Gradient-based rho Parameter for Progressive Hedging

Ulysse Naepels - École Polytechnique, IP Paris
David L. Woodruff - UC Davis

August 12, 2023

Abstract

Watson and Woodruff [29] developed a heuristic for computing variable-dependent values of the penalty parameter ρ from the model itself. We combine this heuristic with a gradient-based method, in order to obtain a new method for calculating ρ values. We then introduce a method for iteratively computing variable-dependent ρ values. This method is based on a first-order condition, and can be implemented with criteria that allow the parameter to be updated as the algorithm progresses. These new approaches for selecting and updating ρ can have an important impact on overall convergence, and on the behavior of decision variables and dual weights.

1 Introduction

1.1 Motivation and context

Stochastic programs efficiently model many optimization problems involving uncertainty. This is the case, for example, in planning and electricity generation unit commitment [27], network interdiction [9, 16], and other problems [4, 21]. A standard way of modeling stochastic programs is based on a discretization of uncertainty. Uncertainty is decomposed into a number of deterministic scenarios, each of which has an associated probability representing the likelihood of occurrence. Decisions are divided into stages, depending on the information available to the decision-maker and the time of the decision. Standard models use non-anticipativity constraints to ensure that decisions are not taken based on information that is not yet available. The algorithms used to solve these stochastic programs lend themselves to the use of parallel-computing because each scenario or sub-problem can be solved independently, enabling the computational load to be distributed using parallel processing units. Such algorithms make it possible to solve large, real-world stochastic programs in tractable time frames [19].

This work focuses on one such algorithm in particular, the Progressive Hedging (PH) algorithm proposed by Rockafellar and Wets [24]. The PH algorithm is efficient for solving multi-stage stochastic programs, especially those with integer variables at each stage [14]. It is sometimes referred to as a horizontal decomposition method, as it decomposes the program into scenarios. Its performance has been shown to be highly sensitive to the selection of the quadratic penalty parameter ρ [1, 12, 21, 23]. To tackle convergence and efficiency issues, the penalty parameter is sometimes increased at each iteration in the case of the Alternating Direction Method of Multipliers (ADMM) [2, 3, 6], but also in the case of PH [11]. Watson and Woodruff [29] have developed a heuristic for computing variable-dependent values of the penalty parameter ρ from the model itself. We propose to combine this heuristic with a gradient-based method, in order to obtain a new method for calculating ρ values. We then introduce a method for iteratively computing variable-dependent ρ values. This method is based on a first-order condition, and can be implemented with criteria that allow the parameter to be updated as the algorithm progresses. These new approaches for selecting and updating ρ have an impact on overall convergence, and on the behavior of decision variables and dual weights.

For this paper's experiments, we make use of the open source `mpi-sppy` package as support. It is comprehensively described in [19], and can be found at `https://github.com/Pyomo/`

`mpi-sppy`, which enables us to use parallel decomposition to solve stochastic programs with algorithms such as progressive hedging. The formulation of stochastic programs in `mpi-sppy` relies on the open-source algebraic modeling language `Pyomo` [7] `http://www.pyomo.org/`. The architecture provides high parallel efficiency by supporting high-scale parallelism. It enables us, for example, to compute upper and lower bounds without negatively impacting computation times. The implementation of this work is done within `mpi-sppy` and can be found on the corresponding github.

1.2 Notation

Our notation is similar to the one used in [19]. We denote by T the number of decision stages and we use $t \in \{1, \dots, T\}$ to index stages; however, we observe that these stages do not always correspond to time periods. Let ξ_t a random variable, which may be vector valued, associated with each decision stage t. This random variable represents the stochastic aspect of the problem. In the case of time periods, we consider that the decisions for stage t are made once the values of the random variables for stages up to and including t are known. Hence we will mostly refer the value ξ_t to only for stage $2, \dots, T$.

Definition 1.1. *We denote by* $\vec{\xi}^{\,t}$ *the realized values of all* ξ_t *up to and including stage* t*. In particular,*

$$\vec{\xi}^{\,T} = (\xi_t,\ t = 2, \dots, T)$$

refers to a full scenario. We will simply use the notation ξ*.*

Definition 1.2. *We denote by* Ξ *the full set of scenarios, where each scenario* ξ *has probability* π_ξ*. We define a tree corresponding to the set of realizations* ξ *such that different scenarios with the same realization up to stage* t *share a node corresponding to that stage* t*. Hence,* $\vec{\xi}^{\,t}$ *refers also to a node in the scenario tree.*

We denote by $\mathcal{G}_t$ *the set of all nodes for stage* t *and by* $\mathcal{G}_t(\xi)$ *the node corresponding to scenario* ξ*. If* $\mathcal{D}$ *is a given node, we denote by* $\mathcal{D}^{-1}$ *the set of scenarios that define the node.*

Similarly, we denote by x^t the decision variable at stage $t \in \{1, \dots, T\}$, and $\vec{x}^{\,t}$ the decisions for all stages up to and including $t \in \{1, \dots, T\}$.

Definition 1.3. *Let* f *be the cost function. More specifically,* $f_1(x^1)$ *corresponds to the first stage cost and* $f_t(x^t; \vec{x}^{\,t-1}, \vec{\xi}^{\,t})$ *each subsequent stage. Let us note that* x^t *is the argument of* f_t *while* $\vec{x}^{\,t-1}, \vec{\xi}^{\,t}$ *are parameters giving the solutions and realizations up to stage* t*.*

Thus, we can express the multi-stage stochastic program as follows:

$$Z^* = \min_{x, \hat{x}} \sum_{\xi \in \Xi} \pi_\xi \left[f_1(x^1(\xi)) + \sum_{t=2}^{T} f_t\left(x^t(\xi); \vec{x}^{\,t-1}, \vec{\xi}^{\,t}\right) \right] \tag{1a}$$

$$x^t(\xi) - \hat{x}^t(\mathcal{D}) = 0, \quad t = 1, \dots, T-1,\ \mathcal{D} \in \mathcal{G}_t,\ \xi \in \mathcal{D}^{-1} \tag{1b}$$

$$x(\xi) \in X_\xi, \quad \xi \in \Xi \tag{1c}$$

with X_ξ a set of constraint for each scenario $\xi \in \Xi$. The condition (1b) enforces the decision variables non-anticipativity. Indeed, it forces x^t to only consider the information available before stage t (i.e. the scenarios that define the nodes $\mathcal{D} \in \mathcal{G}_t$). The condition (1c) summarizes all other constraints.

1.3 The Progressive Hedging algorithm

The Progressive Hedging (PH) algorithm was proposed by Rockafellar and Wets [24] and has been described in many places such as [19, 29]. It decomposes stochastic programs along scenarios in several sub-problems in order to solve them. PH is related to other decomposition algorithms such as Alternating Direction Methods of Multipliers [3, 6]. Under the condition of continuity of all decision variables, the algorithm possesses convergence properties. In the case

of discrete decision variables, convergence and efficiency are issues, even if PH can still be used as a heuristic.

For a multi-stage optimization problem such as (1), the Progressive Hedging algorithm can be stated as described in Algorithm 1. The formulation given here is the same as the one introduced in [19].

Algorithm 1 Progressive Hedging

1: **Initialization**: Let $\nu \leftarrow 0$ and $w^{(t,\nu)}(\xi) \leftarrow 0,\ \forall \xi \in \Xi,\ t = 1, \ldots, T$. Compute for each $\xi \in \Xi$:

$$x^{(\nu+1)}(\xi) \in \operatorname{argmin}_x f_1(x^1) + \sum_{t=2}^{T} f_t\left(x^t; \vec{x}^{t-1}(\xi), \vec{\xi}^{t}\right)$$

2: **Iteration Update**: $\nu \leftarrow \nu + 1$

3: **Aggregation**: Compute for each $t = 1, \ldots, T-1$ and for each $\mathcal{D} \in \mathcal{G}_t$:

$$\bar{x}^{(\nu)}(\mathcal{D}) \leftarrow \sum_{\xi \in \mathcal{D}^{-1}} \pi_\xi x^{(t,\nu)}(\xi) / \sum_{\xi \in \mathcal{D}^{-1}} \pi_\xi$$

4: **Price Update**: Compute for each $t = 1, \ldots, T-1$ and for each $\xi \in \Xi$

$$w^{(t,\nu)}(\xi) \leftarrow w^{(t,\nu-1)}(\xi) + \rho\left[x^{(t,\nu)}(\xi) - \bar{x}^{\nu}(\mathcal{G}_t(\xi))\right]$$

5: **Decomposition**: Compute for each $\xi \in \Xi$

$$\begin{aligned} &x^{(\nu+1)}(\xi) \in \operatorname{argmin}_x f_1(x^1) + \textstyle\sum_{t=2}^{T} f_t\left(x^t; \vec{x}^{t-1}(\xi), \vec{\xi}^{t}\right) \\ &+ \textstyle\sum_{t=1}^{T-1}\left[w^{(t,\nu)}(\xi)^\top x^t + \frac{\rho}{2}\|x^t - \bar{x}^{(\nu)}(\mathcal{G}_t(\xi))\|^2\right] \end{aligned}$$

6: **Termination:** If a criterion is met, Stop. Otherwise go to step 2.

The algorithm is initialized by solving all sub-problems, each corresponding to an individual scenario problem. Then, each PH iteration starts with an aggregation. It involves computing the primal values mean $\bar{x}$ by projecting the solutions of the individual scenarios onto the subspace of non-anticipative policies. From a certain point of view, this is an estimation of the optimal vector system that satisfies the non-anticipativity requirement. The dual prices $w^{(t,\nu)}(\xi)$ function like lagrangian multipliers. They are updated using the previous projection, values of the non-anticipative variables, and the quadratic penalty parameter, ρ. The algorithm then decomposes according to each scenario, perturb each objective function with the dual prices obtained at the previous step and a quadratic penalty term, and solve these individual sub-problem. Finally, algorithm termination can be based on internal PH information such as a relative gap threshold or a tolerance threshold on all non-anticipative constraints. For example, a convergence measure commonly used in practical applications to determine termination is implemented as follows, with $\epsilon > 0$ a termination threshold:

$g^{(\nu)} \coloneqq \sum_{\xi \in \xi} \pi_\xi \left(\sum_{t=1}^{T-1} |x^t(\xi) - \bar{x}^{(\nu)}(\mathcal{G}_t(\xi))|\right)$

If $g^{(\nu)} > \epsilon$, then go to step 2. Otherwise, terminate.

A key aspect of the algorithm is the joint use of the price vectors w and the proximal term

$$\frac{\rho}{2}\|x^t - \bar{x}^{(\nu)}(\mathcal{G}_t(\xi))\|^2.$$

In the following section, we study the impact of the penalty parameter on the Progressive Hedging algorithm progression.

2 Compute effective ρ values

2.1 Impact of ρ on PH convergence

The choice of the penalty parameter ρ value is commonly left to the user, and it is known that its magnitude has a significant impact on the efficiency of the Progressive Hedging algorithm

convergence. We introduce a general method to compute ρ values from the model parameters. Early experiments using PH have led to two observations. Firstly, a good choice for the value of the penalty parameter varies according to the model. For example, Mulvey and Vladimirou (1991) [23] obtained satisfying results for ρ well below 1, and Løkketangen and Woodruff [22] for ρ between 0.4 and 0.8, whereas Listes and Dekker (2005) [21] reported ρ choices ranging between 50 and 100. Secondly, PH convergence performance is substantially impacted by the choice of the penalty parameter. Mulvey and Vladimirou [23] also report that they observe efficiency to be sensitive to the choice of ρ.

More specifically, choosing different ρ values has an impact on both the dual weights and the primal decision variables. Considering step 4 of the Progressive Hedging algorithm, we observe that the difference between two consecutive values of the dual price w is exactly the distance between the decision variable and its mean scaled by the penalty parameter. Hence, the choice of ρ has a direct influence on the progression speed of w. Let us now consider the decomposition step 5 of the algorithm. For a given scenario $\xi \in \Xi$, we minimize the objective function perturbed by two terms, $\sum_{t=1}^{T-1} w^{(t,\nu)}(\xi)^\top x^t$ and $\sum_{t=1}^{T-1} \frac{\rho}{2}\|x^t - \bar{x}^{(\nu)}(\mathcal{G}_t(\xi))\|^2$. These two terms compete with each other during the minimization. In particular, if the first term seeks to make decision variables tend towards a certain point, the second term will influence by bringing x closer to $\bar{x}$. This is particularly the case for large ρ values. Consequently, different choices of penalty parameter values will also modify the progression of the decision variables. By selecting a ρ value, we can now aim for two types of effect on convergence. On the one hand, if we wish to obtain fast primal convergence, setting a large ρ value will increase the weight of the quadratic term in the minimization and accelerate convergence by updating more aggressively x values. On the other hand, if we aim to obtain a smooth and gradual evolution for the dual variables - which can be the case when computing a lower bound for example - selecting a smaller ρ value would be relevant.

2.2 Variable-dependent heuristics for selecting ρ

In order to tackle the issue of a general selection rule for the penalty parameter, Watson and Woodruff (2011) [29] introduce a mathematically-based heuristic to compute ρ in proportion to the unit-cost of the associated decision variable. More specifically, they indicate that an efficient choice for ρ values is a variable-dependent ρ. They distinguish between three ways of selecting a value for ρ. The first is the default: choose a single value for all variables. They refer to the second as cost-proportional: it consists in choosing for ρ the value of the unit-cost multiplied by a re-scale factor. Finally, they provide a formula, details of which are given below. The latter aims to quickly reach good dual weights values. In particular, they want the magnitude of w to reach this good value from below (in the absolute value sense), in order to minimize oscillations. These oscillations can happen when w is updated in a too aggressive manner, and cause the price vector to move past its optimal value. Especially in mixed-integer programs (MIPs), such a convergence coming from both below and above will cause integer variables to change in a certain direction and then reverse, creating oscillations. We retrace the main steps behind the heuristic [29] here.

For a given scenario $\xi \in \Xi$, we consider a single decision variable $x^t(\xi)$ at stage $t \in \{1, \dots, T\}$ and node $\mathcal{D} = \mathcal{G}_t(\xi)$, and $c_\mathcal{D}$ the associated cost. After PH's first iteration, we have an approximation of the optimal value: $\bar{x}^{(0)}(\mathcal{G}_t(\xi))$. Assume we know a value of ρ which will result in $w^t(\xi) = c_\mathcal{D}$, then the proximal term

$$\frac{\rho}{2}\|x^t - \bar{x}^{(\nu)}(\mathcal{G}_t(\xi))\|^2$$

will force the solution to be $\bar{x}^{(0)}(\mathcal{G}_t(\xi))$ in the subsequent PH iteration. Since the value of w is updated by

$$w^{(t,\nu)}(\xi) \leftarrow w^{(t,\nu-1)}(\xi) + \rho\left[x^{(t,\nu)}(\xi) - \bar{x}^{\nu}(\mathcal{G}_t(\xi))\right],$$

They deduce that the value of ρ for a given scenario resulting in $w = c$ is

$$\rho_t(\xi) = \frac{c_{\mathcal{D}}}{|x^t(\xi) - \bar{x}^{(0)}(\mathcal{G}_t(\xi))|}. \tag{2}$$

This formula highlights the proportionality in magnitude to the unit cost. Since we want w to approach their ultimate value from below (in the absolute value sense), using a bound on the denominator can sometimes be useful to get even smaller ρ values. We now describe the use of this formula with a specific gradient-based cost.

3 Gradient-based method for selecting ρ values

3.1 First-order condition for selection ρ

First, we introduce a novel method to obtain price vector values using a first-order condition. We then use this method to compute effective ρ values. We consider the multi-stage stochastic program (1), and assume that the objective function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, with n the dimension of the decision variables, takes on a very large value when its argument violates constraints. Thus, we obtain an equivalent stochastic program and get rid of the formal constraints. We now focus on the decomposition step 5 of the Progressive Hedging algorithm: for each scenario $\xi \in \Xi$, we minimize a perturbed objective function. Ideally, we would aim to reach the optimal value x^* at the next iteration, i.e. obtain x^* as argument of the minimum. In practice, however, we can only approximate this optimum value, which we refer to as $\hat{x}$. If we decide to ignore the proximal term in the first instance, then we want the dual weights $\hat{w}$ to satisfy the following minimization for each $\xi \in \Xi$:

$$\begin{aligned}
&\hat{x}(\xi) \in \operatorname{argmin}_x f_1(x^1) + \sum_{t=2}^{T} f_t\left(x^t;\, \vec{x}^{\,t-1}(\xi),\, \vec{\xi}^{\,t}\right) + \sum_{t=1}^{T-1} \hat{w}^t(\xi)^\top x^t \\
&\text{s.t.} \sum_{\xi \in \mathcal{D}^{-1}} \pi_\xi \hat{w}^t(\xi) = 0 \;\text{ for each } t = 1, \ldots, T-1 \text{ and for each } \mathcal{D} \in \mathcal{G}_t
\end{aligned}$$

The constraint on the dual prices vector reflects the orthogonality of w with the space of non-anticipative variables. This is ensured by the way these prices are updated in the Progressive Hedging algorithm (step 4). If we ignore this null expectation condition at first, and assume that the objective function f is regular enough, a first order condition gives us, for each $\xi \in \Xi$:

$$\nabla f_1(\hat{x}^1) + \sum_{t=2}^{T} \nabla f_t\left(\hat{x}^t;\, \vec{\hat{x}}^{\,t-1}(\xi),\, \vec{\xi}^{\,t}\right) + \sum_{t=1}^{T-1} \hat{w}^t(\xi) = 0$$

Hence, we have a natural candidate for $\hat{w}$ defined as follows:

$$\hat{w}^t(\xi) = -\nabla f_t\left(\hat{x}^t;\, \vec{\hat{x}}^{\,t-1}(\xi),\, \vec{\xi}^{\,t}\right) \;\text{ for each } \xi \in \Xi \text{ and } t = 1, \ldots, T-1 \tag{3}$$

We cannot directly use this formula since PH updates its own dual prices following step 4 of the algorithm. However, according to the previous heuristic [29], we can choose a ρ such that the next iteration of PH would result in dual prices values approximating the above expression (3). Thus, using the Watson-Woodruff heuristic formula (2) with the cost (3), we obtain the following expression:

$$\rho_t(\xi) = \frac{c_{\mathcal{D}}}{|x^t(\xi) - \bar{x}^{(0)}(\mathcal{G}_t(\xi))|} \text{ with } c_{\mathcal{D}} = -\nabla f_t\left(\hat{x}^t;\, \vec{\hat{x}}^{\,t-1}(\xi),\, \vec{\xi}^{\,t}\right) \tag{4}$$

We want to make use of this formula in the Progressive Hedging algorithm in order to get better and faster convergence. We can make a few practical observations. First, the most natural idea is to use expression (4) at the end of the initialization step in order to get all ρ values. This enables us to select effective ρ values when we don't know what values would be effective for a given model. We observe that we don't know the sign of the gradient a priori. Therefore,

using its absolute value for implementation guarantees non-negative ρ values. To compute the gradient, we need to have a feasible solution $\hat{x}$ where to compute the gradient. This solution can be obtained using previous experiments or iteration 0. In the case of mixed-integer programs, we use a continuous relaxation in order to compute the gradient. Such a relaxation will ignore the integer constraints for the computation of the gradient and restore them after. It has been implemented using relaxations provided in `Pyomo` [7].

Note that for linear programs where all non-anticipative variables are in the objective function, the use of the gradient method proposed here gives the same ρ values as the Watson and Woodruff heuristic regardless of the *hatx* that is used.

3.2 Compute scenario-independent ρ values

Contrary to the Alternating Direction Method of Multipliers (ADMM) [6], the Progressive Hedging algorithm requires ρ to not depend on the scenarios. Therefore, we need to compute scenario-independent ρ values, or choose one amongst the ones obtained via (4). In [29], Watson and Woodruff face the same issue of scenario-dependence in ρ computation. They argue that, since we want to approach the optimal value of w from below (in the absolute value sense), we can use a bound on the denominator, and thus get rid of the dependence. They compute ρ values using the expectation of the formula (2):

$$\rho = \frac{c_{\mathcal{D}}}{\max\left(1, \sum_{\xi\in\Xi} \pi_\xi |x^{t,(0)}(\xi) - \bar{x}^{(0)}|\right)}$$

In our case ((4)), the gradient-based cost depends on scenarios. Therefore, using a bound on the denominator as above will not suffice to get rid of the dependence. We have several possibilities: e.g. using the average, minimum, or maximum of $\rho(\xi)$ $\xi \in \Xi$ are natural options. Choosing one or the other will have a different effect on both the primal and the dual behaviors, as it will impact directly the magnitude of dual prices. A natural idea is to weight each of the rhos by the probability of the corresponding scenario. It gives us the following expression:

$$\rho = \sum_{\xi\in\Xi} \pi_\xi \rho(\xi)$$

However, if the objective is to obtain a fast primal convergence, it can be relevant to choose $\rho = \min\{\rho(\xi) : \ \rho(\xi) > 0\ ,\ \xi \in \Xi\}$. It will result in a more aggressive update of the decision variables values. Conversely, if the purpose is to obtain better w convergence, taking the minimum in order to select smaller ρ values would be more appropriate. In the latter situation, note that the smallest ρ value can be zero. In that case, to avoid some of the decision variables remaining at the same point, it can be appropriate to choose the minimum of non-zero values.

3.3 Implementation of gradient-based ρ computation

In order to compute the gradient of a given objective function, we use `PyNumero`, which aims to reduce the time required to develop nonlinear optimization algorithms while maximizing computational performance. It is a Python framework having both the efficiency of libraries like SciPy [17] and AMPL `https://www.ampl.com/` [13], and modeling capabilities of the algebraic modeling language Pyomo [7]. `PyNumero` is open source and can be found at `https://github.com/Pyomo/pyomo/tree/main/pyomo/contrib/pynumero`, and numerical experiments using it at [25].

In this work, we will only use `PyNumero` to compute the gradient of objective functions. To do so, we have to build a `PyomoNLP` instance which is able to use `Pynumero` interfaces to ASL (AMPL Solver Library [13]) in order to compute gradients. If the model is a mixed-integer program, `PyNumero` will not be able to compute gradients. In that case, we apply a `Pyomo` model transformation that relaxes integrality restrictions for the purpose of computing the gradients. The computational cost of the ρ computation comes mainly from the fact that it is necessary to

loop over all the scenarios to compute the gradient, and then loop over all the variable values to compute the denominator.

4 Iterative method for selecting ρ

4.1 Computing an iteration-dependent ρ values

In [29], Watson and Woodruff state that using an iterative ρ may be more appropriate. Crainic et al. [11] make the same observation and suggests a method for updating the penalty parameter at each iteration. They propose multiplying it by a factor $\alpha > 1$ which leads to a progressive increase in ρ magnitude at each iteration. More specifically, as the algorithm progresses towards convergence, x is getting closer to $\bar{x}$ which slows down the update of the dual prices at step 4, and consequently convergence. Increasing ρ therefore counteracts this slowdown effect by enlarging the step size of dual weights. We introduce a novel method for computing iterative ρ values, based on the current non-anticipative variable values. We denote by $\nu \geq 1$ the current PH iteration and by $\xi \in \Xi$ a given scenario. We consider the following minimization, which takes place at step 5, which we re-write here:

$$\min_x f_1(x^1) + \sum_{t=2}^{T} f_t\left(x^t;\, \vec{x}^{\,t-1}(\xi),\, \vec{\xi}^{\,t}\right) + \sum_{t=1}^{T-1}\left[w^{(t,\nu)}(\xi)^\top x^t + \frac{\rho}{2}\|x^t - \bar{x}^{(\nu)}(\mathcal{G}_t(\xi))\|^2\right]$$

As in section 3, we want an approximation $\hat{x}$ of the optimal solution of the stochastic program (1) to be the argument of the minimum for the above minimization. With similar smoothness assumptions, using a first-order condition gives us:

$$\nabla f_1(\hat{x}^1) + \sum_{t=2}^{T} \nabla f_t\left(\hat{x}^t;\, \vec{\hat{x}}^{\,t-1}(\xi),\, \vec{\xi}^{\,t}\right) + \sum_{t=1}^{T-1}\left[w^t(\xi) + \rho(\hat{x}^t - \bar{\hat{x}}^{(\nu)}(\mathcal{G}_t(\xi)))\right] = 0 \tag{5}$$

If we consider a scenario-dependent and variable-dependent penalty parameter, we obtain a natural candidate for selecting ρ:

$$\rho^{(\nu)}(\xi) = \frac{c_{\mathcal{D}} - w^{(t,\nu)}}{x^{(t,\nu)}(\xi) - \bar{x}^{(\nu)}(\mathcal{G}_t(\xi))} \text{ with } c_{\mathcal{D}} = -\nabla f_t\left(\hat{x}^t;\, \vec{\hat{x}}^{\,t-1}(\xi),\, \vec{\xi}^{\,t}\right) \tag{6}$$

This expression of ρ depends on both the current value of the decision variable x and of the dual weight w. During the run of the algorithm, $x^{(t,\nu)}(\xi)$ gets closer to $\bar{x}^{(\nu)}(\mathcal{G}_t(\xi))$ for each $\xi \in \Xi$ and $t = 1, \ldots, T-1$. It induces a decrease in the magnitude of the denominator, and therefore an increase in the penalty parameter ρ. Thus, it corresponds to the evolution we wished to achieve to enable faster primal-dual convergence.

We now show that the formula obtained in (6) is consistent with the heuristic proposed by Watson and Woodruff [29]. As explained in section 2.1, the proximal term and the term in w compete in the minimization at step 5. The heuristic introduced by Watson and Woodruff [29] neglects the proximal term. If we do so in the first-order condition (5), we obtain

$$\hat{w}^t(\xi) = -\nabla f_t\left(\hat{x}^t;\, \vec{\hat{x}}^{\,t-1}(\xi),\, \vec{\xi}^{\,t}\right)$$

which corresponds to the expression of ρ obtained at iteration 0 of PH as detailed in formula (4). Now, if we decide to neglect the w-term instead, the first-order condition gives us:

$$\nabla f_1(\hat{x}^1) + \sum_{t=2}^{T} \nabla f_t\left(\hat{x}^t;\, \vec{\hat{x}}^{\,t-1}(\xi),\, \vec{\xi}^{\,t}\right) + \sum_{t=1}^{T-1} \rho(\hat{x}^t - \bar{\hat{x}}^{(\nu)}(\mathcal{G}_t(\xi))) = 0$$

It results in the following selection of ρ:

$$\rho^{(\nu)}(\xi) = \frac{c_{\mathcal{D}}}{x^{(t,\nu)}(\xi) - \bar{x}^{(\nu)}(\mathcal{G}_t(\xi))} \text{ with } c_{\mathcal{D}} = -\nabla f_t\left(\hat{x}^t;\, \vec{\hat{x}}^{\,t-1}(\xi),\, \vec{\xi}^{\,t}\right)$$

It also corresponds exactly to the formula obtained in (4). Thus, neglecting the proximal term or the w-term in the expression of the penalty parameter (6) gives us the same results as those obtained using Watson and Woodruff's heuristic [29].

With regard to computation efficiency, we observe that for variables that appear in a linear objective, the gradient does not depend on $\hat{x}$ (or the PH iteration): we just need to compute it once at the end of the algorithm initialization. In addition, the values of the variables $w^{(t,\nu)}$, and $x^{(t,\nu)}(\xi)$ and $\bar{x}^{(\nu)}(\mathcal{G}_t(\xi))$, needed to compute the denominator, have already been calculated and are in cached memory at the time of the new setting of ρ. The main computational cost comes from the fact that, since ρ is variable-dependent, we have to loop over all variables to set new ρ values.

Implementing this adaptive ρ setting makes it highly sensitive to primal convergence. In particular, the decision variables can get close to their mean rapidly, before the convergence slows down. In that case, the difference $x^{(t,\nu)}(\xi) - \bar{x}^{(\nu)}(\mathcal{G}_t(\xi))$ will reach a very small norm very quickly, which cause ρ to get disproportionately big very quickly. Such an increase in ρ values is ineffective, and leads to the blocking of primal and dual convergences. To tackle this issue, we define criteria in order to decide when to update ρ.

4.2 Primal-based update criterion

To tackle the issue of when to update ρ values, we leverage measures of convergence, which aim to compute a convergence metric from scenario primal and dual values. These measures are implemented in `PySP` [30] and `mpi-sppy` [19]. A first approach is to consider the primal convergence evolution. If the latter is still decreasing significantly, then the current ρ values is efficient enough and we can let it go on. If the primal convergence metric stagnates, it can be due to two different phenomena. The first option is that the decision variables have converged. In this case updating ρ will not have an impact since the difference $x^{(t,\nu)}(\xi) - \bar{x}^{\nu}(\mathcal{G}_t(\xi))$ will not vary much. The second option is that the Progressive Hedging algorithm is still making progress but slowly because ρ is too small. In this case, we update ρ following the formula (4), which will bring it back to scale. To this end, we introduce a seminorm on the problem variables.

Definition 4.1. *For a given iteration ν and a scenario $\xi \in \Xi$, let $y^{(\nu)}(\xi)$ be a variable vector. We define the norm* N *of y at iteration ν as:*

$$\mathrm{N}(y^{(\nu)}) \coloneqq \left[\sum_{\xi\in\xi} \pi_\xi \left(\sum_{t=1}^{T-1} |y^{(t,\nu)}(\xi)|^2\right)\right]^{1/2}$$

This seminorm allows us to define a convergence metric for primal decision variables:

$$\mathrm{N}\left(\frac{x^{(\nu)} - \bar{x}^{(\nu)}}{\bar{x}^{(\nu)}}\right) = \left[\sum_{\xi\in\xi} \pi_\xi \left(\sum_{t=1}^{T-1} \left|\frac{x^{(t,\nu)}(\xi) - \bar{x}^{(\nu)}(\mathcal{G}_t(\xi))}{\bar{x}^{(\nu)}(\mathcal{G}_t(\xi))}\right|^2\right)\right]^{1/2}$$

The latter is a measure commonly used, for example, to decide on the Progressive Hedging algorithm termination [5, 29]. It corresponds to a re-scaled norm of the primal residuals $x^{(t,\nu)}(\xi) - \bar{x}^{(\nu)}(\mathcal{G}_t(\xi))$. Primal and dual residuals are detailed in [6] and used for the Progressive Hedging algorithm in [5]. Dividing by $\bar{x}$ aims to re-scale the resulting norm. The decision variables values are dependent on the model, and, thus, have significantly different values in different numerical experiments. This criterion takes a threshold $\delta > 0$ and $\nu \geq 1$ the current PH iteration as inputs and returns a Boolean indicating if ρ needs to be updated or not. We can implement the criterion as follows:

This criterion is seeking to improve the primals convergence metric at each iteration. Indeed, when the latter stagnates, it will update ρ, and the new value will be a larger one which should make the decision variables move towards the optimal value. From a computational point of view, the main cost comes from accessing all variables values, which is done by looping over both scenarios and variables.

Rho update: primal criterion

1: **Compute**

$$c_{\text{primal}}(\nu) = |\,\mathrm{N}\left(\frac{x^{(\nu)} - \bar{x}^{(\nu)}}{\bar{x}^{(\nu)}}\right) - \mathrm{N}\left(\frac{x^{(\nu-1)} - \bar{x}^{(\nu-1)}}{\bar{x}^{(\nu-1)}}\right)|$$

2: **If** $c_{\text{primal}}(\nu) \leq \delta$, **Then** update ρ following (6). Otherwise, pass.

4.3 Dual-based update criterion

Iteratively setting the penalty parameter also aims at better dual convergence. The heuristic introduced by Watson and Woodruff [29] sets ρ based on what we would like the dual prices to be at the next iteration. Therefore, it is natural to examine the question of updating ρ from a dual perspective. An iteration-dependent choice of the penalty parameter aims for better convergence of dual variables, i.e. more efficient but gradual and without oscillations. More specifically, if w is still significantly moving, then, according to step 4, we can consider the product

$$\rho\left[x^{(t,\nu)}(\xi) - \bar{x}^{\nu}(\mathcal{G}_t(\xi))\right]$$

to be large enough. This will eventually lead to an update in the decision variables values which will get closer to the optimal value. Now, if w values are not changing significantly, it means that the above product has become too small. This may be due to two phenomena. First, this happens if the magnitude of ρ is too small compared to the difference between decision variables and their average. In this case, it makes particular sense to update the value of ρ using formula (4): this will give it a significant value in relation to the difference $x^{(t,\nu)}(\xi) - \bar{x}^{\nu}(\mathcal{G}_t(\xi))$. Second, this happens if the decision variables are converged, or very close, to the optimal solution. In this case, since the dual prices are not moving either, it means we have reached a primal-dual convergence point. In particular, updating ρ values here would not have any impact since neither the difference between x and $\bar{x}$ nor the value of w would evolve significantly. Therefore, we reuse the norm introduced in definition 4.1 to quantify the difference in w values between the current and the last PH iteration.

Rho update: dual criterion

1: **Compute**

$$c_{\text{dual}}(\nu) = \mathrm{N}\left(\frac{w^{(\nu)} - w^{(\nu-1)}}{w^{(\nu)}}\right)$$

2: **If** $c_{\text{dual}}(\nu) \leq \delta$, **Then** update ρ following (6). Otherwise, pass.

Dividing by the dual prices current values in the computation of $c_{\text{dual}}(\nu)$ aims to re-scale the resulting norm. The dual prices values are highly dependent on the model, and, thus, have significantly different values in different numerical experiments. We remark that the difference $w^{(\nu)} - w^{(\nu-1)}$ is equal to $\rho\left[x^{(t,\nu)}(\xi) - \bar{x}^{\nu}(\mathcal{G}_t(\xi))\right]$ (step 4). As a result, even though this criterion looks more closely at the evolution of dual weights, it doesn't provide any new information compared with the primal-based criterion.

4.4 Primal-dual update criterion

In [6], the authors detail a penalty parameter update scheme for the Alternating Direction Method of Multipliers. Different strategies for this scheme can be found in [15, 28]. This scheme relies on the idea that primal and dual progress should be balance. More specifically, it examines the primal and dual residuals (see [6] for ADMM, [5, 18] for PH), and increases ρ if the dual progression is faster than the primal progression and decreases ρ conversely. As explained in [18], a similar update scheme can be applied to the Progressive Hedging algorithm since it is a variant of ADMM and some similar issues arise.

Definition 4.2. *Let $\nu \geq 1$ be the current PH iteration. We define the primal residuals vector $r^{(t,\nu)}$ and dual residuals vector $s^{(t,\nu)}$ as:*

$$r^{(t,\nu)} = x^{(t,\nu)}(\xi) - \bar{x}^{(\nu)}(\mathcal{G}_t(\xi)), \quad s^{(t,\nu)} = -\rho\left[\bar{x}^{(\nu)}(\mathcal{G}_t(\xi)) - \bar{x}^{(\nu-1)}(\mathcal{G}_t(\xi))\right]$$

These residuals converge to 0 as the algorithm converges [6]. The update scheme for the penalty parameter in ADMM [6, 15, 18] is the following:

$$\rho^{(\nu+1)} = \begin{cases} \rho^{(\nu)} \cdot \tau_{\text{incr}} & \text{if } \mathrm{N}(r^{(\nu)}) > \mu\, \mathrm{N}(s^{(\nu)}) \\ \rho^{(\nu)} \cdot \frac{1}{\tau_{\text{decr}}} & \text{if } \mathrm{N}(s^{(\nu)}) > \mu\, \mathrm{N}(r^{(\nu)}) \\ \rho^{(\nu)} & \text{otherwise,} \end{cases} \tag{7}$$

The constants $\tau_{\text{incr}} > 1$, $\tau_{\text{incr}} > 1$ and $\mu > 1$ are parameters of the scheme. As [6] points out, the scheme (7) suggests that large values of ρ yield a large penalty on primal feasibility violations, and hence tend to reduce the primal residual. Conversely, small values of ρ tend to reduce the dual residual. We propose to combine this scheme and the gradient-based method of selecting ρ (6) to obtain a new update criterion. As in [29], we want the magnitude of w to reach a good value from below (in the absolute value sense). Therefore, we want the penalty parameter values not to grow too quickly during the Progressive Hedging algorithm convergence and too large relative to the model. It implies that we want to update the penalty parameter to bigger values only if dual residuals become excessively small compared to primal residuals.

Rho update: primal-dual criterion

1: **Compute**

$$c_{\text{primal-dual}}(\nu) = \frac{\mathrm{N}(s^{(\nu)})}{\mathrm{N}(r^{(\nu)})}$$

2: **If** $c_{\text{primal-dual}}(\nu) \leq \delta$, **Then** update ρ following (6). Otherwise, pass.

This update criterion aims to keep a certain balance between primal progression and dual progression during PH convergence. The positive parameter δ is intended to be small, and would correspond to $\frac{1}{\mu}$ in the scheme (7). Setting δ to a very small positive value will promote gradual convergence of dual weights and significantly reduce dual residuals before updating ρ. This results in less frequent updates and slower primal convergence. Conversely, a relatively large δ value will tend to favor primal convergence at the expense of dual residuals.

5 Numerical experiments

5.1 Stochastic unit commitment

For our experiments we consider the stochastic unit commitment example. Unit commitment (UC) is a widely-studied optimization problem in planning and power system operations. The unit commitment problem consists of finding an allocation plan for a set of operating power generators to satisfy the demand for electricity while minimizing the final production cost. This operating schedule has to meet both operational and physical constraints. For example, thermal units cannot rapidly change their power output, and large units cannot be turned on and off too frequently. Unit commitment is solved on a daily basis, and determines the future day's operational timeline for all power units, given the predicted next-day demand. There exist several formulations for the UC problem. This work uses the unit commitment formulation implemented in `mpi-sppy` and described in [8, 26]. We leverage PH to exploit the decomposition of uncertainty over a number of scenarios, using the parallel computing features provided by `mpi-sppy`. The unit commitment model in `mpi-sppy` relies on the `egret` (Electrical Grid Research and Engineering Tools) package, which can be fount at `https://github.com/grid-parity-exchange/Egret`. The `egret` package is based on the mixed-integer programming formulations detailed in [20].

We conduct experiments on this unit commitment problem to study the convergence of the Progressive Hedging algorithm 1 with varying ρ settings. To this end, we use the solver CPLEX [10] to solve a 5-scenario stochastic version of unit commitment. These experiments use the `mpi-sppy` architecture [19] for parallelizing the resolution of stochastic programs. Regarding the parallel setup in `mpi-sppy`, our experiments use a PH hub limited to 20 iterations with to additional spoke. The first one, `lagrangian`, computes a lower bound based on a Lagrangian

relaxation of the problem. Details about its convergence can be found in [14]. The second one, `xhatshuffle`, computes upper bounds by shuffling the scenarios and looping over them to try a $\hat{x}$ until the hub provides new variable values.

For our experiments, we define several strategies for selecting ρ values.

- Cost-based rho: reference strategy. It corresponds to the current cost proportional ρ setter used by default for UC in `mpi-sppy`.
- Gradient-based rho: gradient-based ρ values as described in section 3. The penalty parameter is computed once in the beginning and then kept constant.
- Adaptive gradient rho: gradient-based ρ values as described in section 4. The penalty parameter is computed at each PH iteration.

When computing adaptive gradient-based ρ values following (6), the denominator might be zero (or near-zero) for mixed-integer programs. This is the case in particular for unit commitment in which there are many binary variables. If the denominator corresponding to a decision variable is 0 we assign it the value of the maximum of the denominators of the other variables based on heuristic reasoning that if the variable becomes "un-converged" it will need some ρ until the next update.

	cost-based ρ	gradient-based	adaptive gradient
wall clock time (s.)	843.54	893.42	1055.79
PH LB	62613.0172	62628.3674	62627.1023
PH UB	62628.7844	62629.3967	62629.3967
EF	62628.395	62628.395	62628.395
Abs. Gap	15.7672	1.0292	2.2944
Rel. Gap	0.025%	0.002%	0.004%

Table 1: Experiments with cost-based ρ selection and gradient-based methods

We report the results of our first experiment in table 1. For comparison purposes, we also report the solution for the extensive form solved by CPLEX without decomposition. First, we observe that run-times of the default ρ selection and gradient-based selection are almost the same, whereas the adaptive gradient takes more time. This is due to the fact that it re-computes the penalty parameter values at each iteration which requires to loop over every scenario and variable. Moreover, for a fixed number of iterations we reach, we reach significantly smaller gaps using gradient methods: more than 12 times smaller with a gradient-based choice of ρ and 6 times smaller with an adaptive ρ.

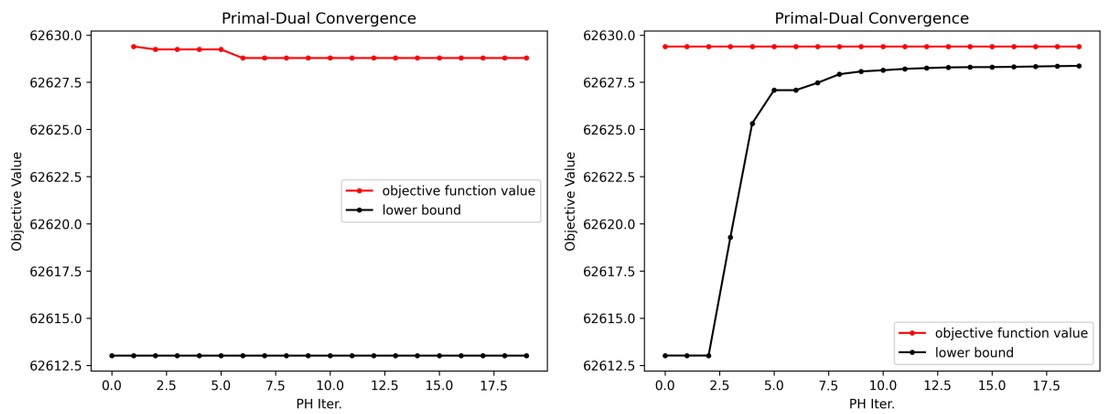

Figure 1: Comparison between convergences for cost-based ρ (left) and gradient-based ρ (right)

We report in figure 1 PH convergence for two selections of ρ values: cost-based and gradient-based. We observe two substantially different behaviors for the primal-dual convergence when using the cost-based ρ or the gradient-based ρ setting. Firstly, in the case of cost-based ρ, the lower bound does not improve at all over the 20 iterations of the algorithm, remaining the

trivial bound calculated at iteration 0. The upper bound decreases slightly. For gradient-based ρ, however, the lower bound improves significantly, while the upper bound remains unchanged. As the lower bound is calculated by the Lagrangian relaxation in the `lagrangian` spoke, these observations are consistent with the previous theoretical discussion. More precisely, the primary objective of the cost-based ρ setting is the convergence of decision variables. It results in a more aggressive approach to primal convergence, to the detriment of dual convergence. Conversely, the gradient-based method aims to achieve a good behavior of dual price vectors: this is illustrated by an efficient computation of the lower bound, which is much improved compared to the cost-based setting.

The iterative method for computing gradient-based ρ values can be implemented with different update criteria as described section 4.

- No criterion: the penalty parameter is updated at each PH iteration following (6).
- Primal criterion: the penalty parameter is updated based on a scaled primal norm.
- Dual criterion: the penalty parameter is updated based on a scaled dual norm.
- Primal-dual criterion: the penalty parameter is updated based on primal and dual residuals norms.

	no crit	primal crit	dual crit	primal-dual crit
wall clock time (s.)	1055.79	886.73	905.76	903.51
PH LB	62627.10	62628.31	62628.36	62628.29
PH UB	62629.40	62629.40	62629.40	62629.40
EF	62628.40	62628.40	62628.40	62628.40
Abs. Gap	2.29	1.09	1.03	1.11
Rel. Gap	0.004%	0.002%	0.002%	0.002%

Table 2: Experiments with different update criteria for an adaptive selection of ρ

We report the results of our second experiment in table 2. First, we observe that the run-time of the ρ adaptive selection without update criteria is longer than that of the other selections. As in the previous case, this is due to the fact that we update ρ at each iteration of PH, whereas we only do so for certain iterations in the other cases. Regarding the relative gaps we obtain, there is a small variability but no major difference. We observe that with these gradient methods, we obtain an efficient lower bound very close to the optimal value for extensive form.

6 Conclusions

We have introduced new methods for selecting the quadratic penalty parameter ρ in the Progressive Hedging algorithm in order improving several aspects of convergence. More specifically, we combine the heuristic proposed by Watson and Woodruff [29] with the computation of the gradient of the objective function to obtain a new variable-dependent formulation of the penalty parameter ρ, which results in more efficient convergence. This extends the Watson and Woodruff hueristic to situations where some or all non-anticipative variables do not appear in the objective function. By continuously relaxing the variables, we also enable this method to be used for mixed-integer programs.

We then propose an iterative way of calculating this parameter based on a first-order condition. This adaptive selection takes place at each step of the algorithm and take into account the current values of non-anticipative variables. It enables ρ values to increase as the algorithm progresses, preventing convergence from stagnating and enhancing performance. To improve the updating of the penalty parameter during PH progression, we introduce criteria based on the behavior of the primal and dual variables. We also combine a adaptive ρ update scheme introduced for ADMM [6] with the gradient-based method of selecting ρ. These criteria allow us to retain the value of ρ if it is still relevant to the convergence of the algorithm towards the optimal solution.

We report the results of experiments carried out using our methods to compute and updateρ, applied to the unit commitment problem. The results suggest that the methods can be effective.

A number of issues remain for further research, such as study of discontinuities in the objective function. Another topic of interest is mixed integer problems with weak relaxations.

References

[1] S. Atakan and S. Sen. "A Progressive Hedging based branch-and-bound algorithm for mixed-integer stochastic programs". In: *Computational Management Science* 15.3 (Oct. 2018), pp. 501–540.

[2] D. P. Bertsekas. "Chapter 2 - The Method of Multipliers for Equality Constrained Problems". In: *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press, 1982, pp. 95–157. ISBN: 978-0-12-093480-5. DOI: https://doi.org/10.1016/B978-0-12-093480-5.50006-4. URL: https://www.sciencedirect.com/science/article/pii/B9780120934805500064.

[3] D. P. Bertsekas. "Multiplier methods: A survey". In: *Automatica* 12.2 (1976), pp. 133–145. ISSN: 0005-1098. DOI: https://doi.org/10.1016/0005-1098(76)90077-7. URL: https://www.sciencedirect.com/science/article/pii/0005109876900777.

[4] J. R. Birge and F. Louveaux. *Introduction to stochastic programming*. Springer series in operations research. Literaturverz. S. 387 - 410. New York: Springer, 1997. XIX, 421. ISBN: 0387982175.

[5] N. Boland, J. Christiansen, B. Dandurand, A. Eberhard, J. Linderoth, and J. Luedtke. "Combining Progressive Hedging with a Frank-Wolfe Method to Compute Lagrangian Dual Bounds in Stochastic Mixed-Integer Programming". In: (Feb. 2017). arXiv: 1702.00880 [math.OC].

[6] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers". In: *Foundations and Trends® in Machine Learning* 3.1 (2011), pp. 1–122. ISSN: 1935-8237. DOI: 10.1561/2200000016. URL: http://dx.doi.org/10.1561/2200000016.

[7] M. L. Bynum, G. A. Hackebeil, W. E. Hart, C. D. Laird, B. L. Nicholson, J. D. Siirola, J.-P. Watson, and D. L. Woodruff. *Pyomo – optimization modeling in Python*. English. 3rd edition. Vol. 67. Springer Optim. Appl. Cham: Springer, 2021. ISBN: 978-3-030-68927-8; 978-3-030-68930-8; 978-3-030-68928-5. DOI: 10.1007/978-3-030-68928-5.

[8] K. Cheung, D. Gade, C. Silva-Monroy, S. M. Ryan, J.-P. Watson, R. J.-B. Wets, and D. L. Woodruff. "Toward scalable stochastic unit commitment". In: *Energy Systems* 6.3 (Sept. 2015), pp. 417–438.

[9] K. J. Cormican, D. P. Morton, and R. K. Wood. "Stochastic Network Interdiction". In: *Oper. Res.* 46.2 (1998), pp. 184–197. DOI: 10.1287/opre.46.2.184.

[10] I. I. Cplex. "V12. 1: User's Manual for CPLEX". In: *International Business Machines Corporation* 46.53 (2009), p. 157.

[11] T. G. Crainic, X. Fu, M. Gendreau, W. Rei, and S. W. Wallace. "Progressive hedging-based metaheuristics for stochastic network design". In: 58 (2011), pp. 114–124. ISSN: 0028-3045. DOI: 10.1002/net.20456.

[12] Y. Fan and C. Liu. "Solving Stochastic Transportation Network Protection Problems Using the Progressive Hedging-based Method". In: *Networks and Spatial Economics* 10.2 (June 2010), pp. 193–208.

[13] R. Fourer, D. M. Gay, and B. W. Kernighan. "A Modeling Language for Mathematical Programming". In: 36 (1990), pp. 519–554. ISSN: 0025-1909. DOI: 10.1287/mnsc.36.5.519.

[14] D. Gade, G. Hackebeil, S. M. Ryan, J. Watson, R. J. Wets, and D. L. Woodruff. "Obtaining lower bounds from the progressive hedging algorithm for stochastic mixed-integer programs". In: *Math. Program.* 157.1 (2016), pp. 47–67. DOI: 10.1007/s10107-016-1000-z.

[15] B. S. He, H. Yang, and S. L. Wang. "Alternating Direction Method with Self-Adaptive Penalty Parameters for Monotone Variational Inequalities". In: *Journal of Optimization Theory and Applications* 106.2 (Aug. 2000), pp. 337–356.

[16] H. Held, R. Hemmecke, and D. L. Woodruff. "A decomposition algorithm applied to planning the interdiction of stochastic networks". In: 52 (2005), pp. 321–328. ISSN: 0894-069X. DOI: 10.1002/nav.20079.

[17] E. Jones, T. Oliphant, and P. Peterson. "SciPy: Open Source Scientific Tools for Python". In: (Jan. 2001).

[18] V. Kaisermayer, D. Muschick, M. Horn, and M. Gölles. "Progressive hedging for stochastic energy management systems". In: *Energy Systems* 12.1 (Feb. 2021), pp. 1–29.

[19] B. Knueven, D. Mildebrath, C. Muir, J. D. Siirola, J.-P. Watson, and D. L. Woodruff. *A Parallel Hub-and-Spoke System for Large-Scale Scenario-Based Optimization Under Uncertainty*. 2020.

[20] B. Knueven, J. Ostrowski, and J.-P. Watson. "On Mixed-Integer Programming Formulations for the Unit Commitment Problem". In: (2020). ISSN: 1091-9856. DOI: 10.1287/ijoc.2019.0944.

[21] O. Listes and R. Dekker. "A Scenario Aggregation–Based Approach for Determining a Robust Airline Fleet Composition for Dynamic Capacity Allocation". In: 39 (2005), pp. 367–382. ISSN: 0041-1655. DOI: 10.1287/trsc.1040.0097.

[22] A. Løkketangen and D. L. Woodruff. "Progressive hedging and tabu search applied to mixed integer (0,1) multistage stochastic programming". In: *Journal of Heuristics* 2.2 (Sept. 1996), pp. 111–128.

[23] J. M. Mulvey and H. Vladimirou. "Applying the progressive hedging algorithm to stochastic generalized networks". English. In: *Annals of Operations Research* 31 (1991), pp. 399–424. ISSN: 0254-5330. DOI: 10.1007/BF02204860.

[24] R. T. Rockafellar and R. J.-B. Wets. "Scenarios and Policy Aggregation in Optimization Under Uncertainty". In: 16 (1991), pp. 119–147. ISSN: 0364-765X. DOI: 10.1287/moor.16.1.119.

[25] J. S. Rodriguez, C. D. Laird, and V. M. Zavala. "Scalable preconditioning of block-structured linear algebra systems using ADMM". In: 133 (2020), p. 106478. ISSN: 0098-1354. DOI: 10.1016/j.compchemeng.2019.06.003.

[26] S. M. Ryan, R. J.-B. Wets, D. L. Woodruff, C. Silva-Monroy, and J.-P. Watson. *Toward scalable, parallel progressive hedging for stochastic unit commitment*. 2013. DOI: 10.1109/pesmg.2013.6673013.

[27] S. Takriti, J. R. Birge, and E. Long. "A stochastic model for the unit commitment problem". In: 11 (1996), pp. 1497–1508. ISSN: 0885-8950. DOI: 10.1109/59.535691.

[28] S. L. Wang and L. Z. Liao. "Decomposition Method with a Variable Parameter for a Class of Monotone Variational Inequality Problems". In: *Journal of Optimization Theory and Applications* 109.2 (May 2001), pp. 415–429.

[29] J. Watson and D. L. Woodruff. "Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems". In: *Comput. Manag. Sci.* 8.4 (2011), pp. 355–370. DOI: 10.1007/s10287-010-0125-4.

[30] J. Watson, D. L. Woodruff, and W. E. Hart. "PySP: modeling and solving stochastic programs in Python". In: *Math. Program. Comput.* 4.2 (2012), pp. 109–149. DOI: 10.1007/s12532-012-0036-1.