2016/38

An Asynchronous Distributed Algorithm for Solving Stochastic Unit Commitment

Ignacio Aravena and Anthony Papavasiliou

CORE
Voie du Roman Pays 34, L1.03.01
Tel (32 10) 47 43 04
Fax (32 10) 47 43 01
Email: immaq-library@uclouvain.be
http://www.uclouvain.be/en-44508.html

An Asynchronous Distributed Algorithm for Solving Stochastic Unit Commitment

Ignacio Aravena and Anthony Papavasiliou
Université catholique de Louvain, CORE
ignacio.aravena@uclouvain.be, anthony.papavasiliou@uclouvain.be

Keywords: *Asynchronous algorithm, coordinate descent method, high performance computing, stochastic programming, unit commitment*
AMS Classification: *68W15, 46N10*

Abstract

We present an asynchronous algorithm for solving the stochastic unit commitment (SUC) problem using scenario decomposition. The algorithm is motivated by the scale of problem and significant differences in run times observed among scenario subproblems, which can result in inefficient use of distributed computing resources by synchronous parallel algorithms. Dual iterations are performed asynchronously using a block-coordinate subgradient descent method which allows performing block-coordinate updates using delayed information. We provide convergence guarantees for the asynchronous block-coordinate subgradient method based on previous results for incremental subgradient methods and stochastic subgradient methods. The algorithm recovers candidate primal solutions from the solutions of scenario subproblems using recombination heuristics.

The asynchronous algorithm is implemented in a high performance computing cluster and we conduct numerical experiments for two-stage SUC instances of the Western Electricity Coordinating Council (WECC) system and of the Central Western European (CWE) system. The WECC system that we study consist of 130 thermal generators, 182 nodes and 319 lines with hourly resolution and up to 1000 scenarios, while the CWE system consist of 656 thermal generators, 679 nodes and 1073 lines, with quarterly resolution and up to 120 scenarios. When using 10 nodes of the cluster per instance, the algorithm provides solutions that are within 2% of optimality to all problems within 47 minutes for WECC and 3 hours, 54 minutes for CWE. Moreover, we find that an equivalent synchronous parallel subgradient algorithm would leave processors idle up to 84% of the time, an observation which underscores the need for designing asynchronous optimization schemes in order to fully exploit distributed computing on real world applications.

1 Introduction

The recent integration of significant shares of renewable energy resources into electric power systems has motivated the industry and academic communities to investigate efficient alternatives for scheduling power systems in the presence of uncertainty. Among the different alternatives that have been proposed (see [TvAFL15] and references therein for a recent survey), stochastic unit commitment (SUC) is a common approach which is able to incorporate uncertainty and estimated probability distributions explicitly into the problem formulation. The use of stochastic programming for short-term scheduling of electric power systems was initially proposed in the seminal work of Takriti *et al.* [TBL96] and Carpentier *et al.* [CGCR96] as a methodology to cope with demand uncertainty. The scope for application of SUC has, since then, been extended to renewable energy fluctuations and system contingencies, among other sources of uncertainty in power systems operations.

SUC studies commonly use decomposition methods to handle the scale of the problem, which increases linearly with the number of scenarios. Takriti *et al.* [TBL96] use Lagrange relaxation of non-anticipativity constraints (i.e. scenario decomposition), a progressive hedging heuristic to obtain non-anticipative solutions. Carpentier *et al.* [CGCR96] use an augmented Lagrange relaxation over generators and a proximal point method to maximize the dual function while recovering primal solutions by assigning assigning a quadratic penalty to demand shedding. Shiina and Birge [SB04] use a column generation approach in order to provide feasible production and commitment schedules to a restricted master SUC problem. Cerisola *et al.* [CBFL$^+$09] compare scenario decomposition with variants of Benders decomposition and stress the importance of finding good initial solutions. Additional decomposition schemes for SUC are described in [TvAFL15].

For the most part, the solution of SUC instances in the literature has been limited to small test cases, even when using decomposition methods. Several recent studies have exploited distributed computing along with decomposition methods in order to advance towards realistic instances. Papavasiliou *et al.* [POR15] implement scenario decomposition in a high performance computing (HPC) cluster to solve instances of the WECC system with up to 1000 scenarios in at most 24 hours within an optimality gap of 1% – 2.5%. Cheung *et al.* [CGSM$^+$15] 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 minutes within an optimality gap of 1.5% – 2.5%. Kim and Zavala [KZ15] 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 hours within an optimality gap of 0.01%.

Other recently proposed methods, which do not exploit distributed computing directly, have also being used to solve large SUC instances. Among them, Schulze *et al.* [SGM15] use a stabilized Dantzig-Wolfe decomposition to solve multi-stage SUC instances of the British system with up to 50 scenarios in 2 hours within an optimality gap of 0.1%. van Ackooij and Malick [vAM16] 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.

Scenario decomposition has inspired several algorithms for general stochastic mixed integer programs. Carøe and Schultz [CS99] propose a scenario decomposition method where non-anticipativity constraints are gradually enforced through a branch-and-bound algorithm. Lubin *et al.* [LMPS13] 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 structure-exploiting interior point solver. Ahmed [Ahm13, Ahm15] 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.* [RRA16] develop several improvements over Ahmed's original algorithm, including a distributed asynchronous implementation.

Other distributed methods for stochastic programming do not stem directly from scenario decomposition. Lubin *et al.* [LHPA13] propose a distributed memory simplex algorithm for stochastic linear programs. Munguía *et al.* [MOR15] 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.* [MPS01] propose a nested asynchronous decomposition algorithm for multistage stochastic

linear programs. Chaturapruek *et al.* [CDR15] 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.* [NB01, NBB01, Ned02] 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 [Wri15] provides a recent survey on coordinate descent methods. Nesterov [Nes12] provides worst-case complexity results of randomized coordinate descent methods for smooth optimization. Fercoq and Richtárik [FR13] propose a parallel synchronous coordinate descent method for minimizing non-differentiable simple composite functions. The authors use Nesterov's smoothing technique [Nes05] to obtain a smooth approximation of the non-decomposable part of the objective and perform a line search separately on each coordinate ot each iteration, as proposed by Tseng [Tse01]. Liu *et al.* [LWR$^+$15], on the other hand, propose an asynchronous parallel stochastic coordinate descent method for minimizing differentiable smooth convex functions over simple separable constraint sets.

Stochastic unit commitment, despite its attractiveness, has failed to become an industry standard due to several reasons, most importantly the difficulty of the mathematical programs involved and the design of a market structure consistent with the treatment of uncertainty. In the present paper we aim at advancing the state-of-the-art with respect of the first problematic by developing an asynchronous scenario decomposition scheme that fully exploits the power of distributed computing. The main innovation of the developed scheme is that we minimize the dual function using a specialized asynchronous block-coordinated subgradient algorithm. Our algorithm does not require differentiability or strong convexity assumptions commonly used in the literature [Nes12, Wri15, LWR$^+$15, CDR15], and it differs from the algorithm of Fercoq and Richtárik [FR13] in that we perform iterations asynchronously and without the need for line search, which would be prohibitively expensive in our context. We provide convergence guarantees for the proposed algorithm based on previous results for stochastic subgradient methods [Erm83] and incremental subgradient methods [NB01, NBB01, Ned02]. Primal recovery is also carried out asynchronously and in parallel to the dual iterations, either by recovering solutions from scenario subproblems, as proposed in [Ahm13, Ahm15], or by using recombination heuristics, similar to those proposed in [CS99]. The proposed asynchronous algorithm allows us to solve limited-size SUC instances faster than the state-of-the-art [CGSM$^+$15] and to solve SUC instances for systems larger than the state-of-the-art [vAM16] within operationally acceptable time frames.

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. Section 3 presents the asynchronous distributed block-coordinate subgradient method and provides convergence guarantees for the dual iterations. Section 4 describes the HPC implementation of the dual algorithm and the recovery of primal feasible solutions. In this section we also cover aspects of communications and load balancing. Section 5 presents the numerical results for the WECC and CWE systems. Finally, section 6 presents the conclusions of the study and points to directions of future research.

2 Scenario decomposition

A general linear stochastic mixed integer program can be written as problem (1) – (4) [LS04].

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

$$\text{s.t.} \quad \boldsymbol{v}_i - \boldsymbol{u} = \boldsymbol{0}, \quad i = 1, \ldots, N \qquad (\boldsymbol{x}_i) \tag{2}$$

$$(\boldsymbol{v}_i, \boldsymbol{w}_i) \in \mathcal{D}_i, \quad i = 1, \ldots, N \tag{3}$$
$$\boldsymbol{u} \in \mathcal{U} \tag{4}$$

In this formulation, i indexes scenarios, $\boldsymbol{u}$ corresponds to non-anticipative variables, $\boldsymbol{v}_i$ are local copies of non-anticipative variables at each scenario, $\boldsymbol{w}_i$ are recourse variables of each scenario, (2) models non-anticipativity constraints, (3) models constraints separable by scenario and (4) corresponds to constraints for state variables. We assume that $\mathcal{U}$ is a bounded convex set and that $\mathcal{D}_i, i = 1, \ldots, N$ are bounded (not necessarily convex) sets. 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. $\boldsymbol{x} = [\boldsymbol{x}_1^T \ \ldots \ \boldsymbol{x}_N^T]^T$ and $\boldsymbol{x}_i \in \mathbb{R}^{n_i}$.

In the context of SUC, $\boldsymbol{v}_i$ tipically correspondsto commitment variables of thermal generators (binary decisions), but it can also include production variables of inflexible thermal generators (continuous decisions), while $\boldsymbol{w}_i$ 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, reactance, flow limits), and $\mathcal{U}$ corresponds to a convex relaxation of the production constraints for variables included in $\boldsymbol{u}$. See [PO13] for a detailed description of the SUC model.

Following [TBL96] we relax problem (1)–(4) by associating multipliers $\boldsymbol{x}_i$ to non-anticipativity constraints, obtaining the dual problem (5), where f_0 and f_i are defined according to (6) and (7), respectively.

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

$$f_0(\boldsymbol{x}) := \sup_{u \in \mathcal{U}} \left(-\sum_{i=1}^{N} \boldsymbol{x}_i^T \right) \boldsymbol{u} \tag{6}$$
$$f_i(\boldsymbol{x}_i) := \sup_{(v,w) \in \mathcal{D}_i} \left((\boldsymbol{c}_i^T + \boldsymbol{x}_i^T)\boldsymbol{v} + \boldsymbol{d}_i^T \boldsymbol{w} \right) \quad i = 1, \ldots, N \tag{7}$$

Scenario decomposition schemes for solving (1)–(4) are based on solving the dual problem (5) and generating primal solutions based on the solution to subproblems. 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 [POR15]. A third important property of problem (5) remains nevertheless ignored, 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 small linear program, hence the evaluation of f_0 requires only a small fraction of the times that it takes to evaluate f_i for any given i. This differ from the usual scope of application of coordinate descent methods, where component functions are easy to evaluate [Nes12, FR13, Wri15, LWR$^+$15]. Even more, 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 $\boldsymbol{x}$'s. Altogether, these differences can render synchronous parallel algorithms ineffective because the time between iterations is limited by the component 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 been inspired by previous work on incremental subgradient algorithms by Nedić *et al.* [NB01, NBB01, Ned02].

For reasons that we explain subsequently, we replace the non-decomposable component function f_0 by a smooth approximation f_0^μ, defined in equation (8), as done in [FR13]. Given $\mu > 0$ and certain $\boldsymbol{u}_0 \in \mathcal{U}$, f_0^μ is a convex differentiable function and its gradient has Lipschitz constant L_0^μ [Nes05, Thm. 1]. We focus then on solving problem (9), where X is a convex set, onto which it is easy to project.

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

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

The solution of problem (9) will be an approximated solution to problem (5) [Nes05]. Note that problem (9), as well as problem (5), are non-differentiable convex optimization problems. Block-coordinate descent methods are able to exploit the separable structure in problems (5) and (9), however, they are not guaranteed to converge to an optimal solution for non-differentiable problems since their convergence results in the literature either depend on smoothness assumptions [Nes12, Wri15, LWR$^+$15, CDR15] or use line search over $\boldsymbol{x}_i$ at each iteration [Tse01, FR13], which is prohibitively expensive in our context.

In what follows, we propose an asynchronous randomized distributed block-coordinate subgradient method and prove that it converges with probability 1 to an optimal solution for problem (9). For ease of presentation, we first introduce a serial variant of the proposed method and then we generalize it to the asynchronous distributed setting.

3.1 Serial method

Let's consider a block version of the randomized coordinate descent method [Nes12] 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

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

where λ_k is the step size, $g(j, \boldsymbol{x}) \in \partial f_j(\boldsymbol{x})$ and 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.

Although f is a non-differentiable function, thanks to the smoothness of f_0^μ, the update rule (10) is equivalent to the update rule of the stochastic subgradient method, as we show 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), $\mathbb{E}[I_J^T (I_J \nabla f_0^\mu\big(\boldsymbol{x}^k\big) + g\big(J, \boldsymbol{x}_J^k\big))| \, \boldsymbol{x}^k]$, coincides with the direction of a subgradient of f at $\boldsymbol{x}^k$.

Proof. By inspection we have that,

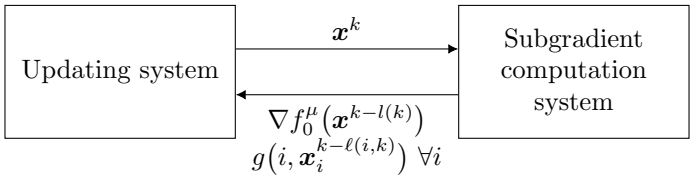

Figure 1: Computation model for the asynchronous distributed method. At each iteration k the *Updating system* communicates the current iterate $\boldsymbol{x}^k$ to the *Subgradient computation system*, which in turn, communicates back the last computed subgradient for each component function. The parallelism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

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

The stochastic subgradient method and its convergence have been well studied in the literature, see for instance [Erm83, 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 update rule (10) requires computing the gradient of f_0^μ and a subgradient of $f_{j(k)}$ at each iteration k, which makes it more expensive than the subgradient method when we consider an entire pass over data, 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 [NBB01], we use the computation model presented in Fig. 1. Note that the 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 the last computation of its subgradient, at iteration k. Considering delays, we propose update rule (11) for the randomized block-coordinate subgradient method, where $j(k)$ is selected uniformly at random from $\{1,\dots,N\}$.

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

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, to 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. Our analysis is based on the Supermartingale Convergence Theorem and the subsequent assumptions.

Theorem 1 (Supermartingale Convergence Theorem [BT96, Prop. 4.2])**.** Let X_t, Y_t and Z_t, $t = 0, 1, 2, \dots$, be three sequences of random variables and let $\mathcal{F}_t$, $t = 0, 1, 2, \dots$, 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 $\mathbb{E}[X_{t+1}|\mathcal{F}_t] \leq X_t - Y_t + Z_t$.

(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.

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

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

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 $j(k)$. Further, 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 SUC 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 subsection 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 approximate subgradient of f at $\boldsymbol{x}^k$.

Proof. Recall that a vector $\boldsymbol{g}$ is said to be an approximated or ϵ-subgradient of a convex function f at $\boldsymbol{x}$ if

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

On the other hand, for the expected direction of update rule (11) we have

$$\begin{aligned} N\cdot(\boldsymbol{x}^k-\boldsymbol{y})^T\mathbb{E}\left[I_J^T\Big(I_J\nabla f_0^{\mu}\big(\boldsymbol{x}^{k-l(k)}\big) + g\big(J,\boldsymbol{x}_J^{k-\ell(J,k)}\big)\Big)\,\Big|\,\mathcal{F}_k\right] = \\ (\boldsymbol{x}^k-\boldsymbol{y})^T\nabla f_0^{\mu}\big(\boldsymbol{x}^{k-l(k)}\big) + \sum_{j=1}^{N}(\boldsymbol{x}_j^k-\boldsymbol{y}_j)^T g\big(j,\boldsymbol{x}_j^{k-\ell(j,k)}\big), \quad \forall \boldsymbol{y}\in X. \end{aligned} \tag{12}$$

The first term in the right-hand side of (12) can be expanded as follows

$$\begin{aligned} \big(\boldsymbol{x}^k-\boldsymbol{y}\big)^T\nabla f_0^{\mu}\big(\boldsymbol{x}^{k-l(k)}\big) &= \big(\boldsymbol{x}^{k-l(k)}-\boldsymbol{y}\big)^T\nabla f_0^{\mu}\big(\boldsymbol{x}^{k-l(k)}\big) + \big(\boldsymbol{x}^k-\boldsymbol{x}^{k-l(k)}\big)^T\nabla f_0^{\mu}\big(\boldsymbol{x}^k\big) - \\ &\qquad \big(\boldsymbol{x}^k-\boldsymbol{x}^{k-l(k)}\big)^T\Big(\nabla f_0^{\mu}\big(\boldsymbol{x}^k\big) - \nabla f_0^{\mu}\big(\boldsymbol{x}^{k-l(k)}\big)\Big) \\ &\geq f_0^{\mu}\big(\boldsymbol{x}^k\big) - f_0^{\mu}(\boldsymbol{y}) - \big(\boldsymbol{x}^k-\boldsymbol{x}^{k-l(k)}\big)^T\Big(\nabla f_0^{\mu}\big(\boldsymbol{x}^k\big) - \nabla f_0^{\mu}\big(\boldsymbol{x}^{k-l(k)}\big)\Big) \\ &\geq f_0^{\mu}\big(\boldsymbol{x}^k\big) - f_0^{\mu}(\boldsymbol{y}) - \big\|\boldsymbol{x}^k-\boldsymbol{x}^{k-l(k)}\big\|_2\Big\|\nabla f_0^{\mu}\big(\boldsymbol{x}^k\big) - \nabla f_0^{\mu}\big(\boldsymbol{x}^{k-l(k)}\big)\Big\|_2 \\ &\geq f_0^{\mu}\big(\boldsymbol{x}^k\big) - f_0^{\mu}(\boldsymbol{y}) - L_0^{\mu}\big\|\boldsymbol{x}^k-\boldsymbol{x}^{k-l(k)}\big\|_2^2, \end{aligned}$$

where in the second line 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 (12) can be expanded as follows

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

where in the second line we use Assumption 1. Furthermore, Assumptions 1 and 2 allow us to bound the difference between current and delayed iterates using the stepsize as

$$\big\|\boldsymbol{x}^k - \boldsymbol{x}^{k-\ell(j,k)}\big\|_2 \leq C \sum_{m=k-\ell(j,k)}^{k-1} \lambda_m \leq C \sum_{m=k-L}^{k-1} \lambda_m, \quad \big\|\boldsymbol{x}^k - \boldsymbol{x}^{k-l(k)}\big\|_2^2 \leq C^2 \sum_{m=k-L}^{k-1} \lambda_m^2,$$

which along with the previous expressions lead us to the relation (13), concluding the proof.

$$\begin{aligned}N \cdot (\boldsymbol{x}^k - \boldsymbol{y})^T \mathbb{E}\left[I_J^T\Big(I_J \nabla f_0^\mu\big(\boldsymbol{x}^{k-l(k)}\big) + g\big(J, \boldsymbol{x}_J^{k-\ell(J,k)}\big)\Big)\Big|\, \mathcal{F}_k\right] \geq \\ f(\boldsymbol{x}^k) - f(\boldsymbol{y}) - \left(C^2 L_0^\mu \sum_{m=k-L}^{k-1} \lambda_m^2 + 2CDN \sum_{m=k-L}^{k-1} \lambda_m\right)\end{aligned} \tag{13}$$

□

Lemma 1 shows that the update direction of rule (11) should, in 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

$$\begin{aligned}\mathbb{E}\big[\|\boldsymbol{x}^{k+1} - \boldsymbol{y}\|_2^2 \big| \mathcal{F}_k\big] \leq &\|\boldsymbol{x}^k - \boldsymbol{y}\|_2^2 - 2\frac{\lambda_k}{N}\big(f(\boldsymbol{x}^k) - f(\boldsymbol{y})\big) + \\ &\lambda_k^2 C^2 + 2\frac{C^2 L_0^\mu}{N}\lambda_k \sum_{m=k-L}^{k-1} \lambda_m^2 + 4CD\lambda_k \sum_{m=k-L}^{k-1} \lambda_m.\end{aligned} \tag{14}$$

Proof. 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

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

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

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

Finally, by substituting $\mathbb{E}\big[\boldsymbol{h}_J \big| \mathcal{F}_k\big]$ in the last relation using (13) (Lemma 1) we obtain the desired inequality. □

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 $\lambda_{-l}=\lambda_0\ \forall l\in\mathbb{N}$.

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

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

where for the last inequality we use the fact that $\{\gamma_k\}$ is non-increasing and, 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. □

Finally, in the following theorem, we apply Theorem 1 to the result of Proposition 2 in order to prove the convergence of the asynchronous method to an optimal solution.

Theorem 2. 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.

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

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

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

$$\sum_{k=0}^{\infty}\lambda_k\big(f(\boldsymbol{x}^k)-f(\boldsymbol{x}^*)\big)<\infty,\tag{15}$$

and with probability 1 the sequence $\big\{\|\boldsymbol{x}^k-\boldsymbol{x}^*\|_2\big\}$ converges to a random variable. The rest of the proof follows exactly the proof of [Ned02, Prop. 3.4] (see also [Erm83, Thm. 2]) and it is repeated here only for self-containedness.

For each $\boldsymbol{x}^*\in X^*$, let Ω_{x^*} denote the set of all sample paths for which equation (15) holds and $\big\{\|\boldsymbol{x}^k-\boldsymbol{x}^*\|_2\big\}$ 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 $\big\{\|\boldsymbol{x}^k-\boldsymbol{v}_j\|_2\big\}$, $j=0,\ldots,p$, converge. Thus, with probability 1, $\{\boldsymbol{x}^k\}$ is bounded, and therefore it has limit points. Furthermore, for each sample path in Ω, by equation (15) 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 $\big\{\|\boldsymbol{x}^k-\boldsymbol{v}_j\|_2\big\}$, $j=0,\ldots,p$, 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. □

The result presented in Theorem 2 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 [Tse01, FR13].

An important remark is that the asynchronous incremental method proposed in [NBB01] 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 leaned towards block-coordinate descent instead of incremental methods because of two main reasons:

- The choice between incremental and block-coordinate subgradient methods can have significant implications for 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 $X = \mathcal{R}^n$ (unconstrained optimization).

  The incremental subgradient method at iteration k selects at random a component function, $j(k) \in \{0, 1, \ldots, N\}$, and perform 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, unavoidable for the problem structure analyzed in [NBB01], 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 very fast and continuously as iterations advance, allowing to maintain small delays throughout the solution process.

- For SUC 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 $i = 1, \ldots, N$, making the incremental method 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 allow us to use a stepsize similar to the dynamic stepsize proposed by Polyak for the subgradient method [Pol69],

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

The original Polyak stepsize requires knowledge of the objective value at the current iterate $f(\boldsymbol{x}^k)$, the optimal value f^* and the norm of the subgradient at the current iterate $\|g(\boldsymbol{x}^k)\|_2$, none of which are available for the asynchronous method. Instead, we use 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

$$f(\boldsymbol{x}^k) \approx \hat{f}_k := f_0\Big(\big[\big(\boldsymbol{x}_1^{k-\ell(1,k)}\big)^T \ldots \big(\boldsymbol{x}_N^{k-\ell(N,k)}\big)^T\big]^T\Big) + \sum_{j=1}^{N} f_j\big(\boldsymbol{x}_j^{k-\ell(j,k)}\big), \tag{16}$$

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

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

where 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 $\check{f}_k$ can be obtained from a feasible solution to the primal problem, computed as described in section 4.2. We assume that this is a strict underestimate, i.e. $\theta \leq f^* - \check{f}_k$ 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,

$$\lambda_k = \frac{p}{(1+rk)^q} \cdot \frac{\min\left\{\xi,\, \hat{f}_k - \check{f}_k\right\}}{\bar{g}_k^2}, \tag{17}$$

where p, r, ξ are 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. The proposed stepsize λ_k, as defined in equation (17), agrees with Assumption 3, as can be seen from the following inequality,

$$p\frac{\theta}{C} \cdot \gamma_k \leq \lambda_k \leq p\frac{\xi}{\sigma} \cdot \gamma_k,$$

therefore, by Theorem 2, the asynchronous algorithm using the proposed stepsize will converge with probability 1 to an optimal solution.

Setting the parameters for the stepsize (17) is a much simpler task than selecting the parameters for a diminishing stepsize of type $1/k^q$. For the proposed stepsize, we only need to decide the proportion of the Polyak stepsize that we would like to have at two different iteration counts and the rate at which we want to decrease this proportion, and then adjust p, r and q accordingly. By contrast, for a diminishing stepsize of type $1/k^q$ it is necessary to determine a 'good' initial stepsize, which can require several trial-and-error runs of the algorithm.

4 High performance computing implementation

The analysis in the previous section provides convergence guarantees for the asynchronous dual optimization algorithm, based on the conceptual computation model of Fig. 1. An actual implementation of the algorithm, on the other hand, requires a description of the different subprocesses running in parallel, the information to be shared between them and a primal recovery scheme in order to obtain feasible solutions for the primal problem (1)–(4). Under these considerations, we implement the asynchronous algorithm for SUC as presented in Fig. 2. The main differences between the conceptual implementation (Fig. 1) and the actual implementation (Fig. 2) are the following:

- There is no clear separation between the *Updating system* and the *Subgradient computation system* in the actual implementation. The *Updating system* is contained within the *Master* process, while the *Subgradient computation system* is split between *Master*, the f_0^μ *gradient evaluator* and the *Subgradient evaluator* l, $l = 1, \ldots, M$.
- The *Upper bound update* process, not present in the conceptual implementation, is a process dedicated exclusively to computing dual function values, which corresponds to upper bounds on the optimal primal objective. This process is commonly absent in the literature of asynchronous algorithms, as it is not always possible or necessary to estimate the value of the dual function.
- The *Primal recovery* and *Recourse evaluator* s, $s = 1, \ldots, R$ processes, also not present in the conceptual implementation, are dedicated to obtaining feasible solutions to the primal problem (1)–(4), which provide lower bounds on the optimal primal objective.

The ability to continuously compute both lower and upper bounds allows establishing a natural termination criterion, $UB - LB \leq \epsilon$, as well as terminating the algorithm early at a certain wall time while still obtaining a feasible solution and a bound.

In the following we present in detail each process of the algorithm and how it communicates with the other processes. Processes are presented in pseudocode, making repeated use of the following procedures/functions:

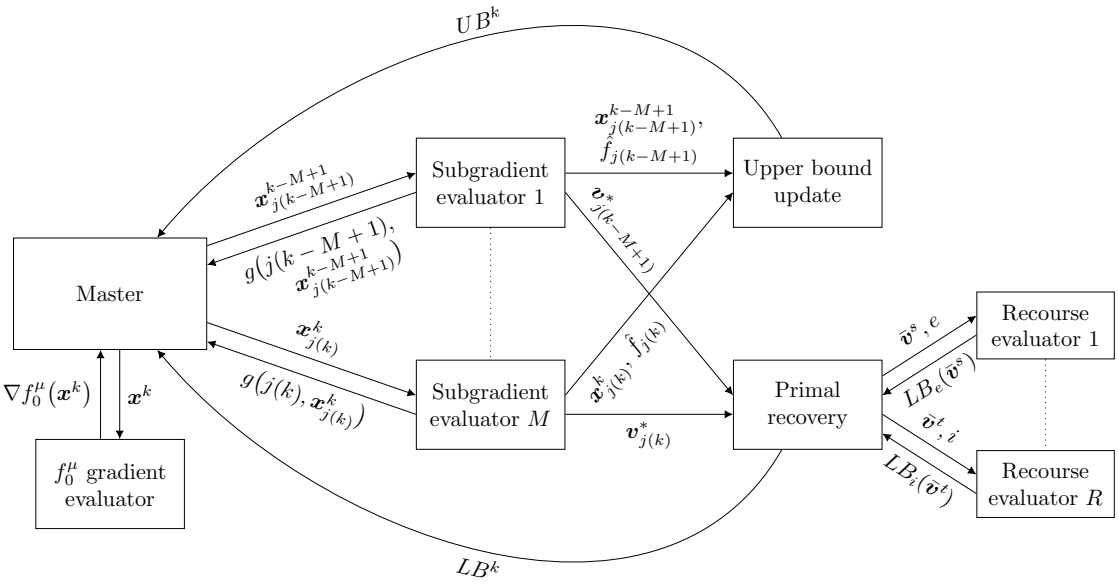

Figure 2: Asynchronous distributed algorithm for stochastic unit commitment. Each box corresponds to a different process running in parallel. Each arrow indicates information transmitted between processes.

launch(*process*). Starts *process* and continues to the next line (does not wait for termination).

send(*message*;*process*). Sends *message* to *process*. *message* is an object whose attributes can be accessed as *message.attribute*. Messages can only contain small amounts of information, larger portions of information are communicated by writing and reading operations on a shared space (hard drives or memory).

get_next_message(*message*), get_all_messages(*list*). Each process stores incoming messages in a queue, preserving the order of arrival, without pausing execution. get_next_message(*message*) assigns the first-arriving message in the queue to *message* and removes it from the queue. get_all_messages(*list*) copies all messages from the queue to *list*. If there are no messages in the queue when executing these procedures, execution is paused until the first message arrives and the procedure is then executed.

read(*local*; *file*). Reads the content of *file* and assigns it to *local*, where *local* is a local variable of the process where the routine is executed.

write(*local*; *file*), write*(*local variable*; *file*). Writes the content of *local* to *file*, overwriting *file* if it already exists. The form write* only overwrites if the content of *local* is newer than information already in *file*.

append(*local*; *file*). Appends the content of *local* to *file*.

extractfirst(*list*). Returns the first element from *list* and removes it from *list*.

4.1 Dual algorithm implementation

The dual algorithm is at the core of the implementation presented in Fig. 2 and it comprises the *Master* process, the f_0^μ *gradient evaluator*, the *Upper bound updater* and M *Subgradient evaluators*.

Algorithm 1: Master.

Data: Starting point $\boldsymbol{x}^0$, estimated subgradients of component functions at $\boldsymbol{x}^0$, $\tilde{\boldsymbol{g}}_i^0$ $\forall i = 0, \ldots, N$, number of dual processors $M + 2$ ($M \leq N$), stepsize parameters p, q, r, ξ, σ, termination gap ϵ and termination time T.

Result: An upper bound on the primal objective UB and a primal solution $(\boldsymbol{u}^\star, \boldsymbol{w}^\star)$ with objective value LB, such that $UB - LB \leq \epsilon$ or run time greater or equal to T.

/* initialize local variables and shared registers */

1 Let $k := 0$, $\boldsymbol{x} := \boldsymbol{x}^0$, $\boldsymbol{g}_i := \tilde{\boldsymbol{g}}_i^0$ $\forall i = 0, \ldots, N$, $LB := -\infty$, $UB := +\infty$, message := `NULL`

2 **for** $i = 1, \ldots, N$ **do** write($\boldsymbol{x}_i$; `'sh:x_'` + i)

3 **for** $i = 1, \ldots, N$ **do** write($[\boldsymbol{x}_i\ UB]$; `'sh:x_UB_'` + i)

/* launch subprocesses */

4 launch(*Upper bound updater*)

5 launch(*Primal recovery*)

6 **for** $i = 1, \ldots, M$ **do**

7 launch(*Subgradient evaluator* i)

8 send(type := new update, block := i; *Subgradient evaluator* i)

9 **end**

10 launch(f_0^μ *gradient evaluator*)

/* main loop */

11 **repeat**

12 get_next_message(message) /* wait until message arrives before proceeding */

13 **if** message.type = new subgradient **then**

14 i := message.block

15 l := message.evaluator

16 read($\boldsymbol{g}_i$; `'sh:g_i'`) /* read new f_i subgradient */

17 read($\boldsymbol{g}_0$; `'sh:g_0'`) /* read last f_0^μ gradient */

18 k += 1

19 j := random$(1, N)$ /* select block to be updated */

20 $\boldsymbol{x}_j := \boldsymbol{x}_j - \lambda(k, p, r, q, \xi, \sigma) \cdot (I_j \boldsymbol{g}_0 + \boldsymbol{g}_j)$ /* perform BCD update */

21 write($\boldsymbol{x}_j$; `'sh:x_'` + j)

22 send(type := new update, block := j; *Subgradient evaluator* l)

23 send(type := new update, block := j; f_0^μ *gradient evaluator*)

24 **else if** message.type = new upper bound **then**

25 UB := message.value /* update upper bound */

26 **else if** message.type = new lower bound **then**

27 LB := message.value /* update lower bound */

28 **end**

29 **until** $UB - LB \leq \epsilon$ **or** `run_time` $\geq T$

30 read(primal_solution, 'sh:incumbent') /* collect primal solution information */

31 stop(*all*) /* terminate all subprocesses */

The execution starts with the *Master* process, presented in pseudocode in Alg. 1. The main tasks of the *Master*, besides initialization and termination, are to perform the block-coordinate updates and to manage the subgradient evaluation processes. Updates are performed following the evaluation of a subgradient for a certain component function by a *Subgradient evaluator*. After an update has taken place on block $j(k)$, the *Master* will instruct the re-computation of the subgradient of component function $j(k)$.

The *Subgradient evaluator* process, presented in Alg. 2, works on the tasks assigned by the *Master*, basically evaluating a component function for a certain iterate and returning the result. The f_0^μ *gradient evaluator*, presented in Alg. 3, on the other hand, is continuously providing new gradients for f_0^μ. Note that multiple updates of $\boldsymbol{x}$ might take place while the f_0^μ *gradient evaluator* is solving problem (8). Additionally, while only an update on block i would require f_i to be re-evaluated, any update of $\boldsymbol{x}$ requires that f_0^μ is re-evaluated in order to maintain the subgradients up-to-date. In order to avoid evaluating f_0^μ for old iterates, lines 7-10 of Alg. 3, refresh the local version of $\boldsymbol{x}$ to the current iterate, prior to the next f_0^μ evaluation.

The proposed implementation of dual iterations is effectively self-regulating the task load balancing, in the sense that it will assign more *Subgradient evaluators* to compute subgradients of the

Algorithm 2: Subgradient evaluator l.

Data: Functions f_i, $i = 1, \ldots, N$, termination time T.
Let message := NULL, $\boldsymbol{y} := 0$, $UB := +\infty$, $\boldsymbol{h} := 0$, primal_information := NULL
repeat
  get_next_message(message) /* receive task from Master */
  i := message.block
  read($\boldsymbol{y}$; 'sh:x_' + i) /* read current multipliers */
  evaluate $f_i(\boldsymbol{y})$ /* solve problem (7) */
  $UB := f_i(\boldsymbol{y})$
  $\boldsymbol{h}$:= get_element($\partial f_i(\boldsymbol{y})$) /* get a subgradient of f_i */
  primal_information := get_primal_solution($f_i(\boldsymbol{y})$)
  /* update f_i subgradient register */
  write*($\boldsymbol{h}$; 'sh:g_' + i)
  send(type:= new subgradient, block:= i; *Master*)
  /* update f_i upper bound register */
  write*([$\boldsymbol{y}$ UB]; 'sh:x_UB_' + i)
  send(type:= new component evaluation, block:= i; *Upper bound updater*)
  /* make primal information available for primal recovery */
  append(primal_information; 'sh:primal_pool')
  send(type:= new component evaluation, block:= i, evaluator:= l; *Primal recovery*)
until run_time $\geq T$

Algorithm 3: f_0^μ gradient evaluator.

Data: Starting point $\boldsymbol{x}^0$, f_0^μ function, termination time T.
Let $\boldsymbol{x} := \boldsymbol{x}^0$, $\boldsymbol{g}_0 := 0$, message_list := []
repeat
  evaluate $f_0^\mu(\boldsymbol{x})$ /* solve problem (8) */
  $\boldsymbol{g}_0 := \nabla f_0^\mu(\boldsymbol{x})$
  write($\boldsymbol{g}_0$; 'sh:g_0') /* update f_0^μ gradient register */
  get_all_messages(message_list) /* collect all messages received */
  for msg **in** message_list**:** msg.type = new update **do**
    i := msg.block
    read($\boldsymbol{x}_i$; 'sh:x_' + i) /* update multipliers information */
  end
  message_list := []
until run_time $\geq T$

most difficult component functions, so that in the long-run subgradients are computed with the same frequency for all component functions. This can be seen by considering an instance where all components functions are equally difficult to evaluate, except for one $f_{j^\star}$ which is much more difficult to evaluate than the rest. Since at each update the block-coordinate is selected by uniform sampling, $j^\star$ might be selected while its subgradient is already being evaluated for a previous iterate by a certain *Subgradient evaluator* process. As the iterations advance, there will be multiple *Subgradient evaluators* working on $j^\star$ for different iterates. The number of *Subgradient evaluators* working on $j^\star$ will continue to grow until the frequency at which we obtain new subgradients for $f_{j^\star}$ matches the frequency of evaluation of the rest of the component functions. This load balancing mechanism contributes to maintaining the delay $\ell(j^\star, k)$ at small values at all times.

The *Upper bound update*, presented in Alg. 4, provides estimates of the current dual function value to the *Master*. These estimates are computed using relation (16), i.e. gathering the last evaluated $\boldsymbol{x}_i$ of each component function $f_i, i = 1, \ldots, N$ and computing f_0 for a composite of the evaluated dual multipliers. The *Upper bound update* works in a very similar fashion to the f_0^μ *gradient evaluator*, but collecting the evaluated dual multipliers from the *Subgradient evaluators* instead of collecting the current iterates from the *Master*. New upper bounds are communicated to the *Master* as soon as they are computed.

The computation of estimated subgradients of component functions at $\boldsymbol{x}^0$ (cf. Alg. 1) and

Algorithm 4: Upper bound update.

Data: Starting point $\boldsymbol{x}^0$, upper bounds to component function values at $\boldsymbol{x}^0$, $UB_i^0\ \forall i = 0, \ldots, N$, f_0^μ function, termination time T.

Let $\boldsymbol{x} := \boldsymbol{x}^0$, $UB_i := UB_i^0\ \forall i = 0, \ldots, N$, message_list := []
repeat
  evaluate $f_0(\boldsymbol{x})$ /* solve problem (6) */
  $UB_0 := f_0(\boldsymbol{x})$
  if $\sum_{i=0}^{N} UB_i < UB$ **then** /* update upper bound */
    $UB := \sum_{i=0}^{N} UB_i$
    send(type:= new upper bound, value:= UB; *Master*)
  end
  get_all_messages(message_list) /* collect all messages received */
  for msg **in** message_list**:** msg.type = new component evaluation **do**
    $i :=$ msg.block
    read($[\boldsymbol{x}_i\ UB_i]$; 'sh:x_UB_' + i) /* evaluated multipliers information */
  end
  message_list := []
until run_time $\geq T$

upper bounds for component function values at $\boldsymbol{x}^0$ (cf. Alg. 4), although not presented in the algorithms for simplicity, is also performed in parallel and using all the computational resources available (that is, resources that will be assigned to the dual algorithm or to primal recovery). At this stage, we evaluate f_i approximately at $\boldsymbol{x}^0$ by substituting $\mathcal{D}_i$ by a relaxed constraint set, for all $i = 1, \ldots, N$, which provides estimates for the subgradients and upper bounds on the function values. In particular, for SUC instances, we relax the intertemporal constraints in $\mathcal{D}_i$ and we obtain the estimates by solving a sequence of optimal power flow problems subject additionally to minimum stable levels and maximum capacity constraints of generators.

4.2 Primal recovery

Primal recovery is an essential component of any Lagrangian relaxation scheme. Although there exist exact methods for recovering primal solutions in the case of linear programs [AW09], these methods do not extend to the mixed integer case and primal recovery relies, generally, on heuristics in the latter case.

In the case of stochastic unit commitment instances, we exploit of the fact that given $\bar{\boldsymbol{v}}$ such that $(\bar{\boldsymbol{v}}, \boldsymbol{w}_i) \in \mathcal{D}_i$ for certain i and some $\boldsymbol{w}_i$, then $\bar{\boldsymbol{v}} \in \mathcal{U}$ and $(\bar{\boldsymbol{v}}, \boldsymbol{w}_j) \in \mathcal{D}_j$ for some $\boldsymbol{w}_j$ for any $j = 1, \ldots, N$. In other words, any solution to a scenario subproblem $\bar{\boldsymbol{v}}$ can be used as a non-anticipative solution (primal candidate) which we can fix, and then compute the recourse solutions $\boldsymbol{w}_i$ in order to obtain a complete primal solution and a lower bound on the objective [Ahm13, Ahm15]. Recovering a feasible non-anticipative solution on every iteration for every scenario, however, leads to an enormous number of primal candidates which then need to enter a queue for evaluation. Within this context, we test three rules to determine the order with which primal candidates are evaluated:

- 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), an approach that takes into account that as dual iterations advance, solutions to scenario subproblems tend to more almost non-anticipative ($\boldsymbol{v}_i = \boldsymbol{v}_j$, $i, j = 1, \ldots, N$), therefore scenario subproblem solutions in later iterations could have better overall performance.
- Additionally, we propose a fourth method based on importance sampling (IS). First, we pick a sample of the scenarios based on their estimated importance. Then, we average the current subproblem solution associated to the sampled scenarios and project the averaged solution

onto the feasible set for $\boldsymbol{v}$. This method 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, to generate different candidates after each dual iteration if necessary. The latter will not be possible if we were to average the solutions to all scenario subproblems, as suggested in [CS99], since the average of the solution to all scenario subproblems does not change significantly from one iteration to the next.

The four primal recovery methods (FIFO, RND, LIFO, IS) are embedded in the *Primal recovery* process presented in Alg. 5, which uses the *Recourse evaluators*, Alg. 6, to obtain complete primal solutions for the primal candidates in a similar fashion as the *Master* uses the *Subgradient evaluators*. The evaluation of primal candidates is also asynchronous, in the sense that a new candidate is generated and starts to be evaluated as soon as computational resources are available, without waiting for another candidate to be fully evaluated. New lower bounds are communicated to the *Master* as soon as the information about the incumbent is collected and saved.

5 Numerical results

We implement the proposed asynchronous algorithm as described in the previous section in Mosel, using the module mmjobs to handle communications and solving the subproblems with XPress [CH14]. We configure XPress to solve the root node in all subproblems using the barrier algorithm and we set the termination gap of subproblems to 1%.

We test the proposed algorithm on instances of the WECC [POR15] and CWE [AP16] systems. Sizes and solution times of scenario subproblems are presented in Table 1, where it can be appreciated that the instances used in the present study are of the same or more difficulty than the state-of-the-art. Numerical experiments ran on the Sierra cluster hosted at the Lawrence Livermore National Laboratory. Each node of the Sierra cluster is equipped with two Intel Xeon EP X5660 processors (12 cores per node) and 24GB of RAM memory.

5.1 Western Electricity Coordinating Council system instances

The WECC system instance [POR15] is composed of 130 thermal generators, 182 nodes and 319 lines. It features multiarea renewable production with hourly resolution over a 24 hour 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.

The solution times of these instances are summarized in Table 2 for different configurations of the asynchronous algorithm. Regarding dual methods, 'diminishing' corresponds to a stepsize of the type $1/k$, 'polyak' corresponds to the Polyak stepsize defined in equation (17), where the diminishing part is set to decrease from 0.5 to 0.25 in $50N$ iterations with $q = 1$, and 'polyak-long' is equivalent to 'polyak' with the sole difference that the stepsize decreases from 1 to 0.75 in $50N$ iterations. Primal methods correspond to the four methods described in subsection 4.2. We use 2 nodes to solve the 10 scenario instances and 10 nodes to solve the 100 and 1000 scenario instances. The number of dual and primal processes, M and R, are determined according to

$$
\begin{aligned}
M &:= \min\left\{N, \text{round}\left(\alpha \cdot \frac{\#\text{ available processors}}{\#\text{ processors per } \textit{Subgradient evaluator}}\right)\right\} \\
R &:= \#\text{ available processors} - M,
\end{aligned}
$$

where α is the proportion of processors dedicated to subgradient computation, given as an input to the script. The number of processors per *Subgradient evaluator* is set to 1 for WECC instances. Stars in the last column indicate that the termination time T was reached without achieving the 1% target optimality gap.

On the one hand, from the perspective of dual optimization, Polyak-like stepsizes outperform $1/k$ stepsizes when considering a 1% termination criterion. In all our instances and test runs, Polyak-like stepsizes provided better results as a default option without the need for tuning.

Algorithm 5: Primal recovery.

Data: Number of primal processors R, initial collection of m^0 primal candidates $\{\tilde{\boldsymbol{v}}^1, \ldots, \tilde{\boldsymbol{v}}^{m_0}\}$ $(R \leq m_0 \cdot N < R + N)$, primal recovery method PRM, termination time T.

Let $m := m^0$, $\bar{\boldsymbol{v}}^l := \tilde{\boldsymbol{v}}^l \ \forall l = 1, \ldots, m$, $F(l, i) := -\infty \ \forall l = 0, \ldots, m, i = 1, \ldots, N$, primal_queue:= [], message := `NULL`, $\bar{\boldsymbol{v}}^{\text{best}} := 0$, $\bar{\boldsymbol{w}}^{\text{best}} := 0$, $LB := -\infty$

primal_queue = $\sum_{l=1}^{m} \sum_{i=1}^{N}$[(index := l, block := i)] `/* generate initial tasks */`

`/* launch recourse evaluators and assign intial recovery tasks */`

for $s = 1, \ldots, R$ **do**
  nc := extractfirst(primal_queue)
  launch(*Recourse evaluator s*)
  send(type:=recovery, candidate:=nc.index, block:=nc.block; *Recourse evaluator s*)
end

`/* main loop */`

repeat
  get_next_message(message)
  if message.type = new component evaluation **then**
    read(primal_pool, `'sh:primal_pool'`) `/* read what's new in the pool */`
  else if message.type = primal recovered **then**
    t := message.candidate
    s := message.evaluator
    $F(t,$ message.block$)$:= message.value
    `/* update incumbent if a candidate is evaluated for all blocks */`
    if is_it_fully_evaluated(t) **then**
      if $\sum_{i=1}^{N} F(t, i) > LB$ **then**
        $\bar{\boldsymbol{v}}^{\text{best}} := \bar{\boldsymbol{v}}^t$
        for $i = 1, \ldots, N$ **do** read($\bar{\boldsymbol{w}}_i^{\text{best}}$, `'sh:primalsol_cand_'` $+ t +$ `_block_` $+ i$)
        write($[\bar{\boldsymbol{v}}^{\text{best}}\ \bar{\boldsymbol{w}}^{\text{best}}]$, 'sh:incumbent')
        $LB := \sum_{i=1}^{N} F(t, i)$
        send(type:= new lower bound, value:= LB; Master)
      else
        for $i = 1, \ldots, N$ **do** delete(`'sh:primalsol_cand_'` $+ t +$ `_block_` $+ i$)
      end
    end
    `/* generate new candidates if necessary */`
    if primal_queue = [] **then**
      m += 1
      $\bar{\boldsymbol{v}}^m$:= generate_candidate(PRM, primal_pool)
      primal_queue += $\sum_{i=1}^{N}$[(index := candidate, block := i)]
    end
    `/* assign next recovery task to the released evaluator */`
    nc := extractfirst(primal_queue)
    send(type:=recovery, candidate:=nc.index, block:=nc.block; *Recourse evaluator s*)
  end
until `run_time` $\geq T$

Algorithm 6: Recourse evaluator s.

Data: Vectors $\boldsymbol{c}_i, \boldsymbol{d}_i$ and sets $\mathcal{D}_i$, $i = 1, \ldots, N$, termination time T.

Let message := NULL, $\bar{\boldsymbol{v}} := 0$, $\bar{\boldsymbol{w}} := 0$, $LB := -\infty$

repeat

  get_next_message(message) /* receive task from Primal recovery */

  $t :=$ message.candidate

  $l :=$ message.block

  read($\bar{\boldsymbol{v}}$; 'sh:v_' + t) /* read primal candidate */

  $\bar{\boldsymbol{w}} := \arg\max_w \left\{ \boldsymbol{d}_l^T \boldsymbol{w} : (\bar{\boldsymbol{v}}, \boldsymbol{w}) \in \mathcal{D}_l \right\}$ /* compute recourse solution */

  $LB := \boldsymbol{c}_l^T \bar{\boldsymbol{v}} + \boldsymbol{d}_l^T \bar{\boldsymbol{w}}$ /* compute candidate value on block l */

  /* make partial primal solution available to Primal recovery */

  write($\bar{\boldsymbol{w}}$; 'sh:primalsol_cand_' + t + _block_ + l)

  send(type := primal recovered, candidate:= t, block:= l, value:= LB, evaluator:= s; *Primal recovery*)

until run_time $\geq T$

Table 1: Scenario subproblem sizes and solution times for different instances used in the present study (highlighted in boldface) and in the literature. Subproblem solution times for WECC [POR15] correspond to the winter weekend instance with 100 scenarios, while subproblem solution times for CWE [AP16] correspond to the spring weekday instance with 120 scenarios.

Instance	Rows	Columns	Non-zeros	Integers	Subproblem solution time [s], avg.	(max.)
WECC [CGSM+15]	69 447	28 943	240 724	4 080	9.4	(25.7)
WECC [POR15]	34 104	23 090	129 216	3 074	10.8	(64.9)
EDF [vAM16]	812 906	73 562	–	26 122	–	–
CWE [AP16]	553 871	389 112	1 769 973	9 752	3 691.5	(5 877.8)

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

N	Dual method	Primal method	M	R	Solution time [s], avg. (max.)			
					2% optimality		1% optimality	
10	diminishing	FIFO	10	11	59.2	(120.1)	>1382.1	(*1800.0)
10	diminishing	RND	10	11	78.0	(195.5)	>1364.7	(*1800.0)
10	diminishing	LIFO	10	11	72.9	(199.2)	>1383.9	(*1800.0)
10	diminishing	IS	10	11	79.6	(183.6)	>1286.6	(*1800.0)
10	polyak	FIFO	10	11	182.7	(629.7)	504.7	(1500.6)
10	polyak	RND	10	11	182.4	(619.2)	500.0	(1380.3)
10	polyak	LIFO	10	11	159.0	(478.1)	445.0	(1249.9)
10	polyak	IS	10	11	135.8	(471.8)	412.3	(1427.8)
10	**polyak-long**	**IS**	**10**	**11**	**76.1**	**(289.5)**	**217.4**	**(651.5)**
100	polyak	FIFO	86	29	1957.7	(9657.9)	>9722.3	(*10800.0)
100	polyak	RND	86	29	180.3	(492.8)	>3283.6	(*10800.0)
100	polyak	LIFO	86	29	221.2	(624.3)	>2514.3	(*10800.0)
100	polyak	IS	86	29	161.8	(453.8)	824.6	(2482.9)
100	**polyak-long**	**IS**	**98**	**17**	**95.4**	**(226.7)**	**413.2**	**(1108.4)**
1000	polyak	IS	86	28	1472.6	(5095.0)	10496.9	(30836.9)
1000	**polyak-long**	**IS**	**97**	**17**	**858.6**	**(2799.9)**	**4364.1**	**(11140.1)**

Table 3: Asynchronous execution statistic for the best performing configuration on WECC instances. The table includes the instances with the smaller and larger average f_i evaluation time, and the instances with the smaller and larger synchronous idle time.

N	Day type	f_i evaluation time [s]		Ratio of passes	Synchronous idle time [%]
		Avg.	Sd.		
10	Autumn, WE	**4.1**	2.0	1.38	52.0%
10	Winter, WE	**10.5**	5.9	1.45	53.1%
10	Summer, WE	4.5	1.1	2.29	**25.0%**
10	Spring, WD	5.2	3.2	2.20	**60.2%**
100	Summer, WE	**4.4**	1.2	2.60	**56.5%**
100	Winter, WE	**10.8**	6.8	1.69	**74.0%**
1000	Autumn, WE	**4.8**	2.9	3.67	82.2%
1000	Winter, WE	**12.0**	8.0	1.97	**84.3%**
1000	Summer, WE	4.9	1.9	4.63	**71.6%**

Moreover, tuning a Polyak-like stepsize, as done in the present work by using a longer stepsize, can bring significant benefits to the overall convergence.

For primal recovery, we observe that the three methods that recover solutions directly from solution 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 method significantly outperforms its counterparts in all instances.

Solution times presented in Table 2, for the best performing configuration (highlighted in boldface), largely outperform the results of [POR15], in which 1000 scenario instances (23.1 million variables, 35.9 million constraints and 3.1 million integers) were solved in up to 24 hours using 1000 processors. Instead, the best performing algorithm in this paper solves the same instance in less than 4 hours using 120 processors. Similar speedups are observed for instances with fewer scenarios with respect to [POR15]. The current state-of-the-art method for systems of similar scale to the WECC instances used in this work, [CGSM$^+$15] (see Table 1), is able to solve instances with up to 100 scenarios within 1.5–2.5% optimality within 25 minutes, while the proposed algorithm solves them to 1% optimality in 18.5 minutes and to 2% suboptimality in less than 5 minutes.

The asynchronous execution statistics presented in Table 3 indicate the importance of the idle time of processors in parallel algorithms. f_i evaluation times exhibit a relative standard deviation (sd./avg.) that ranges between 23.8% and 70.9%, which hints that synchronous algorithms can suffer from important idle times. Note that the number of evaluations of f_i (passes) is different for each i at the termination of the algorithm. We report the ratio between the number of times that f_i is evaluated for most commonly and least commonly evaluated scenarios, which can range from 1.38 to 8. We estimate the idle time of a synchronous subgradient algorithm [POR15] as the time that processors would be idle if we were to have $M = N$ (assigning one scenario per processor) and only moving to the next iteration once f_i for all $i = 1, \ldots, N$ are evaluated. The synchronous idle times range from 25.0% to 84.3% of the total computation time, with the idle proportion of time growing as we increase the number of scenarios. These results indicate that synchronous algorithms, although simpler in conception and implementation, are not the optimal design choice for exploiting high performance computing, especially when tackling problems decomposed in a large number of components.

5.2 Central Western European system instances

The CWE system instance is essentially the same considered in [AP16] with the difference that in the present work we additionally include the commitment of nuclear units and the selection of the set point of CHP power plants (around which CHPs can vary their production within a limited range [DA15], a continuous variable) as first-stage decisions of SUC. The instance constists of 656 thermal generators, 679 nodes and 1037 lines, featuring multiarea renewable production with quarterly resolution over a 24 hour horizon (96 periods) for 8 representative day types and collections of 30, 60 and 120 scenarios fo renewable production. Each scenario subproblem of the CWE instances is almost one order of magnitude larger, in terms of matrix size, than the scenario

Table 4: Solution time statistics for WECC instances, over 8 representative day types. All instances use the Polyak stepsize ('polyak') to perform dual updates.

N	Primal method	M	R	Solution time [s], avg. (max.)			
				2% optimality		1% optimality	
30	RND	30	55	2509.9	(5498.3)	5311.4	(11530.1)
30	LIFO	30	55	2501.8	(5629.8)	5626.0	(13730.1)
30	IS	30	55	2674.3	(5643.4)	4628.2	(8689.2)
60	RND	43	29	3950.6	(6623.8)	>16753.5	(*36000.0)
60	IS	43	29	3545.8	(7021.6)	>10409.9	(*36000.0)
120	IS	43	29	6503.7	(14059.4)	>29058.3	(*54000.0)

Table 5: Worst optimality gaps at different wall-times, for the best configuration and over 8 representative day types for CWE instances.

N	Worst optimality gap			
	20'	1 hour	2 hours	4 hours
30	5.11%	3.86%	1.21%	0.85%
60	100.00%	4.15%	1.71%	1.32%
120	100.00%	4.67%	3.82%	1.95%

subproblems of the largest instance considered in the SUC literature [vAM16] (see Table 1).

Tables 4 and 5 present solution statistics for the proposed algorithm on the CWE instances. We use 10 nodes for all instances and two processors are used by each *Subgradient evaluator*. Solution times are much larger that those observed for the WECC, nevertheless they remain within operationally acceptable time frames for day-ahead scheduling. Moreover, given 4 hours of termination time, the algorithm is able to find solutions within 2% of optimality for all instances.

The increase in overall solution times with respect to the WECC instances results mostly from the time required for solving scenario subproblems, as shown in Table 6. For spring weekday instances, in particular, an important proportion of subproblems reach the MIP time limit (set to 5400 seconds). Whenever this occurs, and the MIP solver is not able to find a good feasible integer solution, we apply a specialized rounding heuristic to the solution of the root in order to obtain an estimated subgradient of the corresponding component function. Note that for certain instances the algorithm did not evaluate all f_i before finishing, which implies that the proposed algorithm terminates before a synchronous algorithm would even perform its first iteration. For theses cases we can only estimate a lower bound on the synchronous idle time, which are indicated in the last column of Table 6. We observe that, as for the WECC instances, synchronous schemes can dramatically underutilize parallel computing infrastructure.

Table 6: Asynchronous execution statistics for best configuration on CWE instances. The table includes the instances with the smaller and larger average f_i evaluation time, and the instances with the smaller and larger synchronous idle time.

N	Day type	f_i evaluation time [s]		f_i eval. reached MIP time limit [%]	Ratio of passes	Synchronous idle time [%]
		Avg.	Sd.			
30	Winter, WE	**1102.6**	451.4	0%	3	>42.7%
30	Spring, WD	**5235.0**	759.9	71.0%	1.5	**6.9%**
30	Winter, WD	1277.2	526.3	0%	2.5	**51.1%**
60	Winter, WE	**954.3**	381.2	0%	5	**>70.0%**
60	Spring, WD	**4805.5**	1273.4	51.0%	3	>29.0%
60	Autumn, WD	1303.0	330.6	0%	4.5	**33.1%**
120	Winter, WE	**975.5**	378.3	0%	4	**>80.2%**
120	Spring, WD	**3690.5**	2167.2	33.6%	5.5	**14.0%**

6 Conclusions

We propose an asynchronous dual decomposition algorithm for stochastic unit commitment, in which dual iterations are performed using a specialized block-coordinate subgradient method, for which we provide convergence guarantees. We propose a high performance computing implementation of the algorithm and test it on instances of the WECC and CWE systems. The algorithm is able to solve all instances within operationally acceptable time frames. Furthermore, we find that synchronous algorithms dramatically underutilize high performance computing infrastructure, resulting in processors idle time of up to 84.3%, which stresses the need for designing asynchronous algorithms to tackle large-scale real world 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 instances of real power systems and (*ii*) detailed deterministic unit commitment problems over large interconnected power systems, integrating the optimization of transmission and distribution systems through convex relaxations of AC power flow constraints [CNH+16].

Acknowledgement

The authors would like to thank Dr. Vitor de Matos (Plan4 Engenharia) for suggesting the idea of evaluating primal solutions in reverse order (LIFO).

The authors would also like to thank the Fair Isaac Corporation FICO for providing licenses for Xpress, and the Lawrence Livermore National Laboratory for granting access and computing time at the Sierra cluster.

This research was funded by the ENGIE Chair on Energy Economics and Energy Risk Management and by the Univeristé catholique de Louvain through an FSR grant.

References

[Ahm13] Shabbir Ahmed. A scenario decomposition algorithm for 01 stochastic programs. *Operations Research Letters*, 41(6):565 – 569, 2013.

[Ahm15] Shabbir Ahmed. Corrigendum to "A scenario decomposition algorithm for 01 stochastic programs" [Oper. Res. Lett. 41(6) (2013) 565 – 569]. *Operations Research Letters*, 43(2):215 – 217, 2015.

[AP16] Ignacio Aravena and Anthony Papavasiliou. Renewable energy integration in zonal markets. *IEEE Transactions on Power Systems*, PP(99):1–1, 2016.

[AW09] Kurt M. Anstreicher and Laurence A. Wolsey. Two "well-known" properties of subgradient optimization. *Mathematical Programming*, 120(1):213–220, 2009.

[BT96] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1st edition, 1996.

[CBFL+09] Santiago Cerisola, Álvaro Baíllo, José M. Fernández-López, Andrés Ramos, and Ralf Gollmer. Stochastic power generation unit commitment in electricity markets: A novel formulation and a comparison of solution methods. *Operations Research*, 57(1):32–46, 2009.

[CDR15] Sorathan Chaturapruek, John C. Duchi, and Christopher R. 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.

[CGCR96] Pierre Carpentier, Guy Gohen, Jean-Christophe Culioli, and Arnaud Renaud. Stochastic optimization of unit commitment: a new decomposition framework. *IEEE Transactions on Power Systems*, 11(2):1067–1073, May 1996.

[CGSM+15] Kwok Cheung, Dinakar Gade, César Silva-Monroy, Sarah M. Ryan, Jean-Paul Watson, Roger J.-B. Wets, and David L. Woodruff. Toward scalable stochastic unit commitment. Part 2: solver configuration and performance assessment. *Energy Systems*, 6(3):417–438, 2015.

[CH14] Y. Colombani and S. Heipcke. Multiple models and parallel solving with Mosel, February 2014. Avalaible at: http://community.fico.com/docs/DOC-1141.

[CNH+16] Michael Caramanis, Elli Ntakou, William W. Hogan, Aranta Chakrabortty, and Jens Schoene. Co-optimization of power and reserves in dynamic t amp;d power markets with nondispatchable renewable generation and distributed energy resources. *Proceedings of the IEEE*, 104(4):807–836, April 2016.

[CS99] Claus C. Carøe and Rüdiger Schultz. Dual decomposition in stochastic integer programming. *Operations Research Letters*, 24(12):37 – 45, 1999.

[DA15] Ilias Dimoulkas and Mikael Amelin. Probabilistic day-ahead CHP operation scheduling. In *2015 IEEE Power Energy Society General Meeting*, pages 1–5, July 2015.

[Erm83] Yuri Ermoliev. Stochastic quasigradient methods and their application to system optimization. *Stochastics*, 9(1-2):1–36, 1983.

[FR13] Olivier Fercoq and Peter Richtárik. Smooth minimization of nonsmooth functions with parallel coordinate descent methods. Optimization Online, 2013.

[KZ15] Kibaek Kim and Victor M. Zavala. Algorithmic innovations and software for the dual decomposition method applied to stochastic mixed-integer programs. Optimization Online, 2015.

[LHPA13] Miles Lubin, J. A. Julian Hall, Cosmin G. Petra, and Mihai Anitescu. Parallel distributed-memory simplex for large-scale stochastic lp problems. *Computational Optimization and Applications*, 55(3):571–596, 2013.

[LMPS13] Miles Lubin, Kipp Martin, Cosmin G. Petra, and Burhaneddin Sandıkçı. On parallelizing dual decomposition in stochastic integer programming. *Operations Research Letters*, 41(3):252 – 258, 2013.

[LS04] Guglielmo Lulli and Suvrajeet Sen. A branch-and-price algorithm for multistage stochastic integer programming with application to stochastic batch-sizing problems. *Management Science*, 50(6):786–796, 2004.

[LWR+15] Ji Liu, Stephen J. Wright, Christopher Ré, Victor Bittorf, and Srikrishna Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. *Journal of Machine Learning Research*, 16(1):285–322, 2015.

[MOR15] Lluís-Miquel Munguía, Geoffrey Oxberry, and Deepak Rajan. PIPS-SBB: A parallel distributed-memory branch-and-bound algorithm for stochastic mixed-integer programs. Optimization Online, 2015.

[MPS01] Hans W. Moritsch, Georg Ch. Pflug, and Mariusz Siomak. Asynchronous nested optimization algorithms and their parallel implementation. *Wuhan University Journal of Natural Sciences*, 6(1):560–567, 2001.

[NB01] Angelia Nedić and Dimitri P. Bertsekas. Incremental subgradient methods for nondifferentiable optimization. *SIAM Journal on Optimization*, 12(1):109–138, 2001.

[NBB01] Angelia Nedić, Dimitri P. Bertsekas, and Vivek S. Borkar. Distributed asynchronous incremental subgradient methods. In D. Butnariu, Y. Censor, and S. Reich, editors, *Inherently Parallel Algorithms in Feasibility and Optimization and their Applications*, Studies in Computational Mathematics, pages 381–407. Elsevier, 2001.

[Ned02] Angelia Nedić. *Subgradient Methods for Convex Optimization*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, June 2002.

[Nes05] Yurii Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103(1):127–152, 2005.

[Nes12] Yu. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

[PO13] Anthony Papavasiliou and Shmuel S. Oren. Multiarea stochastic unit commitment for high wind penetration in a transmission constrained network. *Operations Research*, 61(3):578–592, 2013.

[Pol69] Boris T. Polyak. Minimization of unsmooth functionals. *USSR Computational Mathematics and Mathematical Physics*, 9:14–29, 1969.

[POR15] Anthony Papavasiliou, Shmuel S. Oren, and Barry Rountree. Applying high performance computing to transmission-constrained stochastic unit commitment for renewable energy integration. *IEEE Transactions on Power Systems*, 30(3):1109–1120, May 2015.

[RRA16] K. Ryan, D. Rajan, and S. Ahmed. Scenario decomposition for 0-1 stochastic programs: Improvements and asynchronous implementation. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 722–729, May 2016.

[SB04] Takayuki Shiina and John R. Birge. Stochastic unit commitment problem. *International Transactions in Operational Research*, 11(1):19–32, 2004.

[SGM15] Tim Schulze, Andreas Grothey, and Ken McKinnon. A stabilised scenario decomposition algorithm applied to stochastic unit commitment problems. Optimization Online, 2015.

[TBL96] Samer Takriti, John R. Birge, and Erik Long. A stochastic model for the unit commitment problem. *IEEE Transactions on Power Systems*, 11(3):1497–1508, Aug 1996.

[Tse01] Paul Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of Optimization Theory and Applications*, 109(3):475–494, 2001.

[TvAFL15] Milad Tahanan, Wim van Ackooij, Antonio Frangioni, and Fabrizio Lacalandra. Large-scale unit commitment under uncertainty. *4OR*, 13(2):115–171, 2015.

[vAM16] Wim van Ackooij and Jérôme Malick. Decomposition algorithm for large-scale two-stage unit-commitment. *Annals of Operations Research*, 238(1):587–613, 2016.

[Wri15] Stephen J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.