligned}$$

From (45) and $\Delta_{k,l} > 0$, we have $\lim_{k\to\infty} D(\lambda^*) - D(\lambda^k) = 0$. ∎

3.3. Dynamic subproblem allocation. Algorithm 2 runs for a fixed (static) set J_π for each $\pi \in \Pi$, where each worker process is allocated to certain scenario subproblems for all iterations. The allocation of worker processes is advantageous because each mixed-integer programming (MIP) subproblem solver can take advantage of the warm-start feature given in off-the-shelf MIP solvers. However, the static allocation of subproblems can cause parallel inefficiency, because one of the worker processes might present a computational bottleneck for evaluating the trial points in queue.

The asynchronous computation in Algorithm 2 can be varied by dynamically allocating the subproblems to the worker processes. With dynamic allocation, the worker processes are not dedicated to certain scenario subproblems and can possibly solve different scenario subproblems at each iteration. An asynchronous algorithm with dynamic allocation can be obtained by performing minor modifications to Algorithm 2. In particular, we need only keep track of the status of the evaluation of the trial points in the queue for each scenario subproblem, as compared with that for each worker process in Algorithm 2. Similar to the definition of s_π^q used in Algorithm 2, we define $\tilde{s}_j^q$ as the status of evaluation of trial point λ^q for scenario index $j \in J$, and $\tilde{s}^q := (\tilde{s}_j^q \; \forall j \in J)$. The modified steps are summarized in Algorithm 4.

The initial queue status $\tilde{s}_j^{k,l}$ is set for each scenario $j \in J$ in line 10. We choose a trial point that is not evaluated for some scenarios and send it to a worker process (lines 14–17). For the choice of trial points, we are interested in FIFO and LIFO policies, as mentioned in subsection 3.1. The rest of the algorithm was modified in order to apply the modified notation $\tilde{s}$ for updating and checking the status of evaluating trial points for each scenario (lines 25 and 29–30). We note that the convergence analysis given in subsection 3.2 holds for Algorithm 4.

4. Numerical experiments. In this section, we present numerical experiments for the synchronous and asynchronous BTR algorithms.

4.1. Implementation. We have implemented the proposed algorithms in the open-source and parallel software package `DSP` [14]. The master problem and MIP subproblems were solved using `CPLEX 12.7`. The master problem was solved using a barrier method with 16 parallel cores, whereas the MIP subproblems were solved with default parameter settings. Moreover, the subproblems were solved in parallel by using MPI, and each process uses a single core. The subproblems are distributed to the processes in a round-and-robin fashion (except for the dynamic allocation strategy in subsection 4.5). The processes create a CPLEX solver environment for each subproblem in order to take advantage of the CPLEX warm-starting feature, as well as to avoid data loading time. All computations were performed on the *Blues* cluster—a 630-node computing cluster at Argonne National Laboratory. For both the synchronous and asynchronous BTR algorithms, we set $\lambda^0 = 0$. We also set the parameters $\bar{\rho} = 3$, $\underline{\rho} = 4$, $\underline{\Delta} = 10^{-2}, \bar{\Delta} = 10^4, \Delta_{0,0} = 10^2, \xi = 10^{-4}$, and $\epsilon = 10^{-5}$. We considered different values for parameters $\bar{\Lambda}, \underline{\Pi}$ of Algorithm 2 to evaluate performance, as discussed in subsection 4.4.

4.2. Problem instances. We use a day-ahead stochastic unit commitment (SUC) problem with data representing a test system for the California independent system operator (CAISO) interconnected with the Western Electricity Coordinating Council (WECC), as discussed in [21, 13, 11]. The SUC problem instances embed difficult MIP scenario subproblems that induce strong load imbalances. The test system consists of 225 buses, 375 transmission lines, 130 generators, 40 loads, 5 import

Algorithm 4 Asynchronous subsection algorithm with dynamic subproblem allocation.

Initialize $\lambda^0 \in \mathbb{R}^{Nn_1}, \Delta_{0,0} \in (0, \Delta^{max}], \xi \in (0, 0.5), \epsilon \geq 0, \bar{\Lambda} > 0, \underline{\Pi} \in (0, |\Pi|], \Lambda_{0,0} \leftarrow \emptyset, \Pi_{0,0} \leftarrow \Pi, \mathcal{B}^{0,0} \leftarrow \{0\}, k \leftarrow 0$, and $l \leftarrow 0$.
Solve the Lagrangian subproblem (4) to find $D_j(\lambda^0)$ and x_j^0 for all $j \in J$.
Initialize the model function $\tilde{m}_{0,0}$.
loop
  Solve the master (31) to find $\lambda^{k,l}$.
  if $\tilde{m}_{k,l}(\lambda^{k,l}) - D(\lambda^k) \leq \epsilon(1 + |D(\lambda^k)|)$ **then** ▷ Termination test
    Stop
  end if
  if $|\Lambda_{k,l}| < \bar{\Lambda}$ **then** ▷ Queue new dual variable
    $\tilde{s}_j^{k,l} \leftarrow$ `ready` for all $j \in J$.
    $\Lambda_{k,l} \leftarrow \Lambda_{k,l} \cup \{(\lambda^{k,l}, \tilde{s}^{k,l})\}$.
  end if
  for $\pi \in \Pi_{k,l}$ **do** ▷ Dynamic allocation of dual variables to processes
    if $\exists (\lambda^q, s^q) \in \Lambda_{k,l}$ such that $\tilde{s}_j^q =$ `ready` for any $j \in J$ **then**
      Choose an element $(\lambda^q, \tilde{s}^q) \in \Lambda_{k,l}$ such that $\tilde{s}_j^q =$ `ready` for some $j \in J$.
      SEND λ^q to process π.
      $\tilde{s}_j^q \leftarrow$ `assigned` and $J_\pi \leftarrow \{j\}$.
    else
      SEND $\lambda^{k,l}$ to process π.
    end if
    $\Pi_{k,l} \leftarrow \Pi_{k,l} \backslash \{\pi\}$.
  end for
  repeat ▷ Receive subproblem solutions from processes
    RECEIVE $D_j(\lambda^q)$ and x_j^q from any process $\pi \in \Pi$.
    $\tilde{s}_j^q \leftarrow$ `evaluated` if $(\lambda^q, \cdot) \in \Lambda_{k,l}$.
    $\Pi_{k,l} \leftarrow \Pi_{k,l} \cup \{\pi\}$.
  until $|\Pi_{k,l}| \geq \underline{\Pi}$
  `serious` $\leftarrow$ `false`.
  if $\exists (\lambda, \tilde{s}) \in \Lambda_{k,l}$ such that $\tilde{s}_j =$ `evaluated` $\forall j \in J$ **then**
    $\hat{\lambda}^{k,l} \leftarrow \arg\max\{D(\lambda) : (\lambda, \tilde{s}) \in \Lambda_{k,l},\ \tilde{s}_j =$ `evaluated` $\forall j \in J\}$
    $\Lambda_{k,l} \leftarrow \Lambda_{k,l} \backslash \{(\hat{\lambda}^{k,l},$ `evaluated`$)\}$
    if $D(\hat{\lambda}^{k,l}) \geq D(\lambda^k) + \xi[\tilde{m}_{k,l}(\lambda^{k,l}) - D(\lambda^k)]$ **then** ▷ Serious step
      Choose $\Delta_{k+1,0} \in [\Delta_{k,l}, \Delta^{max}]$.
      Choose $\Lambda_{k+1,0} \subseteq \Lambda_{k,l}$.
      Set $\lambda^{k+1} \leftarrow \hat{\lambda}^{k,l}, \tilde{m}_{k+1,0} \leftarrow \tilde{m}_{k,l}, \Pi_{k+1,0} \leftarrow \Pi_{k,l}, k \leftarrow k+1$ and $l \leftarrow 0$.
      `serious` $\leftarrow$ `true`.
    else ▷ Null step
      Choose $\Delta_{k,l+1} \in (0, \Delta_{k,l}]$.
    end if
  end if
  if `serious` = `false` **then** ▷ Model update
    Update the model function $\tilde{m}_{k,l+1}$ by adding cuts (30b).
    Set $\Pi_{k,l+1} \leftarrow \Pi_{k,l}, \Lambda_{k,l+1} \leftarrow \Lambda_{k,l}, l \leftarrow l+1$.
  end if
end loop

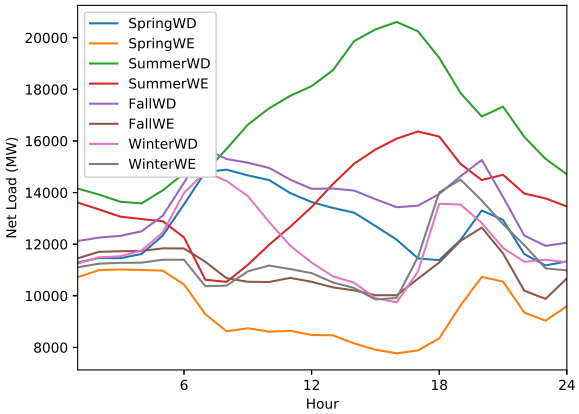


FIG. 1. *Net load of the test system for the 24-hour time horizon for each day type. WD and WE in the legend stand for weekdays and weekends, respectively.*

points, 5 wind farms, and 11 nonwind renewable generators. Of the 130 generators, 90 generators are *fast generators* capable of starting in response to demand. The other 40 generators are *slow generators*. In the SUC problem instances, we consider a 24-hour time horizon with hourly intervals. The amount of power at import points and renewable generators and loads are categorized in eight day types. Each day type is a combination of elements in two sets $\{Spring, Summer, Fall, Winter\}$ and $\{Weekdays, Weekends\}$. The load is calculated by the amount of total system load subtracted by total import power and renewable power, excluding wind power (see Figure 1). In addition to the eight day types, we use 160 scenarios of wind power generation at each wind farm and for each season (see Figure 2).

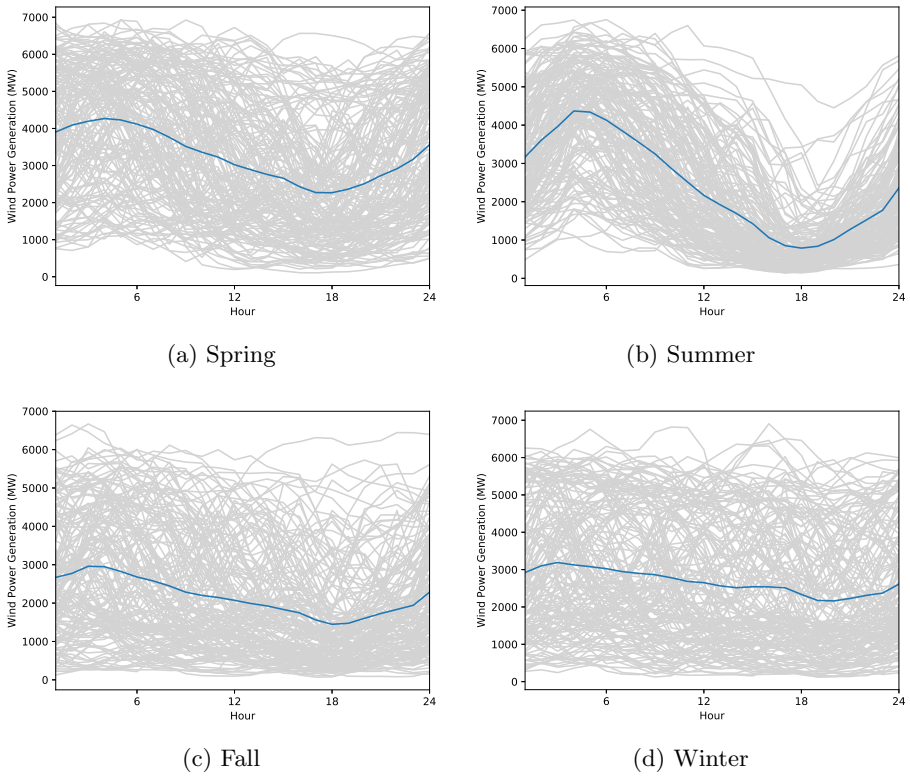
We use $\mathcal{N}, \mathcal{L}, \mathcal{T}$, and $\mathcal{S}$ to represent sets of buses, transmission lines, time periods, and scenarios, respectively. In addition, $\mathcal{L}_n^+$ and $\mathcal{L}_n^- \subset \mathcal{L}$ represent the set of transmission lines to and from bus $n \in \mathcal{N}$, respectively. We use $\mathcal{G}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{R}_n$, and $\mathcal{W}_n$ to denote sets of generators, loads, import points, renewable generators, and wind farms at bus $n \in \mathcal{N}$, respectively. We also define $\mathcal{G} := \sum_{n\in\mathcal{N}} \mathcal{G}_n$ and denote by $\mathcal{G}_S$ the set of slow generators that should be scheduled a day ahead. Let u_{gts}, v_{gts}, and p_{gts} be the decision variables for unit commitment, generator start, and power generation amount, respectively, for $g \in \mathcal{G}, t \in \mathcal{T}, s \in \mathcal{S}$. Let f_{lts} be the decision variables for power flow on transmission line $l \in \mathcal{L}$, and let θ_{nts} be the decision variables for phase angles at bus $n \in \mathcal{N}$. Let $d_{jts}, m_{its}, r_{its}$, and w_{its} be the slack variables representing load shedding, import spillage, renewable generation spillage, and wind generation spillage, respectively.

The formulation of the SUC problem is given by the following two-stage stochastic mixed-binary program:

$$\text{(46a)} \qquad \min \quad \sum_{t\in\mathcal{T}} \sum_{s\in\mathcal{S}} \frac{1}{|\mathcal{S}|} \left[\sum_{g\in\mathcal{G}} (K_g u_{gts} + S_g v_{gts} + C_g p_{gts}) + \sum_{j\in\mathcal{D}} V d_{jts} \right]$$

$$\text{(46b)} \qquad \text{s.t.} \quad u_{gts} = u_{gt,s-1} \ \forall g \in \mathcal{G}_S,\ t \in \mathcal{T},\ s \in \mathcal{S},$$

$$\text{(46c)} \qquad v_{gts} = v_{gt,s-1} \ \forall g \in \mathcal{G}_S,\ t \in \mathcal{T},\ s \in \mathcal{S},$$



(a) Spring (b) Summer

(c) Fall (d) Winter

FIG. 2. *Total amount of wind power for each season. Scenarios are shown in grey and the mean is in blue. (Figure in color online.)*

$$\text{(46d)} \qquad \sum_{q=t-UT_g+1}^{t} v_{gqs} \le u_{gts} \ \forall g \in \mathcal{G},\ t \in \{UT_g, \dots, T\},\ s \in \mathcal{S},$$

$$\text{(46e)} \qquad \sum_{q=t+1}^{t+DT_g} v_{gqs} \le u_{gts} \ \forall g \in \mathcal{G},\ t \in \{1, \dots, T - DT_g\},\ s \in \mathcal{S},$$

$$\text{(46f)} \qquad v_{gts} \ge u_{gts} - u_{g,t-1,s} \ \forall g \in \mathcal{G},\ t \in \mathcal{T},\ s \in \mathcal{S},$$

$$\text{(46g)} \qquad \sum_{l \in \mathcal{L}_n^+} f_{lts} - \sum_{l \in \mathcal{L}_n^-} f_{lts} + \sum_{g \in \mathcal{G}_n} p_{gts} = \sum_{j \in \mathcal{D}_n} (D_{jt} - d_{jts}) - \sum_{i \in \mathcal{I}_n} (M_{it} - m_{its}) - \sum_{i \in \mathcal{R}_n} (R_{it} - r_{its}) - \sum_{i \in \mathcal{W}_n} (W_{its} - w_{its}) \ \forall n \in \mathcal{N},\ t \in \mathcal{T},\ s \in \mathcal{S},$$

$$\text{(46h)} \qquad f_{lts} = B_l(\theta_{mts} - \theta_{nts}) \ \forall l = (m,n) \in \mathcal{L},\ t \in \mathcal{T},\ s \in \mathcal{S},$$

$$\text{(46i)} \qquad u_{gts} \in \{0,1\},\ v_{gts} \in [0,1] \ \forall g \in \mathcal{G},\ t \in \mathcal{T},\ s \in \mathcal{S},$$

$$\text{(46j)} \qquad P_g^{min} u_{gts} \le p_{gts} \le P_g^{max} u_{gts} \ \forall g \in \mathcal{G},\ t \in \mathcal{T},\ s \in \mathcal{S},$$

$$\text{(46k)} \qquad -F^{max} \le f_{lts} \le F^{max} \ \forall l \in \mathcal{L},\ t \in \mathcal{T},\ s \in \mathcal{S},$$

$$-360 \leq \theta_{nts} \leq 360 \; \forall n \in \mathcal{N},\; t \in \mathcal{T},\; s \in \mathcal{S}, \tag{46l}$$

$$0 \leq d_{jts} \leq D_{jt} \; \forall j \in \mathcal{D}_n,\; n \in \mathcal{N},\; t \in \mathcal{T},\; s \in \mathcal{S}, \tag{46m}$$

$$0 \leq m_{its} \leq M_{it} \; \forall i \in \mathcal{I}_n,\; n \in \mathcal{N},\; t \in \mathcal{T},\; s \in \mathcal{S}, \tag{46n}$$

$$0 \leq r_{its} \leq R_{it} \; \forall i \in \mathcal{R}_n,\; n \in \mathcal{N},\; t \in \mathcal{T},\; s \in \mathcal{S}, \tag{46o}$$

$$0 \leq w_{its} \leq W_{its} \; \forall i \in \mathcal{W}_n,\; n \in \mathcal{N},\; t \in \mathcal{T},\; s \in \mathcal{S}, \tag{46p}$$

where (46b) and (46c) are the nonanticipativity constraints on the first-stage variables u_{gts} and v_{gts} for $g \in \mathcal{G}_S, t \in \mathcal{T}, s \in \mathcal{S}$ (i.e., commitment decisions for slow generators). The objective function (46a) minimizes the expected total operating cost that sums commitment cost K_g, generator startup cost S_g, generation cost C_g, and load shedding cost V (set to \$5,000/MWh). Equations (46d) and (46e) are, respectively, the minimum uptime UT_g and downtime DT_g constraints for each generator $g \in \mathcal{G}$. Constraint (46f) imposes the unit commitment logic. Constraint (46g) represents the power balance equation with load D_{jt}, import power M_{it}, renewable generation R_{it}, and wind generation W_{its}. Constraint (46h) represents the direct current power flow equation with line susceptance B_l. Constraints (46j) and (46k) impose the minimum P_g^{min} and maximum P_g^{max} generation capacity and transmission line capacity F^{max}, respectively.

Using the net load data and the wind generation scenarios, we created 80 SUC problem instances. Each instance uses 16 scenarios for wind power generation. The scenario subproblems are distributed in a round-and-robin fashion, and we also explore the dynamic allocation described in subsection 4.5. For each problem instance, the first stage has 1,920 variables and 1,960 constraints, and the second stage has 21,144 variables and 21,930 constraints. Therefore, each SUC instance has 340,224 variables and 352,840 constraints.

4.3. Synchronous computing and load balance. We first present computational times and load balancing for the synchronous BTR algorithm to highlight the impact of load imbalancing on efficiency. We solve the 80 problem instances by using `DSP` with 32 cores on the *Blues* cluster. The master process uses 16 cores, and worker processes use the other 16 cores for solving MIP subproblems in parallel. We define the set of problem instances as $\mathcal{P}$. The load balance is quantified by using the percent imbalance metric, as defined in [22]. Specifically, we define the percent imbalance metric of problem instance $p \in \mathcal{P}$ and for each iteration k as

$$\nu_{pk} := \left(\frac{t_{pk}^{max}}{\bar{t}_{pk}} - 1 \right) \times 100\%, \tag{47}$$

where t_{pk}^{max} and $\bar{t}_{pk}$ are the maximum and mean subproblem solution times, respectively, for problem instance p over all worker processes at iteration k. We also define $\bar{\nu}_p := \max_k \nu_{pk}$ and $\underline{\nu}_p := \min_k \nu_{pk}$. We denote by ν_p^m the mean percent imbalance metric over all iterations. The computational results are reported in Table 1, where t represents the total solution time (in seconds) and t^{LB} denotes the total time spent in the computation of the lower bound.

For the 80 problem instances, the total solution time ranges from 1,406 to 10,782 seconds. The average solution time is 3,193 seconds, of which 3,118 seconds were spent on the lower bounding problem. The average lower bounding time per iteration is 20 seconds. Figure 3 summarizes the distribution of the average percent imbalance metrics, where the y-axis presents the number of problem instances such that the percent imbalance metric is greater than or equal to the x-axis value. The average

percent imbalance metrics range from 27% to 84%. The average percent imbalance is larger than 50% for the 18 problem instances.

Table 1: Computational results and percent imbalance metric from the synchronous BTR algorithm.

Instance	Iter	z_{LD}	t	t^{LB}	$\bar{\nu}_p$	$\underline{\nu}_p$	ν_p^m
SpringWD0	125	1602068	2439	2369	86%	17%	51%
SpringWD1	141	1799653	3026	2937	98%	20%	42%
SpringWD2	108	1787644	2046	1989	56%	15%	29%
SpringWD3	130	1726987	2567	2489	82%	18%	41%
SpringWD4	165	1898284	3182	3061	74%	18%	35%
SpringWD5	132	1999293	2767	2691	57%	14%	31%
SpringWD6	115	1615039	2170	2109	108%	22%	47%
SpringWD7	196	1607014	5209	4963	138%	19%	73%
SpringWD8	179	1725101	4134	3932	91%	22%	47%
SpringWD9	126	1782099	2433	2357	94%	21%	42%
SpringWE0	140	1044900	2397	2337	77%	13%	36%
SpringWE1	121	1114237	1826	1772	73%	10%	32%
SpringWE2	123	1110895	2211	2161	96%	21%	43%
SpringWE3	132	1084464	2410	2345	93%	18%	51%
SpringWE4	111	1199183	1653	1608	77%	12%	30%
SpringWE5	118	1259176	1732	1681	62%	15%	31%
SpringWE6	176	974356	2960	2700	98%	18%	42%
SpringWE7	154	1007195	2682	2598	94%	18%	47%
SpringWE8	138	1090599	2252	2181	76%	10%	39%
SpringWE9	121	1133203	1899	1760	96%	17%	37%
SummerWD0	189	4710799	5086	4874	110%	18%	50%
SummerWD1	149	4753046	4269	4225	115%	20%	48%
SummerWD2	112	4800483	2798	2771	121%	19%	52%
SummerWD3	208	4693318	5948	5532	97%	19%	50%
SummerWD4	149	4743635	3189	3019	60%	12%	34%
SummerWD5	288	4549022	10782	6942	164%	0%	82%
SummerWD6	175	4698568	5719	5497	174%	19%	85%
SummerWD7	134	4644613	3584	3449	103%	24%	51%
SummerWD8	115	4827235	2516	2487	64%	17%	34%
SummerWD9	276	4678698	8794	8181	123%	23%	60%
SummerWE0	160	3159044	3176	3010	61%	15%	34%
SummerWE1	85	3200891	1408	1390	75%	16%	31%
SummerWE2	80	3237591	1406	1390	77%	14%	32%
SummerWE3	129	3162672	2182	2078	67%	12%	28%
SummerWE4	91	3208814	1627	1572	51%	12%	29%
SummerWE5	163	3057287	2992	2838	111%	17%	42%
SummerWE6	151	3141373	2621	2477	61%	14%	30%
SummerWE7	111	3108884	2027	1955	71%	15%	30%
SummerWE8	90	3248660	1719	1662	66%	15%	34%
SummerWE9	206	3161301	3876	3503	63%	13%	32%
FallWD0	165	2444953	5421	5288	75%	29%	52%

Continued on next page

Table 1 – *Continued from the previous page*

Instance	Iter	z_{LD}	t	t^{LB}	$\bar{\nu}_p$	$\underline{\nu}_p$	ν_p^m
FallWD1	150	2555819	5897	5768	129%	19%	63%
FallWD2	144	2495699	4225	4120	101%	25%	49%
FallWD3	265	2470042	10244	9543	123%	23%	61%
FallWD4	175	2562315	5528	5338	95%	23%	48%
FallWD5	184	2563568	5260	5059	92%	23%	48%
FallWD6	159	2481434	4468	4340	79%	15%	46%
FallWD7	154	2624963	4052	3916	113%	18%	54%
FallWD8	164	2525282	4746	4582	84%	15%	43%
FallWD9	168	2406007	4498	4331	90%	23%	48%
FallWE0	121	1542897	1925	1860	75%	16%	40%
FallWE1	108	1622543	1705	1649	76%	16%	37%
FallWE2	149	1550416	3040	2918	75%	22%	44%
FallWE3	134	1541075	2436	2357	100%	22%	46%
FallWE4	107	1627667	1849	1796	79%	16%	39%
FallWE5	115	1631388	1903	1751	63%	16%	35%
FallWE6	112	1567312	1785	1727	71%	14%	38%
FallWE7	97	1687574	1835	1788	92%	21%	44%
FallWE8	101	1589716	1791	1741	102%	13%	43%
FallWE9	113	1503765	1775	1718	75%	18%	35%
WinterWD0	146	1869441	2994	2899	71%	18%	39%
WinterWD1	147	1952864	3243	3141	130%	25%	58%
WinterWD2	110	1674367	1925	1866	105%	20%	47%
WinterWD3	136	1786630	2923	2839	125%	29%	58%
WinterWD4	117	1735955	1992	1926	80%	17%	38%
WinterWD5	142	1932486	2937	2844	73%	21%	38%
WinterWD6	137	2036682	2941	2841	74%	16%	42%
WinterWD7	143	1994766	3170	3075	70%	19%	40%
WinterWD8	138	1738391	2656	2566	97%	25%	47%
WinterWD9	136	1733211	2452	2364	103%	19%	41%
WinterWE0	130	1293480	2108	2038	83%	16%	38%
WinterWE1	147	1341952	2556	2461	87%	16%	36%
WinterWE2	129	1166758	1985	1926	64%	16%	36%
WinterWE3	197	1238547	3505	3302	119%	23%	60%
WinterWE4	119	1209598	1821	1768	72%	16%	39%
WinterWE5	156	1336934	2693	2582	94%	21%	44%
WinterWE6	122	1403328	1843	1778	78%	18%	38%
WinterWE7	141	1375098	2341	2255	101%	18%	38%
WinterWE8	110	1187088	1770	1724	75%	23%	41%
WinterWE9	197	1207103	5494	5249	233%	22%	76%

4.4. Asynchronous computing and performance profiles. We analyze the computational performance of the asynchronous BTR algorithms. In this section, we collect the numerical results based on the variant that statically allocates subproblems to the worker processes and chooses the first element of the queue for trial points (i.e., FIFO). We use the metrics for the performance profile in [6]. We use $\mathcal{C}$ to define the set of solver configurations. For each problem $p \in \mathcal{P}$ and configuration $c \in \mathcal{C}$, we use t_{pc} to denote the solution time for solving problem p under configuration c. We define

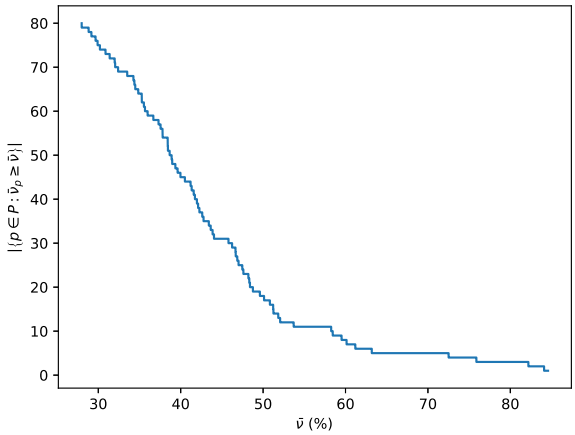


FIG. 3. *Distribution of the average percent imbalance metrics resulting from the synchronous BTR method.*

the performance ratio

$$r_{pc} := \frac{t_{pc}}{\min_{c \in \mathcal{C}} t_{pc}} \tag{48}$$

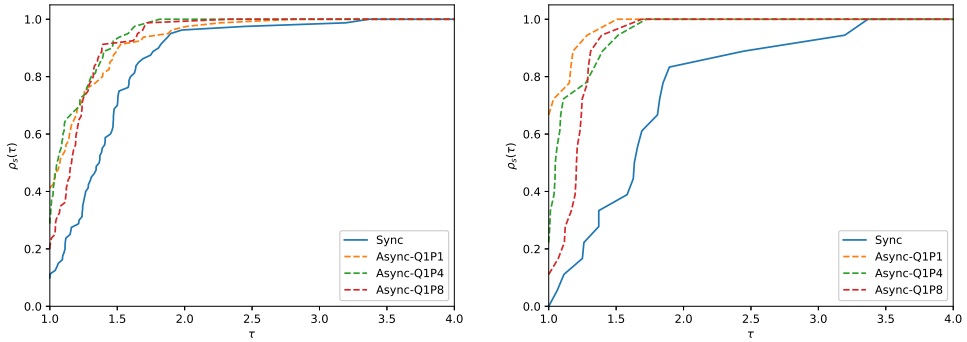
that represents the performance of configuration c as compared with the best performance by any solver configuration on problem p. We now define the performance of configuration c on any given problem as the probability for configuration c that a performance ratio r_{pc} is within a factor τ of the best possible ratio. In other words,

$$\rho_c(\tau) := \frac{1}{|\mathcal{P}|} \left| \{ p \in \mathcal{P} : r_{pc} \leq \tau \} \right|. \tag{49}$$

We compare the performance of the synchronous and asynchronous BTR strategies with different algorithmic settings. In our numerical experiments, we vary the maximum queue size $\bar{\Lambda} \in \{1, 2\}$ and the minimum number of worker processes to receive bundle information $\underline{\Pi} \in \{1, 4, 8\}$.

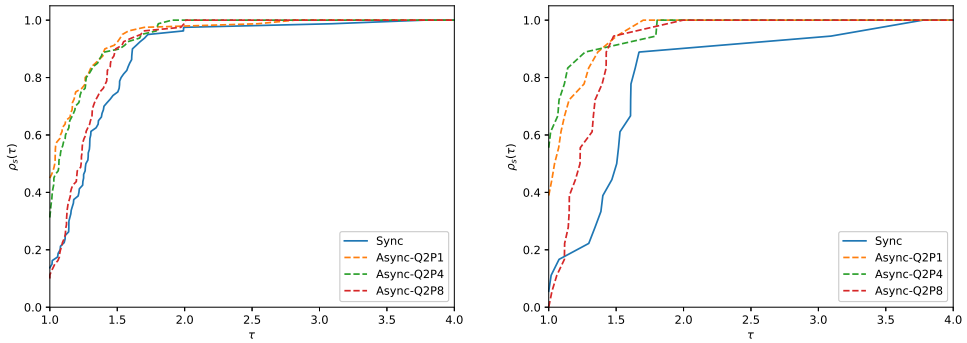
Figure 4 shows the performance of the synchronous and asynchronous BTR algorithms for $\bar{\Lambda} = 1$ and $\underline{\Pi} \in \{1, 4, 8\}$. We label the synchronous method "Sync" and label the asynchronous method with $\bar{\Lambda} = m$ and $\underline{\Pi} = n$ "Async-QmPn." We see that the asynchronous algorithm results in higher probabilities than the synchronous counterpart for any factor τ. In Figure 4a we present profiles for all instances; we can see that Async-Q1P1 has the most wins (with a probability of 0.41) and that Sync has the least wins (with a probability of 0.11). In Figure 4b we profile the solvers for highly imbalanced instances ($\mu_p \geq 50\%$); we see that the probability that Async-Q1P1 is the best solver increases to 0.66, whereas the probability that Sync is the best solver becomes zero. We also observe that the asynchronous algorithms tend to be less competitive with a large value of $\underline{\Pi}$. We also note that the asynchronous algorithm with $\underline{\Pi} = 16$ is equivalent to the synchronous counterpart.

In Figure 5 we present results for the case in which we allow for more capacity for the queue of trial points ($\bar{\Lambda} = 2$). We see that the asynchronous method is faster than the synchronous method in 87% of the problem instances. Async-Q2P4 is more competitive than Async-Q2P1 for the highly imbalanced problem instances. Async-Q2P8 has a lower number of wins than Sync, but the performance becomes much more



(a) All instances (b) Highly imbalanced instances ($\nu_p \geq 50\%$)

FIG. 4. *Performance profile for* $\bar{\Lambda} = 1$ *and* $\underline{\Pi} \in \{1, 4, 8\}$.



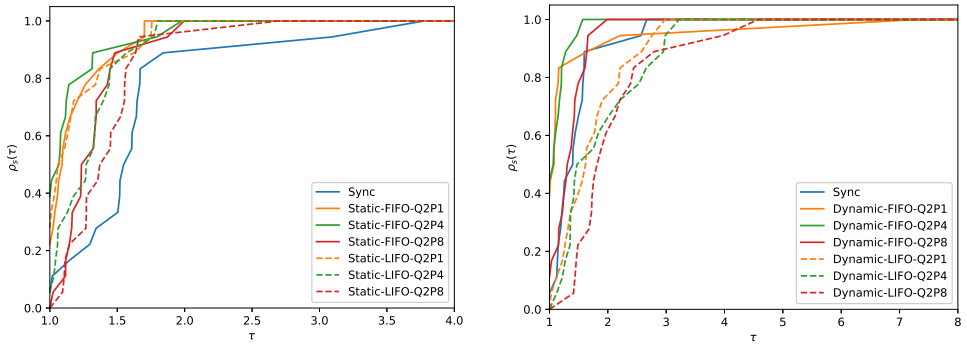
(a) All instances (b) Highly imbalanced instances ($\nu_p \geq 50\%$)

FIG. 5. *Performance profile for* $\bar{\Lambda} = 2$ *and* $\underline{\Pi} \in \{1, 4, 8\}$.

competitive if we extend τ of interest to 1.02 or larger. This implies that the more frequent updating of dual variables is not always advantageous from a computational performance stand-point.

4.5. Variations of asynchronous computing. We present computational results for the asynchronous BTR algorithm with variations in algorithmic settings. In particular, we compare different strategies for choosing trial points (i.e., FIFO versus LIFO) and for allocating subproblems to worker processes (i.e., static versus dynamic). Figures 6 and 7 show the performance plots for the asynchronous algorithms with the different settings. We label the asynchronous algorithm "X-Y-QmPn" for each setting $X \in \{Static, Dynamic\}$ and $Y \in \{FIFO, LIFO\}$. To highlight the impact on performance, we use the highly imbalanced problem instances.

Figure 6 shows the performance profiles, as defined in subsection 4.4, for the synchronous and asynchronous algorithms with trial points chosen based on FIFO and LIFO. We perform the numerical experiments with $\bar{\Lambda} = 2$. Note that FIFO and LIFO are equivalent when $\bar{\Lambda} = 1$. The FIFO policy is faster than the LIFO policy, regardless of the other algorithmic settings (e.g., static versus dynamic subproblem



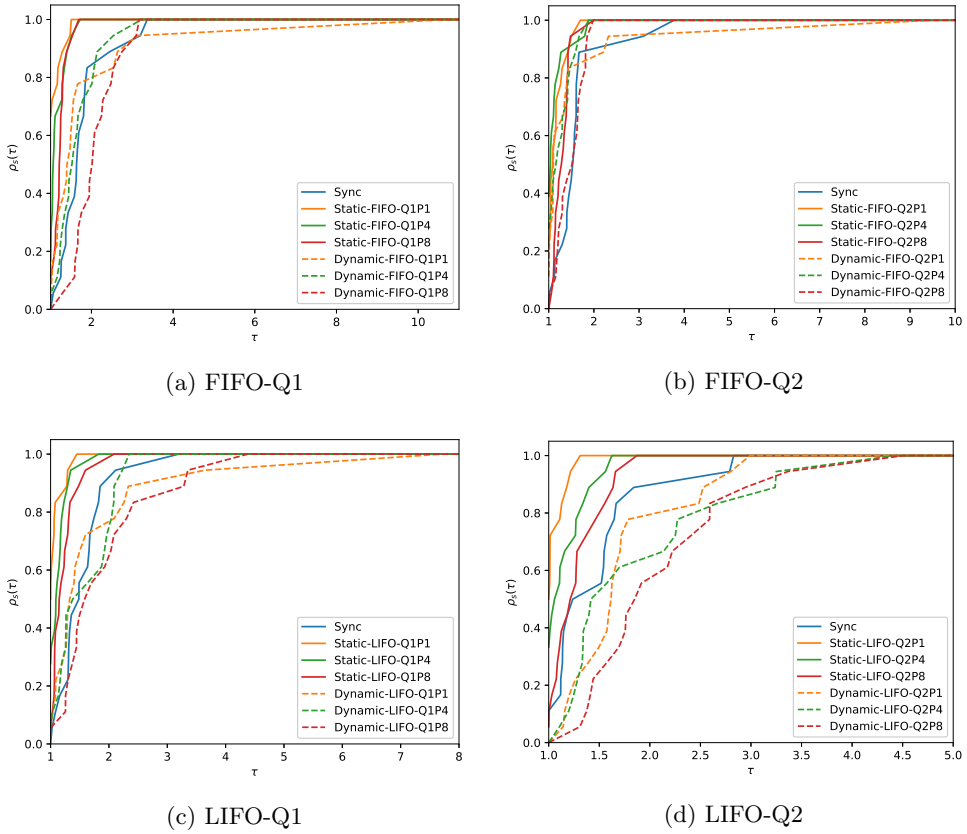
(a) Static (b) Dynamic

FIG. 6. *Performance profile of the asynchronous variations (FIFO versus LIFO) with* $\bar{\Lambda} = 2$ *for highly imbalanced instances* ($\nu_p \geq 50\%$).

allocations). The reason is that the LIFO policy delays the complete evaluation of trial points (i.e., satisfying the condition at line 30 in Algorithm 4) by evaluating new trial points only. In particular, the LIFO policy is slower with the larger queue size (i.e., Q2 versus Q1).

Figure 7 compares the computational performance for the subproblem allocation policies (static versus dynamic). Static allocation is faster than dynamic allocation, regardless of the other settings (e.g., FIFO versus LIFO). The subproblem solution time must be increased in dynamic allocation, for which the wart-starting feature in CPLEX is no longer available when different subproblems are solved from iteration to iteration. We also observe that dynamic allocation is even slower than the synchronous method for many instances. Consistent to the observations in subsection 4.4, we found that the frequent update of dual variables tends to be advantageous, but not always.

4.6. Scalability. We now demonstrate parallel scalability of the asynchronous BTR algorithm for the 80 problem instances. We perform the scaling experiments based on the static subproblem allocation, the FIFO scheme for choosing trial points, $\bar{\Lambda} = 1$, and $\underline{\Pi} = 1$ (i.e., Static-FIFO-Q1P1). We use 2, 4, 8, 16, and 32 computing cores to parallelize the asynchronous method, for which half the computing cores are used in the master problem solution and the others are used in the subproblem solutions. For the instance with eight cores, four of the cores are used to parallelize the subproblem solutions, and the other four are used for solving the master with the barrier solver. Figure 8 shows scaling performance results of the method. We define the speedup as the solution time with N cores to that with two cores. The solution times for each set of 80 instances are shown as a box plot. The linear speedup (red dashed line in Figure 8) is achieved when the speedup increases proportional to the number of cores, which represents the ideal strong scaling efficiency. We observe that the mean scaling plot for the asynchronous method is closely aligned with the linear scaling line, which implies that the asynchronous method scales up as the number of cores increases with respect to the mean solution time. We also highlight that the scalability results are consistent with those of the synchronous variant reported in [14]. Note, however, that the efficiency degrades as the number of cores increases, due to Amdahl's law.



(a) FIFO-Q1 (b) FIFO-Q2

(c) LIFO-Q1 (d) LIFO-Q2

FIG. 7. *Performance profile for the asynchronous variations (Static versus Dynamic) for highly imbalanced instances ($\nu_p \geq 50\%$).*

5. Summary and directions of future work. We have developed synchronous and asynchronous variants of a bundle-trust-region (BTR) algorithm within the context of Lagrangian dual decomposition applied to stochastic mixed-integer programs. The BTR algorithm solves the Lagrangian dual of the SMIP by using a cutting-plane method with a trust region on the dual search space. In the synchronous variant, cutting-planes from all scenario subproblems are used to update the trust region and update the dual search step. We proved that this algorithm converges to the Lagrangian dual bound of the SMIP, and we proved that convergence is independent of the choice of the trust-region norm and of bundle management steps. Unfortunately, this algorithm suffers from parallel inefficiencies due to computational load imbalance in the solution of scenario subproblems (which are solved to obtain the cutting planes). Motivated by this, we developed an asynchronous variant that uses only a subset of the subproblem solutions to update the trust region and compute the dual step. For this method, we devised a trust-region update strategy that uses only trial points of a queue, while the other trial points may be used to update the Lagrangian master problem. We also considered the variations of the algorithmic settings: FIFO/LIFO policies for choosing trial points and static/dynamic subproblem allocations. We proved that all variants of the asynchronous algorithm converge to

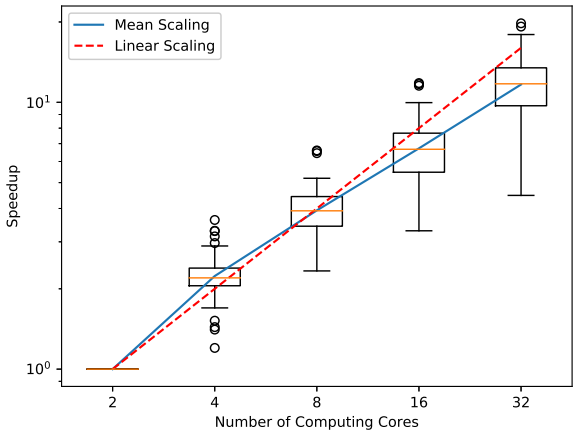


FIG. 8. *Scaling efficiency results for asynchronous BTR algorithm (Static-FIFO-Q1P1).*

the optimal Lagrangian dual bound of the SMIP.

The synchronous and asynchronous BTR algorithms are implemented in the open-source parallel software package `DSP`. In our numerical experiments, we used the WECC test system data and created 80 instances of a SUC problem that schedules a set of power generators and dispatches power to satisfy the demand of the system under uncertain wind power generation. The results show that the asynchronous algorithm solves the problem instances significantly faster than the synchronous counterpart (particularly in the highly imbalanced problem instances). Moreover, we showed that the asynchronous algorithm achieves strong scaling.

If the master problem is relatively more time-consuming than the evaluation of the subproblems, the computational benefit of the asynchronous approach will not be as evident. Such a case can be observed when the subproblems are linear programs and/or when the first stage has a significantly large number of variables. In such a case, the parallelization of master problem solution would improve the computational performance, as shown in [18]. However, the parallelization approach in [18] is based on a parallel Schur complement decomposition that would also suffer from the large number of first-stage variables. Therefore, new parallelization approaches for the master problem are an interesting research path for future work. In addition, a computational comparison with other nonsmooth methods (e.g., [3, 4, 24]) could be of interest. As part of future work, we will also seek to improve the asynchronous method by adaptively changing the parameters $\bar{\Lambda}$ and $\underline{\Pi}$ in order to maximize computational performance. In particular, highly imbalanced instances can be detected after a few synchronous iterations, and this information can be used to tune the parameters of asynchronous iterations. Moreover, inexact evaluation of the Lagrangian subproblems can further be incorporated into the incremental framework to further alleviate load imbalances (e.g., [16, 7, 24]). Motivated by the observation that the dynamic allocation is slower than the static allocation as in subsection 4.5, one can also design a partial dynamic allocation such that each process can take and solve only certain subproblems in the allocation scheme, which would allow us to use the warm-starting feature.

Acknowledgment. We gratefully acknowledge the computing resources provided on Blues, a high-performance computing cluster operated by the Laboratory

Computing Resource Center at Argonne National Laboratory.

REFERENCES

[1] S. Ahmed, *A scenario decomposition algorithm for* 0–1 *stochastic programs*, Oper. Res. Lett., 41 (2013), pp. 565–569.

[2] I. Aravena and A. Papavasiliou, *A distributed asynchronous algorithm for the two-stage stochastic unit commitment problem*, in IEEE Power & Energy Society General Meeting, IEEE, 2015, pp. 1–5.

[3] D. P. Bertsekas, *Incremental proximal methods for large scale convex optimization*, Math. Program., 129 (2011), pp. 163–195.

[4] D. P. Bertsekas, *Incremental Aggregated Proximal and Augmented Lagrangian Algorithms*, preprint, https://arxiv.org/abs/1509.09257, 2015.

[5] C. C. Carøe and R. Schultz, *Dual decomposition in stochastic integer programming*, Oper. Res. Lett., 24 (1999), pp. 37–45.

[6] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Math. Program., 91 (2002), pp. 201–213.

[7] G. Emiel and C. Sagastizábal, *Incremental-like bundle methods with application to energy planning*, Comput. Optim. Appl., 46 (2010), pp. 305–332.

[8] F. Fischer and C. Helmberg, *A parallel bundle framework for asynchronous subspace optimization of nonsmooth convex functions*, SIAM J. Optim., 24 (2014), pp. 795–822, https://doi.org/10.1137/120865987.

[9] A. Frangioni, *Generalized bundle methods*, SIAM J. Optim., 13 (2002), pp. 117–156, https://doi.org/10.1137/S1052623498342186.

[10] M. Gaudioso, G. Giallombardo, and G. Miglionico, *An incremental method for solving convex finite min-max problems*, Math. Oper. Res., 31 (2006), pp. 173–187.

[11] K. Kim, *An Optimization Approach for Identifying and Prioritizing Critical Components in a Power System*, Tech. Report ANL/MCS-P7076-0717, Argonne National Laboratory, 2017.

[12] K. Kim, A. Botterud, and F. Qiu, *Temporal decomposition for improved unit commitment in power system production cost modeling*, IEEE Trans. Power Syst., 33 (2018), pp. 5276–5287, https://doi.org/10.1109/TPWRS.2018.2816463.

[13] K. Kim, F. Yang, V. M. Zavala, and A. A. Chien, *Data centers as dispatchable loads to harness stranded power*, IEEE Trans. Sustain. Energy, 8 (2017), pp. 208–218.

[14] K. Kim and V. M. Zavala, *Algorithmic innovations and software for the dual decomposition method applied to stochastic mixed-integer programs*, Math. Program. Comput., 10 (2018), pp. 225–266, https://doi.org/10.1007/s12532-017-0128-z.

[15] K. C. Kiwiel, *Convergence of approximate and incremental subgradient methods for convex optimization*, SIAM J. Optim., 14 (2004), pp. 807–840, https://doi.org/10.1137/S1052623400376366.

[16] K. C. Kiwiel, *A proximal bundle method with approximate subgradient linearizations*, SIAM J. Optim., 16 (2006), pp. 1007–1023, https://doi.org/10.1137/040603929.

[17] J. Linderoth and S. Wright, *Decomposition algorithms for stochastic programming on a computational grid*, Comput. Optim. Appl., 24 (2003), pp. 207–250.

[18] M. Lubin, K. Martin, C. G. Petra, and B. Sandikçi, *On parallelizing dual decomposition in stochastic integer programming*, Oper. Res. Lett., 41 (2013), pp. 252–258.

[19] A. Nedic and D. P. Bertsekas, *Incremental subgradient methods for nondifferentiable optimization*, SIAM J. Optim., 12 (2001), pp. 109–138, https://doi.org/10.1137/S1052623499362111.

[20] A. Nedić, D. P. Bertsekas, and V. S. Borkar, *Distributed asynchronous incremental subgradient methods*, Stud. Comput. Math. 8, North-Holland, Amsterdam, 2001, pp. 381–407.

[21] A. Papavasiliou and S. S. Oren, *Multiarea stochastic unit commitment for high wind penetration in a transmission constrained network*, Oper. Res., 61 (2013), pp. 578–592.

[22] O. Pearce, T. Gamblin, B. R. De Supinski, M. Schulz, and N. M. Amato, *Quantifying the effectiveness of load balance algorithms*, in Proceedings of the 26th ACM International Conference on Supercomputing, ACM, New York, 2012, pp. 185–194.

[23] K. Ryan, D. Rajan, and S. Ahmed, *Scenario decomposition for* 0–1 *stochastic programs: Improvements and asynchronous implementation*, 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), IEEE, 2016, pp. 722–729.

[24] W. van Ackooij and A. Frangioni, *Incremental bundle methods using upper models*, SIAM J. Optim., 28 (2018), pp. 379–410, https://doi.org/10.1137/16M1089897.