Asynchronous Projective Hedging for Stochastic Programming

Jonathan Eckstein†  Jean-Paul Watson‡  David L. Woodruff§

October 30, 2018

Abstract

This paper proposes a decomposition algorithm for multistage stochastic programming that resembles the progressive hedging method of Rockafellar and Wets, but is capable of asynchronous parallel operation without sacrificing theoretical convergence in the convex case. Perhaps more importantly, each iteration of the decomposition method may process only a subset of the possible scenarios. The method is an application of a class of projective monotone operator splitting methods recently proposed by Combettes and Eckstein. We give a derivation and convergence proof of the method in the case that the problem is convex and the feasible set is compact, subject to some standard regularity conditions. We close by comparing the resulting algorithm to progressive hedging on some standard stochastic linear programming test problems.
1 Introduction

The progressive hedging (PH) method for stochastic programming is a decomposition algorithm for multistage convex stochastic programming problems introduced by Rockafellar and Wets [18]. Consider a stochastic programming problem defined over a finite multistage scenario tree with \( n \) leaf nodes. We henceforth use the term “scenario” to refer to an entire path through this tree from the root to one of the leaves. Progressive hedging cycles through the following steps:

1. Solve each of the \( n \) scenarios as a separate subproblem, with a quadratic perturbation to the objective function.

2. For each non-leaf node in the scenario tree, average the portions of the scenario solutions that correspond to it.


The quadratic perturbation in each scenario subproblem includes both a linear Lagrange multiplier term and a quadratic penalty for deviating from the averaged solution found in step 2 of the previous iteration.

Progressive hedging is a naturally parallel algorithm, since the \( n \) subproblems in step 1 may be solved concurrently, and the remaining calculations are quite simple and may also if necessary be executed in parallel using standard communication patterns whose time requirements grow only logarithmically with the number of processors. However, progressive hedging is also strongly synchronous, in that every scenario subproblem must be fully solved before an iteration can be completed. This synchronousness has a number of drawbacks: for one thing, even at iterations in which a few subproblems diverge significantly from the globally averaged solution, all of them must be processed. Furthermore, if the run times of the subproblems are highly variable, most processors will accumulate significant idle time while waiting for the slowest subproblems to complete. This drawback is particularly acute when progressive hedging is applied to problems with integer variables, as in [22], since the subproblems are mixed-integer programs. Initially such applications of progressive hedging were largely heuristic, but later it has been shown [3, 10] that with suitable adjustments the technique can find useful Lagrangian lower bounds on the global integer-optimal solution value. Although PH has been implemented with asynchronous subproblem solvers in a heuristic setting — see for example [5, 16, 19] — the convergence theory given by Rockafellar and Wets is strongly synchronous.

This paper introduces a class of stochastic programming algorithms that resemble progressive hedging, but are inherently “block iterative” and asynchronous. Being block iterative means that only a subset of subproblems need to be solved at each iteration, a feature that Section 6 shows can result significant speedups over progressive hedging when there are a relatively small numbers of processors.

Our algorithms are an application of the block-iterative, asynchronous projective operator splitting method described in [7], which is essentially a simplification of one of the methods developed in [4]. Although the original convergence proof for progressive hedging in [18] is self-contained, progressive hedging is an application of the alternating direction
method of multipliers (ADMM) [8, 9] in a suitably formulated space with a specially chosen inner product. As noted in [9], the ADMM is in turn an application of a ubiquitous general algorithmic template known as Douglas-Rachford splitting [15]. Broadly speaking, our new class of algorithms is obtained by substituting the new asynchronous projective splitting algorithm of [7] for the more traditional Douglas-Rachford approach. However, there are numerous differences in the details: for example, progressive hedging is obtained by applying a two-operator splitting scheme in an $n$-way product space, where $n$ is the number of scenarios. Our new approach uses an $n$-operator splitting scheme. Because it is an application of asynchronous projective operator splitting, we call our new method “Asynchronous Projective Hedging” (APH).

2 Notation and Problem Setup

Consider a stochastic programming problem defined on a finite scenario tree $\mathcal{T}$ with stages indexed by $t \in 1..T$, and $n$ leaf nodes indexed by $i \in 1..n$. Let $\mathcal{T}_t$ denote the set of tree nodes at each stage $t \in 1..T$, and for each $N \in \mathcal{T}_t$, we let $\mathcal{S}(N) \subseteq 1..n$ denote the set of leaf nodes that have $N$ as an ancestor. The sets $\{\mathcal{S}(N) \mid N \in \mathcal{T}_t\}$ comprise a partition of $1..n$ for each $t \in 1..T$. For each $i \in 1..n$ we denote the probability of leaf node $i$ by $\pi_i \in (0,1]$, subject to $\sum_{i=1}^n \pi_i = 1$. For each node $N$ in the tree, we also let $\pi(N) = \sum_{i \in \mathcal{S}(N)} \pi_i$ denote its probability. We assume that all zero-probability nodes have already been pruned from the scenario tree.

For simplicity (and without loss of generality), we assume that every stage-$t$ node in the tree has the same number of decision variables $m_t \geq 1$, and we let $m = \sum_{t=1}^T m_t$ denote the full number of decision variables associated with each scenario. For each $i \in 1..n$, let $\mathcal{X}_i$ denote the space of vectors of possible decisions for scenario $i$, from all stages. We denote a generic element of $\mathcal{X}_i$ by $x_i$, and its subvector corresponding to each stage $t \in 1..T$ by $x_{it} \in \mathbb{R}^{m_t}$. We give $\mathcal{X}_i$ the inner product

$$\langle x_i, y_i \rangle_{\mathcal{X}_i} = \pi_i (x_i)^\top (y_i) = \pi_i \sum_{t=1}^T (x_{it})^\top (y_{it}), \quad (1)$$

We use the scaled inner product (1) to maximize the resemblance of our algorithm to the original progressive hedging method proposed in [18]. It is also possible to use an unweighted inner product for $\mathcal{X}_i$ in the analysis below, resulting in a very similar algorithm development, but with different averaging operations.

Let $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$, under the inner product induced by (1),

$$\langle x, y \rangle_{\mathcal{X}} = \sum_{i=1}^n \langle x_i, y_i \rangle_{\mathcal{X}_i} = \sum_{i=1}^n \pi_i (x_i)^\top (y_i) = \sum_{i=1}^n \pi_i \sum_{t=1}^T (x_{it})^\top (y_{it}). \quad (2)$$

One may envision the elements of $\mathcal{X}$ as expanding decision vectors defined on the scenario tree into a “grid” by making $|\mathcal{S}(N)|$ replicas of the decision variables at each node $N$ in the tree. Thus, vectors in $\mathcal{X}$ have $n$ sets of decision variables for each stage $t$, regardless of the number of tree nodes $|\mathcal{T}_t|$ at stage $t$. 

3
Next, let \( m' = m - m_T = \sum_{t=1}^{T-1} m_t \). For each \( i \in 1..n \), let \( Z_i \) denote the space of all decision variables for scenario \( i \), except those for stage \( T \). We use a similar indexing scheme and inner product to \( X_i \), so that

\[
\langle z_i, u_i \rangle_{Z_i} = \pi_i(z_i)\top(u_i) = \pi_i \sum_{t=1}^{T-1} (z_{it})\top(u_{it}).
\] (3)

Let \( Z = Z_1 \times \cdots \times Z_n \) with the inner product induced by (3),

\[
\langle z, u \rangle_Z = \sum_{i=1}^{n} \langle z_i, u_i \rangle_{Z_i} = \sum_{i=1}^{n} \pi_i(z_i)\top(u_i) = \sum_{i=1}^{n} \pi_i \sum_{t=1}^{T-1} (z_{it})\top(u_{it}).
\] (4)

Vectors in \( Z \) have the same “grid” interpretation as vectors in \( X \), except that the are missing the variables for the final stage. Essentially, \( X \) is \( \mathbb{R}^{mn} \) and \( Z \) is \( \mathbb{R}^{m'n} \), but with scenario-weighted inner products.

For each \( i \in 1..n \), let \( M_i \) denote the linear map \( X_i \rightarrow Z_i \) that simply drops that last-stage variables, \( M_i(x_{i1}, \ldots, x_{iT}) = (x_{i1}, \ldots, x_{i,T-1}) \). We then let \( M : X \rightarrow Z \) be the linear map given by

\[
M : ((x_{i1}, \ldots, x_{iT}), \ldots, (x_{n1}, \ldots, x_{nT})) \mapsto (M_1x_1, \ldots, M_nx_n)
\] 
\[
= ((x_{11}, \ldots, x_{1,T-1}), \ldots, (x_{n1}, \ldots, x_{n,T-1})),
\] (5)

which simply drops all the last-stage variables from its argument.

As in [18], we define a linear subspace \( N \) of \( Z \) by

\[
N = \{ z \in Z \mid (\forall t \in 1..T) (\forall N \in \mathcal{T}_t) (\forall i, j \in \mathcal{S}(N)) : z_{it} = z_{jt} \}.
\] (6)

Vectors in \( z \in N \) are nonanticipative in the sense that \( z_{it} = z_{jt} \) whenever scenarios \( i \) and \( j \) are indistinguishable at stage \( t \). They thus correspond to implementable plans for stages \( 1, \ldots, T - 1 \).

For each \( i \in 1..n \), let \( h_i : \mathbb{R}^m \rightarrow \mathbb{R} \cup \{+\infty\} \) be a closed proper convex function and consider the optimization problem

\[
\min_{x \in X} \sum_{i=1}^{n} \pi_i h_i(x_i)
\] 

\( \text{ST} \quad Mx \in N. \) (7)

This optimization model can subsume any convex stochastic programming problem defined on the scenario tree \( \mathcal{T} \), in the following manner: within the context of the “clairvoyant” situation in which one knows that leaf node \( i \in 1..n \) of the tree will occur, define \( h_i(x_i) = +\infty \) whenever \( x_i \) is infeasible, and otherwise let \( h_i(x_i) \) be the total cost of the plan described by \( x_i \). In particular, \( h_i(x_i) \) will be \( +\infty \) if \( x_i \) violates any constraint within a stage or any coupling constraint between stages. For all scenarios \( i \in 1..n \), such constraints are embedded within the objective function of (7), while the constraint in (7) requires the selection of an implementable, non-clairvoyant plan.
One can now derive the progressive hedging algorithm by defining convex functions $F : X \to \mathbb{R} \cup \{+\infty\}$ and $G : Z \to \mathbb{R} \cup \{+\infty\}$ as follows:

\[
F(x) = \sum_{i=1}^{n} \pi_i h_i(x_i) \quad \quad G(z) = \begin{cases} 
0, & z \in \mathcal{N} \\
+\infty, & z \notin \mathcal{N}.
\end{cases}
\]

Problem (7) may then be expressed as simply

\[
\min_{x \in X} \{ F(x) + G(Mx) \},
\]

which is a standard form for applying the ADMM. Doing so while keeping in mind the inner products defined in (2) and (4), one may derive the progressive hedging method.

Any method using the function $F$ defined in (8) will naturally tend to be synchronous. So, instead of using (8) and converting the stochastic program into the form (9), we will instead model the stochastic program through the $n$-way decompositional model problem

\[
\min_{z \in \mathcal{H}_0} \left\{ \sum_{i=1}^{n} f_i(L_i z) \right\},
\]

where $\mathcal{H}_0, \ldots, \mathcal{H}_n$ are real Hilbert spaces with respective inner products $\langle \cdot, \cdot \rangle_{\mathcal{H}_0}, \ldots, \langle \cdot, \cdot \rangle_{\mathcal{H}_n}$, and, for all $i \in 1..n$, $f_i : \mathcal{H}_i \to \mathbb{R} \cup \{\infty\}$ is a closed proper convex function and $L_i : \mathcal{H}_0 \to \mathcal{H}_i$ is a bounded linear map. We assign the elements of this model as follows:

S1. $\mathcal{H}_0 = \mathcal{N}$, the nonanticipativity subspace, under the inner product $\langle \cdot, \cdot \rangle_Z$ from (4).

S2. For all $i \in 1..n$, we let $\mathcal{H}_i = \mathcal{Z}_i$, thus using the inner product (3).

S3. For all $i \in 1..n$, we define $L_i$ by $L_i : (z_1, \ldots, z_n) \mapsto z_i$, that is, $L_i$ selects only the portion of its argument corresponding to scenario $i$.

S4. For all $i \in 1..n$, we define $f_i$ by

\[
f_i(z_i) = \pi_i \min \left\{ h_i\left((z_i, x_{iT})\right) \mid x_{iT} \in \mathbb{R}^{m_T} \right\},
\]

that is, given the values of all decision variables $z_i$ for the scenario $i$ except those for the last stage, $f_i$ optimizes over the last-stage variables and returns the resulting objective value scaled by $\pi_i$.

This approach to applying (10) to stochastic programming, in which we implicitly minimize over the last-stage variables within the functions $f_i$, may seem slightly unnatural, but it is necessary to avoid generating undesired proximal terms associated with the last-stage variables in the algorithm we will develop below.

We make the following blanket assumption for the remainder of this paper:

**Assumption 1** For all $i \in 1..n$, $\text{dom } h_i = \{ x_i \in \mathbb{R}^m \mid h_i(x_i) < \infty \}$ is compact.

This assumption will allow us to model any finite-dimensional stochastic programming problem on the tree $\mathcal{T}$ whose feasible region is closed and bounded, a very weak restriction in most practical settings. It is also possible to relax this restriction, but we adopt here in the interest of simplicity.
Proposition 2 Under Assumption 1, the functions \( f_i \) defined by (11) are closed proper convex for all \( i \in 1..n \).

Proof. Closedness is established in [2, Lemma 1.29], while convexity is established in [2, Proposition 8.26]. □

Proposition 3 Suppose that \( \mathcal{H}_0, \ldots, \mathcal{H}_n, f_1, \ldots, f_n, \) and \( L_1, \ldots, L_n \) are constructed as in items S1-S4 above and Assumption 1 holds. Then a vector \( z^* \in \mathbb{R}^{m'} \) is an optimal solution of the problem (10) if and only if it is of the form \( z^* = Mx^* \), where \( x^* \) is an optimal solution of (7) and \( M \) is as defined in (5).

Proof. Under Assumption 1, problem (7) must attain its minimum value, since its feasible region is the intersection of the compact set \( \prod_{i=1}^n \text{dom } h_i \) with the (necessarily closed) linear subspace \( \{ x \in \mathcal{X} \mid Mx \in \mathcal{N} \} \). Thus, it is the minimum value of a closed (lower semicontinuous) function over a compact set and must be attained; see for example [2, Theorem 1.28].

Under the stated hypotheses, problem (10) must also attain its minimum, because, for each \( i \in 1..n \),

\[
\text{dom } f_i = \left\{ z_i \in \mathbb{R}^{m'} \mid \exists x_{iT} \in \mathbb{R}^{m_T} : (z_i, x_{iT}) \in \text{dom } h_i \right\} = M_i \text{dom } h_i. \quad (12)
\]

Since \( \text{dom } h_i \) is compact, this set is also compact. By Proposition 2, the functions \( f_i \) are closed proper convex. The feasible region of problem (10), namely

\[
\mathcal{F} = \{ z \in \mathcal{N} \mid (\forall i \in 1..n) L_i z \in \text{dom } f_i \},
\]

is clearly closed since the \( \text{dom } f_i \) are compact. It also cannot contain an unbounded sequence of points \( \{ z^k \} \), since at least one of the corresponding sequences \( \{ L_i z^k \}, i \in 1..n \), would then have to be unbounded and could not be contained in \( \text{dom } f_i \). Therefore \( \mathcal{F} \) is compact since \( \mathcal{N} \) is finite dimensional. Since the \( f_i \) are lower semicontinuous (closed), it is easily seen that \( \sum_{i=1}^n f_i \circ L_i \) must also be lower semicontinuous. Hence, problem (10) also involves minimization of a lower semicontinuous function over a compact set and attains its minimum. We then calculate

\[
\min_{z \in \mathcal{F}_0} \left\{ \sum_{i=1}^n f_i(L_iz) \right\} = \min_{z \in \mathcal{N}} \left\{ \sum_{i=1}^n f_i(z_i) \right\}
= \min_{z \in \mathcal{N}} \left\{ \sum_{i=1}^n \pi_i \min \left\{ h_i((z_i, x_{iT})) \mid x_{iT} \in \mathbb{R}^{m_T} \right\} \right\}
= \min\left\{ \sum_{i=1}^n \pi_i h_i((z_i, x_{iT})) \mid z \in \mathcal{N}, (\forall i \in 1..n) x_{iT} \in \mathbb{R}^{m_T} \right\}
= \min\left\{ \sum_{i=1}^n \pi_i h_i(x_i) \mid x \in \mathcal{X} : Mx \in \mathcal{N} \right\},
\]

so the objective values of the two problems are equal.
Suppose $x^*$ is an optimal solution of (7). If we let $z^* = Mx^*$, then for all $i \in 1..n$ we have $f(L_i z^*) \leq \pi_i h_i(L_i z^*, x_{iT}^*) = \pi_i h_i(x^*_i)$. Summing, we have $\sum_{i=1}^n f_i(L_i z^*) \leq \sum_{i=1}^n \pi_i h_i(x^*_i)$, which in view of the just-established equivalence of optimal values means that $z^*$ is optimal for (10). Conversely, let $z^*$ be any optimal solution of (10), consider for each $i \in 1..n$ the problem $\min_{x_i T \in R^{mT}} \{ h_i((z^*_i, x_{iT})) \}$, which must be attained at some $x_{iT}^* \in R^{mT}$ since it is again the minimum of a lower semicontinuous function over a compact set. In view of the equality of optimal values of the two problems already established, $x^* = ((z^*_1, x_{1T}^*), \ldots, (z^*_n, x_{nT}^*))$ must be an optimal solution of (7), and we clearly have $Mx^* = z^*$. \hfill \Box

### 3 A General Asynchronous Splitting Method

The primary purpose of this section is to state an abstract asynchronous splitting algorithm from [7] that can be used to solve (10). This algorithm is stated in terms of monotone operator inclusions, so we first need to convert (10) into such a form, and state regularity conditions under which the two problem forms are equivalent.

**Assumption 4** Assume that at least one of the following holds:

1. The functions $h_1, \ldots, h_n$ are polyhedral, or
2. There exists $\bar{x} \in X$ such that, for all $i \in 1..n$,
   \begin{equation}
   (\bar{x}_{i1}, \ldots, \bar{x}_{iT-1}) \in \text{ri } M_i(\text{dom } h_i),
   \end{equation}
   where the operator $\text{ri}$ denotes taking the relative interior of a set.

Assumption 4(1) will hold in the case of linear stochastic programming problems, while the nonlinear case is handled by Assumption 4(2), which is essentially a Slater-class constraint qualification. For a typical nonlinear convex stochastic programming formulation translated into (10) through S1-S4, the assumption essentially states that there exists a point that is strictly feasible with respect to all inequality constraints at stages 1 through $T - 1$.

By [2, Corollary 6.15], the set $\text{ri } M_i(\text{dom } h_i)$ on the right-hand side of (13) is identical to $M_i(\text{ri } \text{dom } h_i)$. It is possible to substitute less restrictive assumptions for Assumption 4, but we use the form above to keep the analysis relatively simple.

**Proposition 5** In the context of problem (10), suppose that $z \in \mathcal{H}_0$ satisfies
\begin{equation}
0 \in \sum_{i=1}^n L_i^* \partial f_i(L_i z),
\end{equation}
where the summation sign denotes the Minkowski sum of sets, $L_i^*$ denotes the adjoint linear operator of $L_i$, and $\partial g$ denotes the subgradient mapping of a convex function $g$. Then $z$ is an optimal solution of (10). Under Assumption 1 and the setup S1-S4, if one subsequently lets, for all $i \in 1..n$,
\begin{equation}
x_{iT} \in \text{Arg min}_{\xi \in R^{mT}} \{ h_i(L_i z, \xi) \} \quad (\forall t \in 1..(T-1)) \quad x_{it} = z_{it},
\end{equation}
Algorithm 1: Abstract algorithm for problem (14)

then the resulting vector $x \in X$ is optimal for (7).

Conversely, suppose that assumptions 1 and 4 hold, and $x$ is an optimal solution to (7). Then $z = Mx$ must satisfy the optimality condition (14).

Parts of the proof of this result are somewhat technical, so we defer it to Appendix A. Our algorithm is based on finding a solution to the optimality conditions (14), which is always a sufficient condition for optimality; if Assumption 4 holds, the condition is also necessary.

Algorithm 1 describes an algorithm to solve the monotone inclusion problem (14). On lines 1 and 12, $W$ denotes the linear subspace of $\mathcal{H}_1 \times \cdots \times \mathcal{H}_n$ given by

$$W = \left\{ w = (w_1, \ldots, w_n) \in \mathcal{H}_1 \times \cdots \times \mathcal{H}_n \mid \sum_{i=1}^n L_i^* w_i = 0 \right\}. \quad (16)$$

On line 12, proj$_W$ denotes the orthogonal projection map from $Z = \mathcal{H}_1 \times \cdots \times \mathcal{H}_n$ onto $W$.

**Proposition 6** For any closed proper convex functions $f_i : \mathcal{H}_i \to \mathbb{R} \cup \{+\infty\}$ and continuous linear maps $L_i : \mathcal{H}_0 \to \mathcal{H}_i$ (for $i \in 1..n$), Algorithm 1 develops the same sequences of iterates as [7, Algorithm 1] with (in the notation of [7]) $\hat{k} = 0$, the mappings $T_i$ set to $\partial f_i$ for all $i \in 1..n$, and the error parameter $e_k^i$ set to 0 for all $i \in 1..n$ and $k \geq 0$. 

8
Proof. Taking \( \hat{k} = 0 \), [7, Algorithm 1] reduces to a method identical to Algorithm 1, except that it has the single assignment

\[
(x^k_i, y^k_i) = \text{Prox}^\mu_{\partial f_i}(L_i z^{d(i,k)} + \mu_i w^{d(i,k)}_i + e^k_i)
\]

in place of lines 7-9. Setting \( T_i = \partial f_i \) and \( e^k_i = 0 \), and renumbering the parameters \( \mu_{ik} \), we may rewrite this equation as

\[
(x^k_i, y^k_i) = \text{Prox}^\mu_{\partial f_i}(L_i z^{d(i,k)} + \mu_i w^{d(i,k)}_i) = \text{Prox}^\mu_{\partial f_i}(r^k_i),
\]

where \( r^k_i \) is as computed in line 7. In [7], \( \text{Prox}^\mu_{\partial f}(r) \) is defined as the unique pair \((x, y)\) such that \( y \in \partial f(x) \) and \( x + \mu y = r \), so (17) is equivalent to the conditions

\[
y^k_i \in \partial f_i(x^k_i) \quad \quad \quad \quad \quad x^k_i + \mu_i y^k_i = r^k_i.
\]

The vectors \( x^k_i \) and \( y^k_i \) computed by lines 8 and 9 satisfy precisely these conditions, since lines 8 and 9 implement the standard operations for computing the proximal map of a closed convex function; see for instance [2, Example 23.3 and Definition 12.23]. □

Proposition 7 Suppose that (14) has a solution and that the parameter choices in Algorithm 1 conform to the following:

A1. There exist \( \nu, \bar{\nu} \in ]0, 2[ \) such that \( \nu \leq \nu_k \leq \bar{\nu} \) for all \( k \).

A2. For some scalars \( 0 < \underline{\mu} \leq \bar{\mu} \), we have \( \underline{\mu} \leq \mu_{ik} \leq \bar{\mu} \) for all \( k \geq 0 \) and \( i \in i..n \).

A3. For some fixed integer \( M \geq 1 \), each index \( i \) is selected for membership in \( I_k \) at least once in every \( M \) iterations, that is,

\[
(\forall j \geq 0): \left( \bigcup_{k=j}^{j+M-1} I_k \right) = 1..n.
\]

A4. For some fixed integer \( D \geq 0 \), we have \( k - d(i,k) \leq D \) for all \( i, k \) with \( i \in I_k \). That is, there is a constant bound on the extent to which the information \( z^{d(i,k)} \) and \( w^{d(i,k)} \) in line 7 may be “out of date”.

Then the sequences developed by Algorithm 1 has the properties below. Here, “\( \rightharpoonup \)” denotes convergence in the weak topology, which is equivalent to ordinary convergence in finite-dimensional spaces.

1. \( z^k \rightharpoonup z^\infty \), where \( z^\infty \) is some solution of (14)

2. \( (\forall i \in 1..n) w^k_i \rightharpoonup w^\infty_i \), where \( w^\infty_i \in T_i(L_i z^\infty) \), and \( \sum_{i=1}^n L^*_i w^\infty_i = 0 \)

3. \( (\forall i \in 1..n) x^k_i \rightharpoonup L_i z^\infty \)

4. \( (\forall i \in 1..n) y^k_i \rightharpoonup w^\infty_i \)
5. \( v^k \to 0 \)

6. \( u^k \to 0 \).

**Proof.** Proposition 6 asserts that Algorithm 1 generates the same sequence of iterates as Algorithm 1 of [7] with \( k = 0 \), \( T_i = \partial f_i \) for all \( i \in 1..n \), and \( e_i^k = 0 \) for all \( i \in 1..n \) and \( k \geq 0 \). The assumed operating conditions A1-A4 are identical to parts 1-4 of Assumption 2.1 of [7]. Since \( e_i^k = 0 \) for all \( i \in 1..n \) and \( k \geq 0 \), part 5 of Assumption 2.1 of [7] is also trivially satisfied. Thus, that entire assumption holds and the sequences developed by Algorithm 1 conform to the conclusions of Proposition 2.3 of [7]. Conclusions 1-4 then follow immediately, and conclusion 5 follows since Algorithm 1 sets \( v^k = \sum_{i=1}^n L_i^* y_i^k \) and Proposition 2.3 of [7] asserts that \( \sum_{i=1}^n L_i^* y_i^k \to 0 \).

Finally, we consider conclusion 6. First, we note that \( (L_1 z^\infty, \ldots, L_n z^\infty) \in W^\perp \), since for any \((w_1, \ldots, w_n) \in W\) we have

\[
\langle (L_1 z^\infty, \ldots, L_n z^\infty), (w_1, \ldots, w_n) \rangle = \langle (z^\infty, \ldots, z^\infty), (L_1^* w_1, \ldots, L_n^* w_n) \rangle = \sum_{i=1}^n \langle z^\infty, L_i^* w_i \rangle_{\mathcal{H}_i^0} = \langle z^\infty, 0 \rangle_{\mathcal{H}_0^0} = 0.
\]

Let \( a = (a_1, \ldots, a_n) \) be any element of \( \mathcal{H} = \mathcal{H}_1 \times \cdots \times \mathcal{H}_n \). Then

\[
\langle \text{proj}_W(a), (x_1^k, \ldots, x_n^k) - (L_1 z^\infty, \ldots, L_n z^\infty) \rangle \to 0
\]

because conclusion 3 asserts that \( (x_1^k, \ldots, x_n^k) \rightharpoonup (L_1 z^\infty, \ldots, L_n z^\infty) \). Using that orthogonal projections are self-adjoint, we then equivalently have

\[
\langle a, \text{proj}_W(x_1^k, \ldots, x_n^k) - \text{proj}_W(L_1 z^\infty, \ldots, L_n z^\infty) \rangle \to 0. \tag{19}
\]

Now, Algorithm 1 sets \( u^k = \text{proj}_W(x_1^k, \ldots, x_n^k) \), and we have \( \text{proj}_W(L_1 z^\infty, \ldots, L_n z^\infty) = 0 \) since we have already established that \( (L_1 z^\infty, \ldots, L_n z^\infty) \in W^\perp \). Therefore, (19) is equivalent to \( \langle a, u^k \rangle \to 0 \). Since the vector \( a \) was arbitrary, it follows that \( u^k \to 0 \) and conclusion 6 is established. \( \square \)

## 4 Deriving an Algorithm for Stochastic Programs

We now derive an algorithm for convex stochastic programming by applying Algorithm 1 to the setup described in items S1-S4 of Section 2. In order to apply this setup, we need to determine the form of the adjoint operators \( L_i^* \) and the subspace \( W \):

**Proposition 8** Under the choices of \( \mathcal{H}_0, \ldots, \mathcal{H}_n \) and \( L_1, \ldots, L_n \) specified in S1-S3,

1. For all \( i \in 1..n \) the adjoint mapping \( L_i^* : \mathcal{H}_i \to \mathcal{H}_0 = \mathbb{N} \) is given by

\[
L_i^* : w_i \mapsto \text{proj}_N(0, \ldots, 0, w_i, 0, \ldots, 0), \tag{20}
\]

where \( w_i \) appears in the \( i^{th} \) position in \((0, \ldots, 0, w_i, 0, \ldots, 0)\) and \( \text{proj}_N \) denotes the orthogonal projection map from \( \mathcal{Z} = \mathcal{H}_1 \times \cdots \times \mathcal{H}_n \) onto \( \mathbb{N} \).
2. For any vectors \( w_1 \in H_1, \ldots, w_n \in H_n \), one has \( \sum_{i=1}^{n} L_i^* w_i = \text{proj}_N(w_1, \ldots, w_n) \).

3. \( W = \mathcal{N}^\perp \), where \( W \) is the subspace defined in (16).

**Proof.** Consider any particular \( i \in 1..n \). The adjoint of \( L_i : H_0 \to H_i \) is the unique linear operator \( L_i^* : H_i \to H_0 \) with the property

\[
(\forall z \in H_0) \ (\forall w_i \in H_i) \ \langle z, L_i^* w_i \rangle_{H_0} = \langle L_i z, w_i \rangle_{H_i}.
\]

Using S1-S3, the above condition is equivalent to

\[
(\forall z \in \mathcal{N}) \ (\forall w_i \in H_i) \ \langle z, L_i^* w_i \rangle_Z = \pi_i(z_i) \langle z, (0, \ldots, 0, w_i, 0, \ldots, 0) \rangle_Z
\]

where the last equality holds because \( z \in \mathcal{N} \). Because \( \text{proj}_N(0, \ldots, 0, w_i, 0, \ldots, 0) \in Z \), we conclude that \( L_i^* \) has the form \( w_i \mapsto \text{proj}_N(0, \ldots, 0, w_i, 0, \ldots, 0) \). Since \( i \in 1..n \) was arbitrary, the first claim is established.

Now take any \( w_1 \in H_1, \ldots, w_n \in H_n \). Using the first claim, we have

\[
\sum_{i=1}^{n} L_i^* w_i = \sum_{i=1}^{n} \text{proj}_N(0, \ldots, 0, w_i, 0, \ldots, 0) = \text{proj}_N \left( \sum_{i=1}^{n} (0, \ldots, 0, w_i, 0, \ldots, 0) \right)
\]

and the second claim is established. Combining the definition of \( W \) from (16) and the second claim, we then have

\[
\mathcal{W} = \{ (w_1, \ldots, w_n) \in Z \mid \sum_{i=1}^{n} L_i^* w_i = 0 \} = \{ w \in Z \mid \text{proj}_N(w) = 0 \} = \mathcal{N}^\perp,
\]

and the third claim is established. □

It remains to consider how to perform the proximal computation on line 8 of Algorithm 1.

**Proposition 9** Under S1-S4, the computation of \( x_i^k \) on line 8 of Algorithm 1 for any \( i \in 1..n \) may be implemented by finding

\[
(x_i^k, x_{iT}^k) \in \text{Arg min}_{x_i \in \mathbb{R}^{m'}, x_{iT} \in \mathbb{R}^{m_T}} \left\{ h_i(x_i, x_{iT}) + \frac{1}{2\mu_{ik}} \| x_i - x_i^k \|_2^2 \right\},
\]

(21)

where \( \| \cdot \|_2 \) denotes the usual Cartesian \( L_2 \)-norm on \( \mathbb{R}^{m_T} \). That is, one finds a minimizer of the function in (21) over all \( (x_i, x_{iT}) \in \mathbb{R}^{m'} \times \mathbb{R}^{m_T} = \mathbb{R}^{m} \), and \( x_i^k \) consists of the first \( m' \) components of the that minimizer (those components not corresponding to the last stage).
Proof. Under S1-S4, line 8 of Algorithm 1 takes the form

\[ x_i^k = \arg \min_{x_i \in \mathcal{N}_i} \left\{ f_i(x_i) + \frac{1}{2\mu_{ik}} \| x_i - r_i^k \|_{2_i}^2 \right\} \]

\[ = \arg \min_{x_i \in \mathcal{N}_i} \left\{ \pi_i \min_{x_i \in \mathcal{N}_i} \{ h_i(x_i, x_iT) \} + \frac{1}{2\mu_{ik}} \| x_i - r_i^k \|_2 \right\} \]

\[ = \arg \min_{x_i \in \mathcal{N}_i} \left\{ \min_{x_i \in \mathcal{N}_i} \{ h_i(x_i, x_iT) \} + \frac{1}{2\mu_{ik}} \| x_i - r_i^k \|_2 \right\}, \]

Now, since

\[ \min_{x_i \in \mathcal{N}_i} \left\{ \min_{x_i \in \mathcal{N}_i} \{ h_i(x_i, x_iT) \} + \frac{1}{2\mu_{ik}} \| x_i - r_i^k \|_2 \right\} = \min_{x_i \in \mathcal{N}_i} \left\{ h_i(x_i, x_iT) + \frac{1}{2\mu_{ik}} \| x_i - r_i^k \|_2 \right\}, \]

the claim follows immediately.

\[ \square \]

Note that a minimizer of (21) is guaranteed to exist by Assumption 1, since each \( h_i \) is a closed (lower semicontinuous) function.

Considering that \( r_i^k = L_i z^d(i,k) + \mu_{ik} w_i^d(i,k) \) on line 7 of Algorithm 1, and using the form of \( L_i \) from S3, the minimand in (21) may be written

\[ h_i(x_i, x_iT) + \frac{1}{2\mu_{ik}} \| x_i - r_i^k \|_2 = h_i(x_i, x_iT) + \frac{1}{2\mu_{ik}} \| x_i - (z_i^d(i,k) + \mu_{ik} w_i^d(i,k)) \|_2 \]

\[ = h_i(x_i, x_iT) - (w_i^d(i,k))^\top (x_i) + \frac{1}{2\mu_{ik}} \| x_i - z_i^d(i,k) \|_2^2. \] (22)

Further, the calculation of \( y_i^k \) in line 9 of Algorithm 1 may be written

\[ y_i^k = \frac{1}{\mu_{ik}} (r_i^k - x_i^k) = \frac{1}{\mu_{ik}} (z_i^d(i,k) + \mu_{ik} w_i^d(i,k) - x_i^k) = w_i^d(i,k) + \frac{1}{\mu_{ik}} (z_i^d(i,k) - x_i^k). \] (23)

If we reverse the signs of the \{ \( w_i^d(i,k) \) \} and \{ \( z_i^d(i,k) \) \} sequences and replace \( \mu_{ik} \) with a reciprocal parameter \( \rho_{ik} = 1/\mu_{ik} \), we conclude that we may implement lines 7-9 of Algorithm 1 through the following calculations:

\[ (x_i^k, x_iT^k) \in \text{Arg min}_{x_i \in \mathcal{N}_i} \left\{ h_i(x_i, x_iT) + \left( w_i^d(i,k) \right)^\top (x_i) + \frac{\rho_{ik}}{2} \| x_i - z_i^d(i,k) \|_2^2 \right\} \] (24)

\[ y_i^k = w_i^d(i,k) + \rho_{ik} (z_i^k - x_i^k). \] (25)

Other than the potential delay encompassed in the iteration counter \( d(i,k) \), the minimization (24) is identical to the subproblem solved by progressive hedging [18], while (25) resembles the progressive hedging Lagrange multiplier update.
1 Start with any $z^0 \in \mathbb{N}$, $w^0 \in \mathbb{N}^+$ and $(x_i^{-1}, y_i^{-1} \in \mathbb{R}^m)_{i \leq 1..n}$
2 for $k = 0, 1, \ldots$ do
3   Select some $I_k \subseteq 1..n$
4   for $i \in I_k$ do
5     Select some integer $d(i, k) \in 0..k$
6     Select some scalar $\rho_{ik} > 0$
7     Find $(x_i^k, x_{iT}) \in \text{Arg min}_{x_i \in \mathbb{R}^m, x_{iT} \in \mathbb{R}^{m_T}} \left\{ h_i(x_i, x_iT) + \left( w_i d(i,k) \right)^T (x_i) + \frac{\rho_{ik}}{2} \| x_i - z_i d(i,k) \|^2 \right\}$
8     $y_i^k = w_i d(i,k) + \rho_{ik} (x_i^k - z_i d(i,k))$
9   end
10  for $i \not\in I_k$ do $(x_i^k, y_i^k) = (x_i^{k-1}, y_i^{k-1})$
11    $u^k = \text{proj}_{\mathbb{N}^+}(x_1^k, \ldots, x_n^k)$
12    $v^k = \text{proj}_{\mathbb{N}^+}(y_1^k, \ldots, y_n^k)$
13    $\tau_k = \sum_{i=1}^n \pi_i \left( \| u_i^k \|^2 + \| v_i^k \|^2 \right)$
14    if $\tau_k > 0$ then
15      Choose some $\nu_k : 0 < \nu_k < 2$
16      $\theta_k = \frac{\nu_k}{\tau_k} \max \left\{ 0, \sum_{i=1}^n \pi_i (L_i z_i^k - x_i^k)^T (w_i^k - y_i^k) \right\}$
17    else $\theta_k = 0$
18    $z^{k+1} = z^k + \theta_k v^k$
19    $w^{k+1} = w^k + \theta_k u^k$
20  end

Algorithm 2: Asynchronous projective hedging (APH) — Algorithm 1 specialized to the setup S1-S4.

Algorithm 2 displays the entire Asynchronous Projective Hedging (APH) method one obtains by specializing Algorithm 1, with the signs of $\{ u_i^k \}$ and $\{ z_i^k \}$ reversed and $\rho_{ik} = 1/\mu_{ik}$ for all $i \leq 1..n$ and $k \geq 1$, to the setup S1-S4. Because it is based on a projective splitting procedure and the delays $d(i, k)$ in lines 5-8 permit it to be implemented asynchronously, we call this method “asynchronous projective hedging”.

In Algorithm 2, the foregoing discussion establishes that lines 7-8 are equivalent to lines 7-9 of Algorithm 1 under the setup S1-S4, after reversing the signs of $\{ u_i^k \}$ and $\{ z_i^k \}$ and substituting the parameter $\rho_{ik} = 1/\mu_{ik}$ for $\mu_{ik}$. Proposition 8 guarantees that lines 11 and 12 are respectively equivalent to lines 12 and 13 of Algorithm 1, although the sign of $v^k$ is reversed from Algorithm 1 due to the reversed signs of the $y_i^k$. Line 13 computes the squared length of the hyperplane normal $\langle -v^k, u^k \rangle$ in the product norm induced by the norm $\langle \cdot, \cdot \rangle_z$ from (4) chosen for $\mathcal{H}_0 = \mathbb{N}$ and $\mathcal{W} = \mathbb{N}^+$. Line 16 is then equivalent to line 17 of Algorithm 1, considering the choice of the inner products on $\mathcal{H}_1, \ldots, \mathcal{H}_n$ in S2 and the sign reversals for $\{ u_i^k \}$ and $\{ z_i^k \}$. Finally, line 18 is equivalent to the $z$ update in line 19 of Algorithm 1 considering the sign reversal in $v^k$ induced by the sign reversal of the $\{ y_i^k \}$, while line 19 is equivalent to the $w$ update in line 20 of Algorithm 1 when considering the sign reversal of the $\{ w_i^k \}$ sequence.
To implement the algorithm, we need to perform the projections onto \( N^\perp \) and \( N \) specified on lines 11 and 12. Due to the probability-weighted form of the inner product \( \langle \cdot, \cdot \rangle_Z \), the projection \( \text{proj}_N \) consists of the same probability-weighted tree averaging operation as in the originally proposed form of the progressive hedging algorithm [18]. Since \( \text{proj}_{N^\perp} = \text{Id} - \text{proj}_N \), where “Id” represents the identity map, the complementary projection onto \( N^\perp \) involves a nearly identical calculation to projection onto \( N \). Specifically, we may implement line 12 by

\[
(\forall t \in 1..T-1) (\forall N \in \mathcal{T}_t) : \quad \text{compute } \bar{y}(N) = \frac{1}{\pi(N)} \sum_{i \in S(N)} \pi_i y_{it}^k, \text{ then } (\forall i \in S(n)) v_{it}^k = \bar{y}(N), \quad (26)
\]

whereas line 11 may be implemented by

\[
(\forall t \in 1..T-1) (\forall N \in \mathcal{T}_t) : \quad \text{compute } \bar{x}(N) = \frac{1}{\pi(N)} \sum_{i \in S(N)} \pi_i x_{it}^k, \text{ then } (\forall i \in S(n)) u_{it}^k = x_{it}^k - \bar{x}(N). \quad (27)
\]

**Proposition 10** Suppose that a stochastic programming problem formulated as (7) has a solution and Assumption 4 holds. Suppose that we operate Algorithm 2 in accordance with the following conditions:

1. There exist \( \nu, \bar{\nu} \in ]0,2[ \) such that \( \nu \leq \nu_k \leq \bar{\nu} \) for all \( k \).
2. For some scalars \( 0 < \rho \leq \bar{\rho} \), we have \( \rho \leq \rho_{ik} \leq \bar{\rho} \) for all \( k \geq 0 \) and \( i \in \{1..n\} \).
3. For some fixed integer \( M \geq 1 \), \( \bigcup_{j=M}^{\infty} I_k = 1..n \) for all \( j > 0 \).
4. For some fixed integer \( D \geq 0 \), we have \( k - d(i,k) \leq D \) for all \( i, k \) with \( i \in I_k \).

Then

\[
\begin{align*}
    z^k &\to x^\infty \\
    x^k &\to x^\infty \\
    v^k &\to 0 \\
    u^k &\to 0,
\end{align*} \quad (28)
\]

where \( x^\infty = M\bar{x}^* \) for some optimal solution \( \bar{x}^* \in \mathbb{R}^m \) of (7), that is, \( x^\infty \) is some optimal solution to (7) with the last-stage variables deleted.

For each \( i \in \{1..n\} \) let \( I_i = \{k \geq 0 \mid i \in I_k\} \), which is infinitely large by assumption 3, and let \( \{x_{iT}^k\} \) be any limit point of the sequence \( \{x_{iT}^k\}_{i \in I_i} \), where \( x_{iT}^k \) is the last-stage solution vector found in line 7 of Algorithm 2. Then the vector

\[
\bar{x}^\infty = \left(\left(x_i^\infty, x_{iT}^\infty\right)\right)_{i=1}^n = \left(\left(x_1^\infty, x_{1T}^\infty\right), \ldots, \left(x_n^\infty, x_{nT}^\infty\right)\right)
\]

is an optimal solution of (7).
Proof. First, Proposition 3 states that any solution \( \bar{x}^* \) of (7) has a corresponding solution \( z^* = M\bar{x}^* \) of (10). Since the Assumption 4 is in force, the second part of Proposition 5 implies that \( z^* \) must be a solution to the inclusion (14). Therefore a solution to (14) exists.

We have established that Algorithm 2 is a special case of Algorithm 1 under the setup S1-S4, with the signs of \( \{w^k_i\} \) reversed and \( \rho_{ik} = 1/\mu_{ik} \). Setting \( \mu = 1/\bar{\mu} \) and \( \bar{\mu} = 1/\rho \), condition 2 here is equivalent to condition A2 of Proposition 7 if \( \rho_{ik} = 1/\mu_{ik} \) for all \( i \in 1..n \) and \( k \geq 0 \). The remaining conditions here are identical to those of Proposition 7, so all of the assumptions of Proposition 7 hold. That proposition then immediately asserts that \( z^k \rightharpoonup z^\infty, x^k \rightharpoonup z^\infty, v^k \rightharpoonup 0 \), and \( u^k \rightharpoonup 0 \), where \( z^\infty \) is some solution of (14). By Proposition 5, such a point is necessarily a solution to (10), and so Proposition 3 guarantees it is of the form \( M\bar{x}^* \) for some optimal solution \( \bar{x}^* \) to (7). Since all the spaces are finite dimensional, weak and ordinary convergence are equivalent, and so we have established (28).

To establish the last claim, let \( \bar{x}^* = ((z^n_i, x^n_{iT}), \ldots, (z^\infty_i, x^*_{iT})) \) be an optimal solution of (7) such that \( z^\infty = M\bar{x}^* \). Consider any \( k \geq 0 \) and \( i \in I_k \). By the calculation in line 7, we must have for all \( k \geq 0 \) and \( i \in I_k \) that

\[
\begin{align*}
    h_i(x^k_i, x^*_iT) + (w^k_i)^\top(x^k_i) + \frac{\rho_{ik}}{2} \|x_i - z^k_i\|^2 & \\
    \leq h_i(z^\infty_i, x^*_iT) + (w^\infty_i)^\top(z^\infty_i) + \frac{\rho_{ik}}{2} \|z^\infty_i - z^k_i\|^2
\end{align*}
\]

Under assumption 4, \( d(i,k) \to \infty \) as \( k \to \infty \) for all \( i \in 1..n \), and from Proposition 7 we know that \( \{w^k\} \) converges to some \( w^\infty \in W \). Let \( \mathcal{K}_{i} \subseteq I_{i} \) denote an infinite set of indices such that \( \lim_{k \to \infty} k \in \mathcal{K}_{i} \{x^k_i\} = x^\infty_{iT} \). Taking limits over \( k \in \mathcal{K}_{i} \) in the above inequality and using that \( \{x^k\} \) and \( \{z^k\} \) both converge to the same limit \( z^\infty \), we conclude that

\[
\lim_{k \to \infty} \sup_{k \in \mathcal{K}_{i}} \{h(x^k_i, x^*_iT)\} + (w^\infty_i)^\top(z^\infty_i) \leq h_i(z^\infty_i, x^*_iT) + (w^\infty_i)^\top(z^\infty_i)
\]

Canceling the identical terms \( (w^\infty_i)^\top(z^\infty_i) \) and using that \( f_i \) is a lower semicontinuous function, we then have that

\[
h_i(z^\infty_i, x^*_iT) \leq \lim_{k \to \infty} \inf_{k \in \mathcal{K}_{i}} \{h(x^k_i, x^*_iT)\} \leq \lim_{k \to \infty} \sup_{k \in \mathcal{K}_{i}} \{h(x^k_i, x^*_iT)\} \leq h_i(z^\infty_i, x^*_iT).
\]

Multiplying by \( \pi_i \) and summing over \( i \), we obtain that

\[
\sum_{i=1}^{n} \pi_i h_i(z^\infty_i, x^*_iT) \leq \sum_{i=1}^{n} \pi_i h_i(z^\infty_i, x^*_iT) = \sum_{i=1}^{n} \pi_i h_i(\bar{x}^*_i)
\]

and thus that \( \bar{x}^\infty \) is an optimal solution to (7). \( \square \)

It is also possible to show that the common limit \( w^\infty \) of \( \{w^k\} \) and \( \{y^k\} \) is an appropriately defined dual solution of (7), but we omit this result as it is not required for our subsequent analysis and experiments.

The following result shows that if the algorithm encounters the condition \( \tau_k = 0 \) encountered in line 14, then it has reached exact convergence.
Proposition 11. In Algorithm 2, if \( u^k = 0 \) and \( v^k = 0 \) (or equivalently \( \tau_k = 0 \)) for any \( k \) sufficiently large that \( \bigcup_{\ell=0}^{k} I_\ell = 1..n \), then \( x^k = (x^1_1, \ldots, x^k_n) \) is an optimal solution of the problem formulation (10).

Proof. If \( \bigcup_{\ell=0}^{k} I_\ell = 1..n \), every subproblem has been processed at least once, and we have \( y_i^k \in \partial f_i(L_i x^k) = \partial f_i(x_i^k) \) for all \( i \in 1..n \). Since \( u^k = \text{proj}_{\bigcup_{\ell=0}^{k}}(x^k) \) from line 11 of the algorithm, the hypothesis \( u^k = 0 \) implies that \( x^k \in \mathbb{N} \). From line 12 and Proposition 8(2-3), we have \( v^k = \text{proj}_{\bigcup}(y_1^k, \ldots, y_n^k) = \sum_{i=1}^{n} L_i^* y_i^k \), so from the hypothesis \( v^k = 0 \) we obtain

\[
0 = \sum_{i=1}^{n} L_i^* y_i^k \in \sum_{i=1}^{n} L_i^* \partial f_i(L_i x^k),
\]

where the inclusion follows because we have already established that \( y_i^k \in \partial f_i(L_i x^k) \) for all \( i \in 1..n \). Therefore the sufficient optimality condition (14) holds and \( x \) is an optimal solution of (10).

In practice, we have never observed the condition \( u^k = 0 \) and \( v^k = 0 \) to hold exactly; however, since both \( u^k \to 0 \) and \( v^k \to 0 \) as established in Proposition 10, we can use small values of \( \|u^k\| \) and \( \|v^k\| \) as numerical termination conditions for the algorithm.

5 Including a Scaling Parameter

Consider two optimization problems \( \mathcal{P} \) and \( \tilde{\mathcal{P}} \), identical except that the objective function of \( \tilde{\mathcal{P}} \) is obtained by multiplying \( \mathcal{P} \)'s objective function by some positive constant \( C \). The PH algorithm has a property we may call parametric scale invariance, as follows: if we apply PH to \( \mathcal{P} \) and \( \mathcal{P} \) with respective penalty parameters \( \rho \) and \( C\rho \) and initialize the second algorithm at \( (z^0, \tilde{w}^0) = (z^0, Cw^0) \), it will generate respective sequences of iterates \( \{(z^k, w^k)\} \) and \( \{(\tilde{z}^k, \tilde{w}^k)\} \) that are equivalent in the sense that \( (\tilde{z}^k, \tilde{w}^k) = (z^k, Cw^k) \) will hold for all \( k \geq 0 \). Essentially, the two instances of PH will follow the same path, except that the dual iterates for \( \tilde{\mathcal{P}} \) will be scaled by \( C \). Thus, the penalty parameter \( \rho \) can be used to compensate for the relative scaling of the primal and dual variables.

Unfortunately, the APH algorithm as we have stated it so far lacks this desirable property. If we initialize \( (\tilde{z}^0, \tilde{w}^0) = (z^0, Cw^0) \), multiply all the proximal parameters for the \( \tilde{\mathcal{P}} \) instance by \( C \), and keep all the other parameters the same, running APH on \( \mathcal{P} \) and \( \tilde{\mathcal{P}} \) will not generate equivalent paths. The reason is that the projection operations in APH minimize the distance to a separating hyperplane in the primal-dual space of iterates \( (z^k, w^k) \). If the dual \( w^k \) iterates become larger (as must eventually happen if \( C > 1 \)), then they will become a larger fraction of the squared distance to each separating hyperplane, so APH will emphasize dual convergence over primal convergence for sufficiently large \( C \). Conversely, for \( C \) sufficiently small, the primal iterates \( z^k \) will start to dominate the distance to the separating hyperplane and APH will emphasize primal convergence over dual convergence.

In projective splitting algorithms, one can compensate for such effects by using a scaled norm for the projections, as discussed in [14]. In the case of the stochastic programming
setup developed in Section 2, this means selecting a scalar parameter \( \gamma > 0 \) and using the following inner product on the space \( H_0 \times H_1 \times \cdots \times H_n = \mathbb{R} \times \mathbb{Z} : \)

\[
\langle (z, w), (z', w') \rangle_\gamma = \sum_{i=1}^{n} \pi_i \cdot \left( \gamma \langle z_i, z_i' \rangle + \langle w_i, w_i' \rangle \right).
\] (29)

Essentially, all the primal components of the inner product are multiplied by \( \gamma \). Using the norm induced by this inner product to perform the hyperplane projections results in the following two changes to Algorithm 2:

Line 13 becomes:

\[
\tau_k = \sum_{i=1}^{n} \pi_i \left( \| u_i^k \|_2^2 + \gamma^{-1} \| u_i^k \|_2^2 \right)
\]

Line 18 becomes:

\[
z^{k+1} = z^k + \gamma^{-1} \theta_k v^k.
\]

Otherwise, Algorithm 2 remains unchanged.

These modifications make it possible to recover a form of parameterized scale invariance: for two hypothetical problems \( \mathcal{P} \) and \( \tilde{\mathcal{P}} \) as described above, if we initialize \( (\tilde{z}^0, \tilde{w}^0) = (z^0, Cw^0) \), multiply all the proximal parameters for the \( \tilde{\mathcal{P}} \) instance by \( C \), multiply its value of \( \gamma \) by \( C \), and leave all the other algorithm parameters unchanged, we will recover the path equivalence that \( (\tilde{z}^k, \tilde{w}^k) = (z^k, Cw^k) \) for all \( k \geq 0 \). More crucially, the parameter \( \gamma \) allows the user to adjust the APH algorithm’s relative emphasis on primal and dual convergence: increasing \( \gamma \) shifts emphasis towards primal convergence, and decreasing \( \gamma \) shifts emphasis toward dual convergence.

For the problem instances used in the experiments of the following section, we found APH to exhibit better practical performance than PH without introducing \( \gamma \) into the calculations. For other problem instances, however, using \( \gamma \neq 1 \) may be critical to the performance of APH.

6 Computational Experiments

We implemented APH within the PySP stochastic programming modeling environment [23], part of the Python-based Pyomo optimization modeling system [12, 11]. PySP has an implementation of PH that may be applied to problems of all types (convex, non-convex, mixed integer), and our APH implementation makes use of the same infrastructure. This results in high flexibility and broad availability of the software for subsequent research, perhaps at some cost in efficiency. The PySP software supports a mechanism referred to as bundling — see for example [22, 6] — in which each subproblem is a “bundle” composed of multiple stochastic programming scenarios. There are explicit nonanticipativity constraints within each bundle.

Our APH implementation is based on an earlier, heuristic attempt to implement PH in an asynchronous manner, with a simple “master-worker” processor organization. The “master” runs the entire coordination process, while the “workers” simply wait to receive subproblems from the master, solve them, and return the results.

Because the linear stochastic programming test problems we are studying here display minimal variability in subproblem solution time, we focus in these experiments on the “block-iterative” capability of the algorithm to process only a subset of the subproblems at each iteration, as opposed to its asynchronous capabilities.
In its first iteration \((k = 0)\), the implementation begins by solving all subproblems, which is equivalent to setting \(I_0 = 1..n\) in Algorithm 2. Subsequently, the operation of the master is controlled by a parameter \(\text{bufferlen}\): the master enters a loop in which it dispatches \(\text{bufferlen}\) subproblems for solution, waits to receive all the results, performs the projective coordination calculations, and repeats. In all our experiments except those described in Table 2 below, the number of available worker processors is effectively equal to \(\text{bufferlen}\). However, if \(\text{bufferlen}\) exceeds the number of available worker processes, subproblems simply queue until a worker is available.\(^1\)

The master selects subproblems for dispatch by a rule that gives priority to the most negative values of \(\phi_i = \pi_i \sum_{t=1}^{T-1} (z_{it} - x_{it})^\top (w_{it} - y_{it})\); a rationale for this kind of “greedy” selection heuristic is given in [14, Section 4.3]. If there are fewer than \(\text{bufferlen}\) indices \(i\) for which \(\phi_i < 0\), then the remaining subproblems are chosen randomly. Duplicates of the same subproblem \(i\) are never dispatched in the same iteration, but the dispatched subproblem may duplicate those already being processed by workers.

We conducted our experiments on a 96-core Intel Xeon workstation with 2.1GHz processors and 1TB of RAM. Subproblems were solved with Gurobi version 8.0.0. Thus, up to \(\text{bufferlen}\) instances of Gurobi may be running simultaneously; however, each Gurobi instance may employ multiple workstation cores.

Our first set of use a standard stochastic programming test problem family known as \(\text{ssn}\) [20, 21] which has 89 first-stage variables and second-stage data defined on a discrete probability space that is too large to avoid sampling scenarios. The problem is very well scaled with an optimal objective function value on the order of 10, so we used a proximal parameter value of \(\rho = 1\) for all experiments. We report on experiments done with 5,000 scenarios in part because Gurobi can solve its extensive form directly in about one hour, providing known optimal reference solutions (however, we were are not able to directly solve the extensive form for 10,000 scenarios on our computer). Our goal is not to demonstrate the fastest method of solving the \(\text{ssn}\) problem; in this regard, there is little hope of competing with the specialized algorithm and implementation by Sen and Liu [21], which uses a compiled language and a sophisticated algorithm designed for a narrower range of problem types than PH and APH. Our goal is simply to compare APH with PH.

Figure 1 shows that with 10 bundles of 500 scenarios each, and 5 solver servers available, APH obtains better solutions faster. In these experiments, APH dispatched and processed 5 subproblems at a time \((i.e., \text{bufferlen} = 5)\). The horizontal axis corresponds the wall-clock time and includes processing the symbolic model and reading data. For these experiments, we terminated each algorithm after a fixed number of iterations.

We could not identify a circumstance in which PH outperformed APH, but when the number of solver servers matches the number of bundles, both algorithms run faster and the separation becomes smaller. Figure 2 depicts the case of 10 worker processors, \(\text{bufferlen} = 10\) and 10 bundles: in this case, the difference between PH and APH is fairly small.

With far fewer solvers than the number of subproblems, the separation between APH and PH becomes greater, as can be seen in Figure 3. For these experiments, we made only

\(^1\)In future experiments for problems with highly variable subproblem solve times, we will consider having \(\text{bufferlen}\) smaller than the number of workers \(W\) and dispatching \(W\) subproblems immediately after iteration 0.
Figure 1: Percent deviation from optimal of the best feasible solution found so far as a function of wall-clock time (seconds) for PH and APH with 5 worker processors, for three randomly generated instances of ssn with 5,000 scenarios grouped into 10 bundles.
two worker processors available to each algorithm, and APH dispatched and processed two sub-problems at a time ($\text{bufferlen} = 2$). The PH algorithm must wait for all subproblems to solve, involving 5 rounds of worker processing, before updating multipliers, resulting in its curve shifting to the right relative to Figure 2. The APH algorithm appears to be able to make good use of information from just a subset of the subproblems, and the APH curve consequently shifts much less in comparison to Figure 2.

To further explore the case in which there are more bundles than solvers, we considered the $\text{ssn}$ problem with 10 workers and 20,000 scenarios grouped into 20 bundles of 1,000 scenarios each. Table 1 shows the results for each of three replicates. These instances are too large to be solved directly, so the deviation is with respect to the best solution encountered during the experiment. In this setting, APH consistently obtains better solutions much faster than PH.

In order to test APH’s sensitivity to the number of subproblems solved per iteration and study the algorithm’s performance without bundling, we used the venerable ALP1P test problem [13, 1] using serial processing (equivalent to a single worker) and either $n = 1,000$ and $n = 2,000$ scenarios. Table 2 shows deviations from the optimal objective value at various times for three randomly generated instances of each size; for these small problems, we can easily calculate the true optimal objective value with GuRoBi. The $\text{bufferlen}$ parameter increases across each row (for example, “APH 10” refers to a having $\text{bufferlen} = 10$. In this case, there is effectively only a single worker, and $\text{bufferlen}$ subproblems are solved serially between each pair of coordination steps. The “Rep” column shows the replicate number and the the “Sec” column indicates the elapsed run time so far, including all PySP preprocessing and overhead. Reading across the rows and comparing results for the same replicate, one concludes that for smaller amounts of time larger values of $\text{bufferlen}$ work best and PH often gets the best results. For larger amounts of time, dependence on $\text{bufferlen}$ is reduced and APH is often far superior to PH.
Figure 3: For the middle instance in Figure 1, the percent deviation of the objective function value as function of time (in seconds) with 10 bundles but only 2 worker processors available.

<table>
<thead>
<tr>
<th>Replicate</th>
<th>Minutes</th>
<th>Method</th>
<th>Deviation %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>93</td>
<td>aph</td>
<td>0.19</td>
</tr>
<tr>
<td></td>
<td>95</td>
<td>ph</td>
<td>8.29</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>aph</td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td>114</td>
<td>aph</td>
<td>0.11</td>
</tr>
<tr>
<td></td>
<td>161</td>
<td>aph</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>189</td>
<td>ph</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>197</td>
<td>ph</td>
<td>0.20</td>
</tr>
<tr>
<td>2</td>
<td>91</td>
<td>aph</td>
<td>0.20</td>
</tr>
<tr>
<td></td>
<td>94</td>
<td>ph</td>
<td>6.03</td>
</tr>
<tr>
<td></td>
<td>103</td>
<td>aph</td>
<td>0.16</td>
</tr>
<tr>
<td></td>
<td>118</td>
<td>aph</td>
<td>0.11</td>
</tr>
<tr>
<td></td>
<td>163</td>
<td>aph</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>202</td>
<td>ph</td>
<td>0.23</td>
</tr>
<tr>
<td></td>
<td>268</td>
<td>ph</td>
<td>0.29</td>
</tr>
<tr>
<td>3</td>
<td>91</td>
<td>ph</td>
<td>6.28</td>
</tr>
<tr>
<td></td>
<td>94</td>
<td>aph</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td>101</td>
<td>aph</td>
<td>0.13</td>
</tr>
<tr>
<td></td>
<td>115</td>
<td>aph</td>
<td>0.09</td>
</tr>
<tr>
<td></td>
<td>164</td>
<td>aph</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>212</td>
<td>ph</td>
<td>0.23</td>
</tr>
<tr>
<td></td>
<td>326</td>
<td>ph</td>
<td>0.31</td>
</tr>
</tbody>
</table>

Table 1: The ssn problem with three replicates of 20,000 scenarios each and 10 workers applied to 20 bundles.
Table 2: Results for problem APL1P. The data entries are percent deviations from the optimal objective function value. For APH, the number in the column header is the value of the bufferlen parameter.

7 Conclusions

This paper introduced an asynchronous projective hedging (APH) class of stochastic programming algorithms that resemble progressive hedging, but are derived from the projective splitting algorithms of [4, 7]. A rigorous proof of convergence was provided for convex problems, and the analysis explicitly accommodates asynchronous and block-iterative implementations, “block-iterative” meaning that only a subset of the subproblems need be solved at each iteration. Computational experiments on two-stage linear problems demonstrate that APH can significantly outperform PH. An implementation of the method is included in the PySP component of the open-source Pyomo project.

APH presents opportunities for further research both in theory and in applications. A dual derivation of the method may be worth pursuing and might yield convergence under different assumptions (for example, without the compactness condition in Assumption 1). Although we have so far only studied the theory of APH for the convex case, the software we have developed can be applied to a very broad range of multi-stage optimization problems with uncertainty represented by scenarios. Progressive hedging has helped advance the applicability of stochastic programming methods and we hope that APH will contribute to further advances. In addition to practically exploring the practical behavior of APH for mixed-integer and other nonconvex problems, its theoretical behavior in such settings also deserves study, and it would be interesting to consider whether it can be adapted to provide lower bounds on the optimal solution value in such cases.
A Proof of Proposition 5

Proof. First, consider any \( z \) solving (14). Applying [2, Proposition 16.5(ii)] \( n \) times, we have that
\[
\sum_{i=1}^{n} L_i^* \partial f_i(L_iz) \subseteq \partial H(z), \quad \text{where } H : u \mapsto \sum_{i=1}^{n} f_i(L_iu).
\]
Therefore, \( 0 \in \partial H(z) \) and hence \( z \) solves problem (10). Further, for each \( i \in 1..n \), some \( x_{iT} \) as defined in (15) must exist by Assumption 1, and the optimality for (7) of \( x \) as constructed in (15) is immediate from the setup S1-S4. This proves the first half of the proposition.

We now commence the proof of the second half of the proposition, which supposes that \( x \) is some optimal solution to (7) and that assumptions 1 and 4 hold. By Proposition 3, \( z = Mx \) must be an optimal solution of (10), which is equivalent to \( \partial H(z) \ni 0 \). We must show that this condition is equivalent to (14).

To this end, we first claim that \( \partial (f_i \circ L_i) = L_i^* \circ \partial f_i \circ L_i \) for all \( i \in 1..n \). To see this, first fix any \( i \in 1..n \). From [2, Corollary 16.42(i)], we know that the claim will hold if \( 0 \in \text{sri}(\text{dom } f_i - \text{rge } L_i) \), where “sri” denotes the strong relative interior of a set, which is identical to the relative interior in finite dimension. From S3, \( \text{rge } L_i = \mathcal{H}_i \), so we have \( \text{sri}(\text{dom } f_i - \text{rge } L_i) = \text{ri } \mathcal{H}_i = \mathcal{H}_i \ni 0 \), and the claim is established.

To complete the proof, it is sufficient to show that \( \partial \left( \sum_{i=1}^{n} f_i \circ L_i \right) = \sum_{i=1}^{n} \partial(f_i \circ L_i) \), since we would then have
\[
\partial H = \partial \left( \sum_{i=1}^{n} f_i \circ L_i \right) = \sum_{i=1}^{n} \partial(f_i \circ L_i) = \sum_{i=1}^{n} L_i^* \circ \partial f_i \circ f_i,
\]
where the last equality follows from the previous claim. Instantiating this equation at \( z \) yields (14).

We establish that \( \partial \left( \sum_{i=1}^{n} f_i \circ L_i \right) = \sum_{i=1}^{n} \partial(f_i \circ L_i) \) by induction on the number of terms in the sum. That is, we will show that for any \( k \in 1..n \), one has \( \partial \left( \sum_{i=1}^{k} f_i \circ L_i \right) = \sum_{i=1}^{k} \partial(f_i \circ L_i) \). This clearly holds for \( k = 1 \). Now suppose it holds for an arbitrary \( k < n \); we will establish that it must then hold for \( k + 1 \).

First, consider the case that Assumption 4(1) holds. We start this case by observing, for each \( i \in 1..n \), that \( h_i \) being a polyhedral function implies that \( f_i \) is polyhedral. Specifically, \((zi, \alpha) \in \text{epi } \mathcal{H}_i \times \mathbb{R} \) is in \text{epi } f_i \) if and only if there exists \( x_{iT} \in \mathbb{R}^m \) such that \((zi, x_{iT}), \alpha/\pi_i \) \( \in \text{epi } h_i \). Thus, \text{epi } f_i \) is the image of \text{epi } h_i \) under the linear mapping \( P : (\text{epi } h_i) \mapsto (zi, x_{iT}, \beta) \mapsto (zi, \pi_i, \beta) \). Since \( h_i \) is defined as \text{epi } h_i \) being a polyhedral set and the spaces involved are all finite-dimensional, the argument of [17, Theorem 19.3] implies that \text{epi } f_i = P(\text{epi } h_i) \) is polyhedral, and \( f_i \) is polyhedral. The argument of [17, Corollary 19.3.1] then implies that \( f_i \circ L_i \) is also polyhedral, and repeated use of the argument of [17, Theorem 19.4] establishes that \( \sum_{i=1}^{k} f_i \circ L_i \) is polyhedral, being a finite sum of polyhedral functions. Now, we are in the situation that \( \sum_{i=1}^{k} f_i \circ L_i \) is polyhedral, \( f_{k+1} \circ L_{k+1} \) is polyhedral, and their domains intersect, both containing the point \( z = Mx \). Since the setting is finite dimensional, we have
\[
\partial \left( \sum_{i=1}^{k+1} f_i \circ L_i \right) = \partial \left( \sum_{i=1}^{k} f_i \circ L_i + f_{k+1} \circ L_{k+1} \right)
\] (30)
\[ \partial \left( \sum_{i=1}^{k} f_i \circ L_i \right) + \partial(f_{k+1} \circ L_{k+1}) \quad \text{[2, Theorem 16.37(iii)]} \]

\[ = \sum_{i=1}^{k} \partial(f_i \circ L_i) + \partial(f_{k+1} \circ L_{k+1}) \quad \text{[By the induction hypothesis]} \]

\[ = \sum_{i=1}^{k+1} \partial(f_i \circ L_i), \quad (31) \]

and the induction is complete.

We next turn to the situation in which Assumption 4(2) holds. We begin this case by observing that for any \( i \in 1..n, \)

\[ \text{dom}(f_i \circ L_i) = \{ u \in \mathcal{H}_0 \mid L_i u \in \text{dom} f_i \} = L_i^{-1} \text{dom} f_i, \]

where \( L_i^{-1} S \) denotes the preimage of the set \( S \) under the linear map \( L_i. \) Since we are in finite dimension, applying the argument of \([17, \text{Theorem 6.7}]\) leads to the conclusion that

\[ (\forall i \in 1..n) \quad \text{ri dom}(f_i \circ L_i) = \text{ri } L_i^{-1} \text{dom } f_i = L_i^{-1} \text{ri dom } f_i. \quad (32) \]

Next, we observe that the constraint qualification (13) is equivalent to

\[ (\forall i \in 1..n) \quad L_i M \bar{x} \in \text{ri } M_i(\text{dom } h_i) = \text{ri dom } f_i, \quad (33) \]

where the equality is in view of (12). Therefore, we know that \( \bar{z} = M \bar{x} \) has the property the \( L_i \bar{z} \in \text{ri dom } f_i \) for all \( i \in 1..n, \) Hence, for any \( i \in 1..n, \) we have \( \bar{z} \in L_i^{-1} \text{ri dom } f_i = \text{ri dom } (f_i \circ L_i). \)

For the Assumption 4(2) case, we adjoin to the induction hypothesis that

\[ \text{ri dom} \left( \sum_{i=1}^{k} f_i \circ L_i \right) = \bigcap_{i=1}^{k} (L_i^{-1} \text{ri dom } f_i), \]

which holds for \( k = 1 \) by letting \( i = 1 \) in (32).

We may rewrite (33) as \( \bar{z} \in \bigcap_{i=1}^{n} (L_i^{-1} \text{ri dom } f_i) \), hence we have both

\[ \bar{z} \in \bigcap_{i=1}^{k} (L_i^{-1} \text{ri dom } f_i) = \text{ri dom} \left( \sum_{i=1}^{k} f_i \circ L_i \right) \]

\[ \bar{z} \in L_{k+1}^{-1} \text{ri dom } f_{k+1} = \text{ri dom } f_{k+1} \circ L_{k+1}. \]

We may then invoke the same inductive logic as in (30)-(31), but using Corollary 16.38(iv) instead of Theorem 16.37(iii) of [2], and conclude that \( \partial \left( \sum_{i=1}^{k+1} f_i \circ L_i \right) = \sum_{i=1}^{k+1} \partial(f_i \circ L_i). \)

To complete the induction in this case, we also need to establish that

\[ \text{ri dom} \left( \sum_{i=1}^{k+1} f_i \circ L_i \right) = \bigcap_{i=1}^{k+1} (L_i^{-1} \text{ri dom } f_i). \]

This relation follows by combining the additional induction hypothesis with [2, Fact 6.14(v)], because \( \bar{z} \) lies in both \( \text{ri dom} \left( \sum_{i=1}^{k} f_i \circ L_i \right) \) and \( \text{ri dom } f_{k+1} \circ L_{k+1} \). \( \square \)
Acknowledgments

This research was funded in part by National Science Foundation grant CCF-1617617.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy’s National Nuclear Security Administration under contract DENA0003525.

References


