Mathematical Programming Computation (2021) 13:1–50
https://doi.org/10.1007/s12532-020-00185-4

FULL LENGTH PAPER

Asynchronous Lagrangian scenario decomposition

Ignacio Aravena[1] · **Anthony Papavasiliou**[2]

Received: 30 August 2017 / Accepted: 24 February 2020 / Published online: 14 March 2020

Abstract
We present a distributed asynchronous algorithm for solving two-stage stochastic mixed-integer programs (SMIP) using scenario decomposition, aimed at industrial-scale instances of the stochastic unit commitment (SUC) problem. The algorithm is motivated by large differences in run times observed among scenario subproblems of SUC instances, which can result in inefficient use of distributed computing resources by synchronous parallel algorithms. Our algorithm performs dual iterations asynchronously using a block-coordinate subgradient descent method which allows performing block-coordinate updates using delayed information, while candidate primal solutions are recovered from the solutions of scenario subproblems using heuristics. We present a high performance computing implementation of the asynchronous algorithm, detailing the operations performed by each parallel process and the communication mechanisms among them. We conduct numerical experiments using SUC instances of the Western Electricity Coordinating Council system with up to 1000 scenarios and of the Central Western European system with up to 120 scenarios. We also conduct numerical experiments on generic SMIP instances from the SIPLIB library (DCAP and SSLP). The results demonstrate the general applicability of the proposed algorithm and its ability to solve industrial-scale SUC instances within operationally acceptable time frames. Moreover, we find that an equivalent synchronous parallel algorithm would leave cores idle up to 80.4% of the time on our realistic test instances, an observation which underscores the need for designing asynchronous optimization schemes in order to fully exploit distributed computing on real world applications.

Keywords Asynchronous algorithm · Stochastic programming · Unit commitment · High performance computing

Mathematics Subject Classification 90C15 · 68W15 · 68W20 · 90C10 · 90C06

This research was partially funded by the ENGIE Chair on Energy Economics and Energy Risk Management, by the Univeristé catholique de Louvain through an FSR Grant, and by the U.S. Department of Energy through the Lawrence Livermore National Laboratory under contract DE-AC52- 07NA27344.

Extended author information available on the last page of the article

1 Motivation

The unit commitment problem is a classical problem in the short-term scheduling of electric power systems. Unit commitment deals with deciding which generating units will supply energy to a power system over a certain time horizon, so as to minimize the operation cost while respecting the technical constraints of the power system. The problem is usually formulated as a mixed integer linear program (MILP) and it is solved on a daily basis by power system operators worldwide.

In recent years, the integration of significant shares of renewable energy resources into electric power systems has motivated the industry and academic community to investigate efficient approaches for scheduling power systems in the presence of uncertainty (see [50] and references therein for a recent survey on the subject). Stochastic unit commitment (SUC) is a widely studied approach to incorporate uncertainty in this scheduling problem. SUC can be formulated as the two-stage stochastic mixed integer program (SMIP) (1)–(4),

$$\max_{u,v,w} \sum_{i=1}^{N} (\boldsymbol{c}_i^T \boldsymbol{v}_i + \boldsymbol{d}_i^T \boldsymbol{w}_i) \tag{1}$$

$$\text{s.t. } \boldsymbol{u} \in \mathcal{U}, \tag{2}$$

$$(\boldsymbol{v}_i, \boldsymbol{w}_i) \in \mathcal{D}_i, \quad i = 1, \ldots, N \tag{3}$$

$$\boldsymbol{v}_i - \boldsymbol{u} = \boldsymbol{0}, \quad i = 1, \ldots, N \tag{4}$$

where i indexes scenarios, $\boldsymbol{u}$ is the vector of non-anticipative decision variables, $\mathcal{U} \subset \mathbb{R}^n$ is a bounded convex set, $\boldsymbol{v}_i$ are local copies of non-anticipative variables at each scenario, $\boldsymbol{w}_i$ are recourse variables of each scenario, $\mathcal{D}_i \subset \mathbb{R}^{n+m_i}$ $i = 1, \ldots, N$ are bounded non-convex sets and constraints (4) model non-anticipativity constraints. We assume that the problem satisfies $\mathcal{P}_{\boldsymbol{v}_i}(\mathcal{D}_i) = \mathcal{P}_{\boldsymbol{v}_j}(\mathcal{D}_j)$ for any $i, j \in \{1, \ldots, N\}$, where $\mathcal{P}_{\boldsymbol{v}_i}$ is the projection of $\mathcal{D}_i$ on the coordinates of $\boldsymbol{v}_i$. We later refer to this property as *weak relatively complete recourse*.[1] Typically, $\boldsymbol{v}_i$ corresponds to commitment variables of thermal generators (binary decisions), but it can also include production variables of inflexible generators (continuous decisions). The vector $\boldsymbol{w}_i$ includes commitment variables of fast generators, production variables of all generators and flows over the network (mixed integer decisions). $\mathcal{D}_i$ describes the feasible operation domain for scenario i in terms of production constraints (minimum stable level, maximum capacity, maximum ramp rates, minimum up/down times) and power grid constraints (power balance, power flow equations, flow limits). $\mathcal{U}$ corresponds to a convex relaxation of the production constraints for variables included in $\boldsymbol{u}$, $\mathcal{U} \supseteq \mathcal{P}_{\boldsymbol{v}_i}(\mathcal{D}_i)$ for an arbitrarily chosen $i \in \{1, \ldots, N\}$. See Appendix A for a detailed description of the SUC model.

Our aim is to solve problem (1)–(4) for real power systems, within the time limits imposed by daily operations. This differentiates SUC from other applications of stochastic programming in that the typical scale of realistic SUC instances (see

[1] This requirement is weaker than relatively complete recourse, as defined in [12], which demands that any feasible first-stage solution leads to a feasible second-stage subproblem for every scenario.

[18,42,53]), as measured by the number of variables and the number of constraints required to describe $\mathcal{U}$ and $\mathcal{D}_i$, is orders of magnitude larger than that of commonly used SMIP test instances [4]. Furthermore, the computational effort required for optimizing over $\mathcal{D}_i$ can vary significantly from one scenario to another, as well as for the same scenario with slight modifications. Therefore, a parallel implementation of a serial decomposition scheme may perform inefficiently in practice.

In order to overcome these challenges, in this paper we propose an asynchronous distributed algorithm for solving (1)–(4), and we present a high performance computing implementation of the algorithm which is used for solving SUC instances of two industrial-scale systems and the largest instances in the SIPLIB collection [4].

1.1 Relevant literature

Stochastic unit commitment was initially proposed in the seminal work of Takriti et al. [51] and Carpentier et al. [15] as a methodology for coping with demand uncertainty in power systems. The scope for application of SUC has, since then, been extended to renewable energy forecast uncertainty and component failures (referred to as contingencies in the power engineering literature), among other sources of uncertainty in power system operations.

SUC studies commonly use decomposition methods for handling the scale of the problem, which increases linearly with the number of scenarios. Takriti et al. [51] use Lagrange relaxation of non-anticipativity constraints (i.e. scenario decomposition), and a progressive hedging heuristic to obtain non-anticipative solutions. Carpentier et al. [15] relax the problem over generators using an augmented Lagrangian. They maximize the dual function with a proximal point method and recover primal solutions by allowing demand shedding at a quadratic penalty. Shiina and Birge [48] use a column generation approach over production schedules, decomposing over generators and solving the slave production scheduling problems using dynamic programming. Cerisola et al. [16] compare scenario decomposition with variants of Benders decomposition and stress the importance of finding good initial solutions. Additional decomposition approaches for SUC can be found in [50].

SUC instances in the literature have often been limited to test systems which fall short of industrial scale instances. In order to advance towards more realistic instances, several recent studies exploit distributed computing alongside decomposition methods. Papavasiliou et al. [42] implement scenario decomposition in an HPC cluster to solve instances of the Western Electricity Coordinating Council (WECC) system with up to 1000 scenarios in at most 24 h within an optimality gap of 1–2.5%. Cheung et al. [18] decompose the problem by scenarios and use progressive hedging on a multi-processor workstation and on an HPC cluster to solve instances of the WECC system with up to 100 scenarios in at most 25 min within an optimality gap of 1.5–2.5%. Kim and Zavala [28] propose an interior point cutting-plane algorithm to handle dual iterations in scenario decomposition and solve SUC instances based on the IEEE 118-bus test system with up to 32 scenarios, using an HPC cluster, in 6 h within an optimality gap of 0.01%.

Other recent methods, which do not exploit distributed computing explicitly, have also being used to solve large SUC instances. Among them, Schulze et al. [47] use a stabilized Dantzig-Wolfe decomposition to solve multi-stage SUC instances of the British system with up to 50 scenarios in 2 h within an optimality gap of 0.1%. van Ackooij and Malick [53] use primal-dual decomposition along with bundle methods to solve SUC instances of the French system with up to 250 scenarios within an optimality gap of 1%, however no solution times are reported.

As our research is focused on making it practical to solve industrial-scale SUC instances, (*i*) we present a method capable of solving SUC instances faster than the state-of-the-art, (*ii*) we release all the industrial-scale test instances used in this study and (*iii*) we provide the time it takes to solve them with the proposed method. These three elements are not found simultaneously in any of the aforementioned studies.

In the broader SMIP class, to which SUC belongs, scenario decomposition has inspired several different decomposition algorithms. Carøe and Schultz [14] propose a scenario decomposition method where non-anticipativity constraints are gradually enforced through a branch-and-bound algorithm. Lubin et al. [31] extend the previous method to a parallel computing setting by solving the dual problems at each node of the branch-and-bound tree using an interior point solver that exploits the structure of the problem. Oliveira et al. [40], and references therein, use scenario decomposition and solve the dual problem using bundle methods. Ahmed [1,2] proposes an alternative approach for stochastic integer programs where solutions to scenario subproblems are directly used for primal recovery while, at the same time, these solutions are separated from subproblems in order to improve the bound of the relaxation. Ryan et al. [46] develop several improvements over Ahmed's original algorithm, including a distributed asynchronous implementation.

Distributed computing has also found applications in stochastic programing outside the realm of scenario decomposition methods. Lubin et al. [30] propose a distributed memory simplex algorithm for stochastic linear programs. Munguía et al. [34] propose a branch-and-bound algorithm for stochastic mixed integer programs on which each node of the tree is solved using Lubin's method. Moritchs et al. [33] propose a nested asynchronous decomposition algorithm for multistage stochastic linear programs. Chaturapruek et al. [17] propose and prove optimal convergence for asynchronous stochastic gradient descent methods on unconstrained stochastic programs with strongly convex and differentiable objective functions.

A crucial aspect to scenario decomposition is the method used to optimize the dual function, which is separable, convex and non-differentiable. Certain specialized methods allow exploiting the separable structure of the dual function in a distributed computing infrastructure. Nedić et al. [35–37] analyze incremental subgradient algorithms, on which each update is made along the direction of the subgradient of a part of the objective function. Coordinate descent methods are a different approach to exploit the structure of minimization problems for which it is cheaper to compute the gradient with respect to a subset of variables (coordinates) than it is to compute the full gradient of the objective. Wright [55] provides a recent survey on coordinate descent methods. Nesterov [39] provides worst-case complexity results for randomized coordinate descent methods for smooth optimization. Fercoq and Richtárik [23] propose a parallel synchronous coordinate descent method for minimizing non-differentiable

simple composite functions. The authors use Nesterov's smoothing technique [38] to obtain a smooth approximation of the non-decomposable part of the objective and perform a line search separately on each coordinate of each iteration, as proposed by Tseng [52]. Fisher and Helmberg [24] propose an asynchronous distributed bundle method for non-differentiable convex optimization, where at each step a worker greedily selects a subset of variables, blocks them from being accessed by the other workers, and performs a proximal bundle iteration on the selected variables.

1.2 Contributions

Stochastic unit commitment, despite its attractiveness, has failed to become an industry standard due to several reasons, including the difficulty of solving the mathematical programs in an operationally acceptable time frame. In the present paper, we aim at overcoming this challenge by developing an asynchronous scenario decomposition scheme based on distributed computing. The main innovation of the developed scheme is that we optimize the dual function using an asynchronous block-coordinated subgradient algorithm. Our algorithm does not require differentiability or strong convexity assumptions that are commonly used in the literature [17,29,39,55]; it differs from the algorithm of Fercoq and Richtárik [23] in that we perform iterations asynchronously and without the need for line search, which would be prohibitive in our context; and it requires less serial coordination overhead than the asynchronous bundle method [24]. We provide convergence guarantees for the proposed algorithm based on previous results for stochastic subgradient methods [21] and incremental subgradient methods [35–37].

We also perform primal recovery asynchronously and in parallel to the dual iterations, either by recovering solutions from scenario subproblems, as proposed in [1,2], or by using recombination heuristics, similar to those proposed in [14]. The proposed asynchronous algorithm allows us to solve limited-size SUC instances faster than the state-of-the-art [18] and to solve SUC instances for systems larger than the state-of-the-art [53] within operationally acceptable time frames.

Even though the proposed algorithm is inspired and tailored to SUC, the proposed framework for asynchronous dual decomposition can be applied in other contexts, such as temporal or spatial decomposition of unit commitment.

In order to make our numerical experiments replicable, we release all SUC instances used in the present study [9] (WECC and CWE, see Sect. 6) along with the source code of our implementation [10].

1.3 Notation and paper organization

Throughout this document we use boldface to denote vectors, lowercase letters to denote variables and uppercase letters to denote parameters or sets. Additionally, we use partial indexation of vectors to keep notation simple, i.e. $\boldsymbol{x} = [\boldsymbol{x}_1^T \ \dots \ \boldsymbol{x}_N^T]^T$ and $\boldsymbol{x}_i \in \mathbb{R}^{n_i}$.

The rest of the paper is organized as follows. Section 2 introduces the stochastic unit commitment problem and its scenario decomposition in a stylized fashion. Sec-

tion 3 presents the asynchronous distributed block-coordinate subgradient method and provides convergence results for the dual iterations. Section 4 describes our primal recovery heuristics. Section 5 describes the HPC implementation of the dual algorithm, where we also cover aspects of communications and load balancing. Section 6 presents the numerical results for the WECC and Central Western European (CWE) systems. Finally, Sect. 7 presents the conclusions of the study and points to directions of future research.

2 Scenario decomposition in stochastic programming

Problem (1)–(4) is a general formulation for two-stage mixed integer stochastic programs with finitely many scenarios and bounded feasible sets [32].

Following [51], we relax problem (1)–(4) by associating multipliers $\boldsymbol{x}_i$ to non-anticipativity constraints (4). We thus obtain the dual problem (5), where f_0 and f_i are defined according to (6) and (7), respectively.

$$\min_{x \in \mathbb{R}^m} \quad f_0(\boldsymbol{x}) + \sum_{i=1}^{N} f_i(\boldsymbol{x}_i) \tag{5}$$

$$f_0(\boldsymbol{x}) := \sup_{u \in \mathcal{U}} \left(- \sum_{i=1}^{N} \boldsymbol{x}_i^T \right) \boldsymbol{u} \tag{6}$$

$$f_i(\boldsymbol{x}_i) := \sup_{(v,w) \in \mathcal{D}_i} \left((\boldsymbol{c}_i^T + \boldsymbol{x}_i^T)\boldsymbol{v} + \boldsymbol{d}_i^T \boldsymbol{w} \right) \; i = 1, \ldots, N \tag{7}$$

Scenario decomposition schemes for solving (1)–(4) work by solving the dual problem (5) and generating primal solutions based on the solution to subproblems (6) and (7). The objective of problem (5) has a separable structure. Moreover, by evaluating the component functions f_0 and f_i, $i = 1, \ldots, N$ at a certain $\bar{\boldsymbol{x}}$, we obtain a subgradient of the objective at $\bar{\boldsymbol{x}}$. These two properties motivate the use of subgradient algorithms for solving (5) and evaluating the component functions in parallel in order to speed up the algorithm [42]. A third important property of problem (5) should be carefully considered, namely, the differences in evaluation times between component functions.

In our context, the evaluation of f_i at a certain $\bar{\boldsymbol{x}}_i$, $i = 1, \ldots, N$, requires the solution of a large mixed integer linear problem, while the evaluation of f_0 at a certain $\bar{\boldsymbol{x}}$ requires solving a medium-size linear program, hence the evaluation of f_0 requires only a small fraction of the time that it takes to evaluate f_i for any i.[2] In addition, for a given $\bar{\boldsymbol{x}}$ and two component functions f_i and f_j, the evaluation times of f_i and f_j can be dramatically different (we have observed differences of more than 7500%). These differences in evaluation time can also arise for the same component function evaluated at two different iterates. Altogether, these differences can render synchronous parallel algorithms ineffective because the time between iterations is limited by the component

[2] This differs from the usual scope of application of coordinate descent methods, where the decomposable part of the objective is easier to evaluate than the non-decomposable part [23,29,39,55].

function which requires the greatest time to be evaluated. The main aim of the present work is to overcome this limitation.

3 Asynchronous distributed block-coordinate subgradient method

In this section we present a minimization method that exploits the special structure of (5) in order to effectively harness distributed computing. The proposed method has been inspired by previous work on incremental subgradient algorithms by Nedić et al. [35–37].

We replace the non-decomposable component function f_0 by a smooth approximation f_0^μ, defined in Eq. (8), as done in [23].

$$f_0^\mu(\boldsymbol{x}) := \sup_{u \in \mathcal{U}} \left(\left(-\sum_{i=1}^{N} \boldsymbol{x}_i^T \right) \boldsymbol{u} - \frac{1}{2}\mu \|\boldsymbol{u} - \boldsymbol{u}_0\|_2^2 \right) \tag{8}$$

We show in Sect. 3.1 that smoothness of the non-decomposable part of the objective is necessary for the convergence of our method. Given $\mu > 0$ and certain $\boldsymbol{u}_0 \in \mathcal{U}$, f_0^μ is a convex differentiable function and its gradient has a Lipschitz constant L_0^μ [38, Thm. 1]. We then focus on solving problem (9), where X is a convex set, onto which it is easy to project.

$$\min_{x \in X} f(\boldsymbol{x}) = f_0^\mu(\boldsymbol{x}) + \sum_{i=1}^{N} f_i(\boldsymbol{x}_i) \tag{9}$$

The solution of problem (9) will be an approximate solution to problem (5) [38]. Note that problem (9), as well as problem (5), are non-differentiable convex optimization problems.

For ease of presentation, we first introduce a serial variant of the proposed method. We then generalize it to an asynchronous distributed setting. Proofs for all results presented in this section are provided in Appendix B.

3.1 Serial method

Let us consider a block version of the randomized coordinate descent method [39] for minimizing f. At iteration k, with current iterate $\boldsymbol{x}^k$, we select uniformly at random a block $j(k)$ from $\{1, \ldots, N\}$ and compute the next iterate $\boldsymbol{x}^{k+1}$ using the following update rule:

$$\boldsymbol{x}^{k+1} := \mathcal{P}_X \left[\boldsymbol{x}^k - \lambda_k \cdot I_{j(k)}^T \left(I_{j(k)} \nabla f_0^\mu(\boldsymbol{x}^k) + g\left(j(k), \boldsymbol{x}_{j(k)}^k\right) \right) \right]. \tag{10}$$

Here, λ_k is the step size, $g(j, \boldsymbol{x}) \in \partial f_j(\boldsymbol{x})$ and $\boldsymbol{I}_j$ is a matrix that maps ∇f_0^μ to its components relevant to block j, i.e.

$$I_j = \left[\mathbf{0}_{n_j \times \left(\sum_{i=1}^{j-1} n_i\right)} \; \mathbf{1}_{n_j \times n_j} \; \mathbf{0}_{n_j \times \left(\sum_{i=j+1}^{N} n_i\right)} \right],$$

with $\mathbf{1}_{n_j \times n_j}$ corresponding to the identity matrix. Simply put, rule (10) updates the multipliers by moving them in the opposite direction to a subgradient of the objective function, restricted to the coordinates related to scenario $j(k)$.

Although f is a non-differentiable function, the update rule (10) is equivalent to the update rule of the stochastic subgradient method, as in the following proposition.

Proposition 1 *Let J be a discrete uniform random variable on the set $\{1, \ldots, N\}$. The expected direction of update rule (10), $\mathbb{E}[I_J^T (I_J \nabla f_0^\mu(\boldsymbol{x}^k) + g(J, \boldsymbol{x}_J^k)) | \boldsymbol{x}^k]$, coincides with the direction of a subgradient of f at $\boldsymbol{x}^k$.*

Note that smoothness of f_0^μ is a necessary condition for Proposition 1. This can be understood intuitively by observing that the expectation in Proposition 1 constructs the expected update direction by concatenating subgradients of f restricted to blocks of coordinates, which are obtained from different calls to a first-order oracle. The concatenation of blocks of coordinates of different subgradients of a non-separable function (e.g. $\sum_{j=1}^N I_j^T I_j g^j$, where $g^j \in \partial f_0(\boldsymbol{x}),\ j = 1, \ldots, N$) is not necessarily a subgradient of the function (see Proposition 4, in Appendix B), whereas the concatenation of blocks of coordinates of the gradient of a differentiable function ($\sum_{j=1}^N I_j^T I_j \nabla f_0^\mu(\boldsymbol{x}) = \nabla f_0^\mu(\boldsymbol{x})$) results in the gradient of that function.

The stochastic subgradient method and its convergence have been well studied in the literature, see for instance [21, Thm. 2]. By extension, with an appropriate selection of the step size λ_k (diminishing, nonsummable, square summable), the method defined by the update rule (10) will converge to an optimal solution with probability 1.

Note that the update rule (10) requires computing the gradient of f_0^μ and a subgradient of $f_{j(k)}$ at each iteration k. This makes the method more expensive than the subgradient method when we consider an entire pass over all scenarios, i.e. N iterations of the method, which is the minimum amount of iterations required to update every block of variables. Nevertheless, since the cost of computing the gradient of f_0^μ is almost negligible compared to the cost of computing a subgradient of f_i for any i, the extra cost in practice would not be noticeable.

3.2 Asynchronous distributed method

The idea behind the asynchronous distributed method is essentially the same as in the serial method presented in the previous subsection, with the exception that subgradients of component functions are computed in parallel to the updates. Specifically, following [37], we use the computation model presented in Fig. 1. In this computation model, only the *Updating system* can increase iteration counters, and updates are performed serially using the information provided by the *Subgradient computation system*. The serial nature of the *Updating system* allows us to describe the algorithm using a single iteration counter k.

Note that subgradients communicated by the *Subgradient computation system* to the *Updating system* might have been computed at previous iterates. In other words, the subgradient information available to the *Updating system* might have delays. We denote these delays by $l(k)$, the total number of updates since the last evaluation of the gradient of f_0^μ, at iteration k; and $\ell(j, k)$, the number of updates to block j since

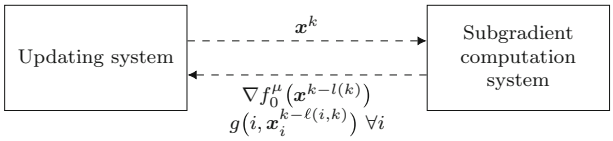

Fig. 1 Computation model for the asynchronous distributed method. At each iteration k, the *Updating system* communicates the current iterate $\boldsymbol{x}^k$ to the *Subgradient computation system*. In turn, the *Subgradient computation system* communicates back the last computed subgradient for each component function. The parallelism of this scheme resides within the *Subgradient computation system*

the last computation of its subgradient, at iteration k. Considering delays, we propose update rule (11) for the randomized block-coordinate subgradient method:

$$\boldsymbol{x}^{k+1} := \mathcal{P}_X\left[\boldsymbol{x}^k - \lambda_k \cdot I_{j(k)}^T\left(I_{j(k)}\nabla f_0^\mu\left(\boldsymbol{x}^{k-l(k)}\right) + g\left(j(k), \boldsymbol{x}_j^{k-\ell(j(k),k)}\right)\right)\right]. \quad (11)$$

Here, $j(k)$ is selected uniformly at random from $\{1, \ldots, N\}$. This update rule is an extension of (10), where we allow for delays in the subgradient information.

The presence of delays implies that Proposition 1 is no longer valid for update rule (11). Nevertheless, we can use essentially the same idea in order to prove convergence of the method defined by rule (11) as the one we used in the previous subsection. That is, we show that the expected update direction coincides with the direction of the *approximate subgradient* of the objective function and that the error in the approximate subgradient vanishes as the iterations advance. In this work, approximate subgradients are defined as follows.

Definition 1 (*Approximate subgradient* [45]) Let f be any convex function and let $\boldsymbol{x} \in \mathrm{Dom}(f)$. A vector $\boldsymbol{g}$ is called an ϵ-subgradient, or approximate subgradient, of f at $\boldsymbol{x}$ if there exists $\epsilon > 0$ such that

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \boldsymbol{g}^T(\boldsymbol{y} - \boldsymbol{x}) - \epsilon, \quad \forall \boldsymbol{y} \in \mathrm{Dom}(f).$$

Our analysis of the iterates of the asynchronous method is based on the *Supermartingale Convergence Theorem* [11, Prop. 4.2] (replicated as Theorem 2 in Appendix B for self-containedness) and the following assumptions:

Assumption 1 (*Subgradient boundedness*) The subgradients of component functions are bounded above by some positive constants. In particular, there exist positive constants C and D such that

$$\sup_{\substack{j\in\{1,\ldots,N\}\\ x,y\in X}} \left\|I_j\nabla f_0^\mu(\boldsymbol{x}) + g\left(j, \boldsymbol{y}_j\right)\right\|_2 \leq C \quad \text{and} \quad \sup_{\substack{j\in\{1,\ldots,N\}\\ x\in X}} \left\|g\left(j, \boldsymbol{x}_j\right)\right\|_2 \leq D.$$

Assumption 2 (*Delay boundedness*) There exist a positive integer L (possibly unknown) such that $l(k) \leq L, \forall k = 1, \ldots, \infty$ and $\ell(j,k) \leq L, \forall j = 1, \ldots, N, \forall k = 1, \ldots, \infty$.

Assumption 3 (*Diminishing-bounded stepsize*) The stepsize λ_k might be a function of $\boldsymbol{x}^k, \boldsymbol{x}^{k-1}, \ldots, \boldsymbol{x}^0$, but not of the block coordinate to be updated $j(k)$. Furthermore, the sequence $\{\lambda_k\}$ is bounded above and below by a deterministic sequence $\{\gamma_k\}$, such that

$$\check{G}\gamma_k \leq \lambda_k \leq \hat{G}\gamma_k, \ \gamma_k = \frac{1}{(1+rk)^q} \ \forall k, \quad \sum_{k=0}^{\infty} \gamma_k = \infty, \quad \sum_{k=0}^{\infty} \gamma_k^2 < \infty,$$

where $\check{G}, \hat{G}, r, q$ are positive constants.

For SMIP instances, Assumption 1 is ensured by the boundedness of the sets $\mathcal{U}$ and $\mathcal{D}_i, i = 1, \ldots, N$ of the primal problem (1)–(4), while Assumption 2 will hold as long as the evaluation time of all component functions is finite. The selection of a stepsize which is consistent with Assumption 3 is discussed in Sect. 3.3.

The following lemma conveys the key idea of our analysis and the rest of the proof follows almost directly from it.

Lemma 1 *Let Assumptions 1 and 2 hold. Additionally, let J be a discrete uniform random variable on the set $\{1, \ldots, N\}$ and $\mathcal{F}_k = \{\boldsymbol{x}^k, \boldsymbol{x}^{k-1}, \ldots, \boldsymbol{x}^0\}$. Then, the expected direction of update rule (11),*

$$\mathbb{E}[I_J^T (I_J \nabla f_0^{\mu}(\boldsymbol{x}^{k-l(k)}) + g(J, \boldsymbol{x}_J^{k-\ell(J,k)}))| \mathcal{F}_k],$$

coincides with the direction of an ϵ-subgradient of f at $\boldsymbol{x}^k$, with

$$\epsilon = C^2 L_0^{\mu} \sum_{m=k-L}^{k-1} \lambda_m^2 + 2CDN \sum_{m=k-L}^{k-1} \lambda_m.$$

Lemma 1 shows that the update direction of rule (11) should, on average, be close to a subgradient of the objective and that the error in the subgradient is bounded by the sum of the last L stepsizes, hence by choosing a stepsize consistent with Assumption 3 this error will vanish as k grows. In the following, Proposition 2 gives an estimate of the progress of the asynchronous method at each iteration, based on the result of Lemma 1, while Proposition 3 presents a straightforward consequence of Assumption 3.

Proposition 2 *Let Assumptions 1, 2 and 3 hold and let $\mathcal{F}_k = \{\boldsymbol{x}^k, \boldsymbol{x}^{k-1}, \ldots, \boldsymbol{x}^0\}$. Then, for the sequence $\{\boldsymbol{x}^k\}$ generated by the update rule (11), we have that*

$$\begin{aligned} \mathbb{E}\big[\|\boldsymbol{x}^{k+1} - \boldsymbol{y}\|_2^2 | \mathcal{F}_k\big] \leq \|\boldsymbol{x}^k - \boldsymbol{y}\|_2^2 &- 2\frac{\lambda_k}{N}\big(f(\boldsymbol{x}^k) - f(\boldsymbol{y})\big) \\ &+ \lambda_k^2 C^2 + 2\frac{C^2 L_0^{\mu}}{N} \lambda_k \sum_{m=k-L}^{k-1} \lambda_m^2 \\ &+ 4CD\lambda_k \sum_{m=k-L}^{k-1} \lambda_m \quad \forall y \in X. \end{aligned} \tag{12}$$

Proposition 3 *Let Assumption 3 hold. Then, we have*

$$\sum_{k=0}^{\infty}\lambda_k=\infty,\quad \sum_{k=0}^{\infty}\lambda_k^2<\infty,\quad \sum_{k=0}^{\infty}\lambda_k\sum_{m=k-L}^{k-1}\lambda_m<\infty,\quad \sum_{k=0}^{\infty}\lambda_k\sum_{m=k-L}^{k-1}\lambda_m^2<\infty,$$

where for notational convenience we let $\lambda_{-l}=\lambda_0\ \forall l\in\mathbb{N}$.

Finally, in the following theorem, we apply the *Supermartingale Convergence Theorem* (Theorem 2, Appendix B) to the results of Propositions 2 and 3 in order to prove the convergence of the asynchronous method to an optimal solution.

Theorem 1 *Let Assumptions 1, 2 and 3 hold, and assume further that the optimal solution set* X^* *is nonempty. Then the sequence* $\{x_k\}$ *generated by the randomized method converges to some optimal solution with probability 1.*

The result presented in Theorem 1 extends the state-of-the-art by providing a convergence guarantee for the asynchronous block-coordinate descent method for non-differentiable optimization problems with the structure of problem (9) without the need for a line search on $\boldsymbol{x}_{j(k)}$ at each iteration [23,52].

An important remark is that the asynchronous incremental method proposed in [37] is guaranteed to converge to an optimal solution for both problem (5) and problem (9), while also exploiting their decomposable structure to a certain extent. We utilize block-coordinate descent instead of incremental methods for two reasons:

– The choice between incremental and block-coordinate subgradient methods can have significant implications on the magnitude of delays whenever we are minimizing a problem with the structure of problem (9), where a gradient of f_0^{μ} is much easier to evaluate than a subgradient of f_i for any $i=1,\ldots,N$, and $X=\mathbb{R}^n$ (unconstrained optimization).
  The incremental subgradient method at iteration k selects at random a component function, $j(k)\in\{0,1,\ldots,N\}$, and performs an update following the direction of the computed subgradient for the selected component function. Whenever $j(k)\geq 1$, the algorithm will update block-coordinate $j(k)$, which will cause the current gradient of f_0^{μ} and the subgradient $f_{j(k)}$ to gain one unit of delay. On the other hand, every time $j(k)=0$, the algorithm will update the entire vector $\boldsymbol{x}$, adding one unit of delay to all the subgradient information available to the *Updating system*. This effect, which is unavoidable for the problem structure analyzed in [37], is undesirable because errors on the update direction depend directly on the magnitude of the delays.
  The block-coordinate subgradient method at iteration k updates only the coordinates of block $j(k)\in\{1,\ldots,N\}$, causing the available gradient of f_0^{μ} and subgradient $f_{j(k)}$ to gain a unit of delay, but leaving unaffected the rest of the subgradient information available to the *Updating system*. Moreover, new gradients for f_0^{μ} can be computed rapidly as iterations advance, which allows us to maintain small delays throughout the solution process.
– For SMIP instances and, in general, for Lagrangian relaxation of constraints linking duplicated variables, as the dual multiplier $\boldsymbol{x}$ approaches an optimal value, $\boldsymbol{u}$ and

$\boldsymbol{v}_i$ start becoming similar for all i. As a consequence, the gradient of f_0^{μ} will tend to point in an opposite direction to the subgradient of f_i for any $i = 1, \ldots, N$, thereby causing the incremental method to be susceptible to oscillations in $\boldsymbol{x}$.

3.3 Stepsize selection and function value estimation

Although Assumption 3 might seem to restrict the stepsize to a diminishing series of the type $1/k^q$, it also allows us to use a stepsize similar to the dynamic stepsize proposed by Polyak for the subgradient method [43],

$$\lambda_k = p\frac{f\left(\boldsymbol{x}^k\right) - f^*}{\|g\left(\boldsymbol{x}^k\right)\|_2^2}, \quad g\left(\boldsymbol{x}^k\right) \in \partial f\left(\boldsymbol{x}^k\right), 0 < p < 2.$$

In order to use the original Polyak stepsize, we need to know the objective value at the current iterate $f\left(\boldsymbol{x}^k\right)$, the optimal value f^*, and the norm of the subgradient at the current iterate $\|g\left(\boldsymbol{x}^k\right)\|_2$, none of which are available for the asynchronous method. Instead, we construct a new stepsize which uses estimates for each of the aforementioned quantities. An estimate of the current objective, which is also an upper bound on the objective of the primal problem (1)–(4) , can be obtained at the cost of evaluating f_0 as follows:

$$f\left(\boldsymbol{x}^k\right) \approx UB_k := f_0\left(\left[\left(\boldsymbol{x}_1^{k-\ell(1,k)}\right)^T \ldots \left(\boldsymbol{x}_N^{k-\ell(N,k)}\right)^T\right]^T\right) + \sum_{j=1}^{N} f_j\left(\boldsymbol{x}_j^{k-\ell(j,k)}\right). \quad (13)$$

On the other hand, an estimate $\bar{g}_k$ of the subgradient norm can be computed using the last known subgradients for the component functions:

$$\|g\left(\boldsymbol{x}^k\right)\|_2 \approx \bar{g}_k := \max\left\{\sigma, \left\|\nabla f_0^{\mu}\left(\boldsymbol{x}^{k-l(k)}\right) + \sum_{j=1}^{N} I_j g\left(j, \boldsymbol{x}_j^{k-\ell(j,k)}\right)\right\|_2\right\}$$

Here, σ is a small positive constant intended to prevent that $\bar{g}_k = 0$ (note that, due to the delays, $\bar{g}_k = 0$ does not imply that $\boldsymbol{x}^k$ is optimal). An underestimate for the optimal value LB_k can be obtained from a feasible solution to the primal problem, which is computed as described in Sect. 4. We assume that this is a strict underestimate, i.e. $\theta \leq f^* - LB_k$ for some $\theta > 0$. In other words, we assume that strong duality is never attained for realistic SUC instances.

Using these estimates, we propose the following dynamic stepsize:

$$\lambda_k = p_k \cdot \frac{\min\left\{\xi,\ UB_k - LB_k\right\}}{\bar{g}_k^2}, \quad (14)$$

where ξ is a positive constant and $p_k = p_0/(1 + rk)^q$, with p_0, r positive constants and $1/2 < q \leq 1$. The goal of ξ is to prevent the method from taking long steps

whenever the underestimate of the optimal value is loose. At the same time, we scale our estimate of the Polyak stepsize by the decreasing sequence p_k. We do this order to ensure that the resulting stepsize λ_k, defined in Eq. (14), respects Assumption 3. This can be verified by observing that λ_k satisfies the following inequality:

$$p_0 \frac{\theta}{C} \cdot \gamma_k \leq \lambda_k \leq p_0 \frac{\xi}{\sigma} \cdot \gamma_k.$$

Therefore, by Theorem 1, the asynchronous algorithm using the proposed stepsize (14) will converge with probability 1 to an optimal solution.

A diminishing stepsize of type $\lambda_k = p_0/(1 + rk)^q$ is also guaranteed to achieve convergence under Assumptions 1–3. However, for such a stepsize to work effectively, it is necessary to determine a 'good' initial stepsize p_0 in absolute terms. The process of selecting a 'good' initial stepsize can require several trial-and-error runs of the algorithm for every instance to be solved, which would not be possible in an industrial implementation with a strict time limitation.

By contrast, in order to set the parameters for the proposed stepsize (14), we only need to decide the proportion p_k of the Polyak stepsize that we would like to have at two different iteration counts (e.g. $p_0 = 0.5$ and $p_{50 \cdot N} = 0.25$) and the rate at which we would like to decrease p_k (e.g. decreasing with a rate $1/k$). Given that these parameters are scale-free, in the sense that they do not depend on the absolute value of the objective function, there is no need for trial-and-error runs when using this stepsize rule in industrial-size applications.

4 Primal recovery

Primal recovery is an essential component of any Lagrangian relaxation method aiming at solving the original problem. Although there exist exact methods for recovering primal solutions in the case of linear programs [7], these methods do not extend to the mixed integer case. Therefore, primal recovery generally relies on heuristics.

In the case of the SMIP instances in this paper, we exploit the weak relatively complete recourse of the problem, i.e. that $\mathcal{P}_{\boldsymbol{v}_i}(\mathcal{D}_i) = \mathcal{P}_{\boldsymbol{v}_j}(\mathcal{D}_j)$ for any $i, j \in \{1, \ldots, N\}$. In other words, any solution to a scenario subproblem $\bar{\boldsymbol{v}}$ can be used as a candidate non-anticipative first-stage solution. We can then compute the second-stage cost $h_i(\bar{\boldsymbol{v}})$ by solving second-stage problems with fixed $\boldsymbol{v}_i = \bar{\boldsymbol{v}}$:

$$h_i(\bar{\boldsymbol{v}}) = \boldsymbol{c}_i^T \bar{\boldsymbol{v}} + \min_{w|(\bar{v},w)\in\mathcal{D}_i} \boldsymbol{d}_i^T \boldsymbol{w}.$$

Thus, we obtain a complete primal solution to (1)–(4) and a lower bound on the objective of (5) [1,2].

Recovering one feasible non-anticipative solution at every dual iteration would require the solution of N^2 second-stage MILPs for every dual pass over data. For medium to large scenario sets, the computational requirements of primal recovery can easily become larger than the requirements of the dual algorithm. If both dual iterations and primal recovery are performed concurrently, primal candidates would need to

enter into a queue for evaluation, which will typically grow as dual iterations advance (assuming similar resources are allocated for dual iterations and primal recovery). Within this context, we test three rules for determining the order of evaluation of primal candidates in the queue:

- First-in-first-out (FIFO).
- Random order (RND), motivated by the possibility that a good solution might appear anywhere in the sequence of solutions to scenario subproblems.
- Last-in-first-out (LIFO). This approach accounts for the fact that, as dual iterations advance, solutions to scenario subproblems tend to be almost non-anticipative ($\boldsymbol{v}_i \approx \boldsymbol{v}_j, i, j = 1, \ldots, N$). Therefore, scenario subproblem solutions in later iterations could achieve better overall performance.

A different approach towards recovering primal solutions is to create primal candidates as they are required, by combining the solutions to different scenario subproblems. Carøe and Schultz [14] propose creating primal candidates by first averaging the solutions to all scenario subproblems and then rounding the result using a heuristic. We use a variant of this idea, combined with importance sampling (IS), to create primal solutions. Let $\mathcal{V} = \mathcal{P}_{\boldsymbol{v}_i}(\mathcal{D}_i)$ for an arbitrarily chosen $i \in \{1, \ldots, N\}$. Our recovery heuristic, then, proceeds as follows:

1. Associate to each scenario $i = 1, \ldots, N$ a probability proportional to an estimate of its importance, e.g. $\rho_i \propto f_i(\boldsymbol{x}_i^{k-\ell(i,k)})$.
2. Pick a sample of the scenarios of size $M < N$, using ρ_i as the probability of sampling scenario i.
3. Average the current scenario subproblem solution $\bar{\boldsymbol{v}}_i$ associated to the sampled scenarios $\{i(m), m = 1, \ldots, M\}$ in order to generate an average $\tilde{\boldsymbol{v}}$:

$$\tilde{\boldsymbol{v}} = \frac{1}{M} \sum_{m=1}^{M} \boldsymbol{v}_{i(m)}.$$

4. Generate a primal candidate $\bar{\boldsymbol{u}}$ by projecting the average of step 3 onto $\mathcal{V}$,

$$\bar{\boldsymbol{u}} = \underset{u \in \mathcal{V}}{\arg\min} \ \|\boldsymbol{u} - \tilde{\boldsymbol{v}}\|_2^2 . \tag{15}$$

In general, problem (15) follows the form of a modified scenario subproblem, because $\mathcal{V} = \mathcal{P}_{\boldsymbol{v}_i}(\mathcal{D}_i) = \{\boldsymbol{v} \in \mathbb{R}^n \mid \exists \boldsymbol{w}, (\boldsymbol{v}, \boldsymbol{w}) \in \mathcal{D}_i\}$, for an arbitrarily chosen $i \in \{1, \ldots, N\}$. Nevertheless, in most cases $\mathcal{V} = \mathcal{U} \cap (\mathbb{Z}^{n_I} \times \mathbb{R}^{n-n_I})$, where n_I indicates the number of integer first-stage variables. Problem (15) then reduces to a quadratic version of problem (6), which is smaller and easier to solve than a scenario subproblem.

The proposed heuristic allows us to create primal candidates that combine the characteristics that make a solution optimal for a representative subset of the scenarios while, at the same time, generating different candidates after each dual iteration if necessary. The latter would not be possible if we were to follow [14], since the average of the solution to all scenario subproblems does not change significantly from one coordinate descent iteration to the next.

5 High performance computing implementation

Section 3 provides convergence guarantees for the asynchronous dual optimization algorithm, based on the conceptual distributed computation model of Fig. 1, while Sect. 4 proposes two primal recovery schemes that can be parallelized. This section specifies the actual implementation of the algorithm, that is, the different processes running in parallel and the information to be exchanged between them.

We implement the algorithm using the *Master/Slave* design presented in Fig. 2. The *Master* coordinates the work of all workers, dynamically assigning tasks (solving optimization problems) to *Slaves* as the algorithm progresses. *Slaves*, on the other hand, limit themselves to perform the tasks demanded by the *Master*, without a view of the global progress of the algorithm.

In contrast to the conceptual computation model presented in Fig. 1, in the actual implementation there is no clear separation between the *Updating system* and the *Subgradient computation system*. The *Updating system* is contained within the *Master*. The *Subgradient computation system* is split between the *Master* and the *Slaves* that are currently evaluating f_0^μ or $f_i, i = 1, \ldots .N$. This part of the process shown to the left of the *Master* in Fig. 2.

Primal recovery is performed concurrently with dual iterations, using a portion of the *Slaves*. This part of the process is shown to the right of the *Master* in Fig. 2. Primal recovery evaluates the second-stage cost of primal candidates and, if using the IS heuristic, it also creates new candidates by projecting averaged first-stage solutions onto the first-stage feasible set.

Performing dual iterations alongside primal recovery enables the algorithm to continuously compute upper bounds (dual function evaluations) and lower bounds (primal recovery) on the optimal value. This allows us to establish a natural termination criterion, $UB - LB \leq \epsilon$. We can also terminate the algorithm at a certain wall time, while still returning the incumbent solution and an optimality gap. These termination guarantees permit the deployment of our asynchronous algorithm in settings with hard limits on execution time, such as day-ahead unit commitment.

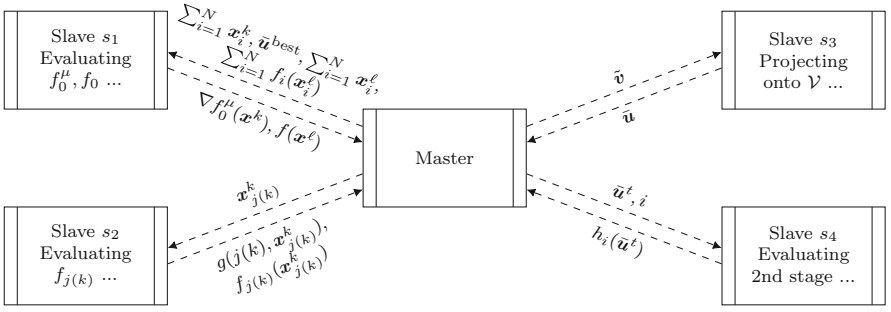

Fig. 2 Execution snapshot of the asynchronous distributed algorithm for stochastic unit commitment. Each box corresponds to a process and each dashed line corresponds to information that is exchanged between processes. The *Master* assigns tasks to each of the *Slaves* dynamically. Not all types of tasks need to be present at all times, and there might be several *Slaves* engaged on the same task, but over different data

In the following, we detail the internal layout of the *Master* and *Slave* processes, and how they interact with each other. The implementation is based on the SMPS file format for stochastic programs [26]. In particular, it uses the concepts of CORE problem, time indexation of variables and constraints (TIME file), and the specification of scenarios by their differences to the CORE problem (STOCH file). In order to maintain a small memory footprint, which is critical for solving large SUC instances, only the CORE problem and the TIME indices are maintained in memory, while the information in the STOCH file is loaded from the hard drive as needed and purged after it has been used.

5.1 Slave

The *Slave* process, presented in Fig. 3, starts by partially reading the instance to be solved (steps 1–2, Fig. 3): it loads the CORE problem, loads all the information in the TIME file (time stages of variables and constraints) and gathers information about the organization of the STOCH file (metadata), e.g. a list of the scenarios and where in the STOCH file are they located.

The reading process respects the classification of constraints within the CORE file. In particular, it differentiates between normal constraints, *delayed constraints* (i.e. constraints that are necessary for feasibility but are unlikely to be binding, also known as *lazy constraints*) and *model constraints* (i.e. constraints redundant at the optimal MIP solution, also known as *user cuts*).[3] Current commercial MILP solvers can take advantage of this classification of constraints to speed up the solution process. Note that, in order to read files with this constraint differentiation, the TIME file must explicitly declare the time index of rows and columns [26].

After every process finishes the reading step (step 3), the control flow is organized around a loop within which the *Slave* receives a task from the *Master*, executes it, and communicates back the result. Subproblems in all tasks are formulated either by modifying the CORE problem, as in steps 6, 10 and 22, or by taking a subset of the constraints of the CORE problem, as in steps 14 and 18. The transformation of steps 6, 10 and 22 uses the metadata to avoid parsing unnecessary parts of the STOCH file.

There are 5 types of tasks, as well as 1 termination signal, that the *Slave* can receive from the *Master*. Among these, *dual scenario* corresponds to evaluating a certain component $j \in \{1, \ldots, N\}$ of the dual objective for certain multipliers, *primal projection* corresponds to projecting an averaged candidate onto the feasible set of first stage decisions [see Eq. (15)], and *second stage scenario* corresponds to solving a recourse problem for a given first stage decision.

The *dual* f_0 task involves the following two actions: computing the gradient of f_0^{μ} and computing an upper bound. These tasks are merged because they involve solving very similar mathematical programs, (6) and (8), none of which has a strict requirement on the frequency with which it must be solved (contrary to the case of *primal projection*, which must be solved whenever we need a new primal candidate). The task uses four pieces of data. The first two, the sum of the current multipliers

[3] *Delayed constraints* and *model constraints* correspond to the terminology used by Xpress, while *lazy constraints* and *user cuts* correspond to the terminology used by Cplex.

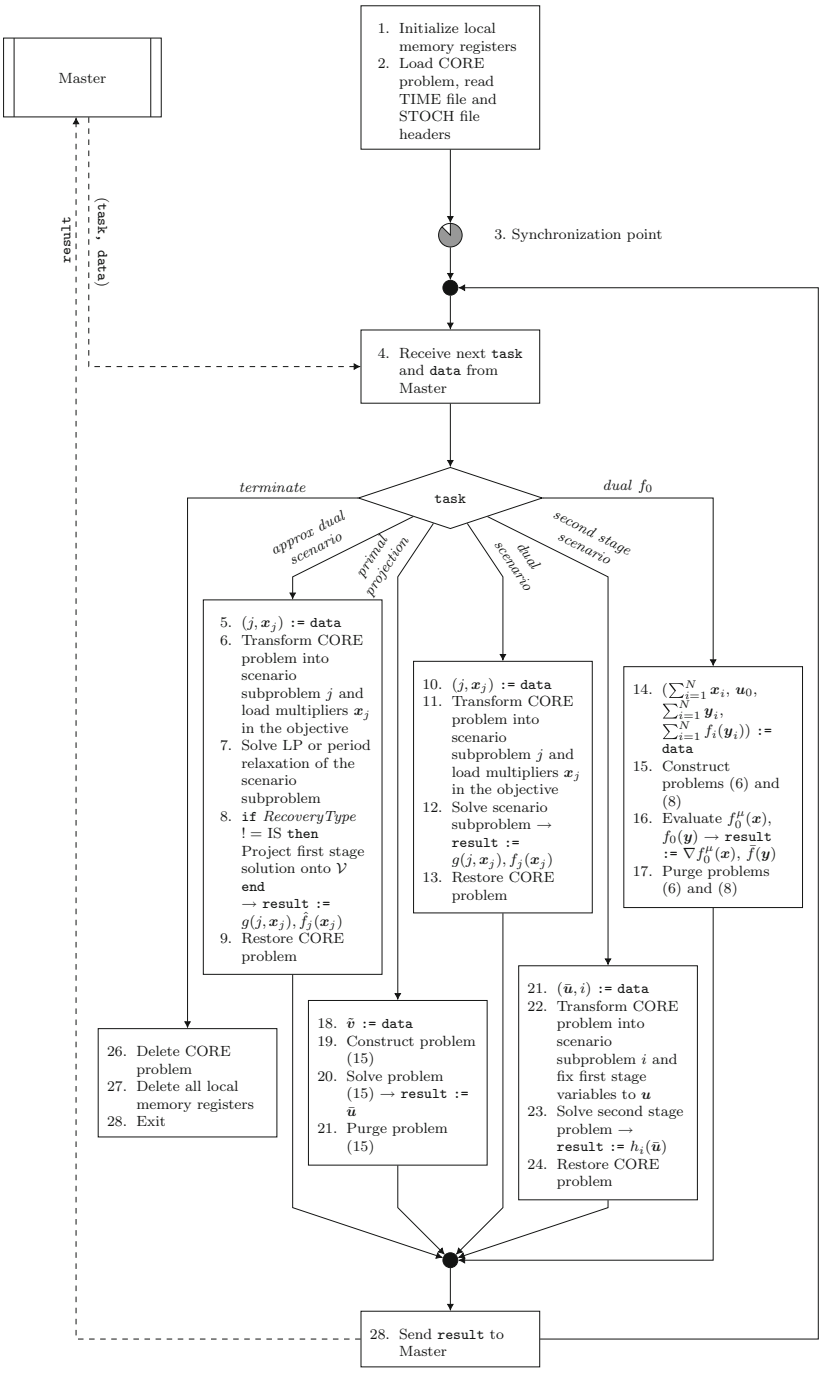

Fig. 3 Control flow of the *Slave* process. Continuous lines show the flow of the program, while dashed lines indicate exchange of information with other processes. Italics denote constants and parameters. The iteration counter k and the candidate index t are dropped because they are not relevant within the *Slave*

$\sum_{i=1}^{N} \boldsymbol{x}_i$ and a center $\boldsymbol{u}_0$, are used to compute the gradient of f_0^{μ} at the current iterate. The other two, the sum of certain multipliers $\boldsymbol{y}$, $\sum_{i=1}^{N} \boldsymbol{y}_i$, and the sum of the scenario component functions of the dual evaluated at $\boldsymbol{y}$, $\sum_{i=1}^{N} f_i(\boldsymbol{y}_i)$, are used to obtain an upper bound on the optimal value of the original program by evaluating $f_0(\boldsymbol{y})$ and letting $\bar{f}(\boldsymbol{y}) = f_0(\boldsymbol{y}) + \sum_{i=1}^{N} f_i(\boldsymbol{y}_i)$, as indicated in Eq. (13).

The *approx dual scenario* task has the same objective as the *dual scenario* task, with the difference that the former solves only a relaxation of the scenario subproblem. Two types of relaxation are considered: (*i*) the linear programming (LP) relaxation, applicable to problems where the sets $\mathcal{D}_i$ are mixed-integer polyhedral sets, and (*ii*) the period relaxation, in which each period of the scenario subproblem is solved independently. The latter is applicable to problems where sets $\mathcal{D}_i$ have a multi-period structure, such as the case of SUC (see Appendix A). The period subproblem is constructed using the indexation of variables and constraints present in the TIME file. These relaxations provide cheap subgradient estimates and upper bounds on component functions. They are used in the initialization procedure described in Sect. 5.3.

5.2 Master

The *Master* process, presented in Figs. 4 and 5, starts in the same manner as the *Slave*, by allocating memory to variables and reading the necessary information from the CORE, TIME and STOCH files, steps 1–2 in Fig. 4. Note that the *Master* does not require further information regarding the subproblems because this is not used directly for dual iterations or primal recovery.

After reaching the synchronization point, step 3, the *Master* uses the *Slaves* to perform the initialization procedure, step 4, which provides initial values for the subgradients and upper bounds. This step is followed by the launching of the initial batch of tasks of the algorithm, in steps 5 and 6. We update as many blocks as possible, launching the corresponding subgradient evaluation tasks. If there are free *Slaves* after updating all blocks, we use them to perform primal recovery tasks, so that all *Slaves* are assigned to a task. The *Master* keeps track of the task assigned to each *Slave*.

The *Master* then enters its main loop, which receives the result of a task from *Slave s* (step 7), processes it (steps 8–16) and assigns a new task to the *Slave s* (steps 21–28). The processing procedure depends on the type of task. For *dual* f_0, we simply overwrite the gradient of f_0 and update the upper bound, while for *primal projection* we add the new candidate to the list of primal candidates.

The processing of *dual scenario* tasks requires checking whether the received result contains new information relative to what is already available to the master before overriding it (step 11). This is necessary, because as the updates of $\boldsymbol{x}$ are performed at random, there might be two or more *Slaves* evaluating the same component function i, each for a different $\boldsymbol{x}_i$. If the evaluation for an older $\boldsymbol{x}_i$ finishes later, it should not overwrite the subgradient information available to the `Master`, since this would only introduce more delays in the subgradient information. Irrespective of its delay, the result is used in step 12 as a primal candidate as long as the primal recovery heuristic is not set to importance sampling.

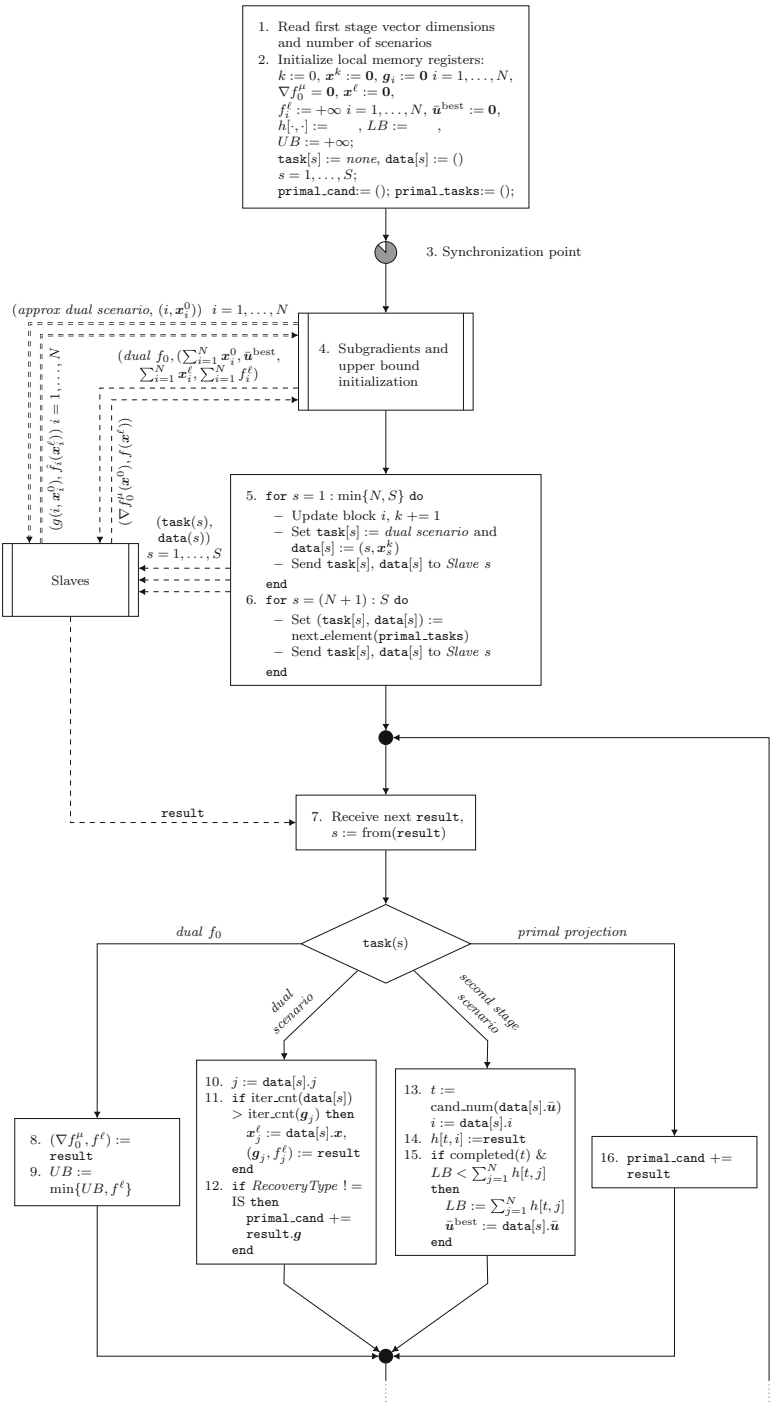

Fig. 4 Control flow of the *Master* process

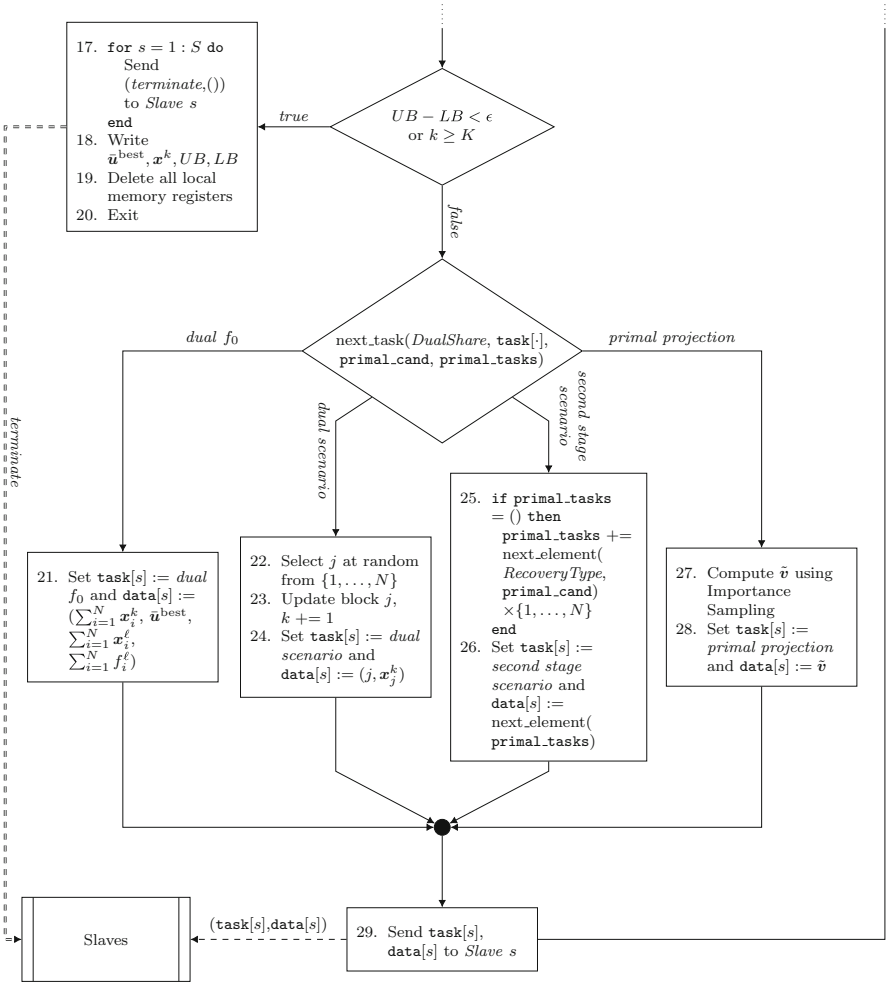

Fig. 5 Control flow of the *Master* process (cont.)

Second stage scenario tasks, on the other hand, return the second stage cost of a given first stage candidate solution at a certain scenario (step 14). If the first stage solution has been evaluated for all scenarios, it is used in step 15 in order to update the lower bound on the primal objective. Note that we do not assume any order in the evaluation of first-stage candidate solutions, allowing for primal recovery to be performed asynchronously on several candidates at the same time.

Once the processing phase is complete, at the top of Fig. 5, the algorithm checks whether the termination criterion is met and terminates the execution of the *Master* and *Slaves* if that is the case. If the termination criterion is not met, the algorithm will proceed to decide which task to assign to *Slave s*. This decision is made according to the following criteria, in the same order of importance as they are presented:

1. Use at most N concurrent processes for *dual scenario* tasks.
2. Maintain the proportion of *Slaves* engaged in dual iterations as close as possible to the value of the configuration parameter *DualShare*.
3. A *dual* f_0 task must be executed every certain number of *dual scenario* tasks.
4. If using the IS primal recovery heuristic, a *primal recovery* task must be executed whenever the number of elements left in `primal_tasks` is small enough to risk `primal_tasks` to be empty before a new candidate is generated.

Before sending the chosen task to *Slave s*, certain preprocessing steps are required. In the case of a *dual scenario* task, a block coordinate descent update is performed on a random scenario j and, only then, a *dual scenario* task for scenario j with the new multipliers is assigned to *Slave s*. This action (steps 22–23) corresponds to the updating system of Fig. 2.

Before assigning a *second stage scenario* task, on the other hand, it is necessary to check whether there are pending primal tasks to be executed. If this is not the case, then, in step 25, new tasks are created using the next candidate, which is chosen according to the order specified by the *RecoveryType* parameter (FIFO, RND, LIFO, IS) from the list of pending candidates, `primal_cand`, and removed from it. If the *RecoveryType* configuration parameter is set to IS, then the most recent candidate in the list is chosen.

The assigned task is then sent to *Slave s* and the *Master* returns to the beginning of its main loop, where it will wait for the next result.

5.3 Initialization

The main objective of the initialization subroutine, executed by the *Master* at step 4 (Fig. 4), is to obtain cheap estimates of the subgradients of component functions, upper bounds on the component function values at the initial multipliers $\boldsymbol{x}^0$ and, optionally, an initial set of primal candidates.

The initialization proceeds as follows. First, subgradient estimates and upper bounds are computed by executing *approx dual scenario* using the *Slaves*, for all scenarios with the initial multipliers. Once the results for all scenarios have been collected, a *dual* f_0 task is executed to obtain a gradient for f_0^{μ} and a valid upper bound on the primal objective.

If the selected primal recovery heuristic is not IS, subgradient estimates can be used as primal candidates (because of the projection onto $\mathcal{V}$, step 8, Fig. 3) and evaluated at step 6 of Fig. 4 (if $S > N$). On the other hand, if the selected primal recovery heuristic is IS, then at step 6 of Fig. 4 the next primal task would correspond to *primal projection*, with averaged candidates generated by the *Master* on the fly.

Without these subgradient estimates, the first round of updates (step 5, Fig. 4) would not modify the multipliers, and the computation of the first upper bound would be delayed until all component functions have been evaluated, that is, at least until the slowest of all component functions is accurately evaluated. Considering that differences in evaluation times of component functions observed in real instances can be as high as 7500%, the lack of an initial upper bound can significantly delay termination

of the algorithm, even when a good primal candidate and lower bound are already available.

Note that primal recovery also benefits from the initialization. The reason for this is that, without the initial set of candidates, primal recovery would be delayed until the results from the accurate evaluation of component functions f_i are returned to the *Master*.

6 Numerical results

We implement the proposed asynchronous algorithm as described in the previous section in C, using Xpress (through its C API) [22] for solving all mathematical programs and MPI [25] for handling communications between processes.

We test the proposed algorithm on SUC instances of the WECC [42] and CWE [8] systems, and on generic SMIP instances from the SIPLIB collection [4]. Sizes and solution times of scenario subproblems of SUC instances are presented in Table 1, where it can be observed that the instances used in the present study are at least as large, or present scenario subproblems as difficult to solve (in terms of solution time) as the models in the literature. We conduct these numerical experiments using multiple high-performance computing clusters, the technical specifications of which are presented in Appendix C.

6.1 Western Electricity Coordinating Council system instances

The WECC system instance [42] is composed of 130 thermal generators, 182 buses and 319 branches. It features multiarea renewable production with hourly resolution over a 24 h horizon for 8 representative day types, one weekday and one weekend day per season. The number of scenarios ranges from 10 to 1000, and each scenario is associated with different renewable production profiles and contingencies.

Numerical experiments on the WECC system were run on the Cab cluster, hosted at the Lawrence Livermore National Laboratory. We configure Xpress to solve the root node in all subproblems using the barrier algorithm, we limit the number of Xpress threads to 1 and set the termination gap of subproblems to 1%. For all WECC instances,

Table 1 Scenario subproblem sizes and solution times for different instances used in the present study (highlighted in boldface) and in the literature

Instance	Rows	Columns	Non-zeros	Integers	Subproblem solution	
					Time (s)	Avg. (max.)
WECC [18]	69,447	28,943	240,724	4080	9.4	(25.7)
WECC [42]	34,441	23,090	139,394	3074	8.3	(67.9)
EDF [53]	812,906	73,562	–	26,122	–	–
CWE [8]	609,589	390,075	1,941,270	9753	3383.2	(7851.8)

Subproblem solution times for WECC [42] correspond to a winter weekend instance with 100 scenarios, while subproblem solution times for CWE [8] correspond to a spring weekday instance with 120 scenarios

we use 1 core per process, so that $S = \#\text{Cores} - 1$, and we limit the run time of the algorithm to 2 h.

The solution times of the WECC instances are summarized in Table 2 for different configurations of the asynchronous algorithm. Regarding stepsizes, 'Dim. 1/k' corresponds to a stepsize of the type $1/k$ and 'Polyak' corresponds to the Polyak stepsize defined in Eq. (14), where the diminishing part is set to decrease from 0.5 to 0.25 in $50N$ iterations with $q = 1$. Primal solution recovery methods correspond to the four methods described in Sect. 4. The modification of the *DualShare* parameter for 1000-scenario instances is motivated by the findings that are presented in Sect. 6.4.

From the perspective of dual optimization, Polyak stepsizes outperform $1/k$ stepsizes when considering a 1% termination criterion. In all our instances and test runs, Polyak stepsizes provided better results without the need for tuning.

For primal recovery, we observe that the three methods that recover solutions directly from solutions to scenario subproblems (FIFO, RND, LIFO) exhibit similar performance for the 10-scenario instances. RND and LIFO outperform FIFO on instances with 100 scenarios. The IS heuristic outperforms its counterparts in all instances.

The best configuration of Table 2, which is highlighted in boldface, outperforms the run time reported in [42], in which 1000 scenario instances (23.1 million variables, 35.9 million constraints and 3.1 million integers) were solved in up to 24 h using 1000 cores. Instead, the best performing algorithm in this paper solves the same instance in less than 2 h using 256 cores. Similar speedups are observed for instances with fewer scenarios with respect to [42]. The progressive hedging method of Cheung et al. [18] solves instances of the WECC with up to 100 scenarios within 1.5–2.5% optimality within 25 min. In comparison, the proposed algorithm solves instances of similar difficulty in terms of solution time of subproblems (see Table 1) to 1% optimality in at most 18.7 min and to 2% suboptimality in less than 5 min. We perform a direct comparison between progressive hedging [18] and the proposed asynchronous algorithm using a subset of the SUC WECC instances. Details of this comparison are presented in Appendix D. We observe that the proposed method is 4.35 times faster than the progressive hedging method of [18], though the latter tends to produce better quality solutions.

Detailed solution statistics per day type for the best configuration are reported in Table 8, in Appendix E. For ease of replicability, in Table 9, which is also included in Appendix E, we report the solution statistics for the 10-scenario instances obtained using a 4-core laptop.

6.2 Central Western European system instances

The CWE system instance is based on [8], with the following adaptations: (*i*) we include the commitment of nuclear units (binary variables) and (*ii*) the selection of a set point for CHP power plants[4] (continuous variables) as first-stage decisions of SUC. The system consists of 656 thermal generators, 679 buses and 1037 branches, and it

[4] Combined Heat and Power (CHP) plants can vary their production of electricity only within a limited range for a given production of heat [20].

Table 2 Solution time statistics for WECC instances, over 8 representative day types

N	Step size	*Recovery type*	# Cores	*Dual share*	Solution time (s), avg. (max.)			
					2% Optimality		1% Optimality	
10	Dim. 1/k	FIFO	16	0.5	228.1	(792.9)	–	
	Dim. 1/k	RND	16	0.5	229.4	(856.1)	–	
	Dim. 1/k	LIFO	16	0.5	200.6	(739.7)	–	
	Dim. 1/k	IS	16	0.5	178.0	(638.0)	–	
	Polyak	FIFO	16	0.5	148.2	(469.3)	424.4	(1361.1)
	Polyak	RND	16	0.5	117.8	(392.6)	–	
	Polyak	LIFO	16	0.5	131.2	(446.2)	384.0	(1326.9)
	Polyak	**IS**	**16**	**0.5**	**118.4**	(441.4)	**364.7**	(1291.5)
100	Polyak	FIFO	160	0.5	267.6	(325.1)	–	
	Polyak	RND	160	0.5	113.2	(345.6)	534.2	(1134.1)
	Polyak	LIFO	160	0.5	99.4	(268.3)	508.9	(1152.4)
	Polyak	**IS**	**160**	**0.5**	**95.5**	(289.8)	**517.9**	(1126.1)
1000	Polyak	LIFO	256	0.75	723.9	(2155.2)	–	
	Polyak	**IS**	**256**	**0.75**	**411.7**	**(1354.5)**	**2535.0**	**(6427.0)**

Statistics for configurations that failed to achieve the target optimality gaps, within the limit wall time, for one or more day types are not reported and are denoted with a dash

Table 3 Solution time statistics for WECC instances, over 8 representative day types

N	# Cores	Solution time (s), avg. (max.)			
		2% Optimality		1% Optimality	
30	96	2580.3	(5908.2)	3806.2	(9279.1)
60	192	2563.7	(5593.3)	3774.2	(8323.4)
120	384	2696.5	(5973.0)	3876.2	(7952.6)

All instances use the Polyak stepsize, the IS primal recovery heuristic, and a *Dual Share* parameter setting of 0.75

features multiarea renewable production with quarterly resolution over a 24-h horizon (96 periods). The problem is solved for 8 representative day types and using 30, 60 and 120 scenarios of renewable production. For all CWE instances, ramp rate constraints within each hour are declared as delayed constraints. Each scenario subproblem of the CWE instances is almost one order of magnitude larger, in terms of matrix size, than the scenario subproblems of the largest instance considered in the SUC literature [53], see also Table 1.

Numerical experiments on the CWE system were run on the Cab cluster of the Lawrence Livermore National Laboratory. We configure Xpress to solve the root node in all subproblems using the barrier algorithm, limit the number of Xpress threads to 2, set the MIP time limit to 1 h and 30 min, and set the termination gap of subproblems to 1%. Each process uses 2 cores (1 core per Xpress thread) hence $S = \#\text{Cores}/2 - 1$. We set the maximum run time of the algorithm to 6 h. Table 3 presents the summarized solution statistics for the proposed algorithm on the CWE instances. Detailed solution statistics per day type are reported in Table 10 of Appendix E.

Solution times are larger that those observed for the WECC, nevertheless they remain within operationally acceptable time frames for day-ahead scheduling (at most 2 h and 34 min are required for obtaining a solution within 1% optimality). This increase in overall solution time with respect to the WECC instances results mostly from the time required for solving dual scenario subproblems which, as shown in Table 1, is two orders of magnitude larger than for WECC.

These results largely outperform reported solution times for instances of the CWE system with up to 16 scenarios [44] (also obtained from [8]), on the same computing environment (Cab cluster). In these references, scenario decomposition along with an optimal grouping approach achieved solutions within 2% of optimality in 15 h.

6.3 SIPLIB instances

SIPLIB [4] is a collection of SMIPs, provided in SMPS format [27], for testing and developing numerical methods for solving SMIPs. Two-stage SMIP instances in the SIPLIB collection include a variable count that ranges from tens up to a few thousand variables. Each scenario subproblem has constraints, which are much fewer than the industrial-size SUC instances considered in the previous sections of thie paper. The SIPLIB problems include from 2 up to 2000 scenarios. We conduct numerical experiments using the largest two-stage SMIP instances in the SIPLIB collection, which

Table 4 Solution time statistics for largest SIPLIB instances

Instance	N	*Dual share*	k/N	Solution time (s)	LB	UB	Gap (%)
dcap233_200	200	0.5	58.3	5.37	− 1839.33	− 1826.28	0.71
dcap233_300	300	0.5	75.3	10.34	− 1655.59	− 1639.14	0.99
dcap233_500	**500**	**0.5**	**200.0**	**41.75**	**− 1776.39**	**− 1734.02**	**2.39**
dcap243_200	200	0.5	40.0	3.70	− 2334.49	− 2311.16	1.00
dcap243_300	300	0.5	62.8	7.99	− 2574.79	− 2551.44	0.91
dcap243_500	500	0.5	146.9	31.59	− 2180.31	− 2161.58	0.86
dcap332_200	**200**	**0.5**	**200.0**	**17.92**	**− 1061.54**	**− 1050.85**	**1.01**
dcap332_300	**300**	**0.5**	**200.0**	**27.58**	**− 1274.11**	**− 1241.07**	**2.59**
dcap332_500	**500**	**0.5**	**200.0**	**43.44**	**− 1611.07**	**− 1583.60**	**1.71**
dcap342_200	200	0.5	34.4	3.32	− 1621.95	− 1605.93	0.99
dcap342_300	300	0.5	166.6	21.80	− 2084.18	− 2063.34	1.00
dcap342_500	500	0.5	167.3	38.31	− 1916.31	− 1899.54	0.88
sslp.5.25.50	50	0.5	57.2	1.91	121.60	122.83	1.00
sslp.5.25.100	100	0.5	28.6	2.82	127.37	128.65	1.00
sslp.10.50.50	50	0.5	20.0	9.42	364.64	368.29	0.99
sslp.10.50.100	100	0.5	27.1	25.32	354.19	357.77	1.00
sslp.10.50.500	500	0.9	27.2	69.99	349.14	352.65	1.00
sslp.10.50.1000	1000	0.9	20.7	105.90	351.71	355.26	1.00
sslp.10.50.2000	2000	0.9	17.8	180.48	347.26	350.77	1.00

All instances use 1% optimality or $200N$ iterations as termination criterion (whichever occurs first), Polyak stepsize, IS primal recovery and 96 cores

originate from two test sets: (*i*) the DCAP test set includes instances with mixed-integer first-stage variables and binary second-stage variables, and (*ii*) the SSLP test set includes instances with binary first-stage variables and mixed-binary second stage variables. These experiments are run on the Lemaitre3 cluster hosted at the Université catholique de Louvain, with Xpress used for solving the root node with dual simplex and a termination gap of 0.1% for subproblems.

Table 4 presents the solution times and solution quality statistics for the DCAP and SSLP instances. Here, k/N corresponds to the total number of coordinate descent iterations normalized by the number of scenarios. For consistency with the rest of the paper, lower bounds LB and upper bounds UB refer to bounds on the objective values of the respective problems, which are posed as maximization problems.

We observe that the proposed asynchronous approach fails to achieve the desired 1% optimality gap required for 4 of the 12 instances. This is highlighted in boldface in Table 4. From the exact solution to these instances [3], obtained using a specialized branch-and-bound algorithm described in [5], we know that this is due to the fact that our heuristics failed to detect a primal candidate with an acceptable objective function value for dcap233_500, dcap332_300 and dcap332_500. Instead, for dcap332_200, the proposed asynchronous algorithm failed to derive a tight upper bound. All other DCAP instances were solved to the desired tolerance.

Table 5 Variation of solution time with the *Dual Share* parameter setting, which varies smoothly between its start value ($k = 0$) and its value after 200 dual passes over data ($k = 200N$)

Primal recovery	*Dual share*		Solution time (s), avg. (max.)			
	$k = 0$	$k = 200N$	2% optimality		1% optimality	
IS	0.1	0.1	150.5	(168.7)	1731.9	(1821.0)
	0.25	0.25	78.9	(82.8)	785.7	(807.8)
	0.5	0.5	52.5	(55.6)	441.4	(467.6)
	0.75	0.75	67.2	(78.0)	307.4	(333.3)
	0.9	0.9	58.1	(72.3)	291.0	(294.2)
LIFO	0.5	0.5	97.5	(120.0)	529.9	(598.3)
	0.75	0.75	92.3	(104.5)	479.9	(624.1)
IS	0.75	0.25	50.9	(59.0)	313.9	(334.5)
	0.9	0.1	66.8	(77.5)	280.3	(320.2)

Results in the table correspond to WECC, spring weekdays, for 100 scenarios. The instance is solved using 96 cores and a Polyak stepsize. Statistics are obtained over 4 runs for each configuration

Regarding SSLP instances, we arrive at a different observation. While we were able to achieve the desired optimality tolerance and solve problems with up to 2000 scenarios, in terms of solution time the proposed approach is outperformed by an order of magnitude by the integer L-shaped method presented in [6].

These results validate the correctness and show the general applicability of our asynchronous algorithm. At the same time, they indicate that, for small-scale problems, enumerative techniques (such as those presented in [5,6]) should be preferred to our asynchronous algorithm. On the other hand, for large-scale problems, the proposed asynchronous algorithm should be preferred, as demonstrated in the previous sections.

6.4 Sensitivity of solution times to the allocation of resources

The *Dual Share* configuration parameter determines how distributed computing resources are allocated between tasks related to dual iterations or to primal solution recovery. This can impact the overall performance of the algorithm significantly, especially when parallel computing resources are limited.

Table 5 shows how variations in the *Dual Share* affect run time, with all other parameters remaining constant. We select the WECC spring weekday instance with 100 scenarios to perform this test. Our choice is due to the fact that it corresponds to a medium-size instance and has the median solution time among WECC instances with 100 scenarios.

The configuration using the IS primal recovery heuristic (first 5 rows) exhibits an important improvement in solution time as we increase the *Dual Share*. For the LIFO heuristic (6th and 7th row), on the other hand, we observe only a minor improvement in solution time with the increase of *Dual Share*. This shows that averaging scenario subproblem solutions can indeed generate better candidates than simply using scenario subproblem solutions. We thus avoid the use of computing power in evaluating the

performance of low-quality primal candidates. This effect was not observed in the 10 and 100-scenario instances of Table 2 because we used almost twice as many cores as scenarios, allowing FIFO, LIFO and RND to carry out a very large number of candidate evaluations.

Our implementation gives us the freedom to change the *Dual Share* during the solution of an instance, allocating fewer or more resources to dual tasks in ealier iterations. This can be beneficial as the subgradient method achieves fast improvements during the first iterations, but it becomes slow as it approaches the optimal solution. Therefore, by starting with a high *Dual Share* and gradually decreasing it, we can take advantage of the rapid bound improvement during the first iterations and use more computing power in later stages in order to recover better primal solutions. Rows 8 and 9 present the results of applying this idea. The improvements with respect to maintaining a fixed resource allocation, as in all other rows of the table, are modest because of a combination of factors. During the first iterations, the Polyak stepsize cannot be computed accurately due to the lack of a good lower bound (see Eq. (14)) and the updates tend to overshoot. On the other hand, during the last iterations, the IS heuristic encounters difficulties in finding new primal candidates, because the frequency at which we obtain solutions to scenario subproblems decreases with the *Dual Share*.

6.5 Parallel computing performance

The performance of a parallel algorithm can be measured against various metrics. We focus on (*i*) how the proposed algorithm compares to a synchronous algorithm, i.e. an algorithm that uses a synchronous method to carry out dual iterations, and (*ii*) how effectively the proposed algorithm scales up with the number of cores.

Solving the instances used in this work with a fully synchronous algorithm may require excessive use of computational resources [41]. In order to avoid using additional computing time for this comparison, we compare the proposed algorithm to a synchronous algorithm in terms of the idle time of slaves. This can be estimated using the solution times of subproblems. This information is already available to us from the experiments of the previous sections. We consider a synchronous algorithm that solves the dual problem with the subgradient method using $\min\{N, \textit{Dual Share} \times S\}$ slaves (rounded to the closest integer, if necessary), and that performs primal recovery synchronously using the remaining slaves. We refer the reader to Appendix F for a complete description of the procedure that we employ for estimating idle times.

Table 6 presents the estimated idle times of slaves for the described synchronous algorithm. Note that idle times generally increase with the number of cores, since with fewer cores tasks can be stacked. In order to obtain a measure of the variation of idle times with the number of cores, Table 6 includes estimates for the synchronous algorithm with (i) the same number of cores as the asynchronous algorithm, (ii) half the number cores of the asynchronous algorithm, and (iii) $1 + 2N$ cores (1 *Master*, $2N$ *Slaves*).

It can be directly observed that synchronous schemes tend to underutilize parallel computing infrastructure, leaving cores corresponding to slaves idle up to 80.4% of

Table 6 Estimated idle time of slaves when solving SUC using a parallel synchronous algorithm. Average and maximum over 8 day types

System	N	# Cores	*Dual share*	Synchronous idle time (%), avg. (max.)					
				Same conditions		Half # cores		$1 + 2N$ cores	
WECC	10	16	0.5	48.0	(53.2)	28.3	(34.5)	54.2	(59.2)
	100	160	0.5	70.6	(80.4)	52.9	(65.0)	74.3	(83.8)
	1000	256	0.75	37.6	(57.0)	21.1	(36.9)	85.4	(92.4)
CWE	30	96	0.75	36.4	(47.0)	27.1	(33.9)	46.5	(53.9)
	60	192	0.75	43.6	(62.2)	33.4	(47.3)	53.3	(67.2)
	120	384	0.75	46.9	(61.2)	36.3	(46.6)	54.7	(65.4)

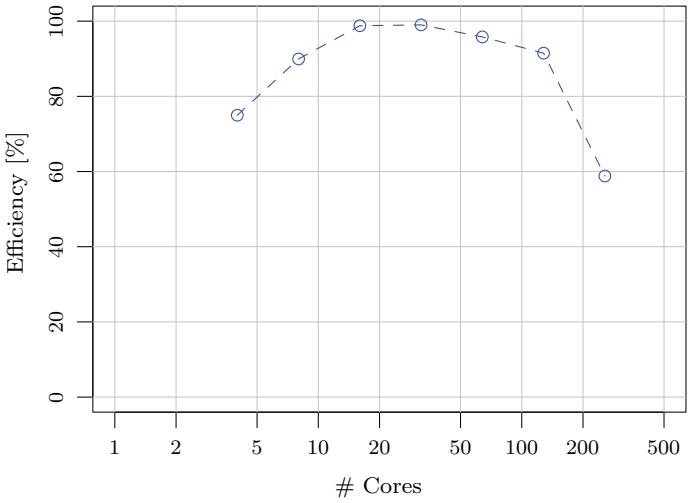

Fig. 6 Parallel efficiency plot of the asynchronous algorithm. Plot drawn using WECC, spring weekdays, 100 scenarios, *Dual Share* 0.75, Polyak stepsize and IS primal recovery, and 1 process per core ($S = \#\text{Cores} - 1$). Wall times are obtained by averaging 4 runs for each core count presented in the plot

the time. For the same problem, the asynchronous algorithm achieves almost zero idle time (the percentage of time dedicated to solving mathematical programs never falls below 97%, see Tables 8 and 10 for details). Decreasing or increasing the number of cores does not change this observation for the synchronous scheme.

In order to determine how well the proposed algorithm scales with respect to the number of cores, we numerically estimate its parallel efficiency (i.e. speedup divided by the number of cores) by running the asynchronous algorithm multiple times with different core counts. The estimated efficiency is presented in Fig. 6. Our baseline is the serial execution of the asynchronous algorithm, i.e. the method of Sect. 3.1, with delays in the gradient of the smooth part of the objective and interleaving subgradient evaluation with primal recovery.

For a small number of cores, low efficiencies result from of the existence of the *Master* process, which is not necessary in the serial execution. For instance, in Fig. 6

with 4 cores, the asynchronous algorithm launches the *Master* and 3 *Slave* processes. Even under ideal conditions, this limits the parallel efficiency to 75%. When the number of cores is between $0.1N$ and $1.3N$, the algorithm achieves efficiency values above 90%, peaking around $0.3N$ with 99.0%. This regime is characterized by very small lags in the dual algorithm, hence a close to sequential performance, and an effective performance of the primal recovery heuristic. For more than $1.3N$ cores, the efficiency decreases as the IS heuristic experiences difficulties in finding new primal candidates.

7 Conclusions

We propose an asynchronous dual decomposition algorithm for stochastic unit commitment, in which dual iterations are performed using a block-coordinate subgradient method, for which we provide convergence guarantees. We also propose primal recovery heuristics and present a high performance computing implementation of the algorithm. The algorithm is able to solve all instances of WECC and CWE within operationally acceptable time frames and presents parallel efficiency above 90% when using between $0.1N$ and $1.3N$ slaves.

We find that synchronous algorithms dramatically underutilize high performance computing infrastructure, resulting in core idle times of up to 80.4%. This finding underscores the need for designing asynchronous algorithms in order to tackle industrial scale unit commitment problems.

Future extensions of the present work will focus on the application of the developed asynchronous decomposition framework for tackling (*i*) multi-stage stochastic unit commitment and (*ii*) detailed deterministic unit commitment problems over large interconnected power systems. The latter class of problem includes the exciting prospect of integrating the optimization of transmission and distribution systems through convex relaxations of AC power flow constraints [13].

Acknowledgements The authors would like to thank Dr. Vitor de Matos (Plan4 Engenharia) for suggesting the idea of evaluating primal solutions in reverse order (LIFO), and Dr. Bernard Knueven (Sandia National Laboratories) for his help in setting up the parameters of progressive hedging for the benchmark presented in Appendix D. The authors would also like to thank the Fair Isaac Corporation FICO for providing licenses for Xpress; the Lawrence Livermore National Laboratory for granting access and computing time at the Sierra, Cab and Quartz clusters; and the the Consortium des Équipements de Calcul Intensif (CÉCI) for granting access and computing time at the Lemaitre3 cluster. CÉCI is funded by the *Fonds de la Recherche Scientifique de Belgique* (F.R.S.-FNRS) under Grant No. 2.5020.11 and by the Walloon Region.

Appendix

A Formulation of the stochastic unit commitment problem

We model the SUC problem following [42]. Minor extensions have been introduced, in order to better capture European electricity markets such as the CWE system. The most important of these extensions is the hybrid time resolution of our model,

where commitment decisions (ON/OFF) follow an hourly resolution (the resolution of the day-ahead market in the CWE), whereas dispatch decisions (production and power flows) follow a quarterly resolution (the resolution of real-time balancing in the CWE). Other extensions include piecewise linear production cost functions, improved ramping formulations following [19], and an improved modeling of nuclear and CHP power plants.

Our complete SUC model is described in the present section, in order to render the paper self-contained. We use the following notation.

Sets

T_{60}	hourly periods, $T_{60} = \{1, \ldots, 24\}$
T_{15}	15 min periods, $T_{15} = \{1, \ldots, 96\}$
N	Buses
L	Lines
G	Thermal generators
G^{SLOW}	Slow thermal generators (e.g. coal and CCGT power plants)
G^{FAST}	Fast thermal generators (e.g. natural gas and diesel turbines)
G^{STAT}	Static thermal generators (e.g. nuclear and CHP power plants)
$L(n, m)$	Lines between buses n and m, directed from n to m
$G(n)$	Thermal generators at node n
I	Scenarios
$\mathcal{P}_{g,i}$	Feasible operation set of slow or fast generator g at scenario i
$\mathcal{P}^{STAT}_{g,i}$	Feasible operation set of static generator g at scenario i

Parameters

$\tau(t)$	Corresponding hour of quarter t, e.g. $\tau(5) = 2$
F_l	Thermal capacity, line l
B_l	Susceptance, line l
$n(l), m(l)$	Outgoing and incoming buses, line l
$D_{n,t}$	Demand at bus n in period t
$\underline{P}_g, \overline{P}_g$	Minimum stable level and maximum run capacity, generator g
TL_g	Maximum state transition level of generator g
R_g	Maximum ramp of generator g
$C_g(\cdot)$	Production cost function, generator g
K_g	No load cost, generator g
S_g	Startup cost, generator g
V	Value of lost load
$\zeta_{g,i}$	Outage indicator, generator g, Monte Carlo sample i
$\zeta_{l,i}$	Outage indicator, line l, Monte Carlo sample i
$\xi_{n,i,t}$	Renewable supply at bus n, Monte Carlo sample i, period t
π_i	Probability of Monte Carlo sample i

Variables

$v^{ON}_{g,i,\tau}, v^{SU}_{g,i,\tau}$	Commitment and startup, slow generator g, Monte Carlo sample i, hour τ

$v_{g,i}^{STAT}$	Commitment of static generator g, Monte Carlo sample i
$p_{g,i}^{STAT}$	Center value of production band of static generator g, Monte Carlo sample i
$z_{g,i,\tau}^{ON}, z_{g,i,\tau}^{SU}$	Commitment and startup, fast generator g, Monte Carlo sample i, hour τ
$p_{g,i,t}$	Production of generator g, at Monte Carlo sample i, period t
$f_{l,i,t}$	Flow through line l, at Monte Carlo sample i, period t
$\theta_{n,s,t}$	Voltage angle, bus n, Monte Carlo sample i, period t
$e_{n,i,t}$	Load shedding at bus n, Monte Carlo sample i, period t
$o_{n,i,t}$	Production shedding at bus n, Monte Carlo sample i, period t
$u_{g,\tau}^{ON}, u_{g,\tau}^{SU}$	Commitment and startup, slow generator g, hour τ
u_{g}^{STAT}	Commitment of static generator g
q_{g}^{STAT}	Center value of production band of static generator g

The SUC problem aims at capturing the scheduling problem faced by power system operators in day-ahead planning. The power system has a fleet of thermal generators divided in 3 groups: (*i*) static generators (nuclear and combined heat-and-power units), (*ii*) slow generators (coal and combined-cycle units) and (*iii*) fast generators (gas and diesel turbines). At day ahead, before knowing the net demand of the system and possible outages, system operators must decide (*i*) the ON/OFF status and a production band for static generators and (*ii*) the hourly ON/OFF status of slow generators. In real time, after the actual net demand of the system and outages are realized, system operators must decide (*i*) the quarterly production of static generators within the day-ahead band, (*ii*) the quarterly production of slow generators, and (*iii*) the hourly ON/OFF status and quarterly production of fast generators, in order to match the net demand at minimum cost.

We formulate SUC using as basis the generic two-stage SMIP formulation (1)–(4), repeated here for convenience.

$$\begin{aligned} \max_{u,v,w} \ & \sum_{i \in I} (\boldsymbol{c}_i^T \boldsymbol{v}_i + \boldsymbol{d}_i^T \boldsymbol{w}_i) \\ \text{s.t.} \ & \boldsymbol{u} \in \mathcal{U} \\ & (\boldsymbol{v}_i, \boldsymbol{w}_i) \in \mathcal{D}_i, \quad i = 1, \ldots, N \\ & \boldsymbol{v}_i - \boldsymbol{u} = \boldsymbol{0}, \quad i = 1, \ldots, N \end{aligned}$$

In the case of SUC, the blocks of first- and second-stage variables correspond to:

$$\begin{aligned} \boldsymbol{u} &:= \left(\boldsymbol{u}^{ON}, \boldsymbol{u}^{SU}, \boldsymbol{u}^{STAT}, \boldsymbol{q}^{STAT} \right), \\ \boldsymbol{v}_i &:= \left(\boldsymbol{v}_{(\cdot,i,\cdot)}^{ON}, \boldsymbol{v}_{(\cdot,i,\cdot)}^{SU}, \boldsymbol{v}_{(\cdot,i)}^{STAT}, \boldsymbol{p}_{(\cdot,i)}^{STAT} \right) \ \forall i \in I, \\ \boldsymbol{w}_i &:= \left(\boldsymbol{z}_{(\cdot,i,\cdot)}^{ON}, \boldsymbol{z}_{(\cdot,i,\cdot)}^{SU}, \boldsymbol{p}_{(\cdot,i,\cdot)}, \boldsymbol{f}_{(\cdot,i,\cdot)}, \boldsymbol{\theta}_{(\cdot,i,\cdot)}, \boldsymbol{e}_{(\cdot,i,\cdot)}, \boldsymbol{o}_{(\cdot,i,\cdot)} \right) \ \forall i \in I. \end{aligned}$$

In words, $\boldsymbol{u}$ is the vector of ON/OFF and startup decisions of slow generators, and ON/OFF and target production decisions of static generators. $\boldsymbol{v}_i$ is the local copy of

$\boldsymbol{u}$ for scenario i. $\boldsymbol{w}_i$ is the vector of ON/OFF and startup decisions of fast generators, real-time production decisions, and power flow decisions under scenario i.

Domain $\mathcal{D}_i$ enforces the operational limits and physics of the power system under scenario i. Operational limits of slow or fast generator g under scenario i are described by $\mathcal{P}_{g,i}$, defined in Eq. (16).

$$\begin{aligned}\mathcal{P}_{g,i} := \Big\{ (\boldsymbol{v}^{ON}, \boldsymbol{v}^{SU}, \boldsymbol{\rho}) \in \mathbb{B}^{|T_{60}|} \times \mathbb{B}^{|T_{60}|} \times \mathbb{R}^{|T_{15}|} \;\Big|\; & v^{SU}_{g,\tau} \geq v^{ON}_{g,\tau} - v^{ON}_{g,\tau-1}, \\ & \sum_{\sigma=\tau-UT_g+1}^{\tau} v^{SU}_{g,\sigma} \leq v^{ON}_{g,\tau}, \quad \sum_{\sigma=\tau-DT_g+1}^{\tau} v^{SU}_{g,\sigma} \leq 1 - v^{ON}_{g,\tau-DT_g} \;\; \forall \tau \in T_{60}; \\ & \underline{P}_g \,(1-\zeta_{g,i})\, v_{g,\tau(t)} \leq \rho_{g,t} \leq \overline{P}_g \,(1-\zeta_{g,i})\, v_{g,\tau(t)}, \\ & \rho_{g,t} - \rho_{g,t-1} \leq TL_g - (TL_g - R_g) v^{ON}_{g,\tau(t-1)}, \\ & \rho_{g,t-1} - \rho_{g,t} \leq TL_g - (TL_g - R_g) v^{ON}_{g,\tau(t)} \\ & \quad \forall t \in T_{15} \Big\} \;\; \forall g \in G^{SLOW} \cup G^{FAST}\end{aligned} \tag{16}$$

The constraints correspond to startup, minimum up time, minimum down time, production bounds, maximum ramp-up and maximum ramp-down constraints.

Operational limits of static generator g at scenario i are described by $\mathcal{P}^{STAT}$, defined in Eq. (17).

$$\begin{aligned}\mathcal{P}^{STAT}_{g,i} := \Big\{ (v^{STAT}, \rho^{STAT}, \boldsymbol{\rho}) \in \mathbb{B} \times \mathbb{R} \times \mathbb{R}^{|T_{15}|} \;\Big|\; & \underline{P}_g \leq \rho^{STAT}_g \leq \overline{P}_g; \\ & \underline{P}_g \,(1-\zeta_{g,i})\, v^{STAT}_g \leq \rho_{g,t} \leq \overline{P}_g \,(1-\zeta_{g,i})\, v^{STAT}_g, \\ & \rho_{g,t} \geq \rho^{STAT}_g - 1/2 H_g - (\overline{P}_g - 1/2\, H_g)(1-\zeta_{g,i}), \\ & \rho_{g,t} \leq \rho^{STAT} + 1/2 H_g - (\underline{P}_g + 1/2\, H_g)(1-\zeta_{g,i}) \;\; \forall t \in T_{15} \Big\} \\ & \quad \forall g \in G^{STAT}\end{aligned} \tag{17}$$

The constraints correspond to limits for target production, ON/OFF limits for actual production, and lower and upper band limits for actual production. Note that both the ON/OFF status and target production for static generators are not indexed by time period, meaning that they are constant throughout the SUC horizon.

Using the previous definitions for $\mathcal{P}_{g,i}$ and $\mathcal{P}^{STAT}_{g,i}$, we can describe $\mathcal{D}_i$ as follows:

$$\begin{aligned}\mathcal{D}_i := \; & \Big\{ \Big(\boldsymbol{v}^{ON}_{(\cdot,i,\cdot)}, \boldsymbol{v}^{SU}_{(\cdot,i,\cdot)}, \boldsymbol{v}^{STAT}_{(\cdot,i)}, \boldsymbol{p}^{STAT}_{(\cdot,i)}, \boldsymbol{z}^{ON}_{(\cdot,i,\cdot)}, \boldsymbol{z}^{SU}_{(\cdot,i,\cdot)}, \boldsymbol{p}_{(\cdot,i,\cdot)}, \boldsymbol{f}_{(\cdot,i,\cdot)}, \boldsymbol{\theta}_{(\cdot,i,\cdot)}, \\ & \boldsymbol{e}_{(\cdot,i,\cdot)}, \boldsymbol{o}_{(\cdot,i,\cdot)} \Big) \in \mathbb{B}^{|G^{SLOW}||T_{60}|} \times \mathbb{B}^{|G^{SLOW}||T_{60}|} \times \mathbb{B}^{|G^{STAT}|} \\ & \times \mathbb{R}^{|G^{STAT}|} \times \mathbb{B}^{|G^{FAST}||T_{60}|} \times \mathbb{B}^{|G^{FAST}||T_{60}|} \times \mathbb{R}^{|G||T_{15}|} \times \mathbb{R}^{|L||T_{15}|} \\ & \times \mathbb{R}^{|N||T_{15}|} \times \mathbb{R}_+^{|N||T_{15}|} \times \mathbb{R}_+^{|N||T_{15}|} \;\Big| \\ & \Big(\boldsymbol{v}^{ON}_{(g,i,\cdot)}, \boldsymbol{v}^{SU}_{(g,i,\cdot)}, \boldsymbol{p}_{(g,i,\cdot)} \Big) \in \mathcal{P}_{g,i} \;\; \forall g \in G^{SLOW};\end{aligned}$$

$$
\begin{aligned}
&\left(\boldsymbol{z}^{ON}_{(g,i,\cdot)}, \boldsymbol{z}^{SU}_{(g,i,\cdot)}, \boldsymbol{p}_{(g,i,\cdot)}\right) \in \mathcal{P}_{g,i} \ \ \forall g \in G^{FAST};\\
&\left(v^{STAT}_{g,i}, p^{STAT}_{g,i}, \boldsymbol{p}_{(g,i,\cdot)}\right) \in \mathcal{P}^{STAT}_{g,i} \ \ \forall g \in G^{STAT};\\
&f_{l,i,t} = B_l(1-\zeta_{l,i})\left(\theta_{n(l),i,t} - \theta_{m(l),i,t}\right),\\
&-F_l \le f_{l,i,t} \le F_l \ \ \forall l \in L, t \in T_{15};\\
&\sum_{g\in G(n)} p_{g,i,t} + \xi_{n,i,t} + \sum_{l\in L(\cdot,n)} f_{l,i,t} + e_{n,i,t}\\
&= D_{n,t} + \sum_{l\in L(n,\cdot)} f_{l,i,t} + o_{n,i,t} \ \ \forall n \in N, t \in T_{15}\Big\},
\end{aligned}
\tag{18}
$$

Here, we enforce the operational limits of all generators, the DC power flow equations [49], and active power balance at each bus. Note that we introduce demand and production shedding variables in all power balance equations, which ensures that SUC formulation has relatively complete recourse.

We define $\mathcal{U}$ as the linear relaxation of the first-stage constraints in $\mathcal{D}_i$, i.e. those constraints involving only $\boldsymbol{v}^{ON}_{(\cdot,i,\cdot)}$, $\boldsymbol{v}^{SU}_{(\cdot,i,\cdot)}$, $\boldsymbol{v}^{STAT}_{(\cdot,i)}$, $\boldsymbol{p}^{STAT}_{(\cdot,i)}$. Therefore, we can describe $\mathcal{U}$ as in Eq. (19), which includes only bounds and minimum up- and down time restrictions.

$$
\begin{aligned}
\mathcal{U} := \Big\{&\left(\boldsymbol{u}^{ON}, \boldsymbol{u}^{SU}, \boldsymbol{u}^{STAT}, \boldsymbol{q}^{STAT}\right) \in [0,1]^{|G^{SLOW}||T_{60}|}\\
&\times [0,1]^{|G^{SLOW}||T_{60}|} \times [0,1]^{|G^{STAT}|} \times \mathbb{R}^{|G^{STAT}|} \ \Big|\\
&u^{SU}_{g,\tau} \ge u^{ON}_{g,\tau} - u^{ON}_{g,\tau-1}, \ \sum_{\sigma=\tau-UT_g+1}^{\tau} u^{SU}_{g,\sigma} \le u^{ON}_{g,\tau},\\
&\sum_{\sigma=\tau-DT_g+1}^{\tau} u^{SU}_{g,\sigma} \le 1 - u^{ON}_{g,\tau-DT_g} \ \ \forall g \in G^{SLOW}, \tau \in T_{60};\\
&\underline{P}_g \le q^{STAT}_g \le \overline{P}_g \ \ \forall g \in G^{STAT}\Big\}
\end{aligned}
\tag{19}
$$

Finally, the part of the objective function associated with scenario i can be written as

$$
\begin{aligned}
\boldsymbol{c}_i^T \boldsymbol{v}_i + \boldsymbol{d}_i^T \boldsymbol{w}_i \equiv \pi_i\Bigg(&\sum_{g\in G}\sum_{t\in T_{15}} C_g(p_{g,i,t}) + \sum_{\substack{g\in G^{SLOW}\cup\\ G^{FAST}}}\sum_{\tau\in T_{60}}\left(K_g v^{ON}_{g,i,\tau} + S_g v^{SU}_{g,i,\tau}\right)\\
&+ \sum_{g\in G^{STAT}} |T_{60}| K_g v^{STAT}_{g,i} + \sum_{n\in N}\sum_{t\in T_{15}} V e_{n,i,t}\Bigg),
\end{aligned}
$$

This corresponds to variable production costs, fixed and startup costs for slow and fast generators, fixed costs for static generators, and penalty terms for demand shedding. All cost components are weighted by the probability of scenario i, π_i.

B Proofs

Proof of Proposition 1 By inspection we have that,

$$\mathbb{E}\left[I_J^T\left(I_J\nabla f_0^\mu(\boldsymbol{x}^k)+g\left(J,\boldsymbol{x}_J^k\right)\right)\middle|\,\boldsymbol{x}^k\right]=\frac{1}{N}\nabla f_0^\mu(\boldsymbol{x}^k)+\frac{1}{N}\sum_{j=1}^{N}I_j^T g\left(j,\boldsymbol{x}_j^k\right)$$
$$=\frac{1}{N}g(\boldsymbol{x}^k),\ g(\boldsymbol{x}^k)\in\partial f(\boldsymbol{x}^k).$$

□

Proposition 4 *Let $f : X \subseteq \mathbb{R}^n \to \mathbb{R}$ be a non-differentiable convex function, P be a partition of $\{1,\ldots,n\}$ and I_p, $p \in P$, be the matrix that maps $\boldsymbol{x} \in \mathbb{R}^n$ into its p coordinates (i.e. $I_p\boldsymbol{x}$ contains only the p coordinates of $\boldsymbol{x}$). For any $x \in X$, define*

$$D^P(\boldsymbol{x})=\left\{\boldsymbol{g}\in\mathbb{R}^n\,\middle|\,\exists\boldsymbol{g}_p\in\partial f(\boldsymbol{x})\ \forall p\in P,\ g=\sum_{p\in P}I_p\boldsymbol{g}_pI_p^T\right\}.$$

Then, $\partial f(\boldsymbol{x}) \subseteq D^P(\boldsymbol{x})$. Furthermore, the inclusion can hold strictly and $g \in D^P(\boldsymbol{x})\backslash\partial f(\boldsymbol{x})$ might not be an approximate subgradient of f at $\boldsymbol{x}$.

Proof It is trivial to observe that the inclusion holds. We proceed to prove the remaining claims via an example. Consider $f : \mathbb{R}^n \to \mathbb{R}$, $f(\boldsymbol{x}) = \max\{x_1 + x_2, -x_1 - x_2\}$ at $\boldsymbol{x} = \mathbf{0}$ and let $P = \{\{1\},\{2\}\}$. Then, $[-1\ 1]^T \in D^P(\boldsymbol{x})$ because $[-1\ 1]^T = [1\ 0]\,g_{\{1\}}\,[1\ 0]^T + [0\ 1]\,g_{\{2\}}\,[0\ 1]^T$, with $g_{\{1\}} = [-1\ -1] \in \partial f(\mathbf{0})$ and $g_{\{2\}} = [1\ 1] \in \partial f(\mathbf{0})$. We also have that $[-1\ 1]^T \notin \partial f(\mathbf{0})$, which proves that the inclusion can hold strictly. Also, note that there is no $\epsilon > 0$ such that

$$f(\boldsymbol{y}) = \max\{y_1 + y_2, -y_1 - y_2\} \geq 0 + [-1\ 1]\boldsymbol{y} - \epsilon$$

holds for all $\boldsymbol{y} \in \mathbb{R}$, since for any ϵ the inequality is violated at $\boldsymbol{y} = [-\epsilon\ \epsilon]$. Therefore, $[-1\ 1]^T$ is not an approximate subgradient of f at $\mathbf{0}$, which concludes the proof. □

Proof of Lemma 1 Note that, for the expected direction of update rule (11), we have

$$N\cdot(\boldsymbol{x}^k-\boldsymbol{y})^T\mathbb{E}\left[I_J^T\left(I_J\nabla f_0^\mu\left(\boldsymbol{x}^{k-l(k)}\right)+g\left(J,\boldsymbol{x}_J^{k-\ell(J,k)}\right)\right)\middle|\,\mathcal{F}_k\right]$$
$$=(\boldsymbol{x}^k-\boldsymbol{y})^T\nabla f_0^\mu\left(\boldsymbol{x}^{k-l(k)}\right)+\sum_{j=1}^{N}(\boldsymbol{x}_j^k-\boldsymbol{y}_j)^T g\left(j,\boldsymbol{x}_j^{k-\ell(j,k)}\right),\quad\forall\boldsymbol{y}\in X. \tag{20}$$

The first term in the right-hand side of (20) can be expanded as follows:

$$\left(\boldsymbol{x}^k-\boldsymbol{y}\right)^T\nabla f_0^\mu\left(\boldsymbol{x}^{k-l(k)}\right)=\left(\boldsymbol{x}^{k-l(k)}-\boldsymbol{y}\right)^T\nabla f_0^\mu\left(\boldsymbol{x}^{k-l(k)}\right)+\left(\boldsymbol{x}^k-\boldsymbol{x}^{k-l(k)}\right)^T\nabla f_0^\mu\left(\boldsymbol{x}^k\right)$$
$$-\left(\boldsymbol{x}^k-\boldsymbol{x}^{k-l(k)}\right)^T\left(\nabla f_0^\mu\left(\boldsymbol{x}^k\right)-\nabla f_0^\mu\left(\boldsymbol{x}^{k-l(k)}\right)\right)$$

$$
\begin{aligned}
&\geq f_0^{\mu}\left(\boldsymbol{x}^k\right) - f_0^{\mu}(\boldsymbol{y}) \\
&\quad - \left(\boldsymbol{x}^k - \boldsymbol{x}^{k-l(k)}\right)^T \left(\nabla f_0^{\mu}\left(\boldsymbol{x}^k\right) - \nabla f_0^{\mu}\left(\boldsymbol{x}^{k-l(k)}\right)\right) \\
&\geq f_0^{\mu}\left(\boldsymbol{x}^k\right) - f_0^{\mu}(\boldsymbol{y}) \\
&\quad - \left\|\boldsymbol{x}^k - \boldsymbol{x}^{k-l(k)}\right\|_2 \left\|\nabla f_0^{\mu}\left(\boldsymbol{x}^k\right) - \nabla f_0^{\mu}\left(\boldsymbol{x}^{k-l(k)}\right)\right\|_2 \\
&\geq f_0^{\mu}\left(\boldsymbol{x}^k\right) - f_0^{\mu}(\boldsymbol{y}) - L_0^{\mu}\left\|\boldsymbol{x}^k - \boldsymbol{x}^{k-l(k)}\right\|_2^2,
\end{aligned}
$$

In the second line above we use the convexity of f_0^{μ} $((\boldsymbol{x}-\boldsymbol{y})^T \nabla f_0^{\mu}(\boldsymbol{x}) \geq f_0^{\mu}(\boldsymbol{x}) - f_0^{\mu}(\boldsymbol{y}))$, in the third line we use the Cauchy-Schwarz inequality, and in the fourth line we use the definition of the Lipschitz constant of the gradient of f_0^{μ}. Following a similar reasoning, each of the terms under the sum on the right hand side of (20) can be expanded as follows:

$$
\begin{aligned}
(\boldsymbol{x}_j^k - \boldsymbol{y}_j)^T g\left(j, \boldsymbol{x}_j^{k-\ell(j,k)}\right) &\geq f_j\left(\boldsymbol{x}_j^k\right) - f_j\left(\boldsymbol{y}_j\right) \\
&\quad - \left\|\boldsymbol{x}^k - \boldsymbol{x}^{k-\ell(j,k)}\right\|_2 \left\|g\left(j, \boldsymbol{x}_j^k\right) - g\left(j, \boldsymbol{x}_j^{k-\ell(j,k)}\right)\right\|_2 \\
&\geq f_j\left(\boldsymbol{x}_j^k\right) - f_j\left(\boldsymbol{y}_j\right) - 2D\left\|\boldsymbol{x}^k - \boldsymbol{x}^{k-\ell(j,k)}\right\|_2,
\end{aligned}
$$

In the second line above we use Assumption 1. Furthermore, Assumptions 1 and 2 allow us to bound the difference between current and delayed iterates using the stepsize assumptions. To see this, note the following:

$$
\begin{aligned}
\left\|\boldsymbol{x}^k - \boldsymbol{x}^{k-\ell(j,k)}\right\|_2 &\leq C \sum_{m=k-\ell(j,k)}^{k-1} \lambda_m \leq C \sum_{m=k-L}^{k-1} \lambda_m, \\
\left\|\boldsymbol{x}^k - \boldsymbol{x}^{k-l(k)}\right\|_2^2 &\leq C^2 \sum_{m=k-L}^{k-1} \lambda_m^2,
\end{aligned}
$$

Using the previous expressions, we arrive to relation (21), which concludes the proof.

$$
\begin{aligned}
&N \cdot (\boldsymbol{x}^k - \boldsymbol{y})^T \mathbb{E}\left[I_J^T\left(I_J \nabla f_0^{\mu}\left(\boldsymbol{x}^{k-l(k)}\right) + g\left(J, \boldsymbol{x}_J^{k-\ell(J,k)}\right)\right) \middle| \mathcal{F}_k\right] \\
&\quad \geq f(\boldsymbol{x}^k) - f(\boldsymbol{y}) - \left(C^2 L_0^{\mu} \sum_{m=k-L}^{k-1} \lambda_m^2 + 2CDN \sum_{m=k-L}^{k-1} \lambda_m\right)
\end{aligned} \tag{21}
$$

□

Proof of Proposition 2 Let $\boldsymbol{h}_j = I_j^T(I_j \nabla f_0^{\mu}(\boldsymbol{x}^{k-l(k)}) + g(j, \boldsymbol{x}_j^{k-\ell(j,k)}))$ and J be a discrete uniform random variable on the set $\{1, \ldots, N\}$. Then for all $k = 1, \ldots, \infty$ and $\boldsymbol{y} \in X$, we have

$$\begin{aligned}\|\boldsymbol{x}^{k+1}-\boldsymbol{y}\|_2^2 &= \left\|\mathcal{P}_X\left[\boldsymbol{x}^k-\lambda_k\boldsymbol{h}_{j(k)}\right]-\boldsymbol{y}\right\|_2^2\\ &\le \left\|\boldsymbol{x}^k-\lambda_k\boldsymbol{h}_{j(k)}-\boldsymbol{y}\right\|_2^2\\ &\le \|\boldsymbol{x}^k-\boldsymbol{y}\|_2^2-2\lambda_k(\boldsymbol{x}^k-\boldsymbol{y})^T\boldsymbol{h}_{j(k)}+\lambda_k^2C_2^2,\end{aligned}$$

where in the second line we use the nonexpansive property of the projection. Therefore, for the expectation conditioned on $\mathcal{F}_k$ it holds that

$$\mathbb{E}\left[\|\boldsymbol{x}^{k+1}-\boldsymbol{y}\|_2^2\middle|\mathcal{F}_k\right]\le\|\boldsymbol{x}^k-\boldsymbol{y}\|_2^2-2\lambda_k(\boldsymbol{x}^k-\boldsymbol{y})^T\mathbb{E}\left[\boldsymbol{h}_J\middle|\mathcal{F}_k\right]+\lambda_k^2C^2.$$

Finally, by substituting $\mathbb{E}\left[\boldsymbol{h}_J\middle|\mathcal{F}_k\right]$ in the last relation using (21) (Lemma 1) we obtain the desired inequality. □

Proof of Proposition 3 The first two inequalities follow directly from Assumption 3. For the third inequality, we have

$$\sum_{k=0}^{\infty}\lambda_k\sum_{m=k-L}^{k-1}\lambda_m\ \le\ \hat{G}^2\sum_{k=0}^{\infty}\gamma_k\sum_{m=k-L}^{k-1}\gamma_m\ \le\ \hat{G}^2L\sum_{k=0}^{\infty}\gamma_{k-L}^2\ <\ \infty,$$

For the last inequality above, we use the fact that $\{\gamma_k\}$ is non-increasing. Therefore, $\sum_{m=k-L}^{k-1}\gamma_m\le L\gamma_{k-L}\ \forall k$. The same reasoning applies to the fourth inequality of the present proposition. □

Theorem 2 (Supermartingale Convergence Theorem [11, Prop. 4.2])

Let X_t, Y_t and Z_t, $t=0,1,2,\ldots$, be three sequences of random variables and let $\mathcal{F}_t$, $t=0,1,2,\ldots$, be sets of random variables such that $\mathcal{F}_t\subset\mathcal{F}_{t+1}$ for all t. Suppose that:

(a) The random variables X_t, Y_t and Z_t are nonnegative, and are functions of the random variables in $\mathcal{F}_t$.
(b) For each t, we have $\mathbb{E}[X_{t+1}|\mathcal{F}_t]\le X_t-Y_t+Z_t$.
(c) There holds $\sum_{t=0}^{\infty}Z_t<\infty$.

Then, we have $\sum_{t=0}^{\infty}Y_t<\infty$, and the sequence X_t converges to a non-negative random variable X with probability 1.

Proof of Theorem 1 From Proposition 2, with $\boldsymbol{y}=\boldsymbol{x}^*$, we obtain

$$\begin{aligned}\mathbb{E}\left[\|\boldsymbol{x}^{k+1}-\boldsymbol{x}^*\|_2^2\middle|\mathcal{F}_k\right]\le\|\boldsymbol{x}^k-\boldsymbol{x}^*\|_2^2&-2\frac{\lambda_k}{N}\left(f(\boldsymbol{x}^k)-f(\boldsymbol{x}^*)\right)\\&+\lambda_k^2\hat{C}^2+2\frac{C^2L_0^{\mu}}{N}\lambda_k\sum_{m=k-L}^{k-1}\lambda_m^2+4CD\lambda_k\sum_{m=k-L}^{k-1}\lambda_m.\end{aligned}$$

Using Proposition 3 and by the Supermartingale Convergence Theorem (Theorem 2), with probability 1 and for each $\boldsymbol{x}^* \in X^*$, we have

$$\sum_{k=0}^{\infty} \lambda_k \big(f(\boldsymbol{x}^k) - f(\boldsymbol{x}^*) \big) < \infty, \tag{22}$$

and with probability 1 the sequence $\left\{ \|\boldsymbol{x}^k - \boldsymbol{x}^*\|_2 \right\}$ converges to a random variable. The rest of the proof follows exactly the proof of [35, Prop. 3.4] (see also [21, Thm. 2]) and it is repeated here only for the sake of making the material self-contained.

For each $\boldsymbol{x}^* \in X^*$, let Ω_{x^*} denote the set of all sample paths for which Eq. (22) holds and $\left\{ \|\boldsymbol{x}^k - \boldsymbol{x}^*\|_2 \right\}$ converges. By convexity of f, the set X^* is convex, so there exist vectors $\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_p \in X^*$ that span the smallest affine set containing X^*, and are such that $\boldsymbol{v}_j - \boldsymbol{v}_0$, $j = 1, \ldots, p$, are linearly independent.

The intersection $\Omega = \cap_{j=1}^{p} \Omega_{v_j}$ has probability 1, and for each sample path in Ω, the sequences $\left\{ \|\boldsymbol{x}^k - \boldsymbol{v}_j\|_2 \right\}$, $j = 0, \ldots, p$, converge. Thus, with probability 1, $\{\boldsymbol{x}^k\}$ is bounded, and therefore it has limit points. Furthermore, for each sample path in Ω, by Eq. (22) and the relation $\sum_{k=0}^{\infty} \lambda_k = \infty$, it follows that

$$\liminf_{k \to \infty} = f^*,$$

implying that $\{\boldsymbol{x}^k\}$ has at least one limit point that belongs to X^* by continuity of f. For any sample path in Ω, let $\bar{\boldsymbol{x}}$ and $\hat{\boldsymbol{x}}$ be two limit points of $\{\boldsymbol{x}^k\}$ such that $\bar{\boldsymbol{x}} \in X^*$. Because the sequences $\left\{ \|\boldsymbol{x}^k - \boldsymbol{v}_j\|_2 \right\}$, $j = 0, \ldots, p$, converge, we must have

$$\|\bar{\boldsymbol{x}} - \boldsymbol{v}_j\|_2 = \|\hat{\boldsymbol{x}} - \boldsymbol{v}_j\|_2, \quad \forall j = 0, 1, \ldots, p.$$

Moreover, since $\bar{\boldsymbol{x}} \in X^*$, the preceding relation can hold only for $\bar{\boldsymbol{x}} = \hat{\boldsymbol{x}}$ by convexity of X^* and the choice of vectors $\boldsymbol{v}_j$. Hence, for each sample path in Ω, the sequence $\{\boldsymbol{x}^k\}$ has a unique limit point in X^*, implying that $\{\boldsymbol{x}^k\}$ converges to some optimal solution with probability 1. □

C Technical specifications of computation platforms used for the numerical experiments

This section list the technical specifications of the computers or high performance computing clusters used in the numerical experiments presented in this paper. Different clusters were used as they were available to the authors.

Sierra cluster: hosted at Lawrence Livermore National Laboratory (LLNL). Comprised of 1944 nodes, each node with 2 × Intel Xeon X5660 processors (6 cores, 12MB cache), totalling 12 cores per node, and 24GB of RAM. Used only for development and testing.

Cab cluster: hosted at LLNL. Comprised of 1296 nodes, each node with 2 × Intel Xeon E5-2670 processors (8 cores, 20MB cache), totalling 16 cores per node, and

Table 7 Progressive hedging results for WECC instances

N	Season	Day	Asynchronous algorithm					Progressive hedging			
			Iters./N	Solution time (s)	LB (MM$)	UB (MM$)	Gap (%)	Iters.	Solution time (s)	LB (MM$)	Gap (%)
10	Autumn	WD	11.4	58.1	− 7.179	− 7.108	1.00	100	1130.3	− 7.192	1.18
	Autumn	WE	11.0	40.5	− 5.153	− 5.101	1.00	100	524.5	− 5.153	1.01
	Spring	WD	12.7	51.8	− 5.939	− 5.880	0.99	39	364.1	− 5.916	0.61
	Spring	WE	75.2	759.3	− 2.875	− 2.846	1.00	100	1680.4	− 2.876	1.02
	Summer	WD	6.3	31.2	− 11.309	− 11.200	0.97	100	1131.1	− 11.261	0.55
	Summer	WE	12.9	52.5	− 7.170	− 7.099	1.00	9	70.1	− 7.134	0.50
	Winter	WD	9.9	49.3	− 7.009	− 6.939	1.00	15	126.2	− 6.99	0.73
	Winter	WE	75.3	872.6	− 3.435	− 3.402	0.97	100	607.0	− 3.44	1.12
100	Autumn	WD	8.6	52.8	− 8.266	− 8.184	1.00	17	203.1	− 8.233	0.59
	Autumn	WE	37.1	241.1	− 5.110	− 5.059	1.00	100	1432.1	− 5.099	0.79
	Spring	WD	51.8	369.3	− 5.798	− 5.740	1.00	100	1014.6	− 5.774	0.59
	Spring	WE	200.0	2058.3	− 3.308	− 3.275	1.00	100	1954.9	− 3.312	1.13
	Summer	WD	53.6	374.9	− 11.379	− 11.268	0.98	100	3706.3	− 11.327	0.52
	Summer	WE	10.8	55.1	− 8.449	− 8.365	1.00	100	1014.3	− 8.423	0.69
	Winter	WD	27.7	209.3	− 6.049	− 5.988	1.00	100	1538.4	− 6.030	0.69
	Winter	WE	200.0	2643.1	− 3.556	− 3.520	1.01	91	1631.9	− 3.560	1.14
Geomeric mean				174.2					758.5		

Optimality gaps for progressive hedging are computed against the upper bound of our asynchronous algorithm

32 GB of RAM. Decommissioned in 2018. Used for results presented in Tables 2, 3, 5, 6, 8 and 10, and Fig. 6.
Quartz cluster: hosted at LLNL. Comprised of 3018 nodes, each node with 2 × Intel Xeon E5-2695 processors (18 cores, 45MB cache), totalling 36 cores per node, and with 128 GB of RAM. Used for results presented in Table 7.
Lemaitre3 cluster: hosted at the Consortium des Équipements de Calcul Intensif (CÉCI). Comprised of 80 nodes, each node with 2 × Intel SkyLake 5118 (12 cores, 16.5MB cache), totalling 24 cores per node, and 96GB of RAM. Used for results presented in Table 4.
4-core laptop: Asus ROG G750JZ, with Intel Core i7 4700HQ processor (4 cores, 6MB cache) and 16GB of RAM. Used for results presented in Table 9.

D Comparison against progressive hedging

This section presents a comparison between our asynchronous algorithm and progressive hedging, as implemented in the software package PySP [54]. We conduct the comparison using SUC instances for the WECC system with 10 and 100 scenarios. We excluded 1000-scenario instances from the comparison due to technical difficulties faced by PySP in scaling to 1000 processes. CWE instances were not considered for the comparison, due to the long solution times of certain scenario subproblems (up to 12 h). The performance of our asynchronous algorithm in these instances benefited significantly from the period relaxation of the initialization phase 5.3, for which there is no counterpart in PySP.

We use the Quartz cluster hosted at the Lawrence Livermore National Laboratory to conduct the numerical experiments presented in this section. Each experiment uses $1+N$ processes (1 *Master*, $S = N$ *Slaves*), 1 core per process, and solves subproblems using the Xpress mathematical programming solver (limited to 2 threads). We set the parameters of progressive hedging following [18]. In particular, we set the termination tolerance to 0.01, we set the maximum number of iterations to 100, we select the value of ρ using the cost proportional rule [18], and we use the *WW* extensions for accelerating convergence [18]. The *WW* extensions set subproblem optimality gaps dynamically (e.g. initial subproblem gap: 0.025, final subproblem gap: 0.0001), fix converged variables and break cycles. For our approach, we use the best configuration found in the experiments presented in Table 2, with the exception of the optimality gap of subproblems and the *Dual Share*, which have been set to 0.005 and 0.75, respectively.

The results of this experimental comparison are presented in Table 7. On average, our approach terminates 4.35 times faster than progressive hedging. Progressive hedging, in most cases, achieves solutions within 1% optimality and its solutions are than those provided by our approach on ∼1/3 of the instances. The quality of these solutions, however, cannot be verified within the same progressive hedging algorithm for free (configuring PySP to compute bounds every 10 progressive hedging iterations increases computation times by 8.5%). Our approach, on the other hand, provides both primal solutions and optimality gaps continuously, exploiting fully the computational power at its disposal thanks to its asynchronous nature. Furthermore, we can acceler-

ate the proposed asynchronous algorithm by allowing it to use more than N parallel workers. For instance, considering an estimate parallel efficiency of 70% at $2N$ parallel workers (see Fig. 6), making $2N$ parallel workers available to the asynchronous algorithm would make it 40% faster than it is with N parallel workers. By contrast, progressive hedging cannot make use of more than N parallel workers, even if they are made available to the algorithm.

Overall, these results demonstrate the ability of the proposed asynchronous approach to solve SUC instances several times faster than the state-of-the-art, while providing optimality guarantees. At the same time, the good quality of the solutions provided by progressive hedging suggests directions for the future development of better primal recovery heuristics.

E Detailed results

This section presents solution statistics per instance for the best configurations found. Table 8 presents solution statistics for WECC instances using the Cab supercomputer at LLNL. Table 9 presents solution statistics for WECC instances using a personal laptop equipped with an Intel Core i7 (4 cores, 8 threads) and 16GB of memory. Table 10 presents solution statistics for CWE instances using Cab.

Table 8 Solution statistics for WECC. Results are obtained using the best configuration in Table 2: Polyak stepsize and IS primal recovery

N	Season	Day	*LB* (MM$)	*UB* (MM$)	Sol. time 1% (s)	Av. *approx* dual scenario time (s)	Av. *dual* f_0 time (s)	Av. *primal projection* time (s)	*Dual scenario* time (s)		*Second stage scenario* time (s)		Math. Prog. time (%)
									Av.	Sd.	Av.	Sd.	
10	Autumn	WD	− 7.161	− 7.090	33.53	1.25	0.25	5.29	3.22	0.67	1.74	0.45	97.4
	Autumn	WE	− 5.152	− 5.100	75.52	1.06	0.23	1.78	2.59	1.11	1.41	1.54	99.1
	Spring	WD	− 5.940	− 5.881	126.25	1.07	0.23	1.32	2.99	1.15	1.38	0.29	99.1
	Spring	WE	− 2.875	− 2.847	1150.14	1.21	0.21	0.92	9.20	6.69	1.42	1.32	99.9
	Summer	WD	− 11.301	− 11.190	92.25	1.17	0.24	3.36	4.00	1.84	1.83	0.37	99.2
	Summer	WE	− 7.166	− 7.095	93.97	1.27	0.24	8.21	3.29	0.74	1.72	0.32	99.2
	Winter	WD	− 7.007	− 6.937	54.11	1.10	0.24	3.06	3.41	0.83	1.63	0.30	98.5
	Winter	WE	− 3.438	− 3.405	1291.48	1.17	0.20	0.58	10.98	6.61	1.49	0.53	99.9
100	Autumn	WD	− 8.256	− 8.174	38.05	1.14	0.19	9.76	3.10	0.57	1.64	0.28	97.0
	Autumn	WE	− 5.104	− 5.054	141.11	1.07	0.18	5.36	2.98	1.89	1.34	0.33	99.1
	Spring	WD	− 5.798	− 5.740	237.72	1.00	0.19	2.90	3.43	1.62	1.37	0.21	99.5
	Spring	WE	− 3.309	− 3.276	892.53	1.12	0.18	7.52	6.17	4.86	1.38	0.75	99.9
	Summer	WD	− 11.380	− 11.267	755.19	1.34	0.19	9.08	4.39	2.17	1.87	0.54	99.8
	Summer	WE	− 8.460	− 8.376	175.44	1.26	0.19	9.65	3.12	0.60	1.75	0.70	99.4
	Winter	WD	− 6.060	− 5.999	776.76	1.09	0.18	0.72	3.90	1.58	1.62	0.23	99.8
	Winter	WE	− 3.556	− 3.522	1126.09	1.26	0.18	4.98	7.99	4.51	1.51	0.45	99.9
1000	Autumn	WD	− 8.106	− 8.026	143.18	1.20	0.20	18.57	3.06	0.66	1.59	0.45	98.5
	Autumn	WE	− 5.391	− 5.337	383.42	1.07	0.19	12.20	2.72	1.95	1.34	0.47	98.3
	Spring	WD	− 5.891	− 5.832	868.55	1.04	0.19	1.39	3.26	1.34	1.42	0.53	98.7
	Spring	WE	− 3.384	− 3.350	5134.57	1.17	0.18	19.53	7.78	6.96	1.36	0.69	99.6
	Summer	WD	− 11.418	− 11.304	3080.87	3.28	0.19	19.22	3.98	2.19	1.84	0.56	99.1
	Summer	WE	− 8.296	− 8.214	433.10	1.17	0.19	20.10	3.19	0.74	1.66	0.26	98.6
	Winter	WD	− 5.792	− 5.737	6427.02	1.07	0.18	0.14	3.96	1.38	1.70	0.78	98.5
	Winter	WE	− 3.621	− 3.585	3809.37	1.18	0.19	4.13	8.32	6.16	1.49	0.48	99.7

Table 9 Solution statistics for the 10-scenario instance of the WECC system, when running on a 4-core laptop

N	Season	Day	LB (MM$)	UB (MM$)	Sol. time 1% (s)	Av. *approx dual scenario* time (s)	Av. *dual* f_0 time (s)	Av. *primal projection* time (s)	*Dual scenario* times (s)		*Second stage* time (s)		Math. prog. time (%)
									Av.	Sd.	Av.	Sd.	time (%)
10	Autumn	WD	−7.169	−7.104	204.53	0.93	0.18	1.95	3.39	1.38	1.42	0.24	99.7
	Autumn	WE	−5.143	−5.091	162.13	0.79	0.18	0.72	2.71	1.52	1.24	0.76	99.7
	Spring	WD	−5.941	−5.883	322.46	0.81	0.17	0.63	2.83	1.12	1.21	0.23	99.9
	Spring	WE	−2.876	−2.848	4323.46	0.89	0.17	0.79	9.78	6.08	1.24	0.79	100.0
	Summer	WD	−11.300	−11.188	167.08	0.87	0.19	4.59	3.63	1.26	1.63	0.26	99.7
	Summer	WE	−7.164	−7.093	180.61	0.95	0.19	6.59	3.05	0.74	1.58	0.95	99.7
	Winter	WD	−7.015	−6.945	258.10	0.82	0.18	0.42	3.51	1.01	1.40	0.21	99.8
	Winter	WE	−3.440	−3.407	4854.34	0.90	0.17	0.42	11.75	7.31	1.33	0.59	100.0

Results are obtained using the best configuration in Table 2: Polyak stepsize and IS primal recovery

Table 10 Solution statistics for CWE

N	Season	Day	*LB* (MM€)	*UB* (MM€)	Sol. time (s)	Av. *approx dual scenario* time (s)	Av. *dual* f_0 time (s)	Av. *primal projection* time (s)	*Dual scenario* time (s)		*Second stage scenario* time (s)		Math. prog. time (%)
									Av.	Sd.	Av.	Sd.	
30	Autumn	WD	− 49.685	− 49.212	3429.7	58.3	2.2	8.8	849.0	379.8	89.3	8.0	99.0
	Autumn	WE	− 37.099	− 36.779	1589.2	60.3	2.3	15.6	879.9	331.2	86.7	12.7	97.6
	Spring	WD	− 33.152	− 32.839	9279.1	79.4	2.2	1.3	3295.8	1408.6	88.6	27.2	99.4
	Spring	WE	− 24.436	− 24.216	6028.2	76.3	2.0	1.3	4473.0	1452.4	81.1	37.7	99.2
	Summer	WD	− 35.805	− 35.470	2493.1	68.5	2.2	1.3	1996.8	212.3	81.3	13.2	98.4
	Summer	WE	− 23.385	− 23.175	3046.2	72.7	2.2	1.4	1086.3	204.7	75.0	20.4	98.1
	Winter	WD	− 30.462	− 30.165	2136.9	62.4	2.2	1.3	849.1	363.7	76.7	12.4	97.6
	Winter	WE	− 25.782	− 25.543	2447.3	62.4	2.0	1.3	820.6	294.1	92.9	32.6	98.5
60	Autumn	WD	− 49.792	− 49.342	3538.7	58.5	2.2	12.3	889.5	508.5	89.1	9.1	99.1
	Autumn	WE	− 37.144	− 36.811	2178.6	60.0	2.3	2.5	989.9	374.8	85.7	10.6	98.2
	Spring	WD	− 33.130	− 32.821	8323.4	79.1	2.2	1.3	3295.1	1476.3	89.7	30.2	99.3
	Spring	WE	− 24.397	− 24.170	6116.6	76.7	2.0	1.4	4361.6	1524.8	79.3	25.4	99.2
	Summer	WD	− 35.742	− 35.500	2547.8	68.6	2.2	1.3	2016.4	240.5	82.6	15.0	98.4
	Summer	WE	− 23.390	− 23.185	2253.3	74.3	2.3	1.4	1176.8	282.2	75.1	18.9	97.4
	Winter	WD	− 30.442	− 30.143	2211.8	62.1	2.2	1.4	860.6	401.9	76.8	12.6	97.5
	Winter	WE	− 25.754	− 25.504	3023.3	62.8	2.0	1.4	815.6	351.6	96.6	42.4	98.5

Table 10 continued

N	Season	Day	*LB* (MM€)	*UB* (MM€)	Sol. time (s)	Av. *approx dual scenario* time (s)	Av. *dual* f_0 time (s)	Av. *primal projection* time (s)	*Dual scenario* time (s)		*Second stage scenario* time (s)		Math. prog. time (%)
									Av.	Sd.	Av.	Sd.	
120	Autumn	WD	− 49.614	− 49.120	2366.0	58.6	2.2	16.5	784.0	295.4	89.2	9.3	98.5
	Autumn	WE	− 37.010	− 36.642	1877.3	59.6	2.2	2.1	874.2	375.2	86.2	10.5	97.9
	Spring	WD	− 33.187	− 32.857	7952.6	78.3	2.1	1.3	3383.2	1408.9	88.9	27.0	99.2
	Spring	WE	− 24.436	− 24.214	5973.0	76.1	2.0	1.4	4135.9	1638.7	78.3	25.7	99.2
	Summer	WD	− 35.806	− 35.468	2497.3	68.4	2.1	1.5	2012.9	226.3	81.5	14.5	98.4
	Summer	WE	− 23.377	− 23.147	3506.5	74.2	2.2	1.5	1258.2	385.4	75.6	20.3	98.4
	Winter	WD	− 30.431	− 30.168	3932.2	61.6	2.1	1.4	1020.4	486.0	76.7	12.6	98.5
	Winter	WE	− 25.780	− 25.522	2904.4	62.2	2.1	1.5	793.6	387.5	93.2	32.2	98.5

Results are obtained using the same configuration as in Table 3: Polyak stepsize and IS primal recovery

F Estimation procedure for idle times of slaves

F.1 A reference synchronous parallel algorithm

In order to estimate the idle times of slaves in a synchronous algorithm, we first need to specify the exact algorithm for which we are performing the estimation. We consider a synchronous parallel algorithm for SMIPs based on the Lagrangian decomposition presented in Sect. 2. This hypothetical synchronous algorithm uses the subgradient method to minimize the dual function, as in [42]. Additionally, the synchronous algorithm uses one of the primal recovery heuristics presented in Sect. 4 to detect high-quality primal candidates.

Let $1+S$ be the number of parallel workers available for executing the synchronous algorithm over a SMIP with N scenarios. Then, 1 worker will be used as *Master*, $DS = \min\{N, \text{round}(\textit{Dual Share} \times S)\}$ workers will be used as *Dual Slaves*, and $PS = S - DS$ workers will be used as *Primal Slaves*.

At each iteration k of the subgradient method, the *Master* uses the *Dual Slaves* to evaluate $f_0(\boldsymbol{x}^k)$ and $f_i(\boldsymbol{x}_i^k)$ for all $i = 1, \ldots, N$, and its respective subgradients. The *Master*, then, has at its disposal DS *Dual Slaves* for performing $N + 1 > DS$ evaluations. The *Master* performs the required function evaluations using a single queue of dual tasks, leaving the evaluation of $f_0(\boldsymbol{x}^k)$ at the end of the queue (f_0 is much easier to evaluate than any f_i). At the beginning of the iteration, as all *Dual Slaves* have no assigned task, the *Master* simultaneously assigns the first DS tasks of the dual queue to the *Dual Slaves*. The next task in the dual queue is then assigned to the first *Dual Slave* to finish its current task, until all tasks are completed, and the *Master* can perform the subgradient update in order to obtain $\boldsymbol{x}^{k+1}$.

Concurrently with carrying out subgradient iterations, the *Master* uses the *Primal Slaves* for evaluating the primal candidates recovered from the scenario subproblems. This evaluation is carried out sequentially, in the sense that the *Master* evaluates a candidate only after completing the evaluation of the previous candidate. However, due to the abundance of primal candidates, we do not impose any locking mechanism between subgradient iterations and primal recovery. That is, the synchronous algorithm can proceed to iteration $l > k$ even if there are primal candidates derived from iteration k pending of evaluation. The evaluation of the p-th primal candidate $\bar{\boldsymbol{v}}^p$ requires evaluating $h_i(\bar{\boldsymbol{v}}^p)$ for all $i = 1, \ldots, N$. If $PS < N$, these evaluations are carried out using a single queue of primal tasks, which operates in an analogous fashion to the dual queue. Once the evaluation of the p-th candidate is complete, the *Master* moves on to the evaluation of the $(p+1)$-th candidate.

The synchronous algorithm terminates once the upper bound (provided by the subgradient method) and the lower bound (provided by primal recovery) are sufficiently close, or when a certain number of subgradient iterations have been completed.

By design, the synchronous algorithm described above has smaller or equal idle times for slaves than synchronous decomposition methods present in the literature [18,28,42] for the same number of parallel processes. This is because of the unlocked execution of dual iterations and primal recovery, which contrasts with the methods in the literature. The latter methods recover primal solutions after each dual/progressive

hedging iteration or after a predetermined number of dual or progressive hedging iterations. The estimated idle times of slaves for the synchronous algorithm are, therefore, optimistic approximations for the methods in the literature.

F.2 Estimation of synchronous idle time using execution data of the asynchronous algorithm

The idle times of slaves in the synchronous algorithm depend on only two factors: (*i*) the solution times of the subproblems (either dual subproblems, f_0, f_i, or second-stage cost evaluations, h_i) and (*ii*) the order of these subproblems in their respective queues. We estimate the idle times of slaves by considering both factors as follows.

We estimate the proportion of idle time for the synchronous algorithm using the recorded solution times of subproblems in the asynchronous algorithm presented in this paper as proxies for the solution times of the synchronous algorithm. Specifically, for each subproblem s solved by the asynchronous algorithm we record (*i*) its $C_s \in \{Dual, Primal\}$, (*ii*) its description D_s with $D_s \in \{0, 1, \ldots, N\}$ for dual subproblems (indicating which dual function $f_i, i = 0, \ldots, N$ was evaluated) and $D_s \in \{1, \ldots, P\} \times \{1, \ldots, N\}$ for primal subproblems (indicating which function $h_i(\boldsymbol{v}^p)$ was evaluated), and (*iii*) its evaluation time T_s. These quantities are organized in an ordered list of tuples $\mathcal{L} = \{(C_s, D_s, T_s), s = 1, 2, \ldots\}$, where indices s are assigned according to the termination timestamp of each task. We can think of $\mathcal{L}$ as the log of the asynchronous algorithm.

The solution times of second-stage cost evaluations can be directly obtained from $\mathcal{L}$. The solution times of the dual subproblems for the reference synchronous algorithm are not readily available, because the asynchronous algorithm does not require to evaluate all component functions over an iteration or a number of iterations. We assume, then, that the list $\mathcal{L}$ contains at least $K > 1$ evaluations of each component function f_i, and that the evaluation time of f_i in the k-th iteration of the synchronous algorithm, where $k < K$, can be approximated by $T_{s(k)}$ where $s(k)$ indicates the k-th occurrence of $(Dual, i, \cdot)$ in the list $\mathcal{L}$. Thus, from the execution of the asynchronous algorithm we obtain estimations for solution times of subproblems over K dual iterations, and P primal candidate evaluations of the synchronous algorithm.

In order to account for the unknown order in which these subproblems may appear in the dual and primal queues of the reference synchronous algorithm, we conduct Monte Carlo simulations of wall time for dual subproblems and second-stage cost evaluations over their respective slaves. Note that we can simulate the wall time of dual subproblems separately from that of second-stage cost evaluations, because these two processes are not locked in the synchronous algorithm.

Let $W_S(\mathcal{T}, R)$ be the wall time for executing the set $\mathcal{T}$ of tasks using S slaves and where tasks are launched as slaves become available in the order indicated by the permutation R. Then we can express the expected wall time W_k for the k-th dual iteration with respect to all possible permutations of $\{1, \ldots, N\}$ with equal probability as $\mathbb{E}_R[W_k] = \mathbb{E}_R[W_{DS}(\{f_1, \ldots, f_N, f_0\}, (R; N + 1))]$. We estimate this quantity for each dual iteration $k = 1, \ldots, K$ using 1000 different Monte Carlo samples of permutations, obtaining estimates $\tilde{W}_k$. Assuming independence of the ordering

between iterations, we can compute an estimate of the total dual wall time $\tilde{W} = \sum_{k=1}^{K} \tilde{W}_k$. At the same time, we can compute the total time actually spent evaluating dual functions T_{Dual} by simply aggregating the solution times of dual subproblems up to iteration K. The proportion of idle time of *Dual Slaves* T^{idle}_{Dual}, then, corresponds to $T^{\text{idle}}_{Dual} = 1 - (DS \cdot \tilde{W})/T_{Dual}$. An analogous procedure is followed for computing the proportion of idle time in *Primal Slaves*, T^{idle}_{Primal}.

Finally, the estimated proportion of idle time of slaves presented in Table 6 is determined as $T^{\text{idle}} = DS/S \cdot T^{\text{idle}}_{Dual} + PS/S \cdot T^{\text{idle}}_{Primal}$.

References

1. Ahmed, S.: A scenario decomposition algorithm for 0–1 stochastic programs. Oper. Res. Lett. **41**(6), 565–569 (2013). https://doi.org/10.1016/j.orl.2013.07.009
2. Ahmed, S.: Corrigendum to "A scenario decomposition algorithm for 0–1 stochastic programs" [Oper. Res. Lett. 41(6) (2013) 565–569]. Oper. Res. Lett. **43**(2), 215–217 (2015). https://doi.org/10.1016/j.orl.2015.01.006
3. Ahmed, S., Garcia, R.: Dynamic capacity acquisition and assignment under uncertainty. Ann. Oper. Res. **124**(1), 267–283 (2003). https://doi.org/10.1023/B:ANOR.0000004773.66339.df
4. Ahmed, S., Garcia, R., Kong, N., Ntaimo, L., Gyana Parija, F.Q., Sen, S.: Siplib: A stochastic integer programming test problem library (2015). http://www.isye.gatech.edu/~sahmed/siplib. Accessed 3 Mar 2020
5. Ahmed, S., Tawarmalani, M., Sahinidis, N.V.: A finite branch-and-bound algorithm for two-stage stochastic integer programs. Math. Program. **100**(2), 355–377 (2004). https://doi.org/10.1007/s10107-003-0475-6
6. Angulo, G., Ahmed, S., Dey, S.S.: Improving the integer l-shaped method. INFORMS J. Comput. **28**(3), 483–499 (2016). https://doi.org/10.1287/ijoc.2016.0695
7. Anstreicher, K.M., Wolsey, L.A.: Two "well-known" properties of subgradient optimization. Math. Program. **120**(1), 213–220 (2009)
8. Aravena, I., Papavasiliou, A.: Renewable energy integration in zonal markets. IEEE Trans. Power Syst. **32**(2), 1334–1349 (2017). https://doi.org/10.1109/TPWRS.2016.2585222
9. Aravena, I., Papavasiliou, A.: Asynchronous Lagragian scenario decompostion: SMIP instances. Zenodo (2020). https://doi.org/10.5281/zenodo.3696477
10. Aravena, I., Papavasiliou, A.: Asynchronous Lagragian scenario decompostion: source code. Zenodo (2020). https://doi.org/10.5281/zenodo.3697310
11. Bertsekas, D.P., Tsitsiklis, J.N.: Neuro-Dynamic Programming, 1st edn. Athena Scientific (1996)
12. Birge, J.R., Louveaux, F.: Introduction to Stochastic Programming. Springer Series in Operations Research and Financial Engineering, vol. 4, 2 edn. Springer, New York (2011). https://doi.org/10.1007/978-1-4614-0237-4
13. Caramanis, M., Ntakou, E., Hogan, W.W., Chakrabortty, A., Schoene, J.: Co-optimization of power and reserves in dynamic t amp;d power markets with nondispatchable renewable generation and distributed energy resources. Proc. IEEE **104**(4), 807–836 (2016). https://doi.org/10.1109/JPROC.2016.2520758
14. Carøe, C.C., Schultz, R.: Dual decomposition in stochastic integer programming. Oper. Res. Lett. **24**(1–2), 37–45 (1999). https://doi.org/10.1016/S0167-6377(98)00050-9
15. Carpentier, P., Gohen, G., Culioli, J., Renaud, A.: Stochastic optimization of unit commitment: a new decomposition framework. IEEE Trans. Power Syst. **11**(2), 1067–1073 (1996). https://doi.org/10.1109/59.496196
16. Cerisola, S., Baíllo, Á., Fernández-López, J.M., Ramos, A., Gollmer, R.: Stochastic power generation unit commitment in electricity markets: a novel formulation and a comparison of solution methods. Oper. Res. **57**(1), 32–46 (2009). https://doi.org/10.1287/opre.1080.0593
17. Chaturapruek, S., Duchi, J.C., R, C.: Asynchronous stochastic convex optimization: the noise is in the noise and SGD don't care. In: Proceedings of the 2015 neural information processing systems (NIPS 2015), Montréal, Canada (2015)

18. Cheung, K., Gade, D., Silva-Monroy, C., Ryan, S.M., Watson, J.P., Wets, R.J.B., Woodruff, D.L.: Toward scalable stochastic unit commitment. Part 2: solver configuration and performance assessment. Energy Syst. **6**(3), 417–438 (2015). https://doi.org/10.1007/s12667-015-0148-6
19. Damcı-Kurt, P., Küçükyavuz, S., Rajan, D., Atamtürk, A.: A polyhedral study of production ramping. Math. Program. **158**(1), 175–205 (2016). https://doi.org/10.1007/s10107-015-0919-9
20. Dimoulkas, I., Amelin, M.: Probabilistic day-ahead CHP operation scheduling. In: 2015 IEEE power energy society general meeting, pp. 1–5 (2015). https://doi.org/10.1109/PESGM.2015.7285962
21. Ermoliev, Y.: Stochastic quasigradient methods and their application to system optimization. Stochastics **9**(1–2), 1–36 (1983). https://doi.org/10.1080/17442508308833246
22. Fair Isaac Corporation (FICO): Xpress-Optimizer Reference Manual (2016)
23. Fercoq, O., Richtárik, P.: Smooth minimization of nonsmooth functions with parallel coordinate descent methods. Optimization Online (2013)
24. Fischer, F., Helmberg, C.: A parallel bundle framework for asynchronous subspace optimization of nonsmooth convex functions. SIAM J. Optim. **24**(2), 795–822 (2014). https://doi.org/10.1137/120865987
25. Forum, M.P.I.: MPI: A Message-Passing Interface Standard, Version 3.1. High Performance Computing Center Stuttgart (2015)
26. Gassmann, H., Schweitzer, E.: A comprehensive input format for stochastic linear programs. Ann. Oper. Res. **104**(1), 89–125 (2001). https://doi.org/10.1023/A:1013138919445
27. Gassmann, H.I.: The SMPS Format for Stochastic Linear Programs, chap. 1, pp. 9–19. SIAM (2005). https://doi.org/10.1137/1.9780898718799.ch2
28. Kim, K., Zavala, V.M.: Algorithmic innovations and software for the dual decomposition method applied to stochastic mixed-integer programs. Optimization Online (2015)
29. Liu, J., Wright, S.J., Ré, C., Bittorf, V., Sridhar, S.: An asynchronous parallel stochastic coordinate descent algorithm. J. Mach. Learn. Res. **16**(1), 285–322 (2015)
30. Lubin, M., Hall, J.A.J., Petra, C.G., Anitescu, M.: Parallel distributed-memory simplex for large-scale stochastic lp problems. Comput. Optim. Appl. **55**(3), 571–596 (2013). https://doi.org/10.1007/s10589-013-9542-y
31. Lubin, M., Martin, K., Petra, C.G., Sandıkçı, B.: On parallelizing dual decomposition in stochastic integer programming. Oper. Res. Lett. **41**(3), 252–258 (2013). https://doi.org/10.1016/j.orl.2013.02.003
32. Lulli, G., Sen, S.: A branch-and-price algorithm for multistage stochastic integer programming with application to stochastic batch-sizing problems. Manag. Sci. **50**(6), 786–796 (2004). https://doi.org/10.1287/mnsc.1030.0164
33. Moritsch, H.W., Pflug, G.C., Siomak, M.: Asynchronous nested optimization algorithms and their parallel implementation. Wuhan Univ. J. Natl. Sci. **6**(1), 560–567 (2001). https://doi.org/10.1007/BF03160302
34. Munguía, L., Oxberry, G., Rajan, D.: PIPS-SBB: a parallel distributed-memory branch-and-bound algorithm for stochastic mixed-integer programs. Optimization Online (2015)
35. Nedić, A.: Subgradient methods for convex optimization. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, USA (2002)
36. Nedić, A., Bertsekas, D.P.: Incremental subgradient methods for nondifferentiable optimization. SIAM J. Optim. **12**(1), 109–138 (2001). https://doi.org/10.1137/S1052623499362111
37. Nedić, A., Bertsekas, D.P., Borkar, V.S.: Distributed asynchronous incremental subgradient methods. In: Butnariu, D., Censor, Y., Reich, S. (eds.) Inherently Parallel Algorithms in Feasibility and Optimization and their Applications, Studies in Computational Mathematics, pp. 381–407. Elsevier, Amsterdam (2001)
38. Nesterov, Y.: Smooth minimization of non-smooth functions. Math. Program. **103**(1), 127–152 (2005). https://doi.org/10.1007/s10107-004-0552-5
39. Nesterov, Y.: Efficiency of coordinate descent methods on huge-scale optimization problems. SIAM J. Optim. **22**(2), 341–362 (2012). https://doi.org/10.1137/100802001
40. Oliveira, W., Sagastizbal, C., Scheimberg, S.: Inexact bundle methods for two-stage stochastic programming. SIAM J. Optim. **21**(2), 517–544 (2011). https://doi.org/10.1137/100808289
41. Papavasiliou, A., Oren, S.S.: Multiarea stochastic unit commitment for high wind penetration in a transmission constrained network. Oper. Res. **61**(3), 578–592 (2013). https://doi.org/10.1287/opre.2013.1174

42. Papavasiliou, A., Oren, S.S., Rountree, B.: Applying high performance computing to transmission-constrained stochastic unit commitment for renewable energy integration. IEEE Trans. Power Syst. **30**(3), 1109–1120 (2015)
43. Polyak, B.T.: Minimization of unsmooth functionals. USSR Comput. Math. Math. Phys. **9**, 14–29 (1969)
44. Rajan, D., Ryan, K., Ahmed, S., Dey, S.: Optimization driven scenario grouping for stochastic unit commitment. In: FERC Software Conference **2017** (2017)
45. Rockafellar, R.T.: Convex Analysis. Princeton University Press, Princeton (1970)
46. Ryan, K., Rajan, D., Ahmed, S.: Scenario decomposition for 0-1 stochastic programs: improvements and asynchronous implementation. In: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 722–729 (2016). https://doi.org/10.1109/IPDPSW.2016.119
47. Schulze, T., Grothey, A., McKinnon, K.: A stabilised scenario decomposition algorithm applied to stochastic unit commitment problems. Optimization Online (2015)
48. Shiina, T., Birge, J.R.: Stochastic unit commitment problem. Int. Trans. Oper. Res. **11**(1), 19–32 (2004). https://doi.org/10.1111/j.1475-3995.2004.00437.x
49. Stott, B., Jardim, J., Alsac, O.: DC power flow revisited. IEEE Trans. Power Syst. **24**(3), 1290–1300 (2009). https://doi.org/10.1109/TPWRS.2009.2021235
50. Tahanan, M., van Ackooij, W., Frangioni, A., Lacalandra, F.: Large-scale unit commitment under uncertainty. 4OR **13**(2), 115–171 (2015). https://doi.org/10.1007/s10288-014-0279-y
51. Takriti, S., Birge, J.R., Long, E.: A stochastic model for the unit commitment problem. IEEE Trans. Power Syst. **11**(3), 1497–1508 (1996). https://doi.org/10.1109/59.535691
52. Tseng, P.: Convergence of a block coordinate descent method for nondifferentiable minimization. J. Optim. Theory Appl. **109**(3), 475–494 (2001). https://doi.org/10.1023/A:1017501703105
53. van Ackooij, W., Malick, J.: Decomposition algorithm for large-scale two-stage unit-commitment. Ann. Oper. Res. **238**(1), 587–613 (2016). https://doi.org/10.1007/s10479-015-2029-8
54. Watson, J.P., Woodruff, D.L., Hart, W.E.: PySP: modeling and solving stochastic programs in python. Math. Program. Comput. **4**(2), 109–149 (2012). https://doi.org/10.1007/s12532-012-0036-1
55. Wright, S.J.: Coordinate descent algorithms. Math. Program. **151**(1), 3–34 (2015). https://doi.org/10.1007/s10107-015-0892-3

Affiliations

Ignacio Aravena[1] · Anthony Papavasiliou[2]

✉ Ignacio Aravena
aravenasolis1@llnl.gov

Anthony Papavasiliou
anthony.papavasiliou@uclouvain.be

1 Lawrence Livermore National Laboratory, Livermore, California, USA

2 CORE, Université catholique de Louvain, Louvain-la-Neuve, Belgium