Randomized Progressive Hedging methods for Multi-stage Stochastic Programming

Gilles Bareilles, Yassine Laguel, Dmitry Grishchenko, Franck Iutzeler, Jérôme Malick

▶ **To cite this version:**

Gilles Bareilles, Yassine Laguel, Dmitry Grishchenko, Franck Iutzeler, Jérôme Malick. Randomized Progressive Hedging methods for Multi-stage Stochastic Programming. Annals of Operations Research, 2020, 295, pp.535-560. 10.1007/s10479-020-03811-5 . hal-02946615

HAL Id: hal-02946615
https://hal.science/hal-02946615

Submitted on 25 Sep 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Noname manuscript No.
(will be inserted by the editor)

Randomized Progressive Hedging methods for Multi-stage Stochastic Programming

Gilles Bareilles · Yassine Laguel · Dmitry Grishchenko · Franck Iutzeler · Jérôme Malick

Revised version Received: date / Accepted: date

Abstract Progressive Hedging is a popular decomposition algorithm for solving multi-stage stochastic optimization problems. A computational bottleneck of this algorithm is that *all* scenario subproblems have to be solved at each iteration. In this paper, we introduce randomized versions of the Progressive Hedging algorithm able to produce new iterates as soon as a *single* scenario subproblem is solved. Building on the relation between Progressive Hedging and monotone operators, we leverage recent results on randomized fixed point methods to derive and analyze the proposed methods. Finally, we release the corresponding code as an easy-to-use Julia toolbox and report computational experiments showing the practical interest of randomized algorithms, notably in a parallel context. Throughout the paper, we pay a special attention to presentation, stressing main ideas, avoiding extra-technicalities, in order to make the randomized methods accessible to a broad audience in the Operations Research community.

1 Introduction

1.1 Context: decomposition of stochastic problems and computational limitations

Stochastic optimization is a rich and active research domain with various applications in science and engineering ranging from telecommunication and medicine to finance; we refer to the two textbooks [18] and [21] for theoretical foundations of this field and pointers to applications. Expressive stochastic models lead to large-dimensional optimization problems, that may be computationally challenging. In many applications, the randomness is highly structured (e.g. in multistage stochastic programming) and can be exploited by decomposition methods [19, Chap. 3.9]. The two main advantages

G. Bareilles, Y. Laguel, D. Grishchenko, F. Iutzeler
Univ. Grenoble Alpes
E-mail: {firstname.lastname}@univ-grenoble-alpes.fr

J. Malick
CNRS and LJK
E-mail: jerome.malick@univ-grenoble-alpes.fr

of decomposition methods are that i) they replace a large and difficult stochastic programming problem by a collection of smaller problems; and ii) these smaller subproblems can usually be solved efficiently with standard off-the-shelf optimization software. As a result, decomposition methods provide an efficient and specialized methodology for solving large and difficult stochastic programming problems by employing readily available tools.

Progressive Hedging is a popular dual decomposition method for multistage stochastic programming. This algorithm was introduced in [17] and can be interpreted as a fixed-point method over a splitting operator [18]. Through this connection, Progressive Hedging is proved to be convergent for solving convex stochastic programs. There are also many applications to mixed-integer stochastic problems where Progressive Hedging acts as an efficient heuristic to get useful bounds; see e.g. [24]. For historical perspectives, theoretical analysis, and references to applications, we refer to [18].

Progressive Hedging tackles multi-stage stochastic problems by decomposing them over the scenarios and solving independently the smaller subproblems relative to one scenario. However the number of these subproblems grows exponentially with the number of stages, so that the computational bottleneck of this algorithm is that *all* scenario subproblems have to be solved at each iteration. As a decomposition method solving scenario subproblems independently, Progressive Hedging is an intrinsically parallel algorithm and admits direct parallel implementations for distributed computing systems (e.g. multiple threads in a machine, or multiple machines in a cluster). In the homogeneous case (where all subproblems are solved with similar duration) such parallel implementations are efficient in practice and require no additional theoretical study; for early works discussing parallelization, see e.g. the doctoral dissertation [22] and the conference papers [6,20]. However, when the subproblems have different difficulties or the computing system is heterogeneous (e.g. with different machines or non-reliable communications between machines), the parallelization speed-up can be drastically degraded. Thus designing efficient, theoretically-grounded, variants of Progressive Hedging for heterogeneous distributed settings is still an on-going research topic (see e.g. the preprint [10]).

1.2 Contribution: accessible, efficient, parallel Progressive Hedging variants

In this paper, we present optimization methods based on Progressive Hedging having efficient parallel implementation and able to tackle large-scale multistage stochastic problems. Our variants are randomized algorithms solving subproblems incrementally, thus alleviating the synchronization barrier of the standard Progressive Hedging. When deployed on computing systems having multiple workers, our algorithms are able to make the most of the computational abilities, synchronously or asynchronously.

This work is based on the interaction of two complementary fields of research:

- applications of Progressive Hedging in the OR community with expressive uncertainty models leading to large-scale multistage stochastic problems;
- recent developments on randomization techniques in the optimization and monotone operators community, motivated by the distributive abilities of modern computing systems.

The connection between these two domains is natural, through the well-known interpretation of Progressive Hedging as a fixed point algorithm (see e.g. [18]). We also build

on this connection to propose our efficient randomized Progressive Hedging algorithms. We pay a special attention to making our developments easily accessible for a broad audience in the OR and stochastic programming community: we explicitly derive the proposed methods from the textbook formulation of Progressive Hedging; we rely on well-established results to highlight fundamental ideas and to hide unnecessary technicalities. Furthermore, we take advantage of the recent distributive abilities of the Julia language [2] (using the *Distributed* module) and provide an easy-to-use toolbox solving multistage stochastic programs with the proposed methods.

2 Multistage stochastic programs: recalls and notation

In this section, we lay down the multistage stochastic model considered in this paper as well as our notation. We follow closely the notation of the textbook [19, Chap. 3].

Stochastic programming deals with optimization problems involving uncertainty, modelled by random variable ξ, with the goal to find a feasible solution $x(\xi)$ that is optimal in some sense relatively to ξ. Considering an objective function f and a risk measure $\mathcal{R}$, the generic formulation of a stochastic problem is

$$\min_{x} \quad \mathcal{R}\big(f(x(\xi),\xi)\big). \tag{2.1}$$

For an extensive review of stochastic programming, see e.g. [21].

In the multistage setting, the uncertainty of the problem is revealed sequentially in T stages. The random variable ξ is split into $T-1$ chunks, $\xi = (\xi_1, .., \xi_{T-1})$, and the problem at hand is to decide at each stage $t = 1, \dots, T$ what is the optimal action, $\mathrm{x}_t(\xi_{[1,t-1]})$, given the previous observations $\xi_{[1,t-1]} := (\xi_1, \dots, \xi_{t-1})$. The global variable of this problem thus writes

$$\mathrm{x}(\xi) = (\mathrm{x}_1, \mathrm{x}_2(\xi_1), \dots, \mathrm{x}_T(\xi_{[1,T-1]})) \in \mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_T} = \mathbb{R}^n$$

where $(n_1, \dots, n_T)$ are the size of the decision variable at each stage and $n = \sum_{t=1}^{T} n_t$ is the total size of the problem.

We focus on the case where the random variable ξ can take a finite number S of values called *scenarios* and denoted by $\xi^1, \dots, \xi^S$. Each scenario occurs with probability $p_s = \mathbb{P}[\xi = \xi^s]$ and is revealed in T stages through one common start and a realization of the random variable $\xi^s = (\xi_1^s, \dots, \xi_{T-1}^s)$. It is thus natural to represent the scenarios as the outcome of a probability tree, as illustrated in Figure 1. For each scenario $s \in \{1, \dots, S\}$, the target of multistage stochastic programming is to provide a decision $\mathrm{x}^s = (\mathrm{x}_1, \mathrm{x}_2(\xi_1^s), \dots, \mathrm{x}_T(\xi_{[1,T-1]}^s))$, and thus the full problem variable writes

$$\mathrm{x} = \begin{pmatrix} \mathrm{x}_1 & \mathrm{x}_2\left(\xi_1^1\right) & \dots & \mathrm{x}_{T-1}\left(\xi_{[1,\dots,T-2]}^1\right) & \mathrm{x}_T\left(\xi_{[1,\dots,T-1]}^1\right) \\ \mathrm{x}_1 & \mathrm{x}_2\left(\xi_1^2\right) & \dots & \mathrm{x}_{T-1}\left(\xi_{[1,\dots,T-2]}^2\right) & \mathrm{x}_T\left(\xi_{[1,\dots,T-1]}^2\right) \\ \vdots & & & & \vdots \\ \mathrm{x}_1 & x_2\left(\xi_1^S\right) & \dots & \mathrm{x}_{T-1}\left(\xi_{[1,\dots,T-2]}^S\right) & \mathrm{x}_T\left(\xi_{[1,\dots,T-1]}^S\right) \end{pmatrix} \in \mathbb{R}^{S \times n}. \tag{2.2}$$

From (2.2), we see that by construction of the randomness, the decision at stage 1 must be the same for all the scenarios. Indeed, as no random variables have been observed, the user does not have any information about the scenarios. In the same

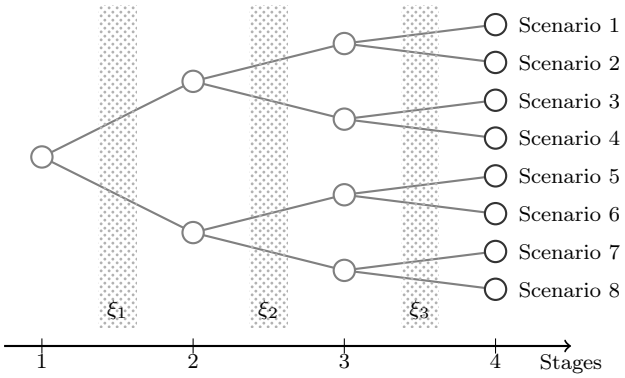

Fig. 1: Scenarios as the outcomes of a probability tree.

vein, given the specificity of these random variables, an important feature of finite multistage problems is that if two scenarios s_1 and s_2 coincide up to stage $t-1$ (i.e. $\xi^{s_1}_{[1,t-1]} = \xi^{s_2}_{[1,t-1]}$), then the obtained decision variables must be equal up to stage t (i.e. $(\mathrm{x}_1, \mathrm{x}_2(\xi^{s_1}_1), \ldots, \mathrm{x}_t(\xi^{s_1}_{[1,t-1]})) = (\mathrm{x}_1, \mathrm{x}_2(\xi^{s_2}_1) \ldots, \mathrm{x}_t(\xi^{s_2}_{[1,t-1]}))$). These constraints are called *non-anticipativity*. Geometrically these constraints define a subspace of $\mathbb{R}^{S\times n}$ that we denote by

$$\mathcal{W} = \left\{ \mathrm{x} \in \mathbb{R}^{S\times n} \colon \forall s_1, s_2 \;\middle|\; \begin{array}{l} \mathrm{x}^{s_1}_1 = \mathrm{x}^{s_2}_1 \;\; (t=1) \\ \quad \text{and} \\ \mathrm{x}^{s_1}_t = \mathrm{x}^{s_2}_t \;\; \text{if } \xi^{s_1}_{[1,t-1]} = \xi^{s_2}_{[1,t-1]} \;\; (t \geq 2) \end{array} \right\}, \tag{2.3}$$

where we denote $\mathrm{x}^s_t \in \mathbb{R}^{n_t}$ the decision variable for scenario s at stage t. We see that the non-anticipativity constraints lead to a variable x with a block structure as depicted in Figure 2.

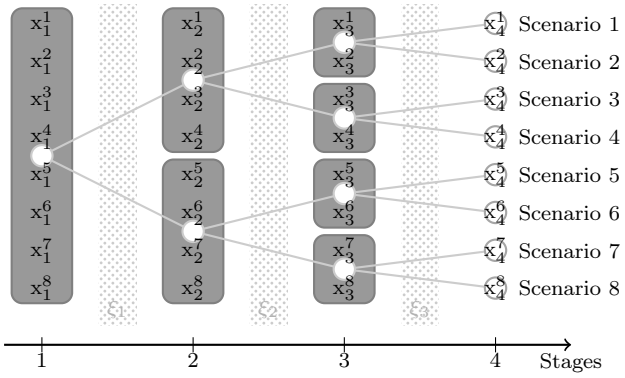

Fig. 2: Structure of the non-anticipativity constraints corresponding to the 4- sstage stochastic problem depicted in Fig. 1. All variables in a dark gray rectangle have to be equal.

For each scenario $s \in \{1, \ldots, S\}$, let us denote by $f^s(\mathrm{x}^s) = f(\mathrm{x}^s, \xi^s)$ the cost of the decision x^s. To simplify notation, we consider that possible constraints are incorporated

in the cost: minimizing a function $\tilde{f}^s$ over a constraint set $\mathcal{C}^s$ is the same as minimizing $f^s = \tilde{f}^s + \iota_{\mathcal{C}^s}$ over the full space (with the indicator function $\iota_{\mathcal{C}^s}$ defined by $\iota_{\mathcal{C}^s}(x) = 0$ if $x \in \mathcal{C}^s$ and $+\infty$ elsewhere). We consider such a constrained problem in our numerical experiments in Section 6.

We consider the "risk-neutral case" where $\mathcal{R}$ is the expectation of the random cost $f^s(\mathrm{x}^s)$ (a "risk-averse" case is discussed later in Remark 2.1). In this setting, Problem (2.1) rewrites as

$$\min_{\mathrm{x} \in \mathcal{W}} \quad \sum_{s=1}^{S} p_s f^s(\mathrm{x}^s), \tag{2.4}$$

which will be our target problem in this paper. The difficulty of this problem comes from the fact that there is an exponentially growing number of scenarios (e.g. in a binary tree, $S = 2^{T-1}$), all linked by the non-anticipativity constraints.

We finish presenting the set-up by formalizing our blanket assumptions on (2.4).

Assumption 1. *The scenario probabilities are positive ($p_s > 0$); the functions f^s are convex, proper, and lower-semicontinuous; and there exists a solution to* (2.4).

Assumption 2. *The subspace $\mathcal{W}$ defined in* (2.3) *intersects the relative interior of the domain of the objective function.*

The convexity in Assumption 1 is used for the convergence analysis (see e.g. [19, Chap. 3]). The technical assumption 2 is the standard non-degeneracy assumption in the multi-stage stochastic programming (see e.g. (9.17) in [19, Chap. 3]) which enables the splitting between scenarios and constraints.

Remark 2.1 (Risk-averse variant) Though we consider in (2.4) a risk-neutral model, this formulation naturally extends to risk-averse models, for which "worst" scenarios are particularly important to take into account. A popular risk-averse measure is the so-called Conditional Value at risk or CVar (see e.g. [16]). Following the idea of [15], the risk-averse problem

$$\min_{\mathrm{x} \in \mathcal{W}} \ \mathrm{CVar}_p(f(\mathrm{x}(\xi), \xi))$$

can be cast in the same form as (2.4). □

3 Progressive Hedging: algorithm and sequential randomization

This section presents an efficient randomization of Progressive Hedging for solving the multi-stage stochastic problem (2.4). We start with recalling the usual Progressive Hedging algorithm and discussing its practical implementation. Then, we propose a *sequential* randomized variant, which is a single-thread method, as the standard Progressive Hedging, but with cheap iterations.

This variant, as well as the other upcoming methods of the next section, is based on the operator view of Progressive Hedging (see e.g. the textbook [19, Chap. 3.9]). More precisely, Progressive Hedging corresponds to the Douglas-Rachford splitting on the subgradient of the dual problem, much like the Alternating Direction Method of Multipliers (ADMM); see [12]. We refer to [9] for a formal link between Douglas-Rachford and ADMM, and to [19, Chap. 3.9] for a formal link between Douglas-Rachford and Progressive Hedging. However, no knowledge on fixed-point theory is required to read this section; we postpone the derivation of the algorithms and the proofs of the theorems in Appendix A.

3.1 Progressive Hedging

Progressive Hedging is a popular decomposition method for solving (2.4) by decoupling the objective (separable among the scenarios) and the constraints (linking scenarios). The method alternates between two steps: i) solving S subproblems (one for each scenario, corresponding to the minimizing f^s plus a quadratic function) independently; ii) projecting onto the non-anticipativity constraint. In order to properly define this second step, it is convenient to define the *bundle* $\mathcal{B}_t^s$ of the scenarios that are indistinguishable from scenario s at time t, i.e.

$$\begin{aligned}\mathcal{B}_t^s &= \left\{\sigma \in \{1,\dots,S\} : \xi^s_{[1,t-1]} = \xi^\sigma_{[1,t-1]}\right\} \\ &= \left\{\sigma \in \{1,\dots,S\} : \mathrm{x} \in \mathcal{W} \Rightarrow \mathrm{x}_t^s = \mathrm{x}_t^\sigma\right\} \qquad \text{(see (2.3))}\ .\end{aligned}$$

Projecting onto the non-anticipativity constraints decomposes by stage and scenario, as an average over the corresponding bundle weighted by the scenarios probabilities.

Algorithm 1 Progressive Hedging

Initialize: $\mathrm{x}^0 \in \mathcal{W}, \mathrm{u}^0 \in \mathcal{W}^\perp, \mu > 0$

For $k = 0, 1, \dots$ **do:**

$$\begin{cases} \mathrm{y}^{k+1,s} = \operatorname{argmin}_{y\in\mathbb{R}^n} \left\{ f^s(y) + \frac{1}{2\mu} \left\| y - \mathrm{x}^{k,s} + \mu\, \mathrm{u}^{k,s} \right\|^2 \right\} & \text{for all } s = 1,\dots,S \\ \mathrm{x}_t^{k+1,s} = \frac{1}{\sum_{\sigma\in\mathcal{B}_t^s} p_\sigma} \sum_{\sigma\in\mathcal{B}_t^s} p_\sigma\, \mathrm{y}_t^{k+1,\sigma} & \text{for all } s = 1,\dots,S \text{ and } t = 1,\dots,T \\ \mathrm{u}^{k+1} = \mathrm{u}^k + \frac{1}{\mu}(\mathrm{y}^{k+1} - \mathrm{x}^{k+1}) \end{cases}$$

Return: x^k

Algorithm 1 presents the Progressive Hedging algorithm; its derivation from the reformulation of (2.4) as fixed-point problem is recalled in Appendix A. This appendix also explains how the convergence of the algorithm can be obtained as an application of existing results for fixed-point algorithms. Here we only formalize the convergence result, and discuss further some implementation details.

Theorem 3.1 *Consider the multistage problem* (2.4) *verifying Assumption 1 and 2. Then, the sequence* (x^k) *generated by Algorithm 1 is feasible (*$\mathrm{x}^k \in \mathcal{W}$ *for all* k*) and converges to an optimal solution of* (2.4).

The costly operation in Algorithm 1 is the update of the variable y which consists in a "proximal" operation for *all* scenarios. In general, there is no closed form expression for this operation, and thus it has to be obtained by a nonlinear optimization solver[1]. For instance, when the scenario costs (f^s) are (constrained) linear or quadratic functions, this operation amounts to solving S (constrained) quadratic programs.

The update of the variable x consists in a projection onto the non-anticipativity constraints $\mathcal{W}$. Though it is rather cheap to compute, it involves the variables (y^s) of all the scenarios. Thus, the update of y has to be completely executed before updating x,

[1] In our toolbox, we solve these problems with IPOPT [23], an open source software package for nonlinear optimization.

resulting in a potential computational bottleneck. Our upcoming randomized variant is aimed at alleviating this practical drawback.

Finally, concerning the initialization of the method, $\mathrm{u}^0 \in \mathcal{W}^\perp$ is primordial for convergence, and $\mathrm{u}^0 = 0$ is a safe choice. The hyperparameter $\mu > 0$ (also present in ADMM) controls the relative decrease of the primal and dual error; for the specific structure of Problem (2.4), $\mu = 1$ seem to be an acceptable choice in most situations.

Remark 3.1 (About leaf nodes) The *leaf nodes* correspond to the variables x_T^s i.e. the decisions at the last stage T for all the scenarios (see Fig. 2). These variables are particular in the optimization problem since they are not linked by the non-anticipativity constraints. This implies that the scenario bundle for scenario s at the last stage T is reduced to the singleton $\mathcal{B}_T^s = \{s\}$. Hence, the weighted average in the update of x in Algorithm 1 reduces to $\mathrm{x}_T^{k+1,s} = \mathrm{y}_T^{k+1,s}$, and as a consequence the dual variable stays unchanged $\mathrm{u}_T^k = \mathrm{u}_T^0$. This modification was present in the original Progressive Hedging algorithm by Rockafellar and Wets [17] since it allows to reduce the storage cost by getting rid of the dual variable corresponding to leaf nodes. This does not hold anymore for randomized versions that we present in this paper. So we choose not to display this modification for simplicity and better compliance with the textbook [19]. □

3.2 Randomized Progressive Hedging

Using the interpretation of Progressive Hedging as a fixed-point algorithm (detailed in appendix A) and results on randomized "coordinate descent" fixed-point methods (see e.g. [11]), we obtain a randomized version of Progressive Hedging. This randomized method consists in updating only a subset of the coordinates at each iteration, corresponding to only *one* scenario, and leaving the other unchanged. By doing so, each iterations is much less demanding computationally than one of Progressive Hedging (roughly S time quicker) since it involves the resolution of one sub-problem compared to S. However, as commonly observed with randomized optimization methods, Algorithm 2 will take more iterations to converge but usually less than S times more due to the progressive improvement brought by each scenario information. Thus, the Randomized Progressive Hedging should in general outperform the Progressive Hedging computationally, with the other advantage that many more iterations are produced per time which can be very useful in practice. Deriving such a method requires a special care as the operator links the variables with each other; these derivations are provided in Appendix B.

At iteration k, our Randomized Progressive Hedging (Algorithm 2) samples one scenario s^k (randomly among all with probabilities (q_s)) and then alternates between the projection over the non-anticipativity constraints $\mathcal{W}$ *associated with* s^k (the full projection on $\mathcal{W}$ is not necessary[2]) and the "proximal" operation over the selected scenario s^k, together with an update of the main variable z. Since a single scenario is involved in the iteration, this algorithm is naturally adapted to single-thread implementations and its incremental nature makes it computationally more efficient than the Progressive Hedging, to almost no additional implementation complications.

Finally, notice that since only the partial projection on the non-anticipativity constraints related to this scenario is needed to perform an iteration, the sequence (x^k),

[2] The full projection can be performed anyway but the variables that are not associated with s^k will not be taken into account by the algorithm anyhow.

Algorithm 2 Randomized Progressive Hedging

Initialize: $\mathrm{z}^0 = \mathrm{x}^0 \in \mathcal{W}, \mu > 0$

For $k = 0, 1, \ldots$ **do:**

Draw a scenario $s^k \in \{1, \ldots, S\}$ with probability $\mathbb{P}[s^k = s] = q_s$

$$\mathrm{x}_t^{k+1,s^k} = \frac{1}{\sum_{\sigma \in \mathcal{B}_t^{s^k}} p_\sigma} \sum_{\sigma \in \mathcal{B}_t^{s^k}} p_\sigma \, \mathrm{z}_t^{k,\sigma} \quad \text{for all } t = 1, \ldots, T \qquad \text{projection only on constraints involving } s^k$$

$$\mathrm{y}^{k+1,s^k} = \operatorname{argmin}_{y \in \mathbb{R}^n} \left\{ f^{s^k}(y) + \tfrac{1}{2\mu} \left\| y - 2\,\mathrm{x}^{k+1,s^k} + \mathrm{z}^{k,s^k} \right\|^2 \right\} \qquad \text{optimization sub-problem only concerning scenario } s^k$$

$$\begin{vmatrix} \mathrm{z}^{k+1,s^k} = \mathrm{z}^{k,s^k} + \mathrm{y}^{k+1,s^k} - \mathrm{x}^{k+1,s^k} \\ \mathrm{z}^{k+1,s} = \mathrm{z}^{k,s} \quad \text{for all } s \neq s^k \end{vmatrix}$$

Return: $\tilde{\mathrm{x}}^{k+1} = \dfrac{1}{\sum_{\sigma \in \mathcal{B}_t^s} p_\sigma} \displaystyle\sum_{\sigma \in \mathcal{B}_t^s} p_\sigma \, \mathrm{z}_t^{k+1,\sigma}$ for all $s = 1, \ldots, S$ and $t = 1, \ldots, T$

although converging to the sought solution, does not verify the non-anticipativity constraints. That is why the output of the algorithm has to be eventually projected onto the (full) non-anticipativity constraints (which introduces variable $\tilde{\mathrm{x}}^k$).

The convergence of this randomized version is formalized by Theorem 3.2 and proved in Appendix B.

Theorem 3.2 *Consider a multistage problem* (2.4) *verifying Assumptions 1 and 2. Then, the sequence* $(\tilde{\mathrm{x}}^k)$ *generated by Algorithm 2 is feasible (*$\tilde{\mathrm{x}}^k \in \mathcal{W}$ *a.s. for all* k*) and converges almost surely to a solution of* (2.4).

In practice, the initialization and parameters are similar to those of Progressive Hedging to the exception of the probabilities (q_s). Two natural choices come to mind:

- *uniform sampling*: taking the same probability $q_s = 1/S$ for all scenarios;
- *p sampling*: taking $q_s = p_s$ and thus sampling more the scenarios with a greater weight in the objective.

Finally, in terms of implementation, this algorithm is by nature *sequential* in the sense that one scenario is sampled, treated, and then incorporated in the master variable. Thus, it suits well single thread setups but is not directly able to benefit from multiple workers. Such an extension is the goal of the next section.

4 Parallel variants of Progressive Hedging

In this section, we discuss the deployment of (variants of) Progressive Hedging algorithms on parallel computing systems. No particular knowledge about distributed computing is required. We consider a generic master-worker framework where M workers collaboratively solve (2.4) under the orchestration of a master. This setting encompasses diverse practical situations such as multiple threads in a machine or multiple machines in a computing cluster (workers can then be threads, machines, agents, oracles, etc.). Our aim is to provide parallel methods that speed-up the resolution of medium-to-large

multi-stage stochastic programs by using a distributed computing system. Fully scalable implementations are problem/system dependent; instead, we take a higher level of abstraction and consider that a worker is a computing procedure that is able to solve any given subproblem. In practice, the algorithms implemented in our toolbox do not need to know the computing system, as they automatically adapt the underlying computing system, thanks to parallelization abilities of the Julia language.

4.1 Parallel Progressive Hedging

The randomized method presented in Section 3.2 is based on the sampling of one scenario per iteration. Using the same reasoning, it is possible to produce an algorithm sampling $M \leq S$ scenarios per iteration. These M scenarios can then be treated in parallel by M workers and then sent to the master for incorporation in the master variable. This algorithm, completely equivalent to the Randomized Progressive Hedging (Algorithm 2) can be formulated in a master-worker setup as follows.

Algorithm 3 Parallel Randomized Progressive Hedging

Master

Initialize: $\mathrm{x}^0 = \mathrm{z}^0 \in \mathcal{W}, \mu > 0$

For $k = 0, 1, \ldots$ **do:**

- Draw M scenarios $(s[1], .., s[M]) \in \{1, \ldots, S\}^M$ with probability $\mathbb{P}[s[i] = s] = q_s$
- $\mathrm{x}_t^{k+1,s} = \frac{1}{\sum_{\sigma \in \mathcal{B}_t^s} p_\sigma} \sum_{\sigma \in \mathcal{B}_t^s} p_\sigma \, \mathrm{z}_t^{k,\sigma}$ for all $t = 1, \ldots, T$ and $s \in (s[1], .., s[M])$
- Send a scenario/point pair $(s[i], 2\,\mathrm{x}^{k+1,s[i]} - \mathrm{z}^{k,s[i]})$ to *each* worker $i = 1, .., M$
- Receive $\mathrm{y}^{s[i]}$ from *all* workers $i = 1, .., M$
  - $\mathrm{z}^{k+1,s} = \mathrm{z}^{k,s} + \mathrm{y}^s - \mathrm{x}^{k+1,s}$ for all $s \in (s[1], .., s[M])$
  - $\mathrm{z}^{k+1,s} = \mathrm{z}^{k,s}$ for all $s \notin (s[1], .., s[M])$

Return: $\bar{\mathrm{x}}^{k+1} = \dfrac{1}{\sum_{\sigma \in \mathcal{B}_t^s} p_\sigma} \displaystyle\sum_{\sigma \in \mathcal{B}_t^s} p_\sigma \mathrm{z}_t^{k+1,\sigma}$ for all $s = 1, \ldots, S$ and $t = 1, \ldots, T$

Worker i

As soon as a scenario/point pair is received:

- Receive scenario/point pair $(s[i], \mathrm{v}[i])$
- $\mathrm{y}^{s[i]} = \mathrm{argmin}_{y \in \mathbb{R}^n} \left\{ f^{s[i]}(y) + \frac{1}{2\mu} \|y - \mathrm{v}[i]\|^2 \right\}$
- Send $\mathrm{y}^{s[i]}$ to the Master

This algorithm presents a simple, yet rather efficient, parallel method to solve multistage stochastic problems based on Progressive Hedging. When the difficulty of the subproblems is highly variable (due to different sizes, data, or initialization), the Progressive Hedging has to wait for the slowest subproblem to be solved, in order to complete an iteration. This is not the case anymore for the parallel randomized variant. However, if the *computing system* is heterogeneous, the parallel version still has to wait for the slowest worker, and thus workers may eventually have idle times. This drawback occurring for heterogeneous setups will be alleviated in the next section by our asynchronous variant.

4.2 Asynchronous Randomized Progressive Hedging

In a parallel computing framework, the Parallel Randomized Progressive Hedging of the previous section can be further extended to generate asynchronous iterations (built on a slightly different randomized fixed-point method [13], as detailed in Appendix C).

The resulting asynchronous Progressive Hedging (Algorithm 4) consists of the same steps per iteration as Algorithm 2, but these steps are performed asynchronously by several workers in parallel. In this case, each of the workers asynchronously receives a global variable, computes an update associated with one randomly drawn scenario, then incorporates it to the master variable[3].

Algorithm 4 Asynchronous Randomized Progressive Hedging

Master

Initialize: $\mathrm{x}^0 = \mathrm{z}^0 \in \mathcal{W}, \mu > 0, k = 0,$

$\hat{\mathrm{x}}[j] = \mathrm{x}^{0,j}$ and $s[j] = j$ for every worker i

Send the scenario/point pair $(s[j], \mathrm{x}[j])$ to every worker j

As soon as a worker finishes its computation:

Receive $\mathrm{y}^{s[i]}$ from an worker, say i

$\mathrm{z}^{k+1,s[i]} = \mathrm{z}^{k,s[i]} + \frac{2\eta^k}{S q_{s[i]}} \left(\mathrm{y}^{s[i]} - \hat{\mathrm{x}}[i]\right)$

$\mathrm{z}^{k+1,s} = \mathrm{z}^{k,s}$ for all $s \neq s[i]$

Draw a new scenario for i : $s[i] \in \{1, \dots, S\}$ with probability $\mathbb{P}[s[i] = s] = q_s$

$\hat{\mathrm{x}}[i] = \frac{1}{\sum_{\sigma \in \mathcal{B}^{s[i]}} p_\sigma} \sum_{\sigma \in \mathcal{B}_t^{s[i]}} p_\sigma \, \mathrm{z}_t^{k+1,\sigma}$ for all $t = 1, \dots, T$

Send the scenario/point pair $(s[i], 2\,\hat{\mathrm{x}}[i] - \mathrm{z}^{k+1,s[i]})$ to worker i

$k \leftarrow k + 1$

Return: $\bar{\mathrm{x}}^{k+1} = \dfrac{1}{\sum_{\sigma \in \mathcal{B}_t^s} p_\sigma} \displaystyle\sum_{\sigma \in \mathcal{B}_t^s} p_\sigma \mathrm{z}_t^{k+1,\sigma}$ for all $s = 1, \dots, S$ and $t = 1, \dots, T$

Worker i

As soon as a scenario/point pair is received:

Receive scenario/point pair $(s[i], \mathrm{v}[i])$

$\mathrm{y}^{s[i]} = \operatorname{argmin}_{y \in \mathbb{R}^n} \left\{ f^{s[i]}(y) + \frac{1}{2\mu} \|y - \mathrm{v}[i]\|^2 \right\}$

Send $\mathrm{y}^{s[i]}$ to the Master

Multiple updates may have occurred between the time of reading and updating. We denote by $\hat{\mathrm{x}}[i]$ (without any time index to avoid confusion) the value of $\mathrm{x}^{k,s}$ lastly used for feeding worker i. When the master performs an update from the information of worker i, $\hat{\mathrm{x}}[i] = \mathrm{x}^{k-d_k,s[i]}$ where $s[i]$ is the scenario treated by worker i for that update and d^k if the number of updates between k and the last time worker i performed an update[4]. We assume here that this delay is uniformly bounded; this is a reasonable assumption for multi-core machines and computing clusters. This assumption allows to

[3] We assume consistent writing, i.e. reading and writing do not clash with each other, extensions to inconsistent reads is discussed in [13, Sec. 1.2]

[4] In Appendix C, following [13], we denote by $\hat{\mathrm{x}}^k = \mathrm{x}^{k-d_k}$ if worker i started its update at time $k - d_k$.

use the convergence analysis of [13] to establish the following convergence result. The proof of this result is given in Appendix C. The intuition behind the result is to use the maximal delay to take cautious stepsizes η_k, guaranteeing convergence of asynchronous updates.

Theorem 4.1 *Consider a multistage problem (2.4) verifying Assumption s 1 and 2. We assume furthermore that the delays are bounded: $d^k \leq \tau < \infty$ for all k. If we take the stepsize η^k as follows for some fixed $0 < c < 1$*

$$0 < \eta_{\min} \leq \eta^k \leq \frac{cSq_{\min}}{2\tau\sqrt{q_{\min}} + 1} \qquad \text{with } q_{\min} = \min_s q_s, \tag{4.1}$$

then, the sequence $(\tilde{\mathrm{x}}^k)$ generated by Algorithm 4 is feasible ($\tilde{\mathrm{x}}^k \in \mathcal{W}$ a.s. for all k) and converges almost surely to a random variable supported by the solution set of (2.4).

Remark 4.1 (Extensions) For sake of clarity, we reduce Algorithm 4 to its simplest formulation with essential ingredients for asynchronous computation with guaranteed convergence. Several extensions and heuristics could be added; among them:

- tuned η_k (we test the simple strategy $\eta_k = 1$ in our numerical experiments),
- adaptive μ (scenario or iteration-wise),
- sending multiple scenario/point pairs (instead of only one) to the updating worker.

□

Remark 4.2 (Comparison with the other existing asynchronous variant)
The preprint [10] proposes another asynchronous variant of Progressive Hedging. This algorithm obviously shares common points with Algorithm 4 but has fundamental differences. The most striking one lies in the primal-dual update: at each iteration we update the primal-dual variable z^k only for the current scenario $s[i]$ while the asynchronous Progressive Hedging of [10] updates the full primal and dual variables. This comes from the fact that our algorithm is based on the asynchronous coordinate-descent method for operators of [13] while [10] is based on the asynchronous splitting method of [8]. Another practical difference is that we only use a partial projection related to the drawn scenario. □

5 The `RPH` toolbox

We release an open-source toolbox for modeling and solving multi-stage stochastic problems with the proposed Progressive Hedging variants. The toolbox is named `RPH` (for Randomized Progressive Hedging) and is implemented on top of `JuMP` [7] the popular framework for mathematical optimization, embedded in `Julia` language [2]. The source code, online documentation, and an interactive demonstration are available on the GitHub page of the project:

`https://github.com/yassine-laguel/RandomizedProgressiveHedging.jl`.

The toolbox `RPH` seems to implement the first publicly-available and theoretically-grounded variant of progressive hedging with randomized/asynchronous calls to the scenario subproblems. Related implementations include the theoretically-grounded one of [10] and the asynchronous heuristic of `ProgressiveHedgingSolvers.jl` publicly-available via the modelling framework `StochasticPrograms.jl` [4].

In this section, we only describe the basic usage of `RPH`; for more details, we refer to the Appendix D and the online material. Notably, we defer the problem modeling (for direct testing, we provide functions directly building toy problems such as `build_simpleexample`). Once a problem is instantiated, its resolution can be launched directly with the sequential Progressive Hedging or Randomized Progressive Hedging methods (Algorithms 1 and 2):

```
using RPH

pb = build_simpleexample() ## Generation of a toy problem

y_PH = solve_progressiveHedging(pb) ## Solving with Progressive Hedging
println("Progressive Hedging output is: ", y_PH)

y_RPH = solve_randomized_sync(pb)  ## Solving with Randomized Progressive Hedging
println("Randomized Progressive Hedging output is:  ", y_RPH)
```

In a parallel environment, one can use the `Distributed` module of `Julia`[5] to setup some number of workers. This can be done either at launch time (e.g. as `Julia -p 8`) or within the `Julia` process (with commands `addprocs()`, `rmprocs()` and `procs()`). Once this is set, the proposed parallel algorithms (Algorithm 3 and Algorithm 4) automatically use all available workers:

```
using Distributed
addprocs(7); length(procs()) # Gives one master + 7 workers

y_par = solve_randomized_par(pb)   ## Solving with Parallel Randomized Progressive Hedging
println("Parallel Randomized Progressive Hedging output is: ", y_par)

y_async = solve_randomized_async(pb)  ## Solving with Asynchronous Randomized Progressive Hedging
println("Asynchronous solve output is: ", y_async)
```

The provided methods in `RPH` rely on three criteria for stopping:

- maximal computing time (default: one hour),
- maximal number of scenario subproblems solved (default: 10^6),
- residual norm (norm of differences of iterates) inferior to the mixed absolute/relative threshold $\varepsilon_{abs} + \varepsilon_{rel}\|z_k\|$, where z_k is the current iterate of the algorithm (default: $\varepsilon_{abs} = 10^{-8}$, $\varepsilon_{rel} = 10^{-4}$).

6 Numerical illustrations

This section presents numerical results obtained with our toolbox `RPH` on multistage stochastic problems. We illustrate the behavior of our methods on a small hydro-thermal scheduling problem. A complete experimental study on real-life problems or modern parallel computing systems is beyond the scope of this work. We release our toolbox to allow reproducibility of our results and to spark further research on these randomized method s.

[5] A detailed explanation of `Julia`'s parallelism is available at `Julia` documentation: https://docs.Julialang.org/en/v1/manual/parallel-computing/. By default, the created workers are on the same machine but can easily be put on a distant machine through an SSH channel.

6.1 A multistage stochastic problem inspired from energy optimization

We consider a simple convex problem modeling a problem of hydro thermal scheduling; it follows [14] and the FAST toolbox[6].

Assume that an energy company wishes to deliver electricity to its clients either produced by several dams or bought externally. The dams produce cheaper energy but can only store a limited amount of water. The randomness of the problem comes from the amount of rain filling the dams at each stage. Mathematically, at each stage $t \in \{1, \dots, T\}$, each dam $b \in \{1, \dots, B\}$ has a quantity $q_t^b \in \mathbb{R}_+$ of water. For each stage t, the company has to decide: i) for each dam b the quantity of water to convert to electricity $y_t^b \in \mathbb{R}_+$; and ii) the quantity of electricity to buy externally $e_t \in \mathbb{R}_+$. The decision variable at stage t thus writes $\mathrm{x}_t = (q_t, y_t, e_t) \in \mathbb{R}_+^B \times \mathbb{R}_+^B \times \mathbb{R}_+$.

At stage t, the random variable ξ^t represents the amount of water that arrived at each of the dams since stage $t-1$. Out of simplicity, ξ_t is equal to r_{dry} with probability p_{dry} or r_{wet} with probability $1 - p_{dry}$. This defines a binary scenario tree (see Fig. 1) leading to 2^{T-1} scenarios.

For a scenario s, i.e. a realization of the sequence of water arrivals $(\xi_1^s, \dots, \xi_{T-1}^s)$, the objective function writes as the sum $f^s = \tilde{f}^s + \iota_{\mathcal{C}^s}$ of the energy generation cost

$$\tilde{f}^s(\mathrm{x}) = \sum_{t=1}^{T} c_{H,t}^\top y_t + c_E e_t$$

and the indicator function of constraints

$$\mathcal{C}^s = \begin{cases} \sum_{b=1}^{B} y_t^b + e_t \geq D & \text{for all } t & \text{(demand is met at each stage)} \\ q_t^b = q_{t-1}^b - y_t^b + \xi_t^s & \text{for all } t \geq 2, b & \text{(evolution of the amount of water)} \\ q_1^b = W_1^b - y_1^b & \text{for all } b & \text{(init. amount of water per dam)} \\ q_t^b \leq W^b & \text{for all } t, b & \text{(max. amount of water per dam)} \end{cases}.$$

For a given scenario, minimizing this objective function amounts to solving a quadratic optimization problem. The variables and constants are summarized in Table 1.

M.S.P.	T	$\mathbb{N}$	number of stages
	S	$\mathbb{N}$	number of scenarios
	ξ	$\mathbb{R}_+^{T-1}$	amount of water brought by the rain since the previous stage
Constants	B	$\mathbb{N}$	number of dams
	c_H	$\mathbb{R}_+^{BT}$	vector of electricity production costs at the dams
	c_E	$\mathbb{R}_+$	cost of buying external electricity
	D	$\mathbb{R}_+$	electricity demand to satisfy at each stage
	W	$\mathbb{R}_+^B$	maximal amount of water at the dams
	W_1	$\mathbb{R}_+^B$	initial amount of water available at the dams
Variables	q	$\mathbb{R}_+^{BT}$	quantity of water at the dams for each stage (directly depends on y and ξ)
	y	$\mathbb{R}_+^{BT}$	amount of water to transform into electricity at the dams at each stage
	e	$\mathbb{R}_+^T$	amount of electricity to purchase at each stage
	x	$\mathbb{R}_+^n$	$\mathrm{x} = (q, y, e)$ and $n = (2B+1)T$

Table 1: Variables and constants for the hydroelectric problem

[6] https://stanford.edu/~lcambier/cgi-bin/fast/index.php

For our computational illustration, we generate randomly one instance of this problem, with $B = 20$ dams, $T = 6$ stages making $S = 2^5 = 32$ scenarios. The 32 quadratic subproblems (one associated to each scenario) are solved by the interior point solver of `Mosek`, with default parameters[7]. We also use the solver `Mosek` to compute the optimal solution with high precision in order to plot the suboptimality of the iterates generated along the run of the algorithms. Since the problem is not big, this computation is quick, of the order of a second.

6.2 Numerical Results

Our illustrative experiments compare the behavior of the different variants of Progressive Hedging implemented in `RPH`. We make two experiments to illustrate the interests of randomization and parallelization for Progressive Hedging: on a sequential/single-thread set-up and on a parallel setup.

We run our experiments on a laptop with an 8-core processor (Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz). For the parallel computation, one core plays the role of master, and the seven others are workers. On each core, solving the small-size quadratic subproblems with an efficient software is rather fast (average $0.02s$). This parallel computing system is thus simple, basic, and homogeneous. In order to reveal the special features of asynchronous algorithms in the experiments, we introduce a small artificial heterogeneity by adding a $0.1s$ waiting time to 4 scenarios. For each experiment, we run each algorithm 10 times and report the median value. In order to display the variability of randomized methods, we also shade the area corresponding to the first and third quartiles for each algorithm.

Sequential experiments In Figure 3, we compare the Progressive Hedging (Algorithm 1, see also remark 3.1) with the randomized variant (Algorithm 2 where we draw 20 scenarios per iteration) with both uniform sampling and p-sampling (see Section 3.2). We thus display the decrease of two quantities:

- the (unconstrained) suboptimality with respect to $\tilde{f} := \sum_{s=1}^{S} p^s \tilde{f}^s$

$$(\tilde{f}(\tilde{x}^k) - \tilde{f}(x^\star))/\tilde{f}(x^\star).$$

- the distance to feasibility, as the distance between $\tilde{\mathrm{x}}^s$ and $\mathcal{C}^s$ over all scenarios

$$\max_{s\in\{1,\dots,S\}} \|\,\mathrm{y}^{k,s} - \tilde{\mathrm{x}}^{k,s}\|;$$

  Note indeed that as for most splitting methods, iterates are asymptotically feasible; more precisely $(\tilde{\mathrm{x}}^k)$ is always in $\mathcal{W}$ but the individual scenario constraints $\mathcal{C}^s$ are verified only asymptotically.

For illustration purposes, we also provide the number of subproblems solved along time, and the steplength of the iterates sequence (i.e. the difference between two successive iterates).

Our first observation is that Progressive Hedging and randomized Progressive Hedging with uniform sampling perform similarly, with respect both to time and to number of subproblems solved (Figures 3(a) and 3(c) respectively). We also notice that

[7] In particular, the (primal) feasibility tolerance is 10^{-8}, and therefore this is the target level of tolerance for the experiments.

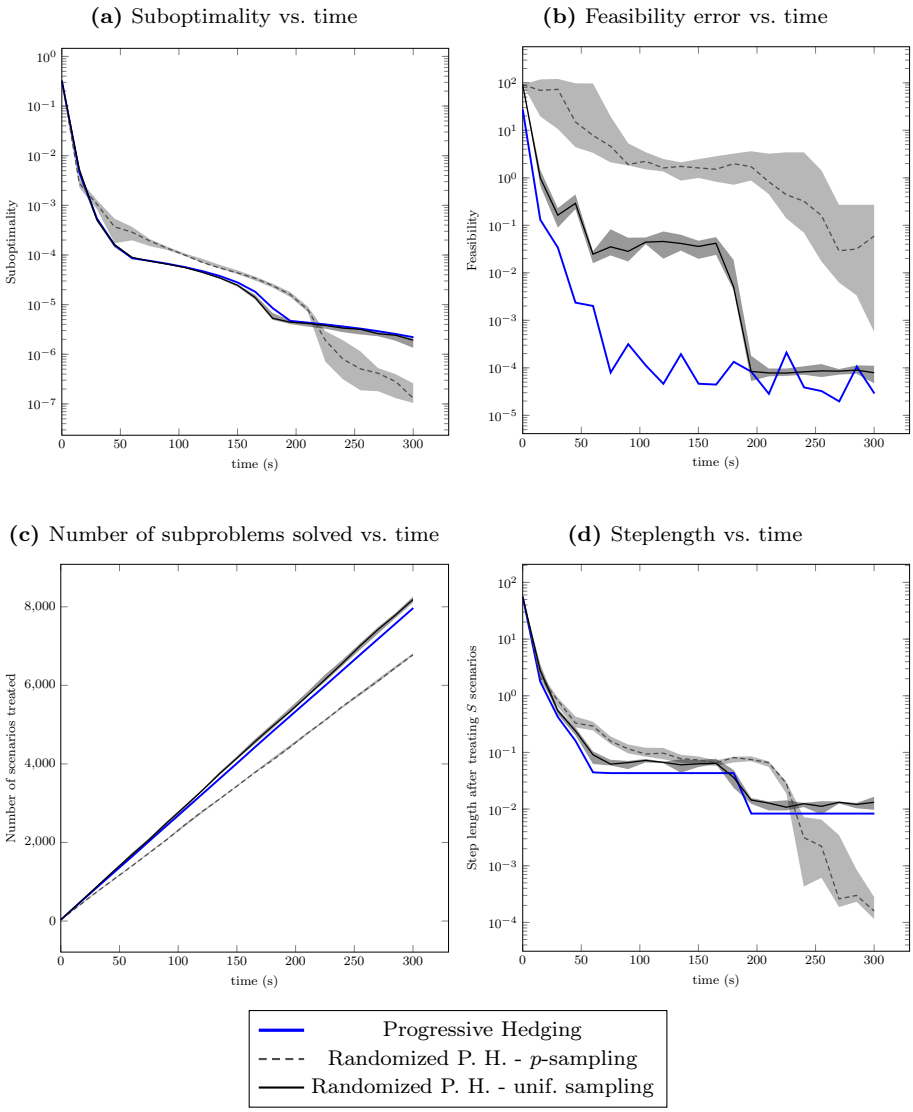

Fig. 3: Comparison of standard vs. randomized Progressive Hedging in a sequential set-up.

p-sampling variant gets to a lower suboptimality but with a larger feasibility gap (displayed on Figure 3(b)). This makes sense since scenarios that are prominent in the objective function are also the ones most often drawn and optimized. Conversely, more work needs to be invested on other scenarios to further reduce feasibility. An interest of the randomized variants is that they produce much more iterates compared to the base algorithm which requires one pass over all scenarios. This can be useful in setups where one iteration of Progressive Hedging is time-consuming.

Parallel experiments After adding seven workers[8], we compare in Figure 4 the Parallel Randomized Progressive Hedging (Algorithm 3) and the Asynchronous Progressive Hedging (Algorithm 4).

We see on Figure 4(c) that with 7 workers, Parallel Randomized Progressive Hedging is able to treat about 1.5 times as many scenarios as the sequential methods (randomized or not). Furthermore, the asynchronous variant lifts the communication bottleneck and is able to treat 4 times as many scenarios as the sequential.

We also see on Figure 4(a) that the Parallel Randomized method converts this higher scenario throughput into efficient iterates: the convergence is faster to the target precision 10^{-8} with a similar feasability gap (Figure 4(b)). Thus, this variant is a simple and efficient way to solve multistage problems on parallel setups.

A final remark from Figure 4(a) is that the theoretical stepsize of Theorem 4.1 (obtained by taking the maximum observed delay) is overly pessimistic, resulting in a slow algorithm. Taking a unit stepsize performs well for this instance. However, we observed in other setups that a unit stepsize may lead to non-convergence; in general, some tuning of this parameter is required for better performance.

Acknowledgments

The authors wish to thank the associate editor and the two anonymous reviewers for their valuable comments, notably with respect to the placement in the literature, which greatly improved the paper. F.I and J.M. thank Welington de Oliveira for fruitful discussions at the very beginning of this project.

References

1. Bauschke, H.H., Combettes, P.L.: Convex analysis and monotone operator theory in Hilbert spaces. Springer Science & Business Media (2011)
2. Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B.: Julia: A fresh approach to numerical computing. SIAM Review **59**(1), 65–98 (2017). DOI 10.1137/141000671
3. Bianchi, P., Hachem, W., Iutzeler, F.: A coordinate descent primal-dual algorithm and application to distributed asynchronous optimization. IEEE Transactions on Automatic Control **61**(10), 2947–2957 (2015)
4. Biel, M., Johansson, M.: Efficient stochastic programming in Julia. arXiv preprint arXiv:1909.10451 (2019)
5. Combettes, P.L., Pesquet, J.C.: Stochastic quasi-fejér block-coordinate fixed point iterations with random sweeping. SIAM Journal on Optimization **25**(2), 1221–1248 (2015)
6. De Silva, A., Abramson, D.: Computational experience with the parallel progressive hedging algorithm for stochastic linear programs. In: Proceedings of 1993 Parallel Computing and Transputers Conference Brisbane, pp. 164–174 (1993)
7. Dunning, I., Huchette, J., Lubin, M.: Jump: A modeling language for mathematical optimization. SIAM Review **59**(2), 295–320 (2017). DOI 10.1137/15M1020575
8. Eckstein, J.: A simplified form of block-iterative operator splitting and an asynchronous algorithm resembling the multi-block alternating direction method of multipliers. Journal of Optimization Theory and Applications **173**(1), 155–182 (2017)
9. Eckstein, J., Bertsekas, D.P.: On the douglas—rachford splitting method and the proximal point algorithm for maximal monotone operators. Mathematical Programming **55**(1-3), 293–318 (1992)

[8] In parallel setups, the respective performance of parallel and asynchronous methods is highly variable. We report the experiments obtained on a rather well behaved setup (all workers are equal), still they reflect the general trend we observed.

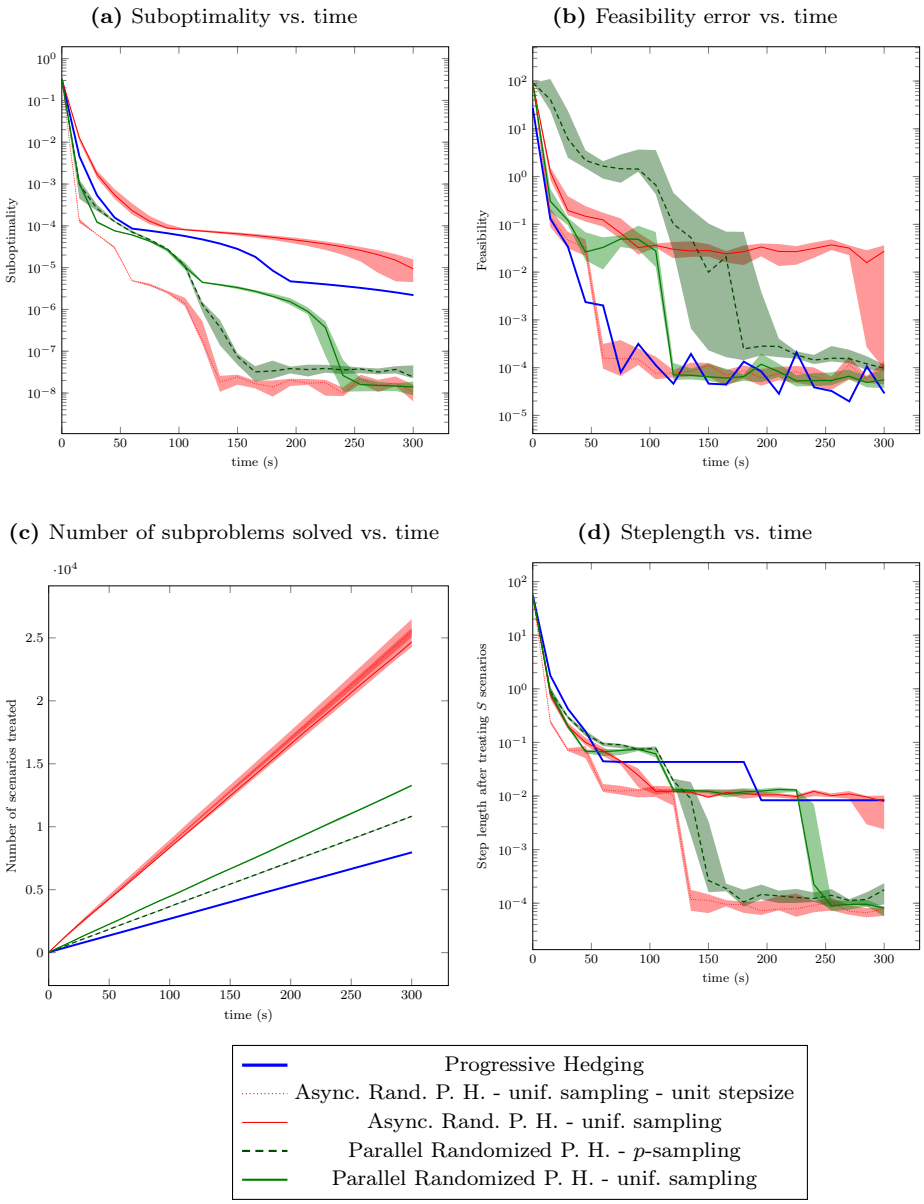

Fig. 4: Comparison of standard vs. randomized Progressive Hedging in a parallel set-up with 7 workers.

10. Eckstein, J., Watson, J.P., Woodruff, D.L.: Asynchronous projective hedging for stochastic programming (2018)
11. Iutzeler, F., Bianchi, P., Ciblat, P., Hachem, W.: Asynchronous distributed optimization using a randomized alternating direction method of multipliers. In: Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on, pp. 3671–3676. IEEE (2013)
12. Lions, P.L., Mercier, B.: Splitting algorithms for the sum of two nonlinear operators. SIAM Journal on Numerical Analysis **16**(6), 964–979 (1979)

13. Peng, Z., Xu, Y., Yan, M., Yin, W.: Arock: an algorithmic framework for asynchronous parallel coordinate updates. SIAM Journal on Scientific Computing **38**(5), A2851–A2879 (2016)
14. Pereira, M.V., Pinto, L.M.: Multi-stage stochastic optimization applied to energy planning. Mathematical programming **52**(1-3), 359–375 (1991)
15. Rockafellar, R.T.: Solving stochastic programming problems with risk measures by progressive hedging. Set-Valued and Variational Analysis **26**(4), 759–768 (2018)
16. Rockafellar, R.T., Royset, J.O.: Superquantile/cvar risk measures: Second-order theory. Annals of Operations Research **262**(1), 3–28 (2018)
17. Rockafellar, R.T., Wets, R.J.B.: Scenarios and policy aggregation in optimization under uncertainty. Mathematics of operations research **16**(1), 119–147 (1991)
18. Ruszczyński, A.: Decomposition methods in stochastic programming. Mathematical programming **79**(1-3), 333–353 (1997)
19. Ruszczyński, A., Shapiro, A.: Stochastic programming models. Handbooks in operations research and management science **10**, 1–64 (2003)
20. Ryan, S.M., Wets, R.J.B., Woodruff, D.L., Silva-Monroy, C., Watson, J.P.: Toward scalable, parallel progressive hedging for stochastic unit commitment. In: 2013 IEEE Power & Energy Society General Meeting, pp. 1–5. IEEE (2013)
21. Shapiro, A., Dentcheva, D., Ruszczyński, A.: Lectures on stochastic programming: modeling and theory. SIAM (2009)
22. Somervell, M.: Progressive hedging in parallel. Ph.D. thesis, Citeseer
23. Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Mathematical programming **106**(1), 25–57 (2006)
24. Watson, J.P., Woodruff, D.L.: Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems. Computational Management Science **8**(4), 355–370 (2011)

A Fixed-point view of Progressive Hedging

This appendix complements Section 3.1: we reformulate the multistage problem (2.4) as finding a fixed point of some operator (see the textbook [19, Chap. 3.9]). For all definitions and results on monotone operator theory, we refer to [1].

Denoting the objective function by $f(\mathrm{x}) := \sum_{s=1}^{S} p_s f^s(\mathrm{x})$ and the indicator of constraints by $\iota_{\mathcal{W}}$ with $\iota_{\mathcal{W}}(\mathrm{x}) = 0$ if $\mathrm{x} \in \mathcal{W}$ and $+\infty$ otherwise, we have that solving (2.4) amounts to finding $\mathrm{x}^\star$ such that

$$0 \in \partial(f + \iota_{\mathcal{W}})(\mathrm{x}^\star) = \partial f(\mathrm{x}^\star) + \partial \iota_{\mathcal{W}}(\mathrm{x}^\star)$$

where we use Assumption 2 for the equality. Then, we introduce the two following operators

$$\mathsf{A}(\mathrm{x}) := P^{-1}\partial f(\mathrm{x}) \quad \text{and} \quad \mathsf{B}(\mathrm{x}) := P^{-1}\partial \iota_{\mathcal{W}}(\mathrm{x}) \tag{A.1}$$

where $P = \mathrm{diag}(p_1, \dots, p_S)$. Using Assumption 1, the operators A and B defined in (A.1) are maximal monotone since so are the subdifferentials of convex proper lower-semicontinuous functions.

Solving (2.4) thus amounts to finding a zero of $\mathsf{A} + \mathsf{B}$ the sum of two maximal monotone operators:

$$\mathrm{x}^\star \text{ solves (2.4)} \iff \mathrm{x}^\star \text{ is a zero of } \mathsf{A} + \mathsf{B} \text{ i.e. } 0 \in \mathsf{A}(\mathrm{x}^\star) + \mathsf{B}(\mathrm{x}^\star). \tag{A.2}$$

We follow the notation of [19, Chap. 3] and the properties of [1, Chap. 4.1 and 23.1]. For a given maximal monotone operator M, we define for any $\mu > 0$ two associated operators:

i) the resolvent $\mathsf{J}_{\mu\mathsf{M}} = (I + \mu\mathsf{M})^{-1}$, (which is well-defined and firmly non-expansive),
ii) the reflected resolvent $\mathsf{O}_{\mu\mathsf{M}} = 2\mathsf{J}_{\mu\mathsf{M}} - I$ (which is non-expansive).

These operators allow us to formulate our multistage problem as a fixed-point problem: with the help of (A.2) and [1, Prop. 25.1(ii)], we have

$$\mathrm{x}^\star \text{ solves (2.4)} \iff \mathrm{x}^\star = \mathsf{J}_{\mu\mathsf{B}}(\mathrm{z}^\star) \text{ with } \mathrm{z}^\star \text{ a fixed point of } \mathsf{O}_{\mu\mathsf{A}} \circ \mathsf{O}_{\mu\mathsf{B}}, \text{ i.e. } \mathrm{z}^\star = \mathsf{O}_{\mu\mathsf{A}} \circ \mathsf{O}_{\mu\mathsf{B}}(\mathrm{z}^\star).$$

We can apply now a fixed-point algorithm to the firmly non-expansive operator[9] $\frac{1}{2}\mathsf{O}_{\mu\mathsf{A}} \circ \mathsf{O}_{\mu\mathsf{B}} + \frac{1}{2}\,\mathrm{Id}$ to find a fixed point of $\mathsf{O}_{\mu\mathsf{A}} \circ \mathsf{O}_{\mu\mathsf{B}}$.

This gives the following iteration (equivalent to Douglas-Rachford splitting)

$$\mathrm{z}^{k+1} = \frac{1}{2}\mathsf{O}_{\mu\mathsf{A}}(\mathsf{O}_{\mu\mathsf{B}}(\mathrm{z}^k)) + \frac{1}{2}\,\mathrm{z}^k \tag{A.3}$$

which converges to a point $\mathrm{z}^\star$ such that $x^\star := \mathsf{J}_{\mu\mathsf{B}}(\mathrm{z}^\star)$ is a zero of $\mathsf{A}+\mathsf{B}$; see [1, Chap. 25.2].

It is well-known (see e.g. the textbook [19, Chap. 3, Fig. 10]) that this algorithm with the operators A and B defined in (A.1) leads to the Progressive Hedging algorithm. We give here a short proof of this property; along the way, we introduce basic properties and arguments used in the new developments on randomized Progressive Hedging of the next two appendices. We provide first the expressions of the reflected resolvent operators for A and B.

Lemma 1 (Operators associated with Progressive Hedging) *Let endow the space $\mathbb{R}^{S\times n}$ of $S\times n$ real matrices with the weighted inner product $\langle A,B\rangle_P = \mathrm{Trace}(A^{\mathrm{T}}PB)$. Then the operators A and B defined in (A.1) are maximal monotone, and their reflected resolvent operators $\mathsf{O}_{\mu\mathsf{A}}$ and $\mathsf{O}_{\mu\mathsf{B}}$ have the following expressions:*

i) $\mathsf{O}_{\mu\mathsf{A}}(\mathrm{z}) = \mathrm{x} - \mu\,\mathrm{u}$ *with*

$$\mathrm{x}^s = \operatorname*{argmin}_{y\in\mathbb{R}^n}\left\{ f^s(y) + \frac{1}{2\mu}\,\|y-\mathrm{z}^s\|^2 \right\} \text{ for all } s=1,\dots,S$$

and $\mathrm{u} = (\mathrm{z}-\mathrm{x})/\mu$ *(hence* $\mathsf{O}_{\mu\mathsf{A}}(\mathrm{z}) = 2\,\mathrm{x}-\mathrm{z}$*);*

ii) $\mathsf{O}_{\mu\mathsf{B}}(\mathrm{z}) = \mathrm{x} - \mu\,\mathrm{u}$ *with*

$$\mathrm{x}_t^s = \frac{1}{\sum_{\sigma\in\mathcal{B}_t^s} p_\sigma} \sum_{\sigma\in\mathcal{B}_t^s} p_\sigma\,\mathrm{z}_t^\sigma \quad \text{for all } s=1,\dots,S \text{ and } t=1,\dots,T$$

and $\mathrm{u} = (\mathrm{z}-\mathrm{x})/\mu$ *(hence* $\mathsf{O}_{\mu\mathsf{B}}(\mathrm{z}) = 2\,\mathrm{x}-\mathrm{z}$*). The point* x *is the* orthogonal projection *of* z *to $\mathcal{W}$. Thus,* z *writes uniquely as* $\mathrm{z} = \mathrm{x} + \mu\,\mathrm{u}$ *with* $\mathrm{x}\in\mathcal{W}$ *and* $\mathrm{u}\in\mathcal{W}^\perp$.

Proof. Since $\partial f(\cdot)$ and $\partial\iota_{\mathcal{W}}(\cdot)$ are the subdifferentials of convex proper lower-semicontinuous functions, they are maximal monotone with respect to the usual inner product, and there so are A and B, with respect to the weighted inner product.

Applying [1, Prop. 23.1] to a maximal monotone operator M, we get that $\mathrm{z}\in\mathbb{R}^{S\times n}$ can be uniquely represented as $\mathrm{z} = \mathrm{x} + \mu\,\mathrm{u}$ with $\mathrm{u}\in\mathsf{M}(\mathrm{x})$, thus $\mathsf{J}_{\mu\mathsf{M}}(\mathrm{z}) = \mathrm{x}$ and $\mathsf{O}_{\mu\mathsf{M}}(\mathrm{z}) = \mathsf{O}_{\mu\mathsf{M}}(\mathrm{x}+\mu\,\mathrm{u}) = \mathrm{x}-\mu\,\mathrm{u}$. This gives the expressions for $\mathsf{O}_{\mu\mathsf{A}}$ and $\mathsf{O}_{\mu\mathsf{B}}$ from the expressions of $\mathsf{J}_{\mu\mathsf{A}}$ and $\mathsf{J}_{\mu\mathsf{B}}$ based on the proximity operators associated with f and $\iota_{\mathcal{W}}$ (see [1, Prop. 16.34]). □

We now apply the general Douglas-Rachford scheme (A.3) with the expressions obtained in Lemma 1. We first get:

$$\begin{cases} \mathrm{x}_t^{k,s} = \frac{1}{\sum_{\sigma\in\mathcal{B}_t^s} p_\sigma} \sum_{\sigma\in\mathcal{B}_t^s} p_\sigma\,\mathrm{z}_t^{k,\sigma} \text{ for all } s=1,\dots,S \text{ and } t=1,\dots,T & \mathrm{x}^k\in\mathcal{W} \\ \mathrm{w}^k = \mathsf{O}_{\mu\mathsf{B}}(\mathrm{z}^k) = 2\,\mathrm{x}^k - \mathrm{z}^k = \mathrm{x}^k - \mu\,\mathrm{u}^k & \text{with } \mathrm{u}^k = (\mathrm{z}^k-\mathrm{x}^k)/\mu \in \mathcal{W}^\perp \\ & \text{thus } \mathrm{u}^k = \mathrm{u}^{k-1} + \frac{1}{\mu}(\mathrm{y}^k - \mathrm{x}^k) \\ \mathrm{y}^{k+1,s} = \operatorname{argmin}_{y\in\mathbb{R}^n}\left\{ f^s(y) + \frac{1}{2\mu}\,\|y-\mathrm{w}^{k,s}\|^2 \right\} \text{ for all } s=1,\dots,S & \\ \mathrm{z}^{k+1} = \frac{1}{2}(2\,\mathrm{y}^{k+1} - \mathrm{w}^k) + \frac{1}{2}\,\mathrm{z}^k = \mathrm{z}^k + \mathrm{y}^{k+1} - \mathrm{x}^{k+1} = \mathrm{y}^{k+1} + \mu\,\mathrm{u}^k & \end{cases}$$

Let us reorganize the equations and eliminate intermediate variables. In particular, we use the fact that, provided that the algorithm is initialized with $\mathrm{x}^0\in\mathcal{W}$ and $\mathrm{u}^0\in\mathcal{W}^\perp$, all iterates

[9] As $\mathsf{O}_{\mu\mathsf{A}}$ and $\mathsf{O}_{\mu\mathsf{B}}$ are non-expansive but not firmly non-expansive, it is necessary to average them with the current iterate (this is often called the Krasnosel'skiĭ–Mann algorithm [1, Chap 5.2]) to make this iteration firmly non-expansive and ensure Fejér monotone convergence.

(x^k) and (u^k) are in $\mathcal{W}$ and $\mathcal{W}^\perp$ respectively. We eventually obtain:

$$\begin{cases} \mathrm{y}^{k+1,s} = \operatorname{argmin}_{y\in\mathbb{R}^n} \left\{ f^s(y) + \frac{1}{2\mu} \left\| y - \mathrm{x}^{k,s} + \mu\, \mathrm{u}^{k,s} \right\|^2 \right\} \text{ for all } s = 1,\dots,S \\ \mathrm{x}_t^{k+1,s} = \frac{1}{\sum_{\sigma\in\mathcal{B}_t^s} p_\sigma} \sum_{\sigma\in\mathcal{B}_t^s} p_\sigma\, \mathrm{y}_t^{k+1,\sigma} \quad \text{for all } s = 1,\dots,S \text{ and } t = 1,\dots,T \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{x}^k \in \mathcal{W} \text{ converges to a solution of (2.4)} \\ \mathrm{u}^{k+1} = \mathrm{u}^k + \frac{1}{\mu}(\mathrm{y}^{k+1} - \mathrm{x}^{k+1}) \end{cases}$$

This is exactly the Progressive Hedging algorithm, written with similar notation as in the textbook [19, Chap. 3, Fig. 10]. The convergence of the algorithm (recalled in Theorem 3.1)) can be obtained directly by instantiating the general convergence result of the Douglas-Rachford method [1, Chap. 25.2].

In the next two appendices, we are going to follow the same line that has brought us from Douglas-Rachford to Progressive Hedging, to go from *randomized* Douglas-Rachford to *randomized* Progressive Hedging, and from *asynchronous* Douglas-Rachford to *asynchronous* Progressive Hedging.

B Derivation and Proof of the Randomized Progressive Hedging

A randomized counterpart of the Douglas-Rachford method (A.3) consists in updating only part of the variable chosen at random; see [11] and extensions [3,5]. At each iteration, this variant amounts to update the variables corresponding to the chosen scenario s^k (randomly chosen with probability q_s^k), the other staying unchanged:

$$\begin{array}{l} \text{Draw a scenario } s^k \in \{1,\dots,S\} \text{ with probability } \mathbb{P}[s^k = s] = q_s \\ \left| \begin{array}{l} \mathrm{z}^{k+1,s^k} = \frac{1}{2}\left[\mathsf{O}_{\mu\mathsf{A}}(\mathsf{O}_{\mu\mathsf{B}}(\mathrm{z}^k))\right]^{s^k} + \frac{1}{2}\,\mathrm{z}^{k,s^k} \\ \mathrm{z}^{k+1,s} = \mathrm{z}^{k,s} \quad \text{for all } s \neq s^k \end{array} \right. \end{array} \tag{B.1}$$

Our goal is to obtain the Randomized Progressive Hedging (Algorithm 2) as an instantiation of (B.1) with the operators defined in Lemma 1 in Appendix A. Before proceeding with the derivation, let us prove the convergence of (B.1) with these operators.

Proposition 1 *Consider a multistage problem (2.4) verifying Assumptions 1 and 2. Then, the sequence (z^k) generated by (B.1) with $\mathsf{O}_{\mu\mathsf{A}}$ and $\mathsf{O}_{\mu\mathsf{B}}$ defined in Lemma 1 converges almost surely to a fixed point of $\mathsf{O}_{\mu\mathsf{A}} \circ \mathsf{O}_{\mu\mathsf{B}}$. Furthermore, $\tilde{\mathrm{x}}^k := \mathsf{J}_{\mu\mathsf{B}}(\mathrm{z}^k)$ converges to a solution of (2.4).*

Proof. First, recall from Lemma 1 that under assumptions 1 and 2, the operators A, B of (A.1) are maximal monotone. Then, the associated operators $\mathsf{O}_{\mu\mathsf{A}}$ and $\mathsf{O}_{\mu\mathsf{A}}$ are then non-expansive by construction (see [1, Chap. 4.1]), and therefore the iteration $\mathsf{T} = (\mathsf{O}_{\mu\mathsf{A}} \circ \mathsf{O}_{\mu\mathsf{B}} + I)/2$ is firmly non expansive. This is the key assumption to use the convergence result [11, Th. 2] which gives that the sequence (z^k) generated by (B.1) converges almost surely to a fixed point of $\mathsf{O}_{\mu\mathsf{A}} \circ \mathsf{O}_{\mu\mathsf{B}}$. Using the continuity of $\mathsf{J}_{\mu\mathsf{B}}$ and the fact that $x^\star := \mathsf{J}_{\mu\mathsf{B}}(\mathrm{z}^\star)$ is a zero of $\mathsf{A} + \mathsf{B}$ (i.e. solves the multi-stage problem (2.4) by (A.2)) gives the last part of the result. □

Now that the convergence of (B.1) with the operators of Appendix A has been proven, let us derive our Randomized Progressive Hedging (Algorithm 2) as an equivalent formulation of (B.1). By doing so, the associated convergence result (Theorem 3.2) directly follows from Proposition 1.

From the specific expressions of operators $\mathsf{O}_{\mu\mathsf{A}}$ and $\mathsf{O}_{\mu\mathsf{B}}$ (Lemma 1), we see that these operators are very different in nature:

- $\mathsf{O}_{\mu\mathsf{A}}$ *is separable by scenario* but involves *solving a subproblem*;
- $\mathsf{O}_{\mu\mathsf{B}}$ *links the scenarios* but only amounts to *computing a weighted average*.

To leverage this structure, we apply the randomized Douglas-Rachford method (B.1) and get:

$$\left\{\begin{array}{l}\text{Draw a scenario } s^k \in \{1,\dots,S\} \text{ with probability } \mathbb{P}[s^k = s] = q_s \\ \mathrm{x}_t^{k,s} = \frac{1}{\sum_{\sigma\in\mathcal{B}_t^s} p_\sigma} \sum_{\sigma\in\mathcal{B}_t^s} p_\sigma\, \mathrm{z}_t^{k,\sigma} \text{ for all } s = 1,\dots,S \text{ and } t = 1,\dots,T \qquad \mathrm{x}^k \in \mathcal{W} \\ \mathrm{w}^k = \mathsf{O}_{\mu\mathsf{B}}(\mathrm{z}^k) = 2\,\mathrm{x}^k - \mathrm{z}^k = \mathrm{x}^k - \mu\,\mathrm{u}^k \qquad \text{with } \mathrm{u}^k = (\mathrm{z}^k - \mathrm{x}^k)/\mu \in \mathcal{W}^\perp \\ \mathrm{y}^{k+1,s} = \operatorname{argmin}_{y\in\mathbb{R}^n} \left\{ f^s(y) + \frac{1}{2\mu}\left\|y - \mathrm{w}^{k,s}\right\|^2 \right\} \text{ for all } s = 1,\dots,S \\ \left|\begin{array}{l} \mathrm{z}^{k+1,s^k} = \frac{1}{2}(2\,\mathrm{y}^{k+1,s^k} - \mathrm{w}^{k,s^k}) + \frac{1}{2}\mathrm{z}^{k,s^k} = \mathrm{z}^{k,s^k} + \mathrm{y}^{k+1,s^k} - \mathrm{x}^{k+1,s^k} = \mathrm{y}^{k+1,s^k} + \mu\,\mathrm{u}^{k,s^k} \\ \mathrm{z}^{k+1,s} = \mathrm{z}^{k,s} \text{ for all } s \neq s^k \end{array}\right. \end{array}\right.$$

Let us carefully prune unnecessary computations. First, only y^{k+1,s^k} needs to be computed, so the other $\mathrm{y}^{k+1,s}$ ($s \neq s^k$) can be safely dropped. The same holds for x^{k,s^k}, w^{k,s^k}, and u^{k,s^k}. However, even though only x^{k,s^k} need to be computed, it depends on all the other scenarios through the projection operator, so the iterates have to be computed successively and with only a partial update of u^k (in contrast with Appendix A, u^k does not belong to $\mathcal{W}$ anymore and thus cannot be dropped out of the projection, thus we keep directly the global variable z^k updated):

$$\left\{\begin{array}{l}\text{Draw a scenario } s^k \in \{1,\dots,S\} \text{ with probability } \mathbb{P}[s^k = s] = q_s \\ \mathrm{x}_t^{k,s^k} = \frac{1}{\sum_{\sigma\in\mathcal{B}_t^{s^k}} p_\sigma} \sum_{\sigma\in\mathcal{B}_t^{s^k}} p_\sigma\, \mathrm{z}_t^{k,\sigma} \text{ for all } t = 1,\dots,T \\ \mathrm{w}^{k,s^k} = 2\,\mathrm{x}^{k,s^k} - \mathrm{z}^{k,s^k} \\ \mathrm{y}^{k+1,s^k} = \operatorname{argmin}_{y\in\mathbb{R}^n} \left\{ f^{s^k}(y) + \frac{1}{2\mu}\left\|y - \mathrm{w}^{k,s^k}\right\|^2 \right\} \\ \left|\begin{array}{l} \mathrm{z}^{k+1,s^k} = \mathrm{z}^{k,s^k} + \mathrm{y}^{k+1,s^k} - \mathrm{x}^{k+1,s^k} \\ \mathrm{z}^{k+1,s} = \mathrm{z}^{k,s} \text{ for all } s \neq s^k \end{array}\right. \end{array}\right.$$

Eliminating intermediate variable w, we obtain the randomized Progressive Hedging:

$$\left\{\begin{array}{l}\text{Draw a scenario } s^k \in \{1,\dots,S\} \text{ with probability } \mathbb{P}[s^k = s] = q_s \\ \mathrm{x}_t^{k+1,s^k} = \frac{1}{\sum_{\sigma\in\mathcal{B}_t^{s^k}} p_\sigma} \sum_{\sigma\in\mathcal{B}_t^{s^k}} p_\sigma\, \mathrm{z}_t^{k,\sigma} \text{ for all } t = 1,\dots,T \\ \mathrm{y}^{k+1,s^k} = \operatorname{argmin}_{y\in\mathbb{R}^n} \left\{ f^{s^k}(y) + \frac{1}{2\mu}\left\|y - 2\,\mathrm{x}^{k+1,s^k} + \mathrm{z}^{k,s^k}\right\|^2 \right\} \\ \left|\begin{array}{l} \mathrm{z}^{k+1,s^k} = \mathrm{z}^{k,s^k} + \mathrm{y}^{k+1,s^k} - \mathrm{x}^{k+1,s^k} \\ \mathrm{z}^{k+1,s} = \mathrm{z}^{k,s} \text{ for all } s \neq s^k \end{array}\right. \end{array}\right.$$

Finally, notice that from Proposition 1, that the variable converging to a solution of (2.4) is $\tilde{\mathrm{x}}^k := \mathsf{J}_{\mu\mathsf{B}}(\mathrm{z}^k)$. From Lemma 1 (and the fact that $\mathsf{O}_{\mu\mathsf{B}} = 2\mathsf{J}_{\mu\mathsf{B}} - \mathsf{I}$), we get that $\tilde{\mathrm{x}}_t^{k,s} = \frac{1}{\sum_{\sigma\in\mathcal{B}_t^s} p_\sigma} \sum_{\sigma\in\mathcal{B}_t^s} p_\sigma\, \mathrm{z}_t^{k,\sigma}$ for all $s = 1,\dots,S$ and $t = 1,\dots,T$ and that $\tilde{\mathrm{x}}^k \in \mathcal{W}$.

C Derivation and Proof of the Asynchronous Randomized Progressive Hedging

Using again the bridge between Progressive Hedging and fixed-point algorithms, we present here how to derive an asynchronuous progressive hedeging from the asynchronous parallel fixed-point algorithm ARock [13]. In order to match the notation and derivations of [13], let us define the operator $\mathsf{S} := I - \mathsf{O}_{\mu\mathsf{A}} \circ \mathsf{O}_{\mu\mathsf{B}}$, the zeros of which coincide with the fixed points of $\mathsf{O}_{\mu\mathsf{A}} \circ \mathsf{O}_{\mu\mathsf{B}}$. Applying ARock to this operator leads to the following iteration:

Every worker asynchronously do

$$\left\{\begin{array}{l}\text{Draw a scenario } s^k \in \{1,\dots,S\} \text{ with probability } \mathbb{P}[s^k = s] = q_s \\ \left|\begin{array}{l} \mathrm{z}^{k+1,s^k} = \mathrm{z}^{k,s^k} - \frac{\eta^k}{S p_{s^k}} \left(\hat{\mathrm{z}}^{k,s^k} - \left[\mathsf{O}_{\mu\mathsf{A}}(\mathsf{O}_{\mu\mathsf{B}}(\hat{\mathrm{z}}^k))\right]^{s^k} \right) \\ \mathrm{z}^{k+1,s} = \mathrm{z}^{k,s} \text{ for all } s \neq s^k \end{array}\right. \\ \text{where } \hat{\mathrm{z}}^k \text{ is the value of } \mathrm{z}^k \text{ used by the updating worker at time } k \text{ for its computation} \end{array}\right. \tag{C.1}$$

Notice that the main difference between this iteration and (B.1) is the introduction of the variable $\hat{z}^k$ which is used to handle delays between workers in asynchronous computations:

- If there is only one worker, it just computes its new point with the latest value so we simply have: $\hat{z}^k = z^k$. We notice that taking $\eta^k = Sp_{s^k}/2$, we recover exactly the randomized Douglas-Rachford method (B.1);
- If there are several workers, $\hat{z}^k$ is usually an older version of the main variable, as other workers may have updated the main variable during the computation of the updating worker. In this case, we have $\hat{z}^k = z^{k-d^k}$ where d^k is the delay suffered by the updating worker at time k.

We derive here our Asynchronous Randomized Progressive hedging (Algorithm 4) as an instantiation of (C.1) with the operators $\mathsf{O}_{\mu\mathsf{A}}$ and $\mathsf{O}_{\mu\mathsf{B}}$ defined in Appendix A. Let us establish first the convergence of this scheme using a general result of [13] which makes little assumptions on the communications between workers and master. The main requirement is that the maximum delay between workers is bounded, which is a reasonable assumption when the algorithm is run on a multi-core machine or on a medium-size computing cluster.

Proposition 2 *Consider a multistage problem (2.4) verifying Assumptions 1 and 2. We assume furthermore that the delays are bounded: $d^k \leq \tau < \infty$ for all k. If we take the stepsize η^k as follows for some fixed $0 < c < 1$*

$$0 < \eta_{\min} \leq \eta^k \leq \frac{cSq_{\min}}{2\tau\sqrt{q_{\min}} + 1} \qquad \text{with } q_{\min} = \min_s q_s. \tag{C.2}$$

Then, the sequence (z^k) generated by (C.1) with $\mathsf{O}_{\mu\mathsf{A}}$ and $\mathsf{O}_{\mu\mathsf{B}}$ defined in Lemma 1 converges almost surely to a fixed point of $\mathsf{O}_{\mu\mathsf{A}} \circ \mathsf{O}_{\mu\mathsf{B}}$. Furthermore, $\tilde{x}^k := \mathsf{J}_{\mu\mathsf{B}}(z^k)$ converges to a solution of (2.4).

Proof. The beginning of the proof follows the same lines as the one of Proposition 1 to show that $\mathsf{O}_{\mu\mathsf{A}}$ and $\mathsf{O}_{\mu\mathsf{A}}$ are non-expansive by construction, which implies that $\mathsf{S} := I - \mathsf{O}_{\mu\mathsf{A}} \circ \mathsf{O}_{\mu\mathsf{B}}$ is also non-expansive with its zeros corresponding to the fixed points of $\mathsf{O}_{\mu\mathsf{A}} \circ \mathsf{O}_{\mu\mathsf{B}}$ (see [1, Chap. 4.1]). We can then apply [13, Th. 3.7] to get that (z^k) converges almost surely to a zero of S. As in the proof of Proposition 1, we use the continuity of $\mathsf{J}_{\mu\mathsf{B}}$ and the fact that $x^\star := \mathsf{J}_{\mu\mathsf{B}}(z^\star)$ is a zero of $\mathsf{A} + \mathsf{B}$ (i.e. solves the multi-stage problem (2.4) by (A.2)) to get the last part of the result. □

Using the expressions of the operators of Lemma 1, (C.1) writes

Every worker asynchronously do

$$\begin{cases} \text{Draw a scenario } s^k \in \{1, \dots, S\} \text{ with probability } \mathbb{P}[s^k = s] = q_s \\ \hat{x}_t^{k,s} = \frac{1}{\sum_{\sigma \in \mathcal{B}_t^s} p_\sigma} \sum_{\sigma \in \mathcal{B}_t^s} p_\sigma \hat{z}_t^{k,\sigma} \quad \text{for all } s = 1, \dots, S \text{ and } t = 1, \dots, T \\ \hat{w}^k = \mathsf{O}_{\mu\mathsf{B}}(\hat{z}^k) = 2\hat{x}^k - \hat{z}^k \\ \hat{y}^{k+1,s} = \operatorname{argmin}_{y \in \mathbb{R}^n} \left\{ f^s(y) + \frac{1}{2\mu} \left\| y - \hat{w}^{k,s} \right\|^2 \right\} \quad \text{for all } s = 1, \dots, S \\ \left[\mathsf{O}_{\mu\mathsf{A}}(\mathsf{O}_{\mu\mathsf{B}}(\hat{z}^k))\right]^{s^k} = 2\hat{y}^{k+1,s^k} - \hat{w}^{k,s^k} \\ z^{k+1,s^k} = z^{k,s^k} - \frac{\eta^k}{Sp_{s^k}} \left(\hat{z}^{k,s^k} - \left[\mathsf{O}_{\mu\mathsf{A}}(\mathsf{O}_{\mu\mathsf{B}}(\hat{z}^k))\right]^{s^k} \right) \\ z^{k+1,s} = z^{k,s} \quad \text{for all } s \neq s^k \end{cases}$$

Pruning unnecessary computations, the asynchronous version of Progressing Hedging boils down to:

Every worker asynchronously do

$$\begin{cases} \text{Draw a scenario } s^k \in \{1, \dots, S\} \text{ with probability } \mathbb{P}[s^k = s] = q_s \\ \hat{x}_t^{k,s^k} = \frac{1}{\sum_{\sigma \in \mathcal{B}^{s^k}} p_\sigma} \sum_{\sigma \in \mathcal{B}_t^{s^k}} p_\sigma \hat{z}_t^{k,\sigma} \quad \text{for all } t = 1, \dots, T \\ \hat{y}^{k+1,s^k} = \operatorname{argmin}_{y \in \mathbb{R}^n} \left\{ f^{s^k}(y) + \frac{1}{2\mu} \left\| y - 2\hat{x}^{k,s^k} + \hat{z}^{k,s^k} \right\|^2 \right\} \\ \left| \begin{array}{l} z^{k+1,s^k} = z^{k,s^k} + \frac{2\eta^k}{Sp_{s^k}} \left(\hat{y}^{k+1,s^k} - \hat{x}^{k,s^k} \right) \\ z^{k+1,s} = z^{k,s} \quad \text{for all } s \neq s^k \end{array} \right. \end{cases}$$

This asynchronous algorithm can be readily rewritten as Algorithm 4, highlighting the master-worker implementation. Theorem 3.2 then follows directly from Proposition 2.

D `RPH` toolbox: Implementations Details

A basic presentation of the toolbox `RPH` is provided in Section 5; a complete description is available on the online documentation. In this section, we briefly provide complementary information on the input/output formats.

The input format is a `Julia` structure, named `problem`, that gathers all the information to solve a given multi-stage problem.

```
struct Problem{T} # Main input class implemented in src files
    scenarios::Vector{T}
    build_subpb::Function
    probas::Vector{Float64}
    nscenarios::Int
    nstages::Int
    stage_to_dim::Vector{UnitRange{Int}}
    scenariotree::ScenarioTree
end
```

The attribute `scenarios` is an array representing the possible scenarios of the problem. `nscenarios` is the total number of scenarios brought by the user and the probability affected to each scenario is indicated by the attribute `probas`. The number of stages, assumed to be equal among all scenarios, is stored in the attribute `nstages`. The dimension of the variable associated to each stage is stored in the vector of couples `stage_to_dim`: if for a fixed stage i, `stage_to_dim[i]` = `p:q`, then the variable associated to stage i is of dimension $q - p + 1$. Each of the scenarios must inherit the abstract structure `AbstractScenario`. This abstract structure does not impose any requirements on the scenarios themselves, so that the user is free to plug any relevant information in these scenarios. Here is an example.

```
abstract type AbstractScenario end  ## Abstract class implemented in src files

struct UserScenario <: AbstractScenario ## Custom class to be designed by the user
    trajcenter::Vector{Float64}
    constraintbound::Int
end

## Class Atributes to be designed by the user
# Instantiation of 4 scenarios
scenario1 = UserScenario([1, 1, 1], 3)
scenario2 = UserScenario([2, 2, 2], 3)
scenario3 = UserScenario([3, 3, 3], 3)
scenario4 = UserScenario([3, 3, 3], 3)
custom_scenarios = [scenario1, scenario2, scenario3, scenario4]

custom_nscenarios = length(custom_scenarios)
custom_stage_to_dim = [1:1, 2:2]
custom_nstages = length(stage_to_dim)
probabilities = [0.1, 0.25, 0.50, 0.15]
```

The function `build_subpb`, provided by the user, informs the solver about the objective function f_s to use for each scenario. This function is assumed to take as inputs a `Jump.model` object, a single `scenario` object and an object `scenarioId`, which corresponds to an integer that identifies the scenario. `build_subpb` must then return the variable designed for the optimization, named `y` below, an expression of the objective function f_s, denoted below `objexpr` as well as the constraints relative to this scenario, denoted below `ctrref`.

```
# Function to be designed by the user
function custom_build_subpb(model::JuMP.Model, s::UserScenario, id_scen::ScenarioId)
    n = length(s.trajcenter)
```

```
    y = @variable(model, [1:n], base_name="y_s"*string(id_scen))
    objexpr = sum((y[i] - s.trajcenter[i])^2 for i in 1:n)
    ctrref = @constraint(model, y .<= s.constraintbound)

    return y, objexpr, ctrref
end
```

Finally, the attribute `scenariotree` is aimed at storing the graph structure of the scenarios. `scenariotree` must be of type `ScenarioTree`, a tree structure designed by the authors. One can build an object scenario tree, by directly stating the shape of the tree with the help of `Julia` set structure.

```
## Instantiation of the scenario tree
stageid_to_scenpart = [
        OrderedSet([BitSet(1:4)]),                    # Stage 1
        OrderedSet([BitSet(1:2), BitSet(3:4)]),       # Stage 2
    ]
custom_scenariotree = ScenarioTree(stageid_to_scenpart)
## Instantiation of the problem
pb = Problem(
        custom_scenarios,  # scenarios array
        custom_build_subpb,
        custom_probabilities,
        custom_nscenarios,
        custom_nstages,
        custom_stage_to_dim,
        custom_scenariotree
    )
```

When the tree to generate is known to be complete, one can fastly generate a scenario tree with the help of the constructor by giving the depth of the tree and the degree of the nodes (assumed to be the same for each node in this case):

```
custom_scenariotree =  ScenarioTree(; depth=custom_nstages, nbranching=2)

pb = Problem(
        custom_scenarios,  # scenarios array
        custom_build_subpb,
        custom_probabilities,
        custom_nscenarios,
        custom_nstages,
        custom_stage_to_dim,
        custom_scenariotree
    )
```

The output of the algorithm is the final iterate obtained together with information on the run of the algorithm. Logs that appear on the console are the input parameters and the functional values obtained along with iterations. If the user wishes to track more information, a callback function can be instantiated and given as an input. This additional information can then either be logged on the console or stored in a dictionary `hist`.

```
pb = build_simpleexample()
hist=OrderedDict{Symbol, Any}()

y_PH = solve_progressiveHedging(pb, maxiter=150, maxtime=40, hist=hist, callback=callback)
```