SAND2016-12289J

BBPH: Using Progressive Hedging Within Branch and Bound to Solve Multi-Stage Stochastic Mixed Integer Programs

Jason Barnett

Department of Applied Mathematics, University of California, Davis, CA 95616-8609, USA

Jean-Paul Watson

Discrete Math and Optimization Department, Sandia National Laboratories, Albuquerque, NM 87185, USA

David L. Woodruff *

Graduate School of Management, University of California, Davis, CA 95616-8609, USA

Abstract

Progressive hedging, though an effective heuristic for solving stochastic mixed integer programs (SMIPs), is not guaranteed to converge in this case. Here, we describe BBPH, a branch and bound algorithm that uses PH at each node in the search tree such that, given sufficient time, it will always converge to a globally optimal solution. In addition to providing a theoretically convergent "wrapper" for PH applied to SMIPs, computational results demonstrate that for some difficult problem instances branch and bound can find improved solutions after exploring only a few nodes.

Keywords: Stochastic Programming, Progressive Hedging, Branch and Bound

1. Introduction

Spurred by important potential applications, researchers have recently devoted considerable energy to developing methods for solving stochastic mixed integer programs (SMIPs). Progressive hedging (PH), though an effective heuristic for SMIPs, is not guaranteed to converge in the integer case. Here, we propose BBPH, a branch and bound (B&B) algorithm that uses PH at each

Email addresses: jpbarnett@math.ucdavis.edu (Jason Barnett), jwatson@sandia.gov (Jean-Paul Watson), dlwoodruff@ucdavis.edu (David L. Woodruff *)

*corresponding author

node of the search tree. Given sufficient time, BBPH will converge to a globally optimal solution. Almost all PH innovations and applications described in the literature can be embedded in this framework.

Originally proposed by Rockafellar and Wets [13], examples of PH in the literature include [3, 4, 5, 7, 11, 12].

Branch and bound for SMIPs is not new, of course. For example, [1] proposes a scenario decomposition algorithm for 0-1 stochastic programs, a specific case of our intended general problem class. The dual decomposition algorithm given in [2] uses B&B for a two stage SMIP. A branch-and-fix algorithm for solving multi-stage, stochastic, mixed 0-1 problems is described in [6]

What is new in this contribution is B&B for PH, which can be applied to a broad class of multi-stage SMIPs. A general-purpose software implementation is available as part of the PySP library in the Pyomo software package (`www.pyomo.org`). Tests of the algorithm and this implementation are described in Section 4. Before describing those tests, we introduce in the remainder of this section the optimization model formulation and the progressive hedging algorithm for multi-stage SMIPs. Section 2 then describes two small examples illustrating the need for branching in PH, which are provided as formal motivation. Our BBPH branch and bound algorithm is described along with a remark whose proof demonstrates its correctness in Section 3.

1.1. A General SMIP Optimization Model

Let T be the number of decision stages for a multi-stage stochastic program. We will use $t \in 1, \ldots, T$ to index stages, although decision stages do not always correspond to time. To make abstract statements about stochastic programming, we make use of a random variable, which may be vector valued, ξ_t, associated with each decision stage t. When the stages correspond to time, we think of the value(s), ξ_t, during or at the end of, stage $t-1$ depending on the application. Hence, we generally refer to ξ_t only for stage $2, \ldots, T$. Consequently, the decisions for the stage t are made once the random variables for the stages up to and including t are known.

We use the symbol $\vec{\xi}_t$ to refer to the realized values of all random variables up to and including stage t. We refer to a full realization of the uncertainty, i.e.,

$$\vec{\xi}^{\,T} = \left(\xi^t,\ t = 2, \ldots, T\right)$$

as a *scenario*. Often, problem data is a function of ξ and the problem data corresponding to a particular value of ξ is also referred to as a scenario.

In abstract formulations, we use $x^t = (u^t, y^t)$ to respectively represent the integer and real parts of the decision vector that corresponds to stage t. We use the notation $\vec{x}^{\,t}$ for $1 \leq t \leq T$ to represent the decisions for all stages up to, and including, stage t. We assume that there are functions of the decision variables for the current stage, parameterized by the decisions and random variable realizations known at the time of the decisions. The functions return the objective function value corresponding to the stage for feasible solutions and a very large number for infeasible solutions. For the first stage we write $f_1(x^1)$

and for subsequent stages $f_t(x^t; \vec{x}^{\,t-1}, \vec{\xi}^{\,t})$. The function f_t takes an argument the partial vector corresponding to stage t and as parameters the solution for previous stages and the realization of the random variables up to stage t. These functions enable us to write the minimization of expected value very succinctly as

$$\min_x f_1\left(x^1\right) + \mathbb{E}\sum_{t=2}^{T} f_t\left(x^t; \vec{x}^{\,t-1}, \vec{\xi}^{\,t}\right) \tag{1}$$

$$\text{subject to } x(\xi) \in Y_\xi, \;\; \xi \in \Xi, \tag{2}$$

where $(u(\xi), y(\xi)) \in Y_\xi \subset \mathbb{Z}^k_{\geq 0} \times \mathbb{R}^l$.

In the multi-stage cases of interest to us, $\xi = \{\xi^t\}_{t=1}^T$ is defined on a discrete probability space $(\Xi, \mathcal{A}, \mathcal{P})$. Each scenario, ξ, has probability π_ξ. We organize realizations, ξ, into a tree with the property that scenarios with the same realization up to stage t share a node at that stage. Consequently, $\vec{\xi}^{\,t}$ refers also to a node in the scenario tree. Let $\mathcal{G}_t$ be the set of all scenario tree nodes for stage t and let $\mathcal{G}_t(\xi)$ be the node at time t for a particular scenario, ξ. For a particular node $\mathcal{D}$ let $\mathcal{D}^{-1}$ be the set of scenarios that define the node.

In the presence of a scenario tree, non-anticipativity must be enforced at each non-leaf node, so using the discrete scenario tree notation, problem (1) becomes

$$\min_{x,\hat{x}} \sum_{\xi\in\Xi} \pi_\xi \left[f_1(x^1(\xi)) + \sum_{t=2}^{T} f_t\left(x^t(\xi); \vec{x}^{\,t-1}, \vec{\xi}^{\,t}\right)\right] \tag{3}$$

$$\text{s.t. } x(\xi) \in Y_\xi, \;\; \xi \in \Xi \tag{4}$$

$$x^t(\xi) - \hat{x}^t(\mathcal{D}) = 0, \;\; t = 1, \ldots, T-1, \; \mathcal{D} \in \mathcal{G}_t, \; \xi \in \mathcal{D}^{-1} \tag{5}$$

We use the notation $x(\xi)$ to emphasize that the decisions can depend on the realizations of the random variables, while constraints (5) use the auxiliary variable $\hat{x}$ to assure that at every node of the decision tree, the portion of the solution corresponding that decision stage is the same. So while x depends on ξ, it does so in a way that is *non-anticipative*. To put it another way: looking only at expression (3), one might get the impression that the optimization is allowed to be prescient and make use of knowledge of the future in setting stage variable values since x^t in these expressions depends on the entire realization, ξ. However, constraints (5) force the decisions to depend only on information that would be available when they are made. Note that in this formulation, there is one full decision vector x for each scenario. The variables $\hat{x}$ are often called *system of vectors* since they are tied to the tree with partial vectors corresponding to each node of the tree.

Sometimes this problem is written without the variable $\hat{x}$ as follows:

$$\min_{x} \sum_{\xi \in \Xi} \pi_\xi \left[f_1(x^1(\xi)) + \sum_{t=2}^{T} f_t \left(x^t(\xi); \vec{x}^{\,t-1}, \vec{\xi}^{\,t} \right) \right] \tag{6}$$

$$\text{s.t. } x(\xi) \in Y_\xi, \;\; \xi \in \Xi \tag{7}$$

$$x^t(\xi) - \bar{x}(\mathcal{G}_t(\xi)) = 0, \;\; t = 1, \ldots, T-1, \; \xi \in \Xi \tag{8}$$

where for each $t = 1, \ldots, T$ and each $\mathcal{D} \in \mathcal{G}_t$

$$\bar{x}^t(\mathcal{D}) := \sum_{\xi \in \mathcal{D}^{-1}} \pi_\xi x^t(\xi) / \sum_{\xi \in \mathcal{D}^{-1}} \pi_\xi.$$

In other words: $\bar{x}$ is a system of node-by-node averages. Unless there are scenarios with zero probability, the formulation with $\bar{x}$ is equivalent to the formulation with $\hat{x}$, so most practical applications we use the second form and remove any zero probability scenarios in a pre-processing step.

1.2. The PH algorithm

In the context of the formulation specified above, we now present the progressive hedging algorithm for multi-stage stochastic mixed-integer programs:

Algorithm 1 The Progressive Hedging Algorithm for Multi-Stage SMIPs

---

1. **Initialization:** Let $\nu \leftarrow 0$ and $w_\nu(\mathcal{G}_t(\xi)) \leftarrow 0$, $\forall \xi \in \Xi$, $\forall t \in \{1, \ldots, T\}$. Compute for each $\xi \in \Xi$:

$$x_{\nu+1}(\xi) \in \operatorname*{argmin}_{x \in X(\xi)} \sum_{\xi \in \Xi} \pi_\xi \left[f_1(x^1(\xi)) + \sum_{t=2}^{T} f_t \left(x^t(\xi); \vec{x}^{\,t-1}, \vec{\xi}^{\,t} \right) \right].$$

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

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

$$\bar{x}^t_\nu(\mathcal{D}) \leftarrow \left(\sum_{\hat{\xi} \in \mathcal{D}^{-1}} \pi_{\hat{\xi}} x^t_\nu(\mathcal{G}_t(\xi)) \right) / \left(\sum_{\hat{\xi} \in \mathcal{D}^{-1}} \pi_{\hat{\xi}} \right).$$

4. **Weight Update:** Compute for each $t \in \{1, \ldots, T-1\}$ and each $\xi \in \Xi$:

$$w_\nu(\mathcal{G}_t(\xi)) \leftarrow w_{\nu-1}(\mathcal{G}_t(\xi)) + \rho[x^t_\nu(\mathcal{G}(\xi)) - \bar{x}^t_\nu(\mathcal{G}^t(\xi))].$$

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

$$x_{\nu+1}(\xi) \in \operatorname*{argmin}_{x \in X(\xi)} f_1(x^1(\xi)) + \sum_{t=2}^{T} f_t \left(x^t(\xi); \vec{x}^{\,t-1}, \vec{\xi}^{\,t} \right) + \sum_{t=1}^{T-1} \left[\langle w^t_\nu(\xi), x^t \rangle + \frac{\rho}{2} \|x^t - \bar{x}^t_\nu(\xi)\|^2 \right].$$

6. **Termination criterion:** If the solutions at the tree nodes are equal (up to a given tolerance ϵ) or the maximum iteration count is reached, stop. Otherwise, return to step 2.

We refer to

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

as the *proximal term*. Our implementation uses the L_2 norm. We refer to systems of weights w that satisfy

$$\sum_{\xi\in\Xi} \pi(\xi)w(\xi) = 0$$

as *qualified weights*. When PH uses a single value of the parameter ρ for all iterations, scenarios, and variables, we refer to the ρ as a *global* ρ. If PH uses a constant vector ρ for all iterations, we refer to the ρ as a *variable-specific* ρ. If ρ is allowed to change at any iteration, we refer to the ρ as a *dynamic* ρ.

2. Motivation for Hybridizing Branch-and-Bound and PH

2.1. A Simple Example

To illustrate what can go wrong with PH convergence in the case of a simple two-stage stochastic binary program, we consider the example found in the standard Pyomo (`www.pyomo.org`) release in the directory:

`pyomo/pysp/tests/examples/test_model/twovarslack`.

The model is given as follows, with x_1, x_2 denoting the first-stage variables and y, α denoting the second-stage variables:

Scenario 1, probability 1/2	Scenario 2, probability 1/2
minimize $10y + 1000\alpha$	minimize $-10y + 1100\alpha$
subject to $y \geq x_i \ \ \forall i \in \{1,2\}$	subject to $y \geq x_i, \ \ i\forall \in \{1,2\}$
$y \leq x_1 + x_2$	$y \leq x_1 + x_2$
$-y + \alpha \geq 0$	$y + \alpha \geq 1$
$\alpha, x_1, x_2 \in \{0,1\}$	$\alpha, x_1, x_2 \in \{0,1\}$

where the first two (shared) constraints require that $y = 0$ if and only if both x_i are zero and y is bounded above by the sum of x_i. Note the importance of the slack variable α: in Scenario 1, a positive y gives slack, but in Scenario 2, $y < 1$ gives slack. An issue arises for PH convergence because no matter how we assign values for x, we will be forced to pay a slack cost in one of the scenarios. Finally, we remark that the globally optimal solution is $x_i = 1$, $\forall i \in \{1,2\}$ which can be seen intuitively by letting our forced slack scenario be Scenario 1, which allows us to utilize the coefficient of -10 in Scenario 2.

Here we consider a global ρ. Depending on the choice of ρ, the outcome of PH is one of the following. For $\rho \leq 1010$, the weights are updated monotonically until reaching a magnitude of 505, where a cycling of three solutions occurs in the following way:

Scenario	1	1	2	2
Solution for	x_1	x_2	x_1	x_2
	0	0	1	1
	1	1	0	1
	1	1	0	1
	0	0	1	1
	1	1	1	0
	1	1	1	0

For $1010 < \rho < 2220$, we obtain the optimal solution in two iterations. For $\rho \geq 2220$, we get a cycling between two solutions, where (x_1, x_2) are $(1, 1)$ in one scenario and $(0, 0)$ in the other.

A question arises: can we find weights (and thus $\rho_s^{(k)}(i)$ values) such that this cycling does not happen? Or even better, can we find weights such that we don't even need the proximal term at all to get an optimal solution? For a problem this simple, we can answer these questions. Since we can easily compute

$$x_1^{(0)} = (0, 0),\ x_2^{(0)} = (1, 1),\ \bar{x}^{(0)} = (1/2, 1/2),$$

the only remaining task is to find the values of $w_s(i)$ such that $x_s = (1, 1)$ is the solution of line 6. If such weights exist, then it is possible to get the solution in only one iteration. Due to our $\bar{x}^{(0)}$, the proximal term will be irrelevant in our minimization as any value of x will yield the same quantity in the norm.

By doing some algebra, one finds that for scenario one, all we need to force the solution of $(1, 1)$ is $w_1^{(1)}(1) + w_1^{(1)}(2) < -1010$ and so this can be satisfied with a global $\rho > 1010$, giving $w_1^{(1)} = (-\rho/2, -\rho/2)$. For scenario two, we need $w_2(i) < 10$, $i = 1, 2$. Notice that since $w_1(i) = -w_2(i)$, we cannot get this weight. However, we can get the solution $(1, 0)$ if $w_2^{(1)}(2) \leq 1110$ or $\rho_2 < 2220$, which gives $w_2^{(1)} = (\rho/2, \rho/2)$.

Now, assuming $1010 < \rho < 2220$, we have the solutions $x_1^{(1)} = (1, 1)$, $x_2^{(1)} = (1, 0)$, so $\bar{x}^{(1)} = (1, \frac{1}{2})$, which gives us $w_s^{(2)} = (w_s^{(1)}, 0)$ for both scenarios. With these weights and ρ, scenario 1 again prefers $(1, 1)$ as $x = (0, 0)$, $(1, 0)$, and $(1, 1)$ give PH objective values of $5\rho/8$ for the first and $1010 - 3\rho/8$ for the next two. Scenario 2 also prefers $(1, 1)$ now as $x = (0, 0)$, $(1, 0)$, and $(1, 1)$ give PH objective values of $1100 + 5\rho/8$, $-10 + 5\rho/8$, and $-20 + 5\rho/8$ respectively. Thus ends the proof of convergence in two iterations of PH.

In many well-behaved examples, the trade-off in choosing a value for ρ is a high iteration count for small values of ρ and possibly sub-optimal solutions for large values of ρ. The above example shows that sometimes there is extra difficulty in choosing ρ in that a ρ outside of a unknown set causes endless cycling. In practice, it is highly impractical to attempt to find this set by hand, and experimentation is necessary, although there exist heuristics and ρ selection innovations in recent years [16]. Furthermore, sometimes this set is empty for global ρ, as can be seen in the following example.

2.2. An Impossible Instance

Consider an instance almost identical to the example in Section 2.1.

Scenario 1, probability 1/2	Scenario 2, probability 1/2
minimize $x_1 + x_2 + 10y + 1000\alpha$	minimize $x_1 + x_2 - 10y + 1100\alpha$
subject to $y \geq x_i \;\; i \in \{1,2\}$	subject to $y \geq x_i, \;\; i \in \{1,2\}$
$y \leq x_1 + x_2$	$y \leq x_1 + x_2$
$-y + \alpha \geq 0$	$y + \alpha \geq 1$
$\alpha, x_1, x_2 \in \{0,1\}$	$\alpha, x_1, x_2 \in \{0,1\}$

with the only difference being the addition of the slight penalty of 1 for each x_i. The optimal solution is still (1,1), but now there is no global ρ that gives us the optimal solution.

First, we will show that there are no qualified weights giving convergence without the proximal term. Again noting that $w_1^{(k)}(i) = -w_2^{(k)}(i)$, then without the proximal term, the modified objective function values for possible solutions is given by the following table.

Solution	Scenario 1	Scenario 2
(1,1)	$1012 + \sum w_1(i)$	$-18 - \sum w_1(i)$
(1,0)	$1011 + w_1(1)$	$-9 - w_1(1)$
(0,1)	$1011 + w_1(2)$	$-9 - w_1(2)$
(0,0)	0	1100

Doing some quick algebra, we can see that for (1,1) to be the solution in both scenarios, we would at least need $\sum w(i) < -1012$ and $w(i) > -9$, $i = 1, 2$ which can clearly never happen.

Next, we point out what occurs in each interval of global ρ. For $\rho \leq 1011$, the weights approach $w_1 = (-505.5, -505.5)$ after which begins a six solution cycle. For $1011 < \rho \leq 1012$, we obtain convergence to the suboptimal solution of $(1, 0)$ in two iterations. For $1012 < \rho < 2217$, we obtain immediate convergence to $x_s^*(1) = 1$ but $x_s(2)$ continually cycles. And for $\rho \geq 2218$, both scenarios cycle between $(0, 0)$ and $(1, 1)$.

These two examples illustrate that even in the simple case of equal-probability, two-stage SMIPs with two binary variables and no general integer variables, no amount of parameter tuning will allow PH to identify an optimal solution given a fixed global value of ρ. This motivates the necessity for embedding PH in the context of a branch-and-bound framework, which we now introduce.

3. Combining PH with B&B

In this section, we use PH and SMIP analysis in order to define an SMIP-specific B&B algorithm. We begin by informally describing our method and

introduce distinctions between the types of nodes encountered in the B&B tree. We branch by fixing discrete variables, and refer to the resulting branched nodes as *outer B&B nodes*. For each outer B&B node, PH is applied in order to obtain the bounds necessary for B&B. In turn, PH repeatedly determines the $x(\xi)$ by solving each scenario realization in the SMIP scenario tree. We refer to these realizations as *scenario subproblems*. Finally, solving each of these subproblems requires the use of a MIP solver, which in turn will use a distinct B&B process. We refer to the nodes created by MIP solvers as *inner B&B nodes*.

We now examine the role of PH in obtaining bounds required by the SMIP B&B algorithm.

3.1. Lower Bounds

Unlike many B&B algorithms for SMIP, we do not compute lower bounds for an outer node using the continuous relaxation of the problem. Gade et al. [8] showed that we can compute lower bounds using the weights w_ν obtained by PH, on the full non-relaxed problem. We note that these lower bounds can be computed at any iteration of PH with a computational cost on the order of one iteration of PH. Indeed, it may be highly beneficial to compute bounds early in order to more quickly prune nodes in the outer B&B tree. Here, we extend their result to the nonlinear multi-stage case.

Theorem 3.1. *Let $-\infty < z^* < \infty$ denote the optimal objective function of a feasible multi-stage SMIP and let $X(\xi) \neq \emptyset \ \forall \xi \in \Xi$. Then for*

$$D_\xi(w(\xi)) = \min_{x \in X(\xi)} f_1(x^1(\xi)) + \sum_{t=2}^{T} f_t\left(x^t(\xi); \vec{x}^{\,t-1}, \vec{\xi}^{\,t}\right) + \sum_{t=1}^{T-1} \langle w^t(\xi), x^t \rangle, \quad (9)$$

where the RHS is simply the decomposition step of PH without the proximal term, then

$$D(w) := \sum_{\xi \in \Xi} \pi_\xi D_\xi(w^t(\xi)) \leq z^*. \quad (10)$$

Proof. Let $(\hat{x}, \{x(\xi), \forall \xi \in \Xi\})$ be an optimal solution. Feasibility implies $x(\xi) \in X(\xi)$ for each $\xi \in \Xi$. Thus,

$$D_\xi(w(\xi)) \leq f_1(x^1(\xi)) + \sum_{t=2}^{T} f_t\left(x^t(\xi); \vec{x}^{\,t-1}, \vec{\xi}^{\,t}\right) + \sum_{t=1}^{T-1} \langle w^t(\xi), x^t \rangle.$$

Then

$$\begin{aligned} D(w) &\leq \sum_{\xi\in\Xi} \pi_\xi \left[f_1(x^1(\xi)) + \sum_{t=2}^{T} f_t\left(x^t(\xi); \vec{x}^{\,t-1}, \vec{\xi}^{\,t}\right) + \sum_{t=1}^{T-1} \langle w^t(\xi), x^t\rangle \right] \\ &= \sum_{\xi\in\Xi} \pi_\xi \left(f_1(x^1(\xi)) + \sum_{t=2}^{T} f_t\left(x^t(\xi); \vec{x}^{\,t-1}, \vec{\xi}^{\,t}\right) \right) + \sum_{\xi\in\Xi} \pi_\xi \sum_{t=1}^{T-1} \langle w^t(\xi), \hat{x}\rangle \\ &= \sum_{\xi\in\Xi} \pi_\xi \left(f_1(x^1(\xi)) + \sum_{t=2}^{T} f_t\left(x^t(\xi); \vec{x}^{\,t-1}, \vec{\xi}^{\,t}\right) \right) \\ &= z^*, \end{aligned}$$

where the first equality follows from implementability and the penultimate equality follows from the assumption that $\sum_{\xi\in\Xi} \pi_\xi w^t(\xi)\ \forall t \in \{1, 2, \ldots, T\}$. □

3.2. Upper Bounds

Upper bounds can be obtained from an admissible, implementable solution for a given outer B&B node. Recall that a solution x is *admissible* if x^t satisfies the constraints for all possible future realizations $\vec{\xi}^{\,t}$, that is, we cannot choose a course of action that could possibly be infeasible in the future. A solution x is *implementable* if it satisfies the non-anticipativity constraint

$$x^t(\xi) = \hat{x}^t,\ t \in \{1, \ldots, T\},\ \mathcal{D} \in \mathcal{G}_t,\ \xi \in \mathcal{D}^{-1},$$

which simply illustrates our requirement that our decision at time t cannot be different for various realizations $\xi_{t+1}, \ldots, \xi_T$. If x^t is both admissible and implementable, we say that it is *feasible*.

PH will naturally only produce admissible solutions, but practical issues can occur in finding an implementable solution. Consequently, mechanisms must be and have been developed for dealing with these issues, particularly in the case of SMIPs. The primary issue is that PH may take an extraordinary amount of time to obtain such a solution due to a poorly chosen ρ, or may not converge at all due to cyclic behaviors. Possible techniques to deal with these difficulties are described in [16], such as "slamming" variables in an attempt to force an implementable solution. In the computational results reported below, our implementation chooses the $x(\xi)$ closest to $\bar{x}$ and checks for feasibility. As with lower bounds, upper bounds can be computed after any PH iteration.

3.3. BBPH: Algorithm and Convergence Proof

Here, we refer to outer B&B nodes with all discrete variables fixed as *terminal nodes*. Further, $\mathcal{L}$ denotes the set of active non-terminal nodes and $\mathcal{N}$ denotes the set of terminal nodes. We refer to variables $u^t(i)$ at a node $N \in \mathcal{N} \cup \mathcal{L}$ with equal lower and upper bounds (obtained from a combination of problem constraints and branching) as *fixed variables* and those with unequal bounds as *non-fixed variables*. A straightforward algorithm is given below with the following parameters: non-terminal node max iteration count $k_{max} \in \mathbb{Z}_+$, tolerance

$\epsilon > 0$, and PH parameter $\rho \in \mathbb{R}_+$. To minimize a multi-stage SMIP,

Algorithm 3 A BBPH algorithm for Multi-Stage SMIPs

1. **Initialize:** Set $\bar{z} = \infty$ and $\underline{z}_0 = -\infty$. Set $\mathcal{L} \leftarrow \{(\text{SMIP})\}$.
2. **Choose node:** Select a node $N^i \in \mathcal{L}$, for example, the node with the smallest lower bound $\underline{z}_i$. If $\mathcal{L} = \emptyset$, proceed to Step 5.
3. **Calculate bounds:** Remove N^i from $\mathcal{L}$. Run PH on the selected outer B&B node N^i with maximum iteration k_{max} and parameter ρ. After each iteration of PH,
   (a) For $\bar{x} = (\bar{u}, \bar{y})$, apply a heuristic (e.g., rounding) to $\bar{u}$ to produce $\hat{x} = (\hat{u}, \bar{y})$, calculate the corresponding objective function value m, set $\bar{z} \leftarrow \min\{m, \bar{z}\}$. Remove all nodes $N^i \in \mathcal{L} \cup \mathcal{N}$ with $\underline{z}_i > \bar{z}$.
   (b) Compute $D(w_\nu)$ as defined in section 3.1. If $\bar{z} - D(w_\nu) < \epsilon$, terminate PH and return to step 2.
4. **Branch:** Select a non-fixed variable $u^1(i)$, for example one with maximum nonintegrality $|\bar{u}^1(i) - \bar{u}^1_{int}(i)|$, and create subnodes N^{i0} and N^{i1} corresponding to the branches $u^1(i) \leq \bar{u}^1(i)$ and $u^1(i) > \bar{u}^1(i)$ respectively. If $u^1(i)$ are fixed $\forall i$, continue with $u^2(i)$ (and so on). If N^{i0} and N^{i1} are fully real-valued problems (i.e. all integer variables have been fixed), add them to $\mathcal{N}$, else add them to $\mathcal{L}$. Return to step 2.
5. **Choose terminal node:** Select a node $N^i \in \mathcal{N}$ and continue to Step 6. If $\mathcal{N} = \emptyset$, terminate BBPH with the following: if $\bar{z} = \infty$, the problem is infeasible, otherwise our solution is $\hat{x}$ with objective value $\bar{z}$.
6. **Solve terminal node:** Remove N^i from $\mathcal{N}$. Run PH on N^i with maximum iteration ∞, tolerance threshold ϵ, and parameter ρ.
   (a) At each iteration of PH, find $\underline{z}_i$. If $\underline{z}_i > \bar{z}$, return to Step 5.
   (b) Upon termination of PH with output $\bar{x}$ and corresponding objective function value m, if $m \leq \bar{z}$, set $\hat{x} \leftarrow \bar{x}$ and $\bar{z} \leftarrow m$ and remove all nodes $N^i \in \mathcal{N}$ with $\underline{z}_i > \bar{z}$. Return to Step 5.

Consider a feasible fixing of the integer variables $u(\xi)$ defining a terminal node $N^i \in \mathcal{N}$. We denote the corresponding set restricting $y(\xi)$ by $Y_\xi(u(\xi)) \subset Y_\xi \subset \mathbb{R}^l$ and the optimal solution for the terminal node x_i^*.

Proposition 3.2. *Suppose that (SMIP) has optimal solution x^* with f_t convex and, for all integer fixings $u(\xi)$, x_i^* exists and $Y_\xi(u(\xi))$ is convex. Then the above BBPH algorithm, for any tolerance threshold $\epsilon > 0$, terminates in finitely many steps with $|\bar{z} - \underline{z}| < \epsilon$.*

Proof. Since, for each $N^i \in \mathcal{L}$, PH will terminate in a finite number of steps and, due to the legitimacy of the bounds in section 3.1, B&B will prune correctly, we have that in a finite number of steps, $\mathcal{L} = \emptyset$ and x^* will be the optimal solution of one of the remaining real SPs $N^i \in \mathcal{N}$, namely that with the smallest objective value. Next, as proven in [13], under the above assumptions, for each $N^i \in \mathcal{N}$,

$$\hat{x}_i^\nu \to x_i^* \text{ and } w_i^\nu \to w^*,$$

that is, PH will solve each SP until $\underline{z}_i \geq \bar{z}$ or $|\bar{z}_i - \underline{z}_i| < \epsilon$. Couple this with the fact that $\bar{z} = \min_i\{\bar{z}_i\}$, $\underline{z} = \min_i\{\underline{z}_i\}$ and we have, in finitely many steps, a solution $\hat{x}$ with $|\bar{z} - \underline{z}| < \epsilon$. □

More practically, for $N^i \in \mathcal{N}$, one could run a commercial solver to solve the (much smaller) extensive form of the fully real stochastic subproblem to 0 duality gap.

3.4. Remarks on the BBPH Implementation

Once PH has terminated at an outer node of the B&B tree, and that node is not pruned, a variable must be selected for branching. We use the PH solution for that node and the execution history of PH to make a selection. Our primary branching rule is to select the non-converged variable with the most diversity over the set of scenarios. For example, in the case of binary variables, this means selecting the variable for which $\hat{x}(\mathcal{G}_t(\xi))$ is closest to $\frac{1}{2}$. In the event that PH terminates with all discrete variables converged, then the secondary rule is to select for branching the variable that converged last during the PH execution. If all discrete variables converged at PH iteration zero and remained converged throughout the run, then an arbitrary variable is selected for branching. The outer node for processing is selected based on the lower bound found when PH was applied to its parent. The goal of this method is to raise the global lower bound.

As an aside, we note that for terminal and near-terminal nodes of the B&B tree, it is often relatively easy to obtain solutions via PH. Assuming that all primary non-leaf variables are discrete, once they have all been selected for branching and have been fixed, the subsequent outer B&B nodes become significantly easier to solve via PH, as the large SMIP subproblem becomes multiple *independent* SMIPs with $T-1$ stages. Of course, our algorithm does not branch on real variables so if there are primary, real-valued, non-leaf variables, this property does not hold. However, the resulting problem would still be much smaller and easier than it was before branching.

There are three main possibilities for parallelism in BBPH: outer node processing, scenario sub-problems for PH to solve at the outer nodes, and nodes of the B&B tree associated with the MIP solver for each scenario sub-problem in each iteration of PH. In our computational experiments and available BBPH implementation, we do not parallelize across BBPH outer nodes. However, we do parallelize the sub-problem solves for PH and the MIP solves for individual PH sub-problems (the latter via shared-memory thread-level parallelism available in commercial MIP solvers such as CPLEX and Gurobi).

4. Computational Experiments

In order to illustrate cases where B&B can improve over the solutions found by PH alone (i.e., equivalent to applying PH only to the root node of the B&B tree), we now conduct some representative computational experiments. In these

	EF	PH		BBPH		PH w / Bundles	
Instance	Objective	Time	Objective	Time	Objective	Time	Objective
1ef50	158653	2579	166848	18161	162163	31891	162946
2ef50	151060	1172	156211	15330	156211	9486	154065
3ef50	161466	2843	167871	33390	165733	30228	166174
4ef50	153854	2296	157229	18150	157229	8349	157697
5ef50	150401	1190	155002	8556	152686	7841	156109

Table 1: Test results for network flow minimization problem; times are wall-clock seconds.

experiments, our primary interest is in analyzing the potential for B&B to enhance PH rather than looking for the best possible way to solve any particular problem. Hence, we use the same parameterization for PH when used stand-alone as when PH is used as the solver for individual BBPH nodes. As a benchmark, we provide results for the direct solution of an extensive form of the SMIPs using an implementation that is available in the PySP library for stochastic programming [18], which is distributed as part of the larger Pyomo [10] Python library for optimization `www.pyomo.org`. The Python code for BBPH is also distributed with Pyomo / PySP. All experiments are conducted on a 96-core Intel Xeon workstation with 2.1GHz processors and 1TB of RAM.

We first consider five problem instances – denoted 1ef50 through 5ef50 – that are 50 scenario versions of a two-stage network design problem, with binary and continuous variables in both stages. These instances are available in the standard Pyomo download. The problem was introduced in [15] as an extension of a problem first introduced by Ruszczyński [14]. The PH algorithm makes use of techniques described in [16], given in configuration files that install with standard versions of PySP; the instances and PH configuration files are also available from the authors. We also consider "bundles" of scenarios as sub-problems in PH, as described in [8]. Bundling can be used to accelerate convergence of PH, at the expense of increased sub-problem solve times. In experiments where bundles are considered, they contain 5 scenarios apiece. PH is executed with 50 parallel solvers (competing for the same memory), each allowed to use up to 2 cores. The bundled variants are allowed more cores per bundle so that its wall clock time is roughly comparable.

Table 1 summarizes the computational results; times are wall-clock times given in seconds. The first column provide a snapshot of the extensive form solution obtained by the CPLEX 12.6.3 MIP solver, after a time limit of 10K seconds. It should be noted that CPLEX finds good solutions well before this time, and then expends significant effort improving them only slightly, while also trying to prove optimality. The subsequent four columns report the time and solution quality obtained by (1) the "standard" PH algorithm and (2) this same PH algorithm embedded in B&B (BBPH) running for ten branch-and-bound nodes. Because BBPH requires significantly more time than PH, we report in the final two columns the results for PH with bundling, to serve as a benchmark for a PH variant allowed a run time more comparable with that of BBPH.

Table 2 shows results analogous to those in Table 1 for a three stage version of

	EF		PH		BBPH	
Instance	Time	Objective	Time	Objective	Time	Objective
1ef10	10,046	160,964	1,934	166,189	18,606	163,647
2ef10	10,045	156,637	2,065	160,849	13,767	160,052
3ef10	3,424	157,025	1,748	166,406	13,581	160,032
4ef10	2,327	170,067	1,428	191,796	13,786	176,127
5ef10	4,295	161,840	2,432	169,287	14,911	168,542

Table 2: Test results for three stage network flow with a branching factor of ten for the second stage and three for the third; times are wall-clock seconds.

the problem that has thirty scenarios. As a practical matter for small problems like these, CPLEX does quite well on the extensive form. For these instances, BBPH offers better solutions than PH, but at the expense of more time.

Our experiments have shown that B&B may be required to improve solutions that PH alone can obtain. As is the case for B&B applied to MIPs, BBPH applied to SMIPs will often spend much of its time at the root node, establishing the best possible incumbent and associated bound. In such situations the proposed BBPH algorithm is still valuable in that it provides (1) assurance that branching could have been used if required and (2) an environment with a guarantee of eventual optimality for general SMIPs.

[1] Shabbir Ahmed. A scenario decomposition algorithm for 0-1 stochastic programs. Technical report, ISYE, Georgia Tech, 2013.

[2] Claus C Carøe and Rüdiger Schultz. Dual decomposition in stochastic integer programming. *Operations Research Letters*, 24(1):37–45, 1999.

[3] Teodor Gabriel Crainic, Xiaorui Fu, Michel Gendreau, Walter Rei, and Stein W. Wallace. Progressive hedging-based metaheuristics for stochastic network design. *Networks*, 58(2):114–124, 2011.

[4] Teodor Gabriel Crainic, Mike Hewitt, and Walter Rei. Scenario grouping in a progressive hedging-based meta-heuristic for stochastic network design. *Comput. Oper. Res.*, 43:90–99, March 2014.

[5] Marcelo L.L. dos Santos, Edson Luiz da Silva, Erlon Cristian Finardi, and Raphael E.C. Gonalves. Practical aspects in solving the medium-term operation planning problem of hydrothermal power systems by using the progressive hedging method. *International Journal of Electrical Power & Energy Systems*, 31(9):546 – 552, 2009. Power Systems Computation Conference (PSCC) 2008Power Systems Computation Conference (PSCC) 200816th Power Systems Computation Conference (PSCC), 2008.

[6] Laureano F. Escudero, Araceli Garín, María Merino, and Gloria Pérez. Bfc-msmip: an exact branch-and-fix coordination approach for solving multi-stage stochastic mixed 0–1 problems. *TOP*, 17(1):96–122, 2009.

[7] Yueyue Fan and Changzheng Liu. Solving stochastic transportation network protection problems using the progressive hedging-based method. *Networks and Spatial Economics*, 10(2):193–208, 2008.

[8] D. Gade, G. Hackebeil, S.M. Ryan, J.P. Watson, R.J.B. Wets, and D.L. Woodruff. Obtaining lower bounds from the progressive hedging algorithm for stochastic mixed-integer programs. *Mathematical Programming, Series B*, 2016.

[9] W.E. Hart, J.P. Watson, and D.L. Woodruff. Pyomo: Modeling and solving mathematical programs in Python. *Mathematical Programming Computation*, 3(3), 2011.

[10] W.E. Hart, J.P. Watson, and D.L. Woodruff. Pyomo: Modeling and solving mathematical programs in Python. *Mathematical Programming Computation*, 3(3), 2011.

[11] Lars M. Hvattum, Arne Lkketangen, and Gilbert Laporte. Solving a dynamic and stochastic vehicle routing problem with a sample scenario hedging heuristic. *Transportation Science*, 40(4):421–438, 2006.

[12] Lars Magnus Hvattum and Arne Løkketangen. Using scenario trees and progressive hedging forstochastic inventory routing problems. *Journal of Heuristics*, 15(6):527–557, 2008.

[13] R Tyrrell Rockafellar and Roger J-B Wets. Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research*, 16(1):119–147, 2004.

[14] A. Ruszczyński. Probabilistic programming with discrete distributions and precedence constrained knapsack polyhedra. *Mathematical Programming*, 93:195–215, 2002.

[15] J.-P. Watson, R. J.-B. Wets, and D.L. Woodruff. Scalable heuristics for a class of chance-constrained stochastic programs. *INFORMS Journal on Computing*, 22(4):543–554, October 2010.

[16] Jean-Paul Watson and David L Woodruff. Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems. *Computational Management Science*, 8(4):355–370, 2011.

[17] J.P. Watson, D.L. Woodruff, and W.E. Hart. PySP: Modeling and solving stochastic programs in Python. *Mathematical Programming Computation*, 4(2), 2012.

[18] J.P. Watson, D.L. Woodruff, and W.E. Hart. PySP: Modeling and solving stochastic programs in Python. *Mathematical Programming Computation*, 4(2), 2012.