A Progressive Hedging Based Branch-and-Bound Algorithm for Mixed-Integer Stochastic Programs

Semih Atakan, Suvrajeet Sen

February 26, 2018

Abstract. Progressive Hedging (PH) is a well-known algorithm for solving multi-stage stochastic convex optimization problems. Most previous extensions of PH for mixed-integer stochastic programs have been implemented without convergence guarantees. In this paper, we present a new framework that shows how PH can be utilized while guaranteeing convergence to globally optimal solutions of mixed-integer stochastic convex programs. We demonstrate the effectiveness of the proposed framework through computational experiments.

Keywords. multi-stage mixed-integer stochastic convex programming; progressive hedging.

1. Introduction

A wide variety of operational and financial problems are modeled as mixed-integer programs (MIPs), where a certain subset of the decision variables are restricted to assume integer values. Examples include scheduling, production and inventory planning, transportation, portfolio optimization, among many others. These problems are inherently difficult to handle as the decisions must be determined over a non-convex feasible set. Nonetheless, over the years, the substantial progress in integer programming has made it possible for many MIP solvers to scale up to formulations with millions of variables (see Bixby, 2002). Today, many real-life problems can be effectively handled with reasonable computational resources.

A still challenging venue for researchers is the incorporation of *data uncertainty* into these models. In real life, much of the input data is usually not known with certainty at the time the decisions are made. Ignoring the inherent uncertainty in the systems (for instance, by using the expected values of the random parameters) can lead to unfavorable decisions, which, in general, underperform in real-life situations. A general methodology to address uncertainty is to consider a sample of *scenarios* as a proxy for the random parameters, and optimize the decisions based on these outcomes. This leads to stochastic programming formulations, which are notorious even without the presence of integrality requirements. The reason is, the large number of samples necessary for statistical accuracy leads to massive scale problems. Consequently, naive optimization attempts on stochastic programming problems rarely succeed in providing high-quality solutions, in a consistent manner.

Effective solution methodologies for stochastic programming rely heavily on decomposition ideas, where the decomposition may either be performed with respect to the scenarios, or the time periods

S. Atakan and S. Sen are with the Daniel J. Epstein Department of Industrial and Systems Engineering, University of Southern California, Los Angeles, CA 90007 USA (e-mail: atakan@usc.edu; s.sen@usc.edu).

This research was funded by the NSF grant ECCS 1548847 and AFOSR grant FA9550-15-1-0267.

(stages) in the decision model. The L-shaped method (Van Slyke and Wets, 1969) for two-stage stochastic linear programs (SLPs), and the nested Benders' decomposition (Louveaux, 1980; Birge, 1985) for multi-stage SLPs can be considered as representatives of the first category. Examples within the context of scenario decomposition involve the *Progressive Hedging* algorithm of Rockafellar and Wets (1991) for multi-stage stochastic convex programs, and the diagonal quadratic approximation of Mulvey and Ruszczyński (1995) for multi-stage SLPs.

For mixed-integer stochastic programs (MISPs), convergence to a globally-optimal solution in a computationally-effective manner is hard-to-achieve due to the non-convex nature of the feasible set. In most cases, global optimality in ensured via branch-and-bound- or cutting-plane-based procedures, although a primal approach based on algebraic arguments have also been proposed in Hemmecke and Schultz (2003). For multi-stage problems, the dual decomposition of Carøe and Schultz (1999) and the branch-and-price algorithm of Lulli and Sen (2004) use scenario decomposition within a branch-and-bound framework. For two-stage problems, cutting-planes are used for the proper convexification of the second-stage value functions. For instance, the algorithms of Qi and Sen (2016) and Ralphs and Hassanzadeh (2014) are applicable to a wide variety of problems. Further assumptions (such as, pure or mixed integers, binary or general integers) on the individual stages lead to computationally faster procedures (see Laporte and Louveaux, 1993; Carøe and Tind, 1998; Sen and Higle, 2005; Sen and Sherali, 2006; Gade et al., 2014; Zhang and Küçükyavuz, 2014; Angulo et al., 2014). A very informative table that summarizes the characteristics of several algorithms can be found in Ralphs and Hassanzadeh (2014) (see also Trapp et al. (2013)).

Despite these advances, the need for an efficient MISP algorithm without restrictive assumptions – especially in the multi-stage case – remains as an important question. In addition, very little is known for the broader class of mixed-integer stochastic convex programs (MISCPs). The purpose of this study is to fill these gaps in the literature. In particular, we seek to address the following question:

> *Is it possible to leverage the advances in the stochastic convex programming literature in order to develop a solution framework for multi-stage MISCP with relatively weaker assumptions?*

To answer this question, we will rely on the Progressive Hedging algorithm of Rockafellar and Wets (1991). The extent at which our proposed framework can address this will be revealed in the following sections.

In light of this discussion, we concern ourselves over finding an optimal *policy* to an optimization problem, where the decision-maker faces data uncertainty, and a subset of its decisions must meet integrality requirements. The uncertainty is characterized by scenarios, which can either be formed by extrapolating from the past data, or more rigorously, by assuming the existence of random variables that govern the modeled system. The uncertainty is represented with the random vector ξ, whose realizations ξ_s, $s = 1 \dots S$, reflect the state of information under scenario s. The use of a random vector necessitates the definition of a probability space $(\Omega, \mathcal{F}, \mathcal{P})$ over which the random variables can be defined. The probability of a scenario is then denoted with $p_s = P(\xi(\omega) = \xi_s)$ where $p_s > 0$, $\forall s$, and $\sum_{s=1}^{S} p_s = 1$. A finite-dimensional problem will be posed by assuming either the sample space Ω has finite support, or letting S represent the finite number of sampled scenarios. Following this discussion, a policy $X(\xi)$ should be interpreted as a random function whose response to scenario s is a decision vector $X(\xi_s) = x_s$. We find it convenient to represent a policy with the mapping $X : s \to x_s$.

A policy X can be partitioned into T components $(X_1, \dots, X_T)$ where X_t should be interpreted as the segment of the policy to be followed at time period t. Likewise, the filtration $\mathcal{F}$ of the probability space is partitioned into periods as $\mathcal{F}_1 \dots \mathcal{F}_T$, where $\mathcal{F}_t \subseteq \mathcal{F}_{t'}$ for $t' \geq t$. A logical requirement is that an *implementable* policy at an arbitrary stage should have no foresight on the exact realizations of future events. Consequently, for any set of scenarios $F \subseteq \mathcal{F}_t$ which are *observationally indistinguishable* at time t, we must have x_{st} equal to each other $\forall s \in F$. For instance, at the initial time period, a policy-maker cannot be anticipative of the future, and therefore must take a single action that is not contingent on any specific scenario (i.e., $x_{s1} = \bar{x}$, $\forall s = 1 \dots S$). In the stochastic programming literature, such restrictions are commonly referred to as the *nonanticipativity constraints.* These constraints will be collectively represented by the set $\mathcal{N}$. Accordingly, we can ensure that a policy is nonanticipative with the restriction $X \in \mathcal{N}$.

For any given scenario s, the decision maker faces the following *scenario subproblem*:

$$f(x_s, \xi_s) = \min_{x_s} \left\{ f_s(x_s) \mid x_s \in C_s,\, x_s \in \mathbb{R}^{n_r} \times \mathbb{Z}^{n_z} \right\}.$$

Above, $C_s \subseteq \mathbb{R}^{n_r + n_z}$ is assumed to be a closed, convex, and nonempty set, whereas $\mathbb{Z}^{n_z}$ is the integer lattice generated by the n_z unit vectors of $\mathbb{R}^{n_z}$. We will use n to refer to the size of the vector x_s, i.e., $n = n_z + n_r$. The objective function $f_s : \mathbb{R}^n \to \mathbb{R}$ is assumed to be a proper convex function, possibly extended real-valued. By defining the mapping $F(X, \xi) : s \to f(x_s, \xi_s)$, we can formally state the problem of interest as follows:

$$\min_{X} \mathbb{E}[\, F(X, \xi) \,] \;\; \text{subject to:} \;\; X \in \mathcal{N}. \tag{$\mathcal{P}$}$$

Throughout this paper, we will assume that $\mathcal{P}$ has at least one solution. A particular solution, $X^\star$, will said to be optimal whenever it is the argument that attains the minimum in problem $\mathcal{P}$. Notice that $X^\star$ being the optimum does not necessarily mean that the decision vectors in its range ($x_s^\star$) are the minimizers of $f(x_s, \xi_s)$, since the nonanticipativity constraints are in effect. Indeed, it should be clear that relaxing $\mathcal{N}$ leads to a problem that can be decomposed into its individual scenarios, whereas relaxing the integrality restrictions converts $\mathcal{P}$ into a convex problem. Our framework will leverage both of these observations.

The specific contributions and features of our study are summarized with the following bullets.

- We develop a novel branch-and-bound framework for using the Progressive Hedging algorithm to compute *optimal solutions* to multi-stage MISCPs. Our study involves a thorough treatment of the convergence of the proposed framework and guarantees optimality under mild assumptions.

- Within our framework, we implement specialized algorithms that can handle two-stage MISCPs with binary first-stage and arbitrary second-stage variables. This family of problems have, to date, continued to challenge the stochastic programming community, and often arises in important fields such as telecommunications (e.g., server location problems) and power-systems planning (e.g., unit commitment problems with quadratic production costs). Furthermore, the implemented algorithms serve to bridge the gap between resource-directive and price-directive decomposition paradigms.

- Our framework is tailored to process, mainly, *decomposed convex-relaxations* of the original MISP. Owing to this design, we expect our algorithms to possess better computational performance than its contenders which process deterministic MIP subproblems. In fact, this feature of our framework may yield the highest benefit on problems involving more arbitrary convex components (e.g., quadratic objectives), as solving mixed-integer convex problems can be substantially more challenging than solving linear ones.

- Among our suggested algorithms, the design of a particular one performs *partial optimizations* of the decomposed convex-relaxations. As a result, we expect our algorithm to spend minimal effort on regions of the feasible set, bearing no promise. In addition, such a design allows our algorithm to be more robust against suboptimal parameter settings of the PH algorithm.

- We provide extensive computational experiments on a variety of instances, including a family of extensively-studied linear benchmark instances, and their quadratic counterparts. Our experiments reveal that the assertions of convergence for our specialized algorithms can be observed in short amounts of time for standard test-instances involving up to two thousand scenarios.

The organization of this paper is as follows. In §2, we present a summary of the Progressive Hedging algorithm and discuss its use in the literature as a heuristic method for MISPs. In §3, we develop our framework for using Progressive Hedging to compute optimal solutions to MISCPs in a somewhat general setting. In §4, we specialize this framework to the aforementioned class of two-stage MISCPs. The developed algorithms allow our framework, still in its infancy, to compete with its contenders. We provide a computational study in §5 which demonstrates the performance of our framework. Concluding remarks are given in §6.

2. Progressive Hedging

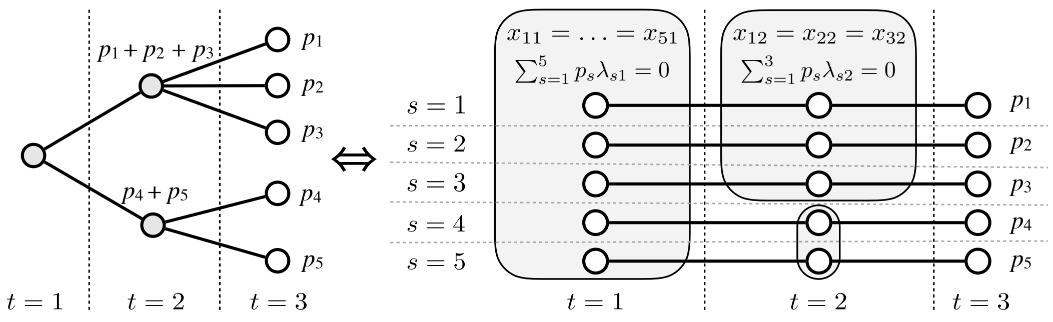
For convex stochastic optimization problems, Rockafellar and Wets (1991) proposed the Progressive Hedging (PH) algorithm which, after more than two decades, remains to be one of the more popular approaches to multi-stage stochastic optimization. In this section, we give a brief overview of their algorithm, along with a discussion of its use for MISPs.

The Algorithm In most stochastic programming problems, the nonanticipativity constraints tie soaring numbers of scenario subproblems to each other. A natural direction is to dualize these constraints, after which, the subproblems can be independently optimized. However, the coordination of the dual variables may not be trivial, and the optimal dual prices may be difficult to obtain. As a result, the presumed benefit of decomposition may never be gained in practice through naive algorithms. Several strategies can be pursued to efficiently coordinate the dual variables. For instance, in the context of mixed-integer stochastic programming problems, the dual decomposition of Carøe and Schultz (1999) uses a bundle method (e.g., Helmberg, 2000), whereas Lulli and Sen (2004) adopt a branch-and-price framework. In their PH algorithm, Rockafellar and Wets (1991) address this concern by using a combination of projection and proximal point algorithms (Rockafellar, 1976).

We begin the discussion by introducing the multipliers λ_s, $\forall s = 1 \dots S$, and the associated mapping $\Lambda : s \to \lambda_s$. Multiplying the outcomes of this mapping with the scenario probabilities, we obtain $p_s\lambda_s$. These values should be interpreted as the dual multipliers of the nonanticipativity constraints associated with scenario s. We denote a nonanticipative policy with $\hat{X} : s \to \hat{x}_s$. Using $\hat{X}$, the nonanticipativity restrictions can be explicitly stated as $\mathcal{N} = \{X \mid X = \hat{X}\}$. Accordingly, letting $\mathcal{Y}$ represent the set of feasible dual multipliers, the ordinary Lagrangian, achieved through the dualization of the constraint $X - \hat{X} = 0$, can be defined as follows:

$$L(X, \hat{X}, \Lambda) = \sum_{s=1}^{S} p_s\big(f_s(x_s) + \lambda_s(x_s - \hat{x}_s)\big) \quad \Leftrightarrow \quad L(X, \Lambda) = \sum_{s=1}^{S} p_s\big(f_s(x_s) + \lambda_s x_s\big),\ \forall \Lambda \in \mathcal{Y}.$$

The equivalence of these two definitions stems from the dual feasibility requirements embedded in $\mathcal{Y}$, which itself is a consequence of the projected optimality condition $0 \in \partial_{\hat{X}} L(X, \hat{X}, \Lambda)$, i.e., the



$p_1 + p_2 + p_3$
p_1
p_2
p_3
$p_4 + p_5$
p_4
p_5
$t = 1$
$t = 2$
$t = 3$
$\Leftrightarrow$
$x_{11} = \ldots = x_{51}$
$\sum_{s=1}^{5} p_s \lambda_{s1} = 0$
$x_{12} = x_{22} = x_{32}$
$\sum_{s=1}^{3} p_s \lambda_{s2} = 0$
$s = 1$
$s = 2$
$s = 3$
$s = 4$
$s = 5$

Figure 1: The scenario tree representation of a three-stage problem with five scenarios (left) and an equivalent lifted formulation for scenario decomposition, based on Jörnsten et al. (1985) and Guignard and Kim (1987) (each circle denotes a decision).

subgradient of $L(X, \hat{X}, \Lambda)$ with respect to $\hat{X}$ while keeping (X, Λ) fixed. More specifically, we have

$$\mathcal{Y} = \left\{ \Lambda \;\middle|\; \mathbb{E}[\,\Lambda_t \mid F\,] = \frac{\sum_{s\in F} p_s \lambda_{st}}{\sum_{s \in F} p_s} = 0,\ \forall F \in \mathcal{F}_t,\ t = 1 \ldots T \right\},$$

where, Λ_t represents the multipliers corresponding to the nonanticipativity constraints of time period t. Then, for any $\Lambda \in \mathcal{Y}$ and (nonanticipative) $\hat{X}$, we have

$$\mathbb{E}[\,\Lambda_t \cdot \hat{X}_t\,] = \mathbb{E}\big[\,\mathbb{E}[\,\Lambda_t \cdot \hat{X}_t \mid F\,]\,\big] = \mathbb{E}\big[\,\hat{X}_t \cdot \mathbb{E}[\,\Lambda_t \mid F\,]\,\big] = 0, \quad \forall F \in \mathcal{F}_t.$$

Therefore, when $\Lambda \in \mathcal{Y}$, the terms $-p_s \lambda_s \hat{x}_s$, $\forall s$, can be omitted from the definition of $L(X, \hat{X}, \Lambda)$. Following this Lagrangian dualization scheme leads to a scenario-wise decomposition of a stochastic program, which we illustrate in Figure 1.

Let k be the iteration index, and let the superscript k denote the solution of the corresponding variable at the end of the k^{th} iteration. We describe an iteration k of the PH algorithm using the following saddle function:

$$\sum_{s=1}^{S} p_s \left(f_s(x_s) + \lambda_s (x_s - \hat{x}_s) + \frac{\rho}{2} \|x_s - \hat{x}_s\|_2^2 - \frac{1}{2\rho} \|\lambda_s - \lambda_s^{k-1}\|_2^2 \right). \tag{1}$$

This function can be interpreted as an extension of $L(X, \hat{X}, \Lambda)$ with two quadratic regularization terms, each having differing objectives. The terms $\|x_s - \hat{x}_s\|_2^2$, $\forall s$, induce better consensus decisions at the end of an optimization, since $\hat{x}_s$ involves components which are shared among certain subsets of scenarios, due to nonanticipativity requirements. The terms $\|\lambda_s - \lambda_s^{k-1}\|_2^2$, $\forall s$, on the other hand, are proximal penalties which introduce stability to the dual iterates by penalizing their deviations between consecutive iterations. The parameter ρ is a positive constant that scales the impact of these regularizers. An undesired feature of (1) is the presence of $\hat{x}_s$ inside the quadratic terms, which destroys the separability of this function with respect to scenarios. The PH algorithm handles this issue by successively projecting two variables and optimizing the third in (1). Accordingly, the optimizations with respect to each variable $(X, \hat{X}, \Lambda)$ can be performed independently across the scenarios. Notice that this is the same pattern followed in the broader genre of algorithms, known as the alternating direction method of multipliers (ADMM) (see, for instance, Boyd, 2010). Indeed, this interpretation of the PH algorithm can be seen as a special instance of an ADMM[*].

[*]An alternative statement of the problem is given in Rockafellar and Wets (1991). The authors pose the problem as a *projected* saddle function that is extended with proximal terms on the $\hat{X}$ and Λ variables. The proof of convergence for the PH algorithm relies on this interpretation.

An iteration k of the PH algorithm begins with fixing $(\hat{X}, \Lambda)$ to their most recent solutions $(\hat{X}^{k-1}, \Lambda^{k-1})$, and solving the following *modified subproblems* for X:

$$\min_{x_s} \left\{ f_s(x_s) + \lambda_s^{k-1} x_s + \frac{\rho}{2} \|x_s - \hat{x}_s^{k-1}\|_2^2 \;\middle|\; x_s \in C_s \right\}, \qquad \forall s = 1 \dots S. \tag{$\mathcal{P}_s$}$$

Above, $\hat{X}^{k-1}$ can be perceived as a (nonanticipative) *incumbent*, with respect to which, the variables X are regularized. From a computational perspective, this is generally the most expensive step of the PH algorithm, as it requires the optimization of S convex programs. Next, X is fixed to the solution obtained from the modified subproblems X^k, Λ is kept at Λ^{k-1}, and the updated form of (1) is optimized with respect to $\hat{X}$. The optimal solution of this minimization admits a closed form solution, which is given by

$$\hat{x}_{st}^k = \mathbb{E}[\, X_t^k \mid F\,],\ \forall s \in F,\ F \in \mathcal{F}_t,\ t = 1 \dots T,\ \text{ where, } \mathbb{E}\left[\, X_t \mid F\,\right] = \frac{\sum_{s \in F} p_s x_{st}^k}{\sum_{s \in F} p_s}. \tag{2}$$

It is easy to see that the above operation is a projection from the set of arbitrary policies to the set of nonanticipative solutions. Therefore, the resulting policy $\hat{X}^k$ will always be nonanticipative. However, this should not necessarily suggest that $\hat{X}^k$ is feasible with respect to C_s, $\forall s$. The final step of an iteration is the update of the dual multipliers, which is performed by fixing $(X,\, \hat{X})$ to $(X^k,\, \hat{X}^k)$ and maximizing (1) with respect to Λ. Then, we have

$$\lambda_s^k = \lambda_s^{k-1} + \rho(x_s^k - \hat{x}_s^k), \quad \forall s = 1 \dots S. \tag{3}$$

This completes one iteration of the PH algorithm. Observe that individual steps of the algorithm can be performed in parallel, which can be exploited to obtain computationally-powerful implementations.

The PH algorithm is carried out until some termination criteria, such as $\sum_{s=1}^S p_s \|x_s^k - \hat{x}_s^k\|_2 < \varepsilon_{ph}$ for a sufficiently small but positive ε_{ph} is achieved. It is common practice to set x_s^0 as a minimizer in $f(x_s, \xi_s)$, since by doing so, we also get an immediate lower bound $\sum_{s=1}^S p_s f_s(x_s^0)$ to the optimal objective value of the original problem. Similarly, letting $\lambda_s^0 = 0$, $\forall s$, leads to $\Lambda^0 \in \mathcal{Y}$, and by induction, one can show that Λ^k will then belong to $\mathcal{Y}$ for any arbitrary k. Finally, using x_s^0, $\forall s$, one can also determine an initial incumbent $\hat{X}^0$ through the procedure in (2). A pseudocode of PH is provided in Algorithm 1.

Algorithm 1: Progressive Hedging

```
1 Input: $\hat{X}^0$, $X^0$, $\Lambda^0$, $\varepsilon_{ph} > 0$.
2 Let $k = 1$
3 while $\sum_{s=1}^S p_s \|x_s^{k-1} - \hat{x}_s^{k-1}\| < \varepsilon_{ph}$ do
4     Solve $\mathcal{P}_s$ for each $s = 1 \dots S$, and obtain $X^k$.
5     Form the nonanticipative incumbent $\hat{X}^k$ according to (2).
6     Update the dual multipliers according to (3).
7     Let $k = k + 1$
8 end
```

The proof of convergence for the PH algorithm relies on the theory of proximal point algorithms. For details, we refer the reader to the original paper. In what follows, we summarize two important results which will be necessary for the development of our algorithm. We begin with an assumption.

A1. All modified subproblems ($\mathcal{P}_s$, $\forall s = 1 \dots S$) encountered during the course of the PH algorithm can be optimized in finitely-many steps.

Problems involving linear or quadratic objectives and polyhedral feasible sets clearly satisfy this requirement. Under this assumption, the main convergence result of the PH algorithm is given below.

Theorem 1 (Rockafellar and Wets (1991)). *Consider a convex stochastic optimization problem which satisfies **A1** and the following constraint qualification condition:*

$$\textit{The only } \Lambda \textit{ satisfying } -\lambda_s \in N_{C_s}(x_s^*) \;\; \forall s = 1 \dots S \textit{ is } \Lambda = 0, \tag{$\mathcal{Q}$}$$

where $N_{C_s}(x_s^*)$ is the normal cone associated with the set C_s. Assume that the problem has an optimal solution $(\hat{X}^*, \Lambda^*)$, which is not necessarily unique. Let $\{\hat{X}^k, \Lambda^k\}_{k=1}^{\infty}$ be the sequence generated by Algorithm 1 starting at an arbitrary point $(\hat{X}^0, \Lambda^0)$. Then, when $k \to \infty$, we have

$$\hat{X}^k \to \hat{X}^* \textit{ and } \Lambda^k \to \Lambda^*.$$

Proof. See Theorem 5.1 of Rockafellar and Wets (1991). ■

Remark 1. The constraint qualification condition ($\mathcal{Q}$) is superfluous when the problem involves linear or quadratic objective and polyhedral feasible set (see Theorem 4.3 in Rockafellar and Wets (1991)).

The existence of finite algorithms for arbitrary families of convex optimization problems is not evident. In such cases, Rockafellar and Wets (1991) suggest optimizing $\mathcal{P}_s$ in an *approximate sense*. Following their scheme, **A1** can be relaxed and, under fairly general conditions, the convergence result in Theorem 1 will remain valid (see Theorem 5.4 and the related discussion in Rockafellar and Wets (1991)). For this reason and to keep this paper concise, **A1** will be presumed throughout our study.

Notice that an optimal solution pair $(\hat{X}^*, \Lambda^*)$ leads to an optimal solution X^*, since the condition $X^* = \hat{X}^*$ is necessary for the PH algorithm to terminate. The following theorem provides a saddle point characterization for an optimal solution (X^*, Λ^*), obtained by the PH algorithm, to the underlying stochastic convex optimization problem.

Theorem 2 (Rockafellar and Wets (1991)). *The optimal solution pair $(X^*, \Lambda^*) \in \mathcal{N} \times \mathcal{Y}$ obtained by the PH algorithm is a saddle point of the ordinary Lagrangian $L(X, \Lambda)$, relative to minimizing over $x_s \in C_s$, $\forall s = 1 \dots S$, and maximizing over $\Lambda \in \mathcal{Y}$.*

Proof. See Theorem 4.2 of Rockafellar and Wets (1991). ■

Use of PH on MISPs Contrary to the case of convex stochastic optimization problems, strong theoretical guarantees on the convergence of the PH algorithm for non-convex problems are not available. In the case of MISPs, the modified subproblems solved at each iteration become non-convex because the feasible set of $\mathcal{P}_s$ ($C_s \cap (\mathbb{R}^{n_r} \times \mathbb{Z}^{n_z})$) is discrete. Empirical evidence indicates that executing the PH algorithm with MIP subproblems may not always lead to an optimal solution, and can lead to oscillatory behavior of the iterates (Gade et al., 2016). However, such difficulties can be overcome with algorithmic novelties, and the PH algorithm can be used as a viable heuristic approach for solving MISPs. For a variety of mixed-integer stochastic linear programming problems, it has been shown that the PH algorithm indeed provides promising solutions (see, for instance, Løkketangen and Woodruff, 1996; Ryan et al., 2013). Nevertheless, due to non-convexity, the convergence of the algorithm to a (near)-optimal solution relies heavily on the quality of the implementations, rather than mathematical assertions. In the remainder of this paper, we will refer to similar uses of the PH algorithm on MISPs as PH-Heuristic.

Several studies aim to improve the computational effectiveness of PH-Heuristic. For instance, Watson and Woodruff (2010) analyze the effect of the penalty parameter ρ on the speed and the solution quality of the algorithm. Another line of research aims to reduce the duality gap that arise due to the relaxation of the nonanticipativity constraints in a mixed-integer program. Gade et al. (2016); Ryan et al. (2016) show that enforcing the nonanticipativity restrictions for subsets of scenarios *(senario bundling)* is shown to improve the quality of solutions. While not directly related to PH-Heuristic, studies on eliminating the duality gap through augmented Lagrangians (Boland and Eberhard, 2014; Feizollahi et al., 2015) may carry significant potential in providing a theoretical basis for such algorithms.

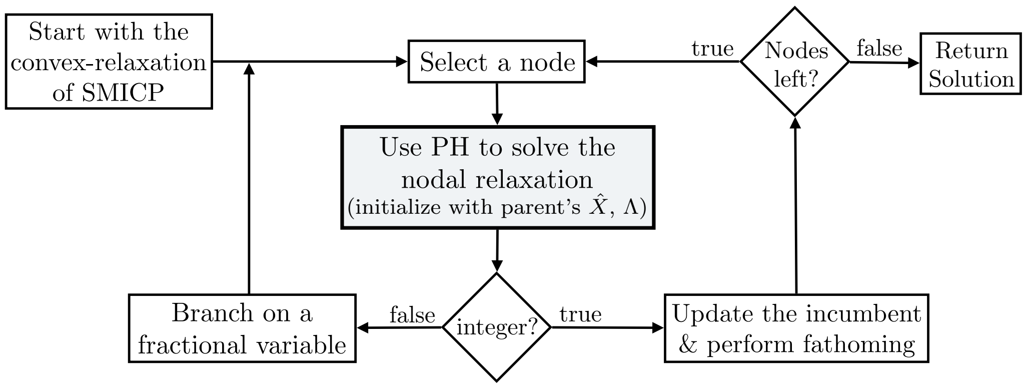
For the purpose of achieving convergence guarantees, Guo et al. (2015) hybridize the PH-Heuristic with the dual decomposition of Carøe and Schultz (1999). Boland et al. (2016) propose an extension of the PH algorithm, which converges to the optimal objective value of the Lagrangian dual problem, achieved through the dualization of the nonanticipativity constraints in $\mathcal{P}$. Their study incorporates a simplicial decomposition method (SDM) into the PH algorithm. The SDM successively provides inner-approximations of the convex hull of $C_s \cap (\mathbb{R}^{n_r} \times \mathbb{Z}^{n_z})$, therefore resolves the non-convexity and leads to a convergent algorithm. The suggested method is designed to solve one MIP subproblem per scenario, therefore achieves somewhat similar computational performance as the aforementioned uses of the PH algorithm. Finally, the recent work of Barnett et al. (2017) propose to use a branch-and-bound procedure as a wrapper for the PH-Heuristic. Their framework uses the incumbents $(\hat{X}^k)$ to partition the feasible region over which PH-Heuristic is iterated. The components of this partition are fathomed based on Lagrangian bounds on their optimal objective values, which are obtained by solving S mixed-integer subproblems (see Gade et al., 2016). The provided computational experiments reveal that the suggested framework spends substantially more computational effort than the standard PH-Heuristic in order to achieve higher-quality solutions, however lags somewhat behind the performance of a commercial mixed-integer programming solver that is directly applied to $\mathcal{P}$. These outcomes might be interpreted as symptoms of heavy-reliance on mixed-integer programming technology in iterative decomposition algorithms.

We emphasize that the PH-Heuristic algorithm and its aforementioned extensions require the solution at least S mixed-integer programming subproblems at each iteration. In general, a stochastic program is only meaningful when a large-enough set of scenarios are considered. Depending on how large the sample size is, the computational effort expended on iterating the algorithm may actually defeat the purpose of decomposition. We intend our framework to remedy this.

3. General Framework of PH with Branch and Bound

The power of branch-and-bound strategies in dealing with the non-convex feasible sets of MIPs is undisputed. In our framework, we integrate their potential with the scenario-decomposition capability of the PH algorithm. The resulting framework, which we refer to as PH-BAB, can be perceived as a branch-and-bound method that uses the PH algorithm to solve the nodal relaxations, approximately (see Figure 2).

Let $Q_s = \{x_s \in \mathbb{R}^n \mid l_s \leq x_s \leq u_s\}$ denote the feasible set defined by bounds on the vector x_s. The mapping $Q : s \rightarrow Q_s$ will be convenient to collectively represent the bounds across all scenarios. Accordingly, at any given branch-and-bound node σ, the corresponding mapping will be represented with $Q^\sigma : s \rightarrow Q^\sigma_s$. The feasible sets of individual scenario subproblems at node σ, i.e., $C_s \cap Q^\sigma_s$, will be denoted with the short-hand notation C^σ_s. Whenever we say that node σ is being solved, we mean



Start with the convex-relaxation of SMICP
Select a node
true
Nodes left?
false
Return Solution
Use PH to solve the nodal relaxation (initialize with parent's $\hat{X}$, Λ)
Branch on a fractional variable
false
integer?
true
Update the incumbent & perform fathoming

Figure 2: A basic PH-BAB algorithm.

that the following nodal relaxation is being optimized:

$$z(\sigma) = \min_{X} \mathbb{E}[\, f_\sigma(X, \xi)\,] \text{ s.t. } X \in \mathcal{N}, \text{ where, } f_\sigma(x_s, \xi_s) = \min_{x_s} \{\, f_s(x_s) \mid x_s \in C_s^\sigma \,\}. \qquad (\mathcal{R}_\sigma)$$

The root node of the branch-and-bound tree, denoted with o, will possess the bound constraints Q^o. Without loss of generality, we assume that $C_s \equiv C_s^o$, $\forall s = 1 \dots S$. Therefore, node o should be interpreted as the convex relaxation of $\mathcal{P}$, achieved by removing the integrality restrictions. The pseudocode of a basic PH-BAB algorithm is provided in Algorithm 2.

Algorithm 2: A Basic PH-BAB Algorithm

```
1  Initialization: Create the root node $o$, let $\mathcal{H} \leftarrow \{o\}$. Set $z_U = +\infty$, $z_L = -\infty$, $\hat{X}_U = \texttt{NULL}$.
2  while $\mathcal{H} \neq \emptyset$ do
3      Pick node $\sigma$ from $\mathcal{H}$, and let $\mathcal{H} \leftarrow \mathcal{H} \setminus \{\sigma\}$.
4      Determine the feasibility of $\sigma$.
5      if $\sigma$ is feasible then                                        // else, $\sigma$ is fathomed
6          Invoke the PH algorithm to solve $\sigma$.
7          Get solution $\hat{X}^\sigma$ of $\sigma$, and its optimal objective value $z(\sigma)$.
8          if $z(\sigma) < z_U$ then                                       // else, $\sigma$ is fathomed
9              if $\hat{X}^\sigma$ is fractional then
10                 Pick a fractional component $\hat{x}^\sigma_{stj}$ of $\hat{X}^\sigma$.
11                 Create nodes $\sigma^+$ and $\sigma^-$. Let $Q^{\sigma^+} = Q^{\sigma^-} = Q^\sigma$, while introducing the additional
                    restrictions $l_{stj} = \lceil \hat{x}^\sigma_{stj} \rceil$, $\forall s \in F \subseteq \mathcal{F}_t$ to $Q^{\sigma^+}$, and $u_{stj} = \lfloor \hat{x}^\sigma_{stj} \rfloor$, $\forall s \in F \subseteq \mathcal{F}_t$ to $Q^{\sigma^-}$.
12                 Let $\mathcal{H} \leftarrow \mathcal{H} \cup \{\sigma^-, \sigma^+\}$.
13             else
14                 Let $\hat{X}_U = \hat{X}^\sigma$, and $z_U = z(\sigma)$.
15                 Fathom: $\mathcal{H} \leftarrow \mathcal{H} \setminus \{\sigma' \mid z(\sigma') \geq z_U\}$.
16             end
17         end
18     end
19     Let $z_L = \min\{z(\sigma) \mid \sigma \in \mathcal{H}\}$.
20 end
```

We draw important distinctions on the use of the branch-and-bound method among the PH-BAB and PH-Heuristic algorithms. Both algorithms leverage the capability of branch-and-bound for handling non-convex feasible sets, however, their goals are different. In PH-BAB, we seek an optimal solution to $\mathcal{P}$, and form a single branch-and-bound tree for the feasible set of the *entire problem*. In the latter, branch-and-bound trees are formed to find optimal solutions to *modified subproblems*, which are modeled as mixed-integer convex programs (MICPs). In fact, in the PH-Heuristic algorithm, the branching decisions are performed on X variables – independently – across the scenarios. As a result, the optimal solutions for the modified subproblems need not agree with each other with respect to the nonanticipativity requirements, and may even produce incumbents $\hat{X}$ that disobey integrality restrictions (e.g., consider the averaging of binary variables). Moreover, after an iteration is completed, the branch-and-bound trees are usually discarded, and information are not conveyed from one iteration to another. In contrast, the single branch-and-bound tree of the PH-BAB algorithm progressively partitions the feasible set of $\mathcal{P}$ into finer feasible regions. The branching essentially occurs on $\hat{X}$, which is reflected on X by *simultaneously* branching on the associated observationally-indistinguishable decisions (see line 11 of Algorithm 2). With this branching strategy, solutions that violate the nonanticipativity restrictions are progressively discarded from future consideration. As a result, as the depth of the branch-and-bound tree grows, the nonanticipativity requirements imposed on the integer variables can be attained with greater ease. Efficient implementations of PH-BAB can easily exploit this feature by giving branching priorities on the variables in the increasing order of their time periods.

As in any other branch-and-bound-based method, the PH-BAB can significantly benefit from warm starts when solving its nodal relaxations. Indeed, the initialization of the PH algorithm that is suggested in §2, should only be invoked at the root node, as it involves the optimization of S minimization problems. For all other branch-and-bound nodes, the final primal and dual solutions of the parent node can be used as a starting point. For faster primal convergence, the primal solution inherited from the parent node should be projected onto the set defined by the new variable bounds. These simple considerations can carry significant potential in improving the performance of PH-BAB implementations.

We leave further computational details to later sections, and continue with a discussion on the convergence of the proposed method. The following assumptions will be necessary for this purpose:

A2. All integer variables in problem $\mathcal{P}$ have finite lower and upper bounds.

A3. Each feasible nodal relaxation satisfies the constraint qualification condition $\mathcal{Q}$.

A4. A certificate of feasibility or infeasibility for each nodal relaxation (i.e., for the set $\mathcal{N} \cap \left(\times_{s=1}^{S} C_s^{\sigma}\right)$) can be computed in finite time.

A5. For each feasible node, the PH algorithm yields an optimal solution in finite time.

The presence of **A2** does not restrict the practicality of our framework as arbitrarily loose bounds can be enforced on unbounded integer variables. Assumption **A3** is a technical condition that is satisfied in most problem formulations, and holds when the sets C_s, $\forall s$, are polyhedral (see Remark 1). Assumption **A4** may lead to concern as infeasibility detection for arbitrary convex problems is not trivial. For well-understood families of convex optimization problems (e.g., linear or quadratic programs), the state-of-the-art solvers are equipped with mechanisms which can detect the consistency of C_s^{σ} and determine the feasibility of the corresponding subproblem. Pathological cases may still occur when $C_s^{\sigma} \neq \emptyset$, $\forall s = 1 \ldots S$, but $\mathcal{N} \cap \left(\times_{s=1}^{S} C_s^{\sigma}\right) = \emptyset$, in which case the infeasibility must be detected with alternative means, for instance, by considering a feasibility problem before decomposing the nodal relaxation. For most practical purposes, such issues seldom occur, and therefore will not be elaborated further, but instead, left for future investigation.

While these three assumptions will be necessary throughout this paper, we state **A5** exclusively for the next proposition.

Proposition 1. *For problem $\mathcal{P}$ which satisfies **A2-A5**, Algorithm 2 yields an optimal solution in finite number of steps.*

Proof. Algorithm 2 differs from a standard branch-and-bound algorithm only with respect to its simultaneous branching rule on observationally-indistinguishable decisions. However, in a feasible solution, these variables must assume equal values due to nonanticipativity restrictions, therefore the branching rule can only discard nodes that are surely infeasible. Accordingly, the proof of convergence follows that of a standard branch-and-bound algorithm, performing exact evaluations. For a rigorous proof, we refer the reader to Horst and Tuy (2013), but we provide the basic argument below.

At every iteration of the algorithm, a single node σ is removed from $\mathcal{H}$ for processing. By **A4**, an infeasible node can be detected and discarded in finite time. Due to **A5**, a feasible node σ can be optimized by the PH algorithm in finite time. If this optimization leads to an integer-feasible solution, the cardinality of $\mathcal{H}$ must reduce in the next iteration, as $\mathcal{H}$ will no longer contain σ. If the optimization produces a fractional solution, branching will occur and $\mathcal{H}$ will gain one more element. Due to **A2**, the feasible set of the problem can only contain a finite number of integer points. As a result, branching can only occur finitely-many times, therefore $\mathcal{H}$ must empty in a finite number of iterations. At this point, the incumbent solution will be optimal to $\mathcal{P}$. ■

A critical concern can be raised to **A5**, as the finite termination of the PH algorithm is not guaranteed by the assertion given in Theorem 1. Alternative PH-BAB implementations can be studied to ensure global convergence. In the rest of this section, we study a particular design where the nodes of the branch-and-bound tree are partially solved, and fathoming is performed according to bounds on their optimal objective values. In particular, after each (partial) optimization of node σ, a lower bound ($z_L(\sigma)$) and an upper bound ($z_U(\sigma)$) will be computed on its optimal objective value ($z(\sigma)$). In our setup, these bounds will be derived from the solutions generated by the PH algorithm. We require that, when necessary, these bounds can be made arbitrarily tight, so that $z(\sigma)$ can be determined by observing the equality of $z_L(\sigma)$ and $z_U(\sigma)$, perhaps, in the limit. We begin with our lower-bounding procedure, which utilizes the solution for Λ at an arbitrary iteration of the PH algorithm. The following proposition is based on Lagrangian duality, and its proof follows Gade et al. (2016).

Proposition 2. *Consider a feasible nodal relaxation $\mathcal{R}_\sigma$ with an optimal solution $\hat{X}^*$, and assume it satisfies **A1**. Suppose that the PH algorithm is used to solve the problem. Let the multipliers Λ^k, obtained at the end of the k^{th} iteration of the algorithm, satisfy $\Lambda^k \in \mathcal{Y}$. Define the function $D_s(\lambda_s^k)$ as*

$$D_s(\lambda_s^k) = \min_{x_s} \left\{ f_s(x_s) + \lambda_s^k x_s \mid x_s \in C_s^\sigma \right\} \qquad \forall s = 1 \dots S. \tag{4}$$

Then, $D(\Lambda^k) = \sum_{s=1}^{S} p_s D_s(\lambda_s^k)$ is a lower bound on the optimal objective function value $z(\sigma)$ of the nodal relaxation.

Proof. Since each subproblem in (4) is minimized with respect to x_s, the following inequality is true:

$$\begin{aligned} D(\Lambda^k) &\le \sum_{s=1}^{S} p_s \left(f_s(\hat{x}_s^*) + \lambda_s^k \hat{x}_s^* \right) = \sum_{s=1}^{S} p_s f_s(\hat{x}_s^*) + \sum_{s=1}^{S} p_s \lambda_s^k \hat{x}_s^* \\ &= z(\sigma) + \mathbb{E}[\, \Lambda^k \cdot \hat{X}^* \,]. \end{aligned} \tag{5}$$

Furthermore, for any time period t, we have

$$\begin{aligned} \mathbb{E}[\, \Lambda_t^k \cdot \hat{X}_t^* \,] = \mathbb{E}\big[\, \mathbb{E}[\, \Lambda_t^k \cdot \hat{X}_t^* \mid F \,] \big] &= \mathbb{E}\big[\, \hat{X}_t^* \cdot \mathbb{E}[\, \Lambda_t^k \mid F \,] \big] && \textit{(since } \hat{X}^* \in \mathcal{N}\textit{)} \\ &= 0. && \textit{(since } \Lambda^k \in \mathcal{Y}\textit{)} \end{aligned}$$

Applying this argument for every t leads the expectation in (5) to vanish, which completes the proof. ■

It should be noted that the Lagrangian bound suggested in Gade et al. (2016) and adopted by Barnett et al. (2017) replaces the convex set C_s^σ with $C_s^\sigma \cap (\mathbb{R}^{n_r} \times \mathbb{Z}^{n_z})$ in (4). As mentioned earlier, our framework intends to avoid non-convexity in its subproblems, even though the bounds that they provide may be significantly better.

A6. For any feasible node σ with an optimal objective value $z(\sigma)$, there exists a finite value $U^\sigma \geq z(\sigma)$.

The above assumption is not essential for the convergence of our algorithm, however it ensures that the fathoming in the branch-and-bound tree is not delayed for infinitely-many steps on any node. Our upper-bounding procedure, which utilizes the solutions to $\hat{X}$ at any iteration of the PH algorithm, is provided below.

Proposition 3. *Consider a feasible nodal relaxation $\mathcal{R}_\sigma$. Assume that it satisfies **A1**, and admits an upper bound U^σ, as defined in **A6**. Suppose the PH algorithm is used to solve this problem. Let $\hat{X}^k$ denote the incumbent obtained at the end of the k^{th} iteration of the algorithm. Define $V_s(\hat{x}_s^k)$ as follows:*

$$V_s(\hat{x}_s^k) = \begin{cases} f_s(\hat{x}_s^k) & \text{for } \hat{x}_s^k \in C_s^\sigma \\ U^\sigma & \text{otherwise.} \end{cases}$$

Then, $V(\hat{X}^k) = \sum_{s=1}^S p_s V_s(x_s^k)$ is an upper bound on the optimal objective value $z(\sigma)$ of the nodal relaxation.

Proof. Since $V(\hat{X}^k)$ simply evaluates the objective value of the (nonanticipative) policy $\hat{X}^k$, the statement is obviously true. ∎

Remark 2. The a priori knowledge of U^σ for all σ may appear restrictive, however, within a branch-and-bound procedure, they can be replaced with a finite *global* upper bound on the optimal objective value of problem $\mathcal{P}$ (e.g., through a feasible-solution heuristic). This may lead to situations where $V(\hat{X}^k) < z(\sigma)$ for some node σ, however, such nodes can never contain a better integer-feasible solution than the one on hand, and should be fathomed in any case.

The following theorem formalizes the convergence condition on the suggested bounds that is sought in our analysis.

Theorem 3. *Consider a feasible nodal relaxation $\mathcal{R}_\sigma$. Assume that it satisfies **A1** and the constraint qualification condition $\mathcal{Q}$, and admits an upper bound U^σ, as defined in **A6**. Let $(\hat{X}^*, \Lambda^*)$ denote an optimal solution to this problem, and $z(\sigma)$ represent its optimal objective value. Suppose the PH algorithm is used to solve this problem. Consider the sequence of lower bounds $\{D(\Lambda^k)\}_{k=1}^\infty$ and upper bounds $\{V(\hat{X}^k)\}_{k=1}^\infty$ generated at the end of the k^{th} iteration of the PH algorithm, according to Propositions 2 and 3, respectively. Then, for $k \to \infty$, we have*

$$D(\Lambda^k) \leq z(\sigma) \leq V(\hat{X}^k), \quad \text{and} \quad D(\Lambda^k) \to z(\sigma), \ V(\hat{X}^k) \to z(\sigma). \tag{6}$$

Proof. Let $L(X, \Lambda)$ represent the ordinary Lagrangian for node σ, and define $\mathcal{I}_\sigma$ as 0 if $\hat{x}^s \in C_s^\sigma$, $\forall s = 1 \ldots S$, and U^σ if otherwise. Then, for any iteration k, the assertions of Propositions 2 and 3 can be stated as follows:

$$\begin{aligned} D(\Lambda^k) = \min{}_{x_s \in C_s^\sigma, \forall s} \ L(X, \Lambda^k) \leq L(\hat{X}^*, \Lambda^k) & \\ \leq L(\hat{X}^*, \Lambda^*) & \qquad \big(= z(\sigma)\big) \\ = L(\hat{X}^*, \Lambda^*) + \mathcal{I}_\sigma(\hat{X}^*) & \qquad \big(\textit{since } \hat{x}_s^* \in C_s^\sigma, \forall s\big) \\ \leq L(\hat{X}^k, \Lambda^k) + \mathcal{I}_\sigma(\hat{X}^k) & \qquad \big(\textit{since } \hat{X} \in \mathcal{N} \textit{ and } \mathcal{I}_\sigma(\hat{X}^k) \geq 0\big) \end{aligned}$$

$$= V(\hat{X}^k).$$

This proves the inequality in (6). By Theorem 1, we have $(\hat{X}^k, \Lambda^k) \to (\hat{X}^*, \Lambda^*)$ as $k \to \infty$, implying that $L(\hat{X}^k, \Lambda^k) \to L(\hat{X}^*, \Lambda^*)$. By Theorem 2, $(\hat{X}^*, \Lambda^*)$ is a saddle point of $L(\hat{X}, \Lambda)$, therefore $\hat{X}^* \in \mathcal{N} \cap \big(\times_{s=1}^{S} C_s^\sigma \big)$, and $L(\hat{X}^*, \Lambda^*) = \min_{x_s \in C_s^\sigma, \forall s} L(X, \Lambda^*) = D(\Lambda^*) = V(\hat{X}^*)$. As a result, as $k \to \infty$, we have $V(\hat{X}^k) - D(\Lambda^k) \to 0$, which completes the proof. ∎

Propositions 2 and 3 can be invoked at any arbitrary iteration of the PH algorithm, even before it approaches to an acceptable solution. The convergence of the PH algorithm can be monitored by periodically evaluating these bounds, and a solution of a node can be labeled as optimal when the gap between these bounds fall within numerical tolerances. When a node is only partially solved, fathoming of the branch-and-bound tree can be performed according to these lower bounds. If a (partial) optimization produces a primal solution that satisfy the integrality requirements, then the upper bounding procedure will determine the objective value of a candidate solution to the original problem (despite the fact that the node may still need to be solved to optimality). Alternative bounding procedures can also be investigated. For instance, in Proposition 2, f_s can be replaced with its minorants, and in Proposition 3, special problem structures can be exploited. However, these alternatives must satisfy the convergence condition (6), stated in Theorem 3.

The following specifications must be fulfilled in the design of our next algorithm.

A7. The branch-and-bound node selection is performed according to the least lower bound rule.

A8. When the PH algorithm (partially) solves a node σ for the i^{th} time, it is terminated, either,

(a) according to a finite iteration limit, or

(b) with a convergence tolerance $\varepsilon_{ph}^i > 0$ where $\varepsilon_{ph}^i \to 0$ as $i \to \infty$, or

(c) with an optimality tolerance $\delta_{ph}^i > 0$ where $\delta_{ph}^i \to 0$ as $i \to \infty$.

Furthermore, the next optimization of σ is initialized with the final solutions of $(X, \hat{X}, \Lambda)$, obtained at the end of the i^{th} run (along with all other possible algorithmic settings of this run).

A9. For each node σ, the computed lower bounds $(z_L(\sigma))$ and upper bounds $(z_U(\sigma))$ satisfy the convergence condition (6) given in Theorem 3.

The latter two assumptions ensure that the PH algorithm can continue solving each nodal relaxation *seamlessly*, despite being interrupted after a finite number of steps. As a result, at each node, the convergence of the PH algorithm to an optimal solution will follow Theorem 1, and the convergence of the bounds $z_L(\sigma)$ and $z_U(\sigma)$ will follow Theorem 3.

We present a PH-BAB algorithm that performs partial solves of its nodes in Algorithm 3. In the ensuing proposition, we show that implementing Algorithm 3 according to **A7-A9** leads to an exact method for optimizing multi-stage MISCPs.

Proposition 4. *Consider problem $\mathcal{P}$ which satisfies **A1-A4**, and let Algorithm 3 be implemented according to **A7-A9**. Assume that the problem has at least one optimal solution with the objective value $z^\star$. Define j as the iteration index of Algorithm 3, and let $\{z_L^j\}_{j=0}^\infty$ and $\{z_U^j\}_{j=0}^\infty$ denote the sequences of best lower and upper bounds produced according to lines 23 and 16, respectively. Also, let $\{X_U^j\}_{j=0}^\infty$ be the sequence of candidate solutions (X_U) produced by the algorithm in line 16. Then, as $j \to \infty$, we have*

$$z_L^j \nearrow z^\star \quad \text{and} \quad z_U^j \searrow z^\star. \tag{7}$$

Furthermore, every accumulation point of the sequence $\{X_U^j\}_{j=0}^\infty$ will be an optimal solution to $\mathcal{P}$.

Algorithm 3: A PH-BAB with Partial Node Solves

```
Initialization: Create the root node $o$, let $\mathcal{H} \leftarrow \{o\}$. Set $z_U = +\infty$, $z_L = -\infty$, $\hat{X}_U =$ NULL.
while $\mathcal{H} \neq \emptyset$ do
    Pick node $\sigma$ from $\mathcal{H}$ and let $\mathcal{H} \leftarrow \mathcal{H} \setminus \{\sigma\}$.
    Determine the feasibility of the sets $C_s^\sigma$, $\forall s = 1 \dots S$.
    if $C_s^\sigma$, $\forall s$, is feasible then                                  // else, $\sigma$ is fathomed
        Invoke PH to (partially) solve $\sigma$.
        Get solution $\hat{X}^\sigma$ of $\sigma$, and the lower bound $z_L(\sigma)$.
        if $z_L(\sigma) < z_U$ then                                                 // else, $\sigma$ is fathomed
            if $\hat{X}^\sigma$ is fractional then
                Pick a fractional component $\hat{x}_{stj}^\sigma$ of $\hat{X}^\sigma$.
                Create nodes $\sigma^+$ and $\sigma^-$. Let $Q^{\sigma^+} = Q^{\sigma^-} = Q^\sigma$, while introducing the additional
                  restrictions $l_{stj} = \lceil \hat{x}_{stj}^\sigma \rceil$, $\forall s \in F \subseteq \mathcal{F}_t$ to $Q^{\sigma^+}$, and $u_{stj} = \lfloor \hat{x}_{stj}^\sigma \rfloor$, $\forall s \in F \subseteq \mathcal{F}_t$ to $Q^{\sigma^-}$.
                Let $\mathcal{H} \leftarrow \mathcal{H} \cup \{\sigma^-, \sigma^+\}$.
            else
                Obtain an upper bound $z_U(\sigma)$ and the corresponding solution $\hat{X}_U^\sigma$.
                if $z_U(\sigma) < z_U$ then
                    Let $\hat{X}_U = \hat{X}_U^\sigma$, $z_U = z_U(\sigma)$.
                    Fathom: $\mathcal{H} \leftarrow \mathcal{H} \setminus \{\sigma' \,|\, z_L(\sigma') \geq z_U\}$
                end
                Let $\mathcal{H} \leftarrow \mathcal{H} \cup \{\sigma\}$.                           // $\sigma$ is replaced
            end
        end
    end
    Let $z_L = \min\{z_L(\sigma) \,|\, \sigma \in \mathcal{H}\}$.
end
```

Proof. Due to **A4**, all infeasible nodes that are encountered by the algorithm will be discarded in finite time, and due to **A8**, all feasible nodes that are encountered will be (partially) optimized in finite time. As a result, Algorithm 3 expends only finite amount of time on each node that it decides to process. In fact, by design, the algorithm performs branching after each (partial) optimization of a feasible node, when the node produces a fractional solution. Due to **A2**, $\exists$ a finite iteration after which the branching process must stop producing nodes. From this iteration forward, all nodes that stay in $\mathcal{H}$ must satisfy the integrality restrictions. Without loss of generality, the rest of the proof will only consider the nodes that are processed subsequently. For the rest of the proof, we define $\mathcal{J}_\sigma$ to be the set of iteration indices, at which, a particular node σ is selected and processed by the algorithm.

We begin with showing that $\{z_L^j\}_{j=0}^\infty$ converges to $z^\star$. By way of contradiction, suppose this is not true. This implies that $\exists$ a node $\sigma \in \mathcal{H}$ which determines $\{z_L^j\}_{j=0}^\infty$ infinitely-many times, and has the subsequence of lower bounds $\{z_L^j(\sigma)\}_{j\in\mathcal{J}_\sigma}$ not converging to $z^\star$. Due to **A9**, this subsequence must have the limit $z(\sigma)$. Suppose $z(\sigma) = z^\star + c$. Clearly, $c < 0$ contradicts the optimality of $z^\star$, leaving the only possibility $c \geq 0$. For $c > 0$, we must have $z_L^{j'}(\sigma) \geq z^\star + \varepsilon$ for finite j' and $\varepsilon \in (0, c)$, since the subsequence $\{z_L^j(\sigma)\}_{j\in\mathcal{J}_\sigma}$ can only have finitely-many elements in any positive neighborhood of $z^\star + c$. However, at least one other node $\sigma^\star \in \mathcal{H}$ must contain an optimal solution, and for this node, we can write $z_L^{j'}(\sigma^\star) \leq z^\star < z^\star + \varepsilon \leq z_L^{j'}(\sigma)$. As a result, **A7** ensures that node σ cannot determine z_L^j when $j \geq j'$, unless $c = 0$. In fact, if $c > 0$, an upper-bounding sequence $\{z_U^j\}_{j=0}^\infty \to z^\star$ will ensure that such σ will be removed from $\mathcal{H}$ after a finite number of iterations. We conclude that, only the nodes with the optimal objective value $z^\star$ can determine the best lower bound for infinitely-many times, and for such σ, **A9** ensures that $\lim_{j\to\infty, j\in\mathcal{J}_\sigma} z_L^j(\sigma) = z^\star$. Since $z_L^j = z_L^j(\sigma)$, $\forall j \in \mathcal{J}_\sigma$ and $\sigma \in \mathcal{H}$, we have every subsequence of $\{z_L^j\}_{j=0}^\infty$ converging to $z^\star$, therefore $\lim_{j\to\infty} z_L^j = z^\star$.

Next, we show the convergence of $\{z_U^j\}_{j=0}^{\infty}$ to $z^\star$. Since all $\sigma \in \mathcal{H}$ satisfy the integrality restrictions, line 14 of Algorithm 3 will be invoked at every iteration. Moreover, due to **A7** and the above discussion, all nodes that the algorithm processes infinitely-many times, will have the optimal objective value $z^\star$. For any such σ, **A9** ensures that $\lim_{j\to\infty,\, j\in\mathcal{J}_\sigma} z_U^j(\sigma) = \lim_{j\to\infty,\, j\in\mathcal{J}_\sigma} z_L^j(\sigma) = z^\star$. Since $z_U^j = \min_\sigma \{\, z_U^j(\sigma) \mid \sigma \in \mathcal{H}\,\}$, we can only have $\lim_{j\to\infty} z_U^j = z^\star$ without invalidating the optimality of $z^\star$. As a result of this (and the previous discussion), the assertion in (7) must be true.

Finally, our claim on the sequence $\{X_U^j\}_{j=0}^{\infty}$ can be trivially confirmed by noting that the best upper bound z_U^j, produced at iteration j, correspond to the objective value of the feasible solution $\hat{X}_U^j$, which, by virtue of Theorem 1, converges to some optimal solution of $\mathcal{P}$. ■

Proposition 4 suggests that Algorithm 3 can resolve the complexity caused by the discrete nature of the feasible set of $\mathcal{P}$, in finite time, and thereafter, convergence to an optimum must happen. Observe, also, that assumption **A6** plays an implicit role in the course of the algorithm, by guaranteeing that all suboptimal nodes will be discarded after finitely-many steps.

Remark 3. Algorithm 3 *always* performs branching on nodes whose (partial) optimizations produce fractional solutions. Depending on the application, it might be beneficial to delay branching until the node at hand is sufficiently explored. In such designs, the assertion in Propositions 4 will remain valid, provided that branching on any node is never postponed for infinitely-long.

4. PH-BAB for a Special Class of Two-Stage Stochastic Programs

Similar to all branch-and-bound-based methods, the PH-BAB framework relies on enumeration, and straightforward implementations are unlikely to provide promising results for large-scale problems. Computational performance can be enhanced through the use of cutting planes, improved branching and node-selection rules, and many other adjunct procedures, all of which have proved themselves in the field of deterministic mixed-integer programming. Moreover, it is well known that special problem structures, which commonly appear in many stochastic programming formulations, can lead to powerful algorithms. The goal of this section is to make use of these facts and develop a computationally effective PH-BAB implementation. The resulting implementation will resemble the branch-and-fix coordination of Escudero et al. (2009) (see, also, Pages-Bernausa et al., 2015). We will focus on a special problem structure, which allows us to directly utilize the capabilities of state-of-the-art mixed-integer programming solvers for efficiently solving MICP subproblems. The structure we consider is a two-stage stochastic program with only binary variables in its first-stage. Remarks will be provided for future extensions.

For clarity of exposition, we express some of our notation within the two-stage context. We denote the number of continuous and integer variables at t^{th} stage of the stochastic program as n_r^t and n_z^t, respectively, and let $n^t = n_r^t + n_z^t$. The nonanticipativity restrictions are not meaningful for the second-stage variables and will be omitted throughout this section. Accordingly, a nonanticipative policy can be constructed from the expectation of the first-stage decisions x_{s1}, $\forall s$, alone. We refer to the outcome of this expectation as $\bar{x}$. With respect to our previous notation, we have $\hat{X} : s \to \hat{x}_s = (\bar{x},\, x_{s2})$, $\forall s$, suggesting that $\bar{x}, \hat{x}_{11} \ldots \hat{x}_{S1}$ are all synonyms of each other. The set of nonanticipative solutions can then be represented as follows:

$$\mathcal{N} = \Big\{\, (x_{11} \ldots x_{S1}) \in \times_{s=1}^{S} \mathbb{R}^{n^1} \mid x_{s1} = \bar{x},\, \forall s = 1 \ldots S \,\Big\}.$$

This state-vector representation of nonanticipativity allows separable Lagrangians as in Higle and Sen (2006) (see, also Lubin et al., 2013).

At each scenario, we partition the feasible region of the problem as $C_s = (C_{\cdot 1} \times \mathbb{R}^{n^2}) \cap C_{s2}$, where $C_{\cdot 1} \subseteq \mathbb{R}^{n^1}$ and $C_{s2} \subseteq \mathbb{R}^{n^1+n^2}$ are closed and convex sets. The dropped scenario indices indicate that the corresponding set is not affected by the realization of the randomness. We define the objective function of the problem as

$$f_s(x_s) = f_{\cdot 1}(x_{s1}) + \min_{x_{s2}} \left\{ f_{s2}(x_{s2}) \mid (x_{s1}, x_{s2}) \in C_{s2},\, (x_{s1}, x_{s2}) \in \mathbb{R}^{n^1} \times \left(\mathbb{R}^{n_r^2} \times \mathbb{Z}^{n_z^2} \right) \right\}, \tag{8}$$

where $f_{\cdot 1} : \mathbb{R}^{n^1} \to \mathbb{R}$ and $f_{s2} : \mathbb{R}^{n^2} \to \mathbb{R}$ are both convex functions. The inner minimization in (8) is assumed to have the objective value $+\infty$ when its input x_{s1} does not lead to a feasible x_{s2}. This minimization has no effect on the way the PH iterations are performed, however gives an alternative interpretation on the objective of the problem. Minimizing f_s can be perceived as searching for the best *first-stage* solution, since the inner minimization in (8) (a.k.a., the second-stage subproblem) will be shaped according to its value. Moreover, the altered form of f_s affects the way the upper bound evaluations in Proposition 3 are performed. For instance, for a (nonanticipative) first-stage decision $\bar{x}^\sigma$ produced from some node σ, the evaluations take the form

$$f_{\cdot 1}(\bar{x}^\sigma) + \sum_{s=1}^{S} p_s \min_{x_{s2}} \left\{ f_{s2} \mid (\bar{x}^\sigma, x_{s2}) \in C_{s2}^\sigma \right\},$$

where $C_{s2}^\sigma = C_{s2} \cap Q_s^\sigma$. Accordingly, the procedure essentially assesses the quality of the first-stage decisions. While computing an upper bound now requires the solution of S second-stage convex subproblems, the likelihood of finding a feasible upper bound is much higher. Following this discussion, we let $\mathbb{B} = \{0,\, 1\}$, and pose a two-stage MISCP with pure binary first-stage variables as follows:

$$\begin{aligned} &\min_X \ \left\{ \mathbb{E}[\, f(X, \xi)\,] \mid x_{s1} = \bar{x},\, \forall s = 1 \ldots S \right\}, \\ &\qquad \text{where,} \quad f(x_s, \xi_s) = \min_{x_s} \left\{ f_s(x_s) \mid x_{s1} \in C_{\cdot 1},\, x_{s1} \in \mathbb{B}^{n^1} \right\}. \end{aligned} \tag{$\mathcal{P}$-2}$$

As suggested by the previous discussion, a key characteristic of stochastic programming problems is that for a given nonanticipative solution for the first-stage, the second-stage subproblems can be evaluated independently. Indeed, stochastic programming algorithms that are in the stagewise-decomposition context mainly rely on this fact. The main purpose of this section will be to investigate under which circumstances the PH-BAB algorithm can also leverage this property, so that promising integer-feasible solutions can be easily produced using the state-of-the-art mixed-integer programming solvers.

Despite performing scenario-wise decomposition, the PH-BAB algorithm can eventually end up with nodes, where the nonanticipativity is guaranteed to hold. Such situations occur when the branch-and-bound procedure restricts the lower and upper bounds of all first-stage variables to unique points. In these cases, the optimization of a nodal relaxation σ reduces to solving

$$\min_{x_{s2}} \left\{ f_{\cdot 1}(x_{s1}) + f_{s2}(x_{s2}) \mid (x_{s1}, x_{s2}) \in C_{s2}^\sigma \right\}, \quad \forall s = 1 \ldots S,$$

where, for a *fixed* first-stage solution $\bar{x}^\sigma$, Q^σ involves the nonanticipativity restriction $x_{s1} = \bar{x}^\sigma$, $\forall s$. The above subproblems can be independently optimized using any convex programming solver. When f_{s2} is linear and C_{s2} is polyhedral, these subproblems turn into linear programs, which can be effectively handled in finite-time. Moreover, from this point forward, the role of the PH algorithm will essentially be over for node σ and its children. As a result, deterministic mixed-integer programming solvers may now be employed to fulfill the integrality restrictions in the second-stage subproblems.

Although the benefit of a fixed first-stage solution is apparent, whether such situations can be efficiently induced must be clarified. For problems involving continuous first-stage variables, our algorithm, in its current form, deems this impossible, and must be extended to allow branching on continuous variables. For pure integer first-stage variables, we require the branch-and-bound tree to be exhausted for all promising branches, therefore the performance of the resulting algorithm will be unsatisfactory. However, when the first-stage only involves pure binaries (as we have assumed in this section), an efficient branching scheme can be developed. Our scheme does not disrupt the overall flow of the branch-and-bound procedure, but is invoked only when a node produces an integral first-stage solution.

Consider a node σ, whose first-stage variables are not completely fixed by the branch-and-bound algorithm. For the sake of argument, suppose that the PH algorithm terminates with an integral first-stage solution ($\bar{x}^\sigma$) but the second-stage is still fractional. For these situations, our recommendation is to follow a modified branching strategy. We create two nodes σ^f and $\sigma^\neg$ in such a way that $\bar{x}^\sigma$ is not feasible in $\sigma^\neg$, but it is the only feasible first-stage solution in σ^f. In other words, these two nodes are created according to the unconventional disjunction $x_{s1} = \bar{x}^\sigma$ or $x_{s1} \neq \bar{x}^\sigma$. The new bounds for σ^f can be expressed effortlessly as

$$Q_s^{\sigma^f} = Q_s^\sigma \cap \left\{ x_s \in \mathbb{R}^{n^1+n^2} \mid \bar{x}^\sigma \leq x_{s1} \leq \bar{x}^\sigma \right\}, \quad \forall s = 1 \ldots S. \tag{9}$$

On the other hand, describing $Q_s^{\sigma^\neg}$ is not straightforward since the imposition $\bar{x} \neq \bar{x}^\sigma$ is non-convex. At this point, the pure-binary-variables assumption at the first stage allows the use of the inequality

$$\sum_{j:\bar{x}_j^\sigma=1} x_{s1j} - \sum_{j:\bar{x}_j^\sigma=0} x_{s1j} \leq |\{j : \bar{x}_j^\sigma = 1\}| - 1, \tag{10}$$

which cuts off $\bar{x}^\sigma$ from the corresponding feasible set. The above inequality has been commonly used for similar purposes in the stochastic-programming literature (see, for instance, Sen and Sherali, 2006; Ahmed, 2013). With a little stretch on the meaning of $Q_\omega^{\sigma^\neg}$, we can define the variable bounds in $\sigma^\neg$ as follows:

$$Q_s^{\sigma^\neg} = Q_s^\sigma \cap \left\{ x_s \in \mathbb{R}^{n^1+n^2} \mid (10) \text{ holds} \right\}, \quad \forall s = 1 \ldots S. \tag{11}$$

This concludes our branching procedure.

As a consequence of the above suggestions, our algorithm can now produce first-stage solutions, with respect to which, the second-stage value functions can be evaluated. An exact evaluation would require imposing the integrality restrictions on the corresponding second-stage variables, and solving the resulting subproblems using deterministic MICP solvers. In essence, such an operation forms a branch-and-bound sub-tree that emanates from the node involving fixed first-stage solutions. Determining an optimal solution to this sub-tree (if such a solution exists) will produce a candidate optimal solution to the original problem. However, it should be noted that the exact evaluations of the second-stage value functions may be computationally expensive, despite the power of commercial solvers. Therefore, these evaluations should only be pursued when the corresponding first-stage solutions are deemed promising. Indeed, instead of frequently invoking MICP solvers, algorithms that successively provide tighter approximation of the second-stage value functions (such as Sen and Higle, 2005; Sen and Sherali, 2006; Gade et al., 2014; Zhang and Küçükyavuz, 2014) could be preferable.

Their potential within the PH-BAB scheme should be investigated in the future.

Along these lines, we consider all nodes σ, which involves fixed first-stage variables $\bar{x}^{\sigma}$, as MICPs of the following form:

$$f_{\cdot 1}(\bar{x}^{\sigma}) + \min_{x_{s2}} \left\{ f_{s2}(x_{s2}) \mid (\bar{x}^{\sigma}, x_{s2}) \in C^{\sigma}_{s2},\ (\bar{x}^{\sigma}, x_{s2}) \in \mathbb{R}^{n^1} \times \left(\mathbb{R}^{n^2_r} \times \mathbb{Z}^{n^2_z}\right) \right\}, \quad \forall s = 1 \ldots S. \qquad (12)$$

Using a presumably fast MICP solver, we optimize the above subproblems in two phases. First, we perform a partial optimization to obtain lower bounds on the optimal objective value of the subproblems. These lower bounds immediately lead to a lower bound on the objective value of the best integer-feasible solution that σ can contain (if it contains any). We refer to these bounds as $z_L(\sigma)$, although this notation should not be confused with the lower bounds generated for other arbitrary nodes according to Proposition 2. Using $z_L(\sigma)$, it is possible to assess whether σ needs to be fathomed, by comparing it with the best upper bound z_U of the PH-BAB algorithm. If the node may indeed contain a better integer-feasible solution, we continue with the second phase of our node algorithm where the subproblems (12) are completely optimized. An outline of our procedure is given in Algorithm 4.

Algorithm 4: A Node Algorithm

```
1  if node $\sigma$ does not have fixed first-stage variables then
2      Determine the feasibility of $\sigma$.
3      if $\sigma$ is feasible then Invoke PH to (partially) solve $\sigma$.        // else, $\sigma$ will be fathomed
4  else
5      Enforce the integrality restrictions on the second-stage variables to form (12).
6      Determine the feasibility of (12).
7      if $\sigma$ is feasible then                                                  // else, $\sigma$ will be fathomed
8          Partially solve (12), and obtain a lower bound $z_L(\sigma)$.
9          if $z_L(\sigma) > z_U$ then                                              // else, $\sigma$ will be fathomed
10             Completely solve (12), and obtain the optimal objective value $z(\sigma)$.
11         end
12     end
13 end
```

The strategies presented in this section can be incorporated to both Algorithm 2 and 3, in order to increase their computational efficiencies. In Algorithm 5, we present the pseudocode for the extension of the latter. In comparison to Algorithm 3, this version of the PH-BAB method can converge to an optimal solution in finite time, provided that the following assumption holds.

A10. For any set of problems of the form (12), there exists an MICP solver which can either determine the infeasibility of any one of the subproblems, or yields an optimal solution to all, both in finite number of steps.

The assumption clearly holds when the feasible sets C_s, $\forall s$, are polyhedral, and the functions f_{s2}, $\forall s$, are linear or quadratic. As a consequence of **A10**, lines 5-12 of Algorithm 4 can be executed in finite time.

Proposition 5. *Consider problem $\mathcal{P}$-2 which satisfies **A1**-**A4**, **A10**, and let Algorithm 5 be implemented according to **A7**-**A9**. Then, Algorithm 5 yields an optimal solution in finite number of steps.*

Proof. The proof is mainly based on the convergence of Algorithm 3, as stated in Proposition 4. Indeed, Algorithms 3 and 5 differ mainly with respect to the introduced branching strategy and the

Algorithm 5: A PH-BAB Algorithm for 2-Stage MISCPs with Pure Binary First-Stage Variables

```
1  Initialization: Create the root node $o$, let $\mathcal{H} \leftarrow \{o\}$. Set $z_U = +\infty$, $z_L = -\infty$, $\hat{X}_U =$ NULL.
2  while $\mathcal{H} \neq \emptyset$ do
3      Pick node $\sigma$ from $\mathcal{H}$ and let $\mathcal{H} \leftarrow \mathcal{H} \setminus \{\sigma\}$.
4      Invoke Algorithm 4 to process $\sigma$.
5      if $\sigma$ is feasible then                                      // else, $\sigma$ is fathomed
6          Get solution $\hat{X}^\sigma$ of $\sigma$, and a lower bound $z_L(\sigma)$.
7          if $z_L(\sigma) < z_U$ then                                   // else, $\sigma$ is fathomed
8              if $\hat{X}^\sigma$ has a fractional first-stage solution then
9                  Pick a fractional first-stage component $\bar{x}^\sigma_j$ of $\hat{X}^\sigma$.
10                 Create nodes $\sigma^+$ and $\sigma^-$. Let $Q^{\sigma^+} = Q^{\sigma^-} = Q^\sigma$, while introducing the additional
                     restrictions $l_{stj} = \lceil \hat{x}^\sigma_{stj} \rceil$, $\forall s \in F \subseteq \mathcal{F}_t$ to $Q^{\sigma^+}$, and $u_{stj} = \lfloor \hat{x}^\sigma_{stj} \rfloor$, $\forall s \in F \subseteq \mathcal{F}_t$ to $Q^{\sigma^-}$.
11                 Let $\mathcal{H} \leftarrow \mathcal{H} \cup \{\sigma^-, \sigma^+\}$.
12             else if $\hat{X}^\sigma$ has a fractional second-stage solution then
13                 Create nodes $\sigma^f$ according to (9), and $\sigma^\neg$ according to (11).
14                 Let $\mathcal{H} \leftarrow \mathcal{H} \cup \{\sigma^f, \sigma^\neg\}$.
15             else
16                 Obtain an upper bound $z_U(\sigma)$ and the corresponding solution $\hat{X}^\sigma_U$.
17                 if $z_U(\sigma) < z_U$ then
18                     Let $\hat{X}_U = \hat{X}^\sigma_U$, $z_U = z_U(\sigma)$.
19                     Fathom: $\mathcal{H} \leftarrow \mathcal{H} \setminus \{\sigma' \,|\, z_L(\sigma') \geq z_U\}$
20                 end
21                 Let $\mathcal{H} \leftarrow \mathcal{H} \cup \{\sigma\}$.                         // $\sigma$ is replaced in $\mathcal{H}$
22             end
23         end
24     end
25     Let $z_L = \min\{z_L(\sigma) \,|\, \sigma \in \mathcal{H}\}$.
26 end
```

node algorithm. Algorithm 5 iterates until all of the first-stage variables are branched on, either through ordinary branching (lines 9-11) or the new one (lines 13-14). In either case, the number of nodes can increase only finitely-many times, since the branching causes the nodes to have (at most) one less integer first-stage solution than their parent node. As a result, a fixed first-stage solution must be obtained in finite time, at which point, the node algorithm invokes the MICP solver instead of the PH algorithm. Due to **A10**, in finite time, the MICP solver either generates the next best solution for problem $\mathcal{P}$*-2*, or proves that no better solution exists in the optimized node (perhaps, by showing that it is infeasible). Since both of these are assumed to happen in finite numbers of steps, the finite convergence of Algorithm 5 follows. ∎

Remark 4. The results presented in this section can be extended to a special class of multi-stage MISCPs, where, at each stage, all nonanticipative decisions (that connect the stages) are pure binaries. The finiteness of a corresponding algorithm can be shown with a similar argument to Proposition 5.

5. Computational Experiments

In this section, we study the performance of the PH-BAB framework with comparisons to two other solution methods for MISPs, which are commonly used in the literature. We will consider two PH-BAB implementations, where the first one is an exact PH-BAB method, whereas the latter is an approximation. Our study will be conducted on instances which involve two-stages and satisfy the pure binary first-stage variables assumption of §4. Our choice of instances was motivated by the fact that, for this class of problems, benchmark instances are available and have been extensively studied in the literature. We begin with presenting our experimental design, which includes a discussion on the test instances, analyzed algorithms, and considered performance metrics. The rest of the section is devoted to the presentation of the outcomes of our computational experiments.

5.1. Experimental Design

Our main experiments are conducted on two sets of stochastic server location problems (SSLPs). The first set, due to Ntaimo and Sen (2005), has been used as a common benchmark in the stochastic programming literature. These instances are labeled according to `SSLP.n.m.S` where `n` and `m` represents certain problem parameters, whereas `S` stands for the number of scenarios. Each problem has `n` binary variables in its first stage; `nm` binary and `n` continuous variables in the second stage. These instances can be obtained from Ahmed et al. (2015). As our second dataset, we introduce the quadratic extensions of these instances. In particular, we consider a quadratic objective function in the second stage of the problem. We label these instances as `SSLP.Q.n.m.S`. Further details on these instances can be found in Appendix A. We note that both of these datasets have pure binary variables in the first stage, which allows the use of the developments given in §4.

A second tier of experiments are also conducted on instances that do not satisfy the assumptions of §4, in order to demonstrate the potential of our framework on different families of MISCPs. To this end, we consider the five network-design problem instances that appeared in Barnett et al. (2017), which are originally based on Ruszczyński (2002). Each instance contains 30 binary and 30 continuous variables in its first stage; 30 binary and 3,000 continuous variables in the second-stage. The number of scenarios in these instances are limited to 50, but the scenario subproblems are much more challenging due to their sizes, and the many disjunctions – modeled using Big-M's – that the formulations contain.

We assess the performance of four different algorithms. The first two of these are variants of PH-BAB implementations which are specialized to two-stage problems with pure binary first-stage variables (see §4). The third algorithm is the PH-Heuristic which is described in §2. Finally, we solve problem $\mathcal{P}$*-2* without any decomposition, through a deterministic mixed-integer programming solver. All algorithms, and the details within, are summarized in Table 1.

In all our experiments, the (relative) optimality gap is calculated according to $(\hat{z}_{UB} - \hat{z}_{LB})/|\hat{z}_{UB}|$, where $\hat{z}_{LB}$ is the final lower bound and $\hat{z}_{UB}$ is the final upper bound produced by the *corresponding* algorithm. We consider a relative optimality tolerance of 10^{-4} and 10^{-2} for the linear and the quadratic test instances, respectively. All other tolerances are consistent with that of major commercial mixed-integer programming solvers. In order not to underestimate the performance of the PH-Heuristic, we compute the relative optimality gap of the produced upper bounds ($\hat{z}_{UB}$) according to $(\hat{z}_{UB} - z^\star)/|z^\star|$, where $z^\star$ denotes the optimal objective value of the corresponding instance. When $z^\star$ is not available (for instance, for some quadratic instances), the objective value of the best upper bound ($z^\star_{UB}$), obtained from any of the exact algorithms, is used instead. Finally, noting that PH-BAB-Apx is not an exact algorithm, we measure the error in its produced solution according to

$$\text{Error} = \max\left(\ \max(\hat{z}_{LB} - z^\star_{UB}, 0),\ \ \max(z^\star_{LB} - \hat{z}_{UB}, 0)\ \right).$$

Above, $z^\star_{LB}$ denotes the best lower bound obtained from any of the exact algorithms. In simple terms, this measure indicates the maximum absolute deviation of the produced lower and upper bounds, from their (best available) theoretical upper and lower limits, respectively.

Algorithm	Description
PH-BAB	Algorithm 5. A single PH iteration is allowed per node. The default choice for ρ is 100.
PH-BAB-Apx	An approximate variant of Algorithm 5. The PH algorithm optimizes nodal relaxations until a specified convergence tolerance is reached. No bound computations are performed on the nodal relaxations, and the final objective value from the PH algorithm is assumed to be the optimal objective value of the processed node. The considered convergence criterion for the PH algorithm is $\sum_{s=1}^{S} p_s \|x_s - \bar{x}_s\|_2 \leq 10^{-1}$. The default choice for ρ is 100.
PH-Heuristic	The PH algorithm which is executed with modified subproblems that are modeled as MICPs (see §2). The considered convergence criterion is $\sum_{s=1}^{S} p_s \|x_s - \bar{x}_s\|_2 \leq 10^{-1}$. Cycle detection is performed through monitoring the objective values from the last 10 iterations, and terminating the algorithm when all of them have already been recorded in earlier iterations. Unless specified otherwise, ρ is selected adaptively, according to the SEP heuristic given in Watson and Woodruff (2010). Upon termination, an integral first-stage solution is ensured by rounding the produced solution $\bar{x}$ to the nearest integer. This solution is used to compute an upper bound according to Proposition 3, *with the integrality restrictions imposed*. A lower bound is computed according to Proposition 2, again, with the integrality restrictions imposed (see Gade et al. (2016)). The computational time associated with these bound computations are *not reflected* into our reports.
`CPLEX`	Optimization of problem $\mathcal{P}$-*2*, without decomposition, but with all of the default deterministic mixed-integer programming technologies of the solver `CPLEX`.

Table 1: Descriptions of the tested algorithms.

Our runs were performed on a single thread of a Dell Desktop PC with Intel® Core™ i7-3770S CPU @ 3.10 GHz, 7.68 GB of RAM, and running Ubuntu Linux 12.04.3 LTS. For all algorithms, a time limit of 3600 seconds is imposed. The mathematical programs were solved through the `Concert Technology` class library of `IBM ILOG CPLEX 12.6.2` (`CPLEX`). In the PH-BAB, PH-BAB-Apx, and PH-Heuristic algorithms, the linear and quadratic problems were solved using the dual simplex algorithm due to its superior performance. The remaining parameters of `CPLEX` were preserved at their default values.

5.2. Performance on Linear MISP Instances

In Table 2, we present the solution times and the optimality gaps reported by all tested algorithms. Our main takeaways are presented in bullets, with ensuing paragraphs that provide further details.

	Optimal	PH-BAB		PH-BAB-Apx			PH-Heuristic			`CPLEX`	
Instance	Objective	Time	Gap	Time	Gap	Error	Time	Gap	UB-Gap	Time	Gap
`SSLP.5.25.50`	-121.600	0.6	-	0.5	-	-	5.1	6.7	-	1.3	-
`SSLP.5.25.100`	-127.370	1.2	-	1.1	-	-	11.8	6.6	-	3.9	-
`SSLP.5.50.100`	-323.700	1.7	-	1.5	-	-	3.9	1.7	-	6.9	-
`SSLP.5.50.500`	-320.800	8.1	-	7.0	-	-	14.7	1.5	-	181.3	-
`SSLP.5.50.1000`	-320.148	16.0	-	13.8	-	-	29.5	1.4	-	523.7	-
`SSLP.5.50.2000`	-320.602	32.0	-	27.8	-	-	59.0	1.4	-	3600.1*	42.5
`SSLP.10.50.50`	-364.640	9.4	-	8.7	-	-	40.8	3.9	-	198.8	-
`SSLP.10.50.100`	-354.190	19.5	-	18.6	-	-	89.0	4.9	-	1685.6	-
`SSLP.10.50.500`	-349.136	86.1	-	80.8	-	-	331.6	4.7	-	3601.7*	0.2
`SSLP.10.50.1000`	-351.711	172.4	-	163.6	-	-	614.8	4.4	-	3600.2*	64.3
`SSLP.10.50.2000`	-347.262	333.1	-	309.6	-	-	1192.4	4.5	-	3600.2*	112.6
`SSLP.15.45.5`	-262.400	2.0	-	1.4	-	-	5.2	3.4	0.5	3.4	-
`SSLP.15.45.10`	-260.500	10.9	-	2.4	-	-	33.7	5.8	-	1.0	-
`SSLP.15.45.15`	-253.600	5.4	-	3.2	-	-	3601.5*	8.7	0.2	11.9	-

Table 2: Performance of algorithms for a class of stochastic linear programs (relative optimality tolerance = 0.01%).

*: Time limit of 3600 seconds is exceeded.

- *Among all the contenders, the PH-BAB and the PH-BAB-Apx algorithms perform significantly faster for almost all of the linear instances.*

We observe that the combined solution time of these algorithms on the whole dataset is less than 25 minutes, whereas the contender algorithms may spend more time than this on particular instances. The recorded speed-ups with respect to the PH-Heuristic average to a factor of four, even after excluding the outlier instance `SSLP.15.45.15`. Aligned with our expectations, the performance of `CPLEX` is the poorest when the problems involve more than a handful of scenarios. In particular, we report four cases where the algorithm cannot certify optimality within the time allotted. The one exception is `SSLP.10.45.10` where `CPLEX` performs the fastest among all of the contenders. However, this instance consists of very few scenarios, and for such instances, we should not have any claim for superiority for any of the decomposition-based algorithms.

These outcomes should not come at a surprise when the steps of each algorithm are carefully scrutinized. For instance, the iterations of both the PH-BAB and PH-BAB-Apx algorithms mainly comprise solving series of quadratic optimization subproblems. In contrast, the PH-Heuristic deals with nonconvex mixed-integer quadratic subproblems. As a consequence, despite the benefit of decomposition, its performance is weakened by its expensive iterations. In the case of `CPLEX`, the poor performance can only be explained by noting that it does not take advantage of the stochastic-programming structure of the tested problems. It is interesting to note that, even though `CPLEX` operates on a *linear* program, it can still be beaten by methods that perform quadratic optimizations.

- *For all linear instances, the PH-BAB and the PH-BAB-Apx algorithms have produced optimal solutions. The PH-Heuristic algorithm was also successful in identifying optimal solutions with minor exceptions.*

Across the PH-based methods, solution quality does not appear to be a significant determinant on the performances of the algorithms. Indeed, the self-reported optimality gaps of the PH-Heuristic underestimates its performance. In the UB-GAP column of Table 2, we see that the gaps between the objective values of the optimal and the produced solutions are all zero, with only two minor exceptions. Finally, with `CPLEX`, instances with large numbers of scenarios may result into gaps exceeding 100%.

- *With respect to the PH-BAB algorithm, the PH-BAB-Apx spends on average 30% less computational effort on the linear instances.*

In the PH-BAB algorithm, the bound computations on the nodes – which provide the exactness guarantee to the algorithm – leads to more computational effort than the gain achieved from the partial processing of the nodes. We observe that the PH-BAB-Apx algorithm is faster than the PH-BAB, without any exception. This indicates for the tested instances that small numbers of iterations are sufficient to attain the specified convergence tolerance in the PH-BAB-Apx algorithm. Tighter tolerances will surely invalidate the conclusion drawn in here.

- *For applications where the optimality of the produced solution is not the utmost concern, the PH-BAB-Apx can be used as an effective heuristic algorithm.*

While the risk of sub-optimality may not be eliminated, our experiments reveal that the PH-BAB-Apx algorithm produces optimal solutions fast and without any error. Moreover, in comparison to the PH-Heuristic, the PH-BAB-Apx is not prone to oscillating iterates, or, as is the case for `SSLP.15.45.15`, unexpectedly-slow convergence rates. The algorithm has minimal tuning requirements (mainly, ρ and the convergence tolerance for PH optimizations). At least for the class of instances we have experimented with, PH-BAB-Apx algorithm appears to be an effective solution approach.

We now focus on the behavior of the PH-BAB algorithm on an individual instance. Figure 3, illustrates the progress of the algorithm for our largest instance `SSLP.10.50.2000`. The gray-shaded areas visualize the absolute optimality gaps computed after (partial) optimizations of nodal relaxations with the PH algorithm. We observe that in general the PH-BAB algorithm spends a uniform amount of effort on the individual nodes with one important exception. After the first 300 seconds, no optimality gap is reported for almost 30 seconds. The underlying reason for this is the mixed-integer programming optimizations of the scenario subproblems (see §4). This procedure – however expensive it may be – results into the first (and the optimal) integer-feasible solution for the considered problem. Immediately after this, the PH-BAB algorithm attempts to perform mixed-integer programming optimizations for another node. In this case, however, the first phase of the proposed node algorithm detects in short amount of time that a lower bound for this node exceeds the objective value of the best integer-feasible solution (see Algorithm 4 in §4). Figure 3 also reveals that the optimality gaps of the nodes tend to be smaller as the PH-BAB algorithm proceeds. More importantly, even if the gaps are not small (for instance, as in the last processed node), the finite iteration limit on the PH algorithm and the computed lower bounds prevent the algorithm to spend unnecessary effort on nodes with no promise.

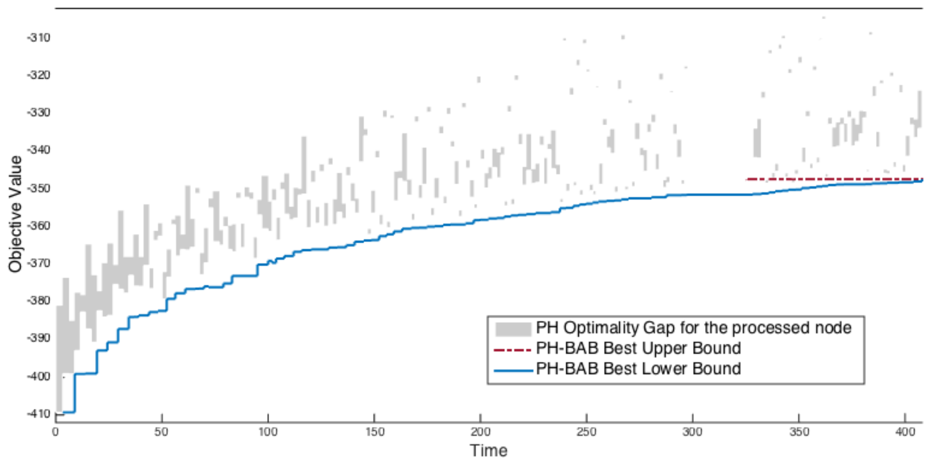


Figure 3: Progress of the PH-BAB algorithm and the optimality gaps of the processed nodal relaxations (`SSLP.10.50.2000`; $\rho = 100$; single PH iteration is allowed per node)

5.3. Sensitivity of PH-based Algorithms to ρ

In the literature, it has been shown that the convergence behavior of the PH algorithm differs according to the magnitude of ρ. For a class of convex problems, Mulvey and Vladimirou (1991) observe that high values of ρ lead to X moving towards nonanticipativity fast, but slowing down later as $X \to \hat{X}^*$, whereas smaller values of ρ typically imply a more gradual convergence. For the mixed-integer case, Watson and Woodruff (2010) report large numbers of iterations for small values of ρ, supporting the latter claim of Mulvey and Vladimirou (1991).

In this section, we aim to understand the effects of ρ on all PH-based methods under consideration. We conduct tests on three different settings: $\rho = 10$, 100, and 1000. All runs were performed on linear instances. For each algorithm, performance measures of interest are reported in Table 3.

- *With respect to inferior choices of ρ, the PH-BAB is the most robust algorithm among other assessed PH-based methods.*

By design, the partial exploration of the nodes and the embedded bounding mechanisms allow the

	PH-BAB				PH-BAB-Apx					PH-Heuristic			
Instance	Nodes	PH-Itr	Time	Gap	Nodes	PH-Itr	Time	Gap	Error	Itr	Time	Gap	UB-Gap
$\rho = 10$													
SSLP.5.25.50	31	26	0.5	-	35	130	1.4	-	-	14	7.8	2.5	-
SSLP.5.25.100	27	24	0.8	-	35	96	2.2	-	-	12	15.0	2.3	-
SSLP.5.50.100	35	30	1.4	-	35	76	2.6	-	-	4	5.0	0.5	-
SSLP.5.50.500	35	30	6.9	-	35	82	13.5	-	-	3	19.4	0.5	-
SSLP.5.50.1000	35	30	13.1	-	35	81	26.5	-	-	3	38.4	0.4	-
SSLP.5.50.2000	35	30	27.2	-	35	82	53.5	-	-	3	75.5	0.4	-
SSLP.10.50.50	275	264	11.6	-	229	1011	31.3	-	-	13	91.3	2.3	-
SSLP.10.50.100	287	274	25.1	-	235	1122	70.2	-	-	19	284.6	2.3	-
SSLP.10.50.500	273	262	110.1	-	211	1035	310.3	-	-	11	751.9	2.4	-
SSLP.10.50.1000	275	266	211.2	-	217	1053	610.4	-	-	8	1119.7	2.3	-
SSLP.10.50.2000	255	250	370.2	-	199	1003	1170.2	-	-	8	2147.5	2.2	-
SSLP.15.45.5	263	252	2.7	-	201	1115	6.5	-	-	7	4.0	1.1	-
SSLP.15.45.10	467	455	14.1	-	171	1021	13.1	-	-	76	256.8	3.4	-
SSLP.15.45.15	495	486	17.1	-	113	592	13.7	-	-	111	785.6	3.3	-
$\rho = 100$													
SSLP.5.25.50	33	28	0.6	-	31	36	0.5	-	-	24	4.3	12.9	2.2
SSLP.5.25.100	34	28	1.2	-	35	35	1.1	-	-	14	6.7	10.1	1.4
SSLP.5.50.100	37	30	1.7	-	35	30	1.5	-	-	5	3.2	2.7	-
SSLP.5.50.500	37	30	8.1	-	35	31	7.0	-	-	4	13.2	2.1	-
SSLP.5.50.1000	37	30	16.0	-	35	31	13.8	-	-	4	27.1	2.0	-
SSLP.5.50.2000	37	30	32.0	-	35	31	27.8	-	-	4	54.4	2.0	-
SSLP.10.50.50	221	218	9.4	-	199	266	8.7	-	-	12	30.4	3.9	-
SSLP.10.50.100	225	220	19.5	-	213	286	18.6	-	-	11	51.5	4.9	-
SSLP.10.50.500	209	208	86.1	-	195	269	80.8	-	-	7	188.5	4.7	-
SSLP.10.50.1000	215	214	172.4	-	201	268	163.6	-	-	8	412.2	4.4	-
SSLP.10.50.2000	201	200	333.1	-	187	261	309.6	-	-	10	918.9	4.5	-
SSLP.15.45.5	181	176	2.0	-	153	186	1.4	-	-	3	1.2	4.4	1.2
SSLP.15.45.10	205	203	10.9	-	141	213	2.4	-	-	7	5.3	5.8	-
SSLP.15.45.15	137	135	5.4	-	97	143	3.2	-	-	7	11.6	8.6	-
$\rho = 1000$													
SSLP.5.25.50	35	28	0.7	-	29	23	0.5	-	-	4	0.8	12.9	2.2
SSLP.5.25.100	37	28	1.5	-	30	26	0.8	-	1.8	3	1.3	23.0	11.7
SSLP.5.50.100	37	30	1.9	-	33	29	1.3	-	-	4	2.1	15.0	10.6
SSLP.5.50.500	35	30	8.1	-	33	29	6.1	-	-	3	8.8	6.7	4.3
SSLP.5.50.1000	37	30	17.8	-	33	29	12.0	-	-	3	17.9	6.8	4.5
SSLP.5.50.2000	37	30	35.8	-	33	29	23.9	-	-	3	36.0	6.9	4.6
SSLP.10.50.50	259	252	11.7	-	143	141	4.7	-	-	4	10.1	9.8	5.4
SSLP.10.50.100	265	258	22.6	-	153	149	10.0	-	-	3	14.7	18.9	11.8
SSLP.10.50.500	239	234	102.6	-	125	125	36.8	-	-	5	86.8	11.2	5.8
SSLP.10.50.1000	243	238	202.3	-	149	149	83.7	-	-	4	176.9	10.7	5.7
SSLP.10.50.2000	229	228	370.8	-	125	125	145.6	-	-	4	329.5	11.3	6.1
SSLP.15.45.5	181	179	2.9	-	43	45	0.4	-	1.2	3	0.9	4.4	1.2
SSLP.15.45.10	275	273	4.5	-	105	104	2.2	-	1.2	3	3.7	7.5	1.6
SSLP.15.45.15	185	181	10.3	-	61	60	1.9	-	-	7	9.0	9.1	0.5

Table 3: Performance of the algorithms under three different ρ settings.

PH-BAB algorithm not to waste effort on nodes with little promise. For both small and large values of ρ, the PH-BAB algorithm improves its precision, only when necessary, through increasing the number of nodes explored. In comparison, the effect of ρ can be significant for the PH-BAB-Apx and PH-Heuristic algorithms. For the PH-BAB-Apx algorithm, we observe that larger values of ρ can significantly speed up the convergence, but have the potential to reduce the accuracy. Table 3 reveals that the setting $\rho = 1000$ leads to three cases, where the reported solution of the algorithm was off by nearly 2 units from the optimal objective values of these instances. This outcome can be explained with the findings of Mulvey and Vladimirou (1991). When ρ is large and the convergence tolerance is not sufficiently tight, the PH-BAB-Apx can confuse the rapid convergence of the iterates towards the non-anticipativity, with their convergence to optimality. Although the accuracy can be improved with a tighter convergence tolerance, this would necessitate more computational effort, and the tradeoff between speed and accuracy could not be eliminated. Similar conclusions can be made for the PH-Heuristic as well. We observe that the solution times and the number of iterations are significantly higher when ρ is small, but the solutions produced by the method are always optimal. When ρ is large, the solution times become very competitive with respect to the PH-BAB algorithm, however the optimality gaps of the produced solutions are far from satisfactory.

5.4. Performance on Quadratic MISP Instances

In our experiments on quadratic instances, we abandon the adaptive ρ selection scheme SEP of the PH-Heuristic algorithm, as it was neither prescribed for a quadratic setting nor provided a better performance. We consider a weaker relative optimality tolerance, namely 1%, as tighter tolerances render a portion of the mixed-integer programs unsolvable with reasonable computational effort. The optimal objective values of the test instances are obtained using the PH-BAB algorithm, without imposing a time limit, and with further instance-specific tunings. For three instances, however, an optimal solution cannot be certified as the memory requirements exceeded the limit in our computing environment. We summarize the performances of all algorithms in Table 4.

	Optimal	**PH-BAB**		**PH-BAB-Apx**			**PH-Heuristic**			CPLEX	
Instance	**Objective**	Time	Gap	Time	Gap	Error	Time	Gap	UB-Gap	Time	Gap
SSLP.Q.5.25.50	-117.160	0.7	0.7	0.6	0.7	-	3.4	6.6	0.0	3.2	0.9
SSLP.Q.5.25.100	-123.395	1.9	0.7	1.2	0.7	-	6.2	5.3	0.0	8.6	0.9
SSLP.Q.5.50.100	-319.025	3.6	0.8	2.7	0.8	-	13.3	3.1	0.1	178.3	1.0
SSLP.Q.5.50.500	-315.139	19.8	0.9	13.3	0.9	-	60.7	3.2	0.1	3600.1*	4.2
SSLP.Q.5.50.1000	-314.936	39.9	0.8	31.2	0.8	-	122.1	2.9	0.1	1074.3	error
SSLP.Q.5.50.2000	-315.171	79.5	0.7	62.5	0.7	-	248.7	3.0	0.1	2909.3	error
SSLP.Q.10.50.50	-319.010	41.4	1.0	19.5	1.0	-	253.0	8.2	1.4	3600.0*	1.4
SSLP.Q.10.50.100	-310.840	40.0	1.0	58.4	1.0	-	389.6	7.3	0.2	3600.1*	4.0
SSLP.Q.10.50.500	-307.177†	322.1	1.0	280.8	0.6	-	1702.3	6.8	0.0	1196.1	error
SSLP.Q.10.50.1000	-309.126†	639.6	0.9	286.4	0.9	-	3684.3*	6.7	0.0	3600.4*	123.8
SSLP.Q.10.50.2000	-305.045†	1275.3	1.0	1100.3	1.0	-	4164.0*	25.3	14.8	2173.5	error
SSLP.Q.15.45.5	-186.700	3.2	0.9	2.4	0.8	-	40.7	0.7	0.0	173.4	1.0
SSLP.Q.15.45.10	-182.650	291.4	0.9	289.9	0.8	-	517.4	6.9	0.1	3600.0*	1.5
SSLP.Q.15.45.15	-177.367	1162.7	0.7	838.8	0.7	-	732.7	8.4	0.2	3600.0*	3.4

Table 4: Solution times and percentage optimality gaps for quadratic test instances (relative optimality tolerance = 1%)

*: Time limit of 3600 seconds is exceeded.
†: A certificate of optimality is not available.

Even though the specified optimality tolerance is weaker, in general, we observe that the optimization of the quadratic instances require much more computational effort than their linear counterparts. The algorithms behave similarly as they did with the linear instances, however, their differences are more pronounced in the quadratic setting.

- *Among all the contenders, the PH-BAB and the PH-BAB-Apx algorithms perform significantly faster for almost all of the quadratic instances. Both the PH-Heuristic and* CPLEX *may occasionally hit the alloted time limit and may produce low-quality solutions.*

A particular example is the SSLP.Q.10.50.2000 instance, where the PH-Heuristic hits the time limit, and produces a solution that has 14.8% worse objective value than the best-known upper bound. For this instance, the PH-BAB and the PH-BAB-Apx algorithms can provide a solution (within the desired tolerances) in half an hour. `CPLEX`, on the other hand, spends more time than this and terminates with no feasible solution. Similar errors have been observed for three other instances, which emphasizes the importance of decomposition on stochastic programming problems.

5.5. Performance of PH-BAB Beyond Specially-Structured Problems

Our final analysis demonstrates the performance of our framework on a family of instances that does not satisfy the assumptions of §4. To this end, we consider the instances that were experimented with in Barnett et al. (2017). The presence of continuous variables in the first stage of these problems makes them particularly challenging, as the convergence of the PH algorithm can no longer be aided by a branch-and-bound process. Accordingly, exact evaluations of the second-stage value functions may no longer be an option, since continuous variables cannot be fixed unlike their binary counterparts. As a result, the PH-BAB algorithm may need to process significant (but finite) numbers of nodes until discreteness is resolved, after which, the PH algorithm can run its course to produce feasible solutions.

In view of the above comment (regarding the large numbers of nodes explored during a branch-and-bound process), it is not an uncommon phenomenon for branch-and-bound schemes to employ heuristics to discover feasible solutions. Along these lines, we periodically invoke a feasible-solution heuristic which resembles Algorithm 4. In particular, after a partial node optimization, we round and fix all first-stage integer variables, and solve the second-stage subproblems as mixed-integer programs, subject to the variable bounds of the node. Then, the solutions to *all* integer variables are fixed, and the PH algorithm is iterated on these subproblems to ensure the nonanticipativity of the (first-stage) continuous variables.

The scenario subproblems in the experimented instances are more difficult than that of the SSLP instances. To surmount the challenge, we upgrade our computing environment so that meaningfully-large numbers of subproblems can be processed, in parallel. To this end, experiments in this subsection are conducted using 32 threads of a Dell Desktop PC with Intel® Xeon® E5-2630 v3 CPU @ 2.40 GHz, 32 GB of RAM, and running Ubuntu Linux 12.04.3 LTS. In other words, this computer should be considered as an ordinary desktop machine.

As noted in §5.1, the experimented instances in this section contain many Big-M constraints, which can commonly induce poor linear-programming relaxations. In order to give emphasis to the quality of our lower bounds, we set $\rho = 1$ so that the PH algorithm can produce better lower-bounding sequences. In contrast, in the PH iterations of our feasible-solution heuristic, we set $\rho = 100$ so that the convergence of primal variables occurs faster. We invoke our heuristic after every 25 (feasible) node optimizations, and label a solution as nonanticipative according to the convergence criterion, $\sum_{s \in S} p_s \|x_s - \bar{x}\|_2 \leq 10^{-1}$.

For brevity, the focus of this subsection will only be on the performance of the PH-BAB algorithm. Our main conclusion is stated below.

- *The PH-BAB algorithm achieves rapid convergence to high-quality feasible solutions, and thereafter shows a gradual progress towards closing the optimality gap.*

In Figure 4, we observe that the most significant improvements in optimality gaps occur within the first 10 minutes of the PH-BAB algorithm. While the algorithm continues to close the gaps afterwards, the paces of improvements slow down, perhaps because most first-stage variables have already been branched on, and branching on second-stage variables makes less difference.

In Table 5, we present snapshots of the best upper and lower bounds that the PH-BAB algorithm obtained within three different allocations of solution time. The table shows that the feasible solutions

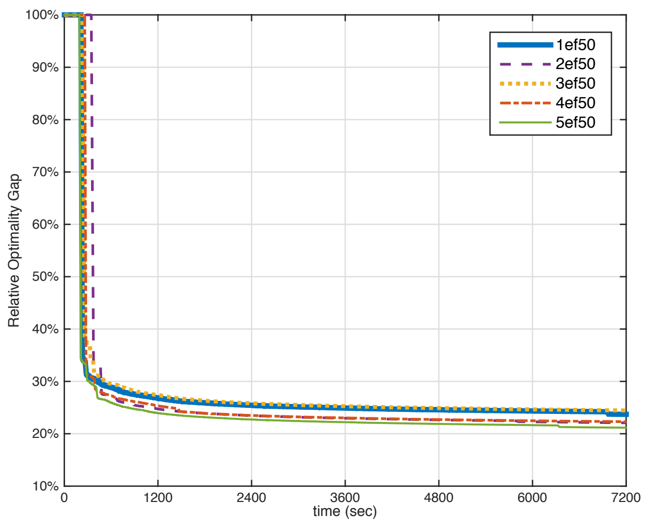


Figure 4: Progress of the relative optimality gaps reported by the PH-BAB algorithm.

provided by the PH-BAB algorithm are high-in-quality, and improves only marginally with increasing computational budget. In contrast, the best lower bounds exhibit very gradual progress towards closing the optimality gaps. Such an outcome is not particularly surprising, as finding high-quality feasible solutions, quickly, and expending significant effort on improving them, is a typical behavior of many (commercial) branch-and-bound algorithms (see, for instance, Barnett et al. (2017), for a discussion on the behavior of `CPLEX` for these problems). For benchmarking purposes, we also consider the performance of the framework suggested in Barnett et al. (2017). The metrics, presented in Table 5, are taken directly from their paper, where the computations therein were performed on a significantly superior computing environment[†]. We observe that the upper bounds obtained by the PH-BAB framework (within 600 seconds) are superior than those obtained by this contender. Similar outcomes can also be drawn by comparing the performance of the PH-Heuristic, presented in Barnett et al. (2017), to the results obtained here.

	CPLEX[‡]	Barnett et al. (2017)		PH-BAB – Best UBs			PH-BAB – Best LBs		
Instance	**Best UB**	**Best UB**	**(Time)**	**600s**	**1800s**	**7200s**	**600s**	**1800s**	**7200s**
`1ef50`	158,653	162,163	(18,161s)	161,682	161,682	160,730	114,990	119,816	122,616
`2ef50`	151,060	156,211	(15,330s)	152,100	152,100	151,827	111,410	115,733	118,287
`3ef50`	161,466	165,733	(33,390s)	164,426	164,426	164,426	115,932	121,104	124,214
`4ef50`	153,854	157,229	(18,150s)	155,944	154,794	154,794	113,576	117,763	120,260
`5ef50`	150,401	152,686	(8,556s)	151,581	151,581	151,006	112,231	116,487	119,055

Table 5: Performance of PH-type algorithms that considers a branch-and-bound framework.

6. Conclusion

In this paper, we have developed a new decomposition framework for computing optimal solutions to multi-stage mixed-integer stochastic convex programs (MISCP). Our framework has been specialized and implemented for a challenging class of two-stage MISCPs. Computational experiments reveal

[†]The authors use a 96-core Intel Xeon workstation with 2.1 GHz processors and 1 TB of RAM, where 50 instances of the PH algorithm are executed in parallel, and each instance is allowed to use up to 2 cores.

[‡]The best objective values for these instances are taken from Barnett et al. (2017), where the authors use `CPLEX` to solve $\mathcal{P}$, without decomposition, for 10,000 seconds within their computing environment.

the effectiveness of the proposed framework on a common mixed-integer stochastic linear programming benchmark, and its newly-introduced quadratic counterpart.

Acknowledgements

This paper is written in honor of Professor Maarten van der Vlerk. His invention took a very elegant approach to a very difficult problem: stochastic programs with simple integer recourse (SIR). Maarten's research inspired the authors to study a problem that he referred to as Mixed-Integer Stochastic Programming (MISP), emphasizing the SP side of the problem. The approach of our paper (PH-BAB), while very different from algorithms for SIR problems, reflects Maarten's MISP point of view: the mixed-integer aspects of our algorithm work within an essentially SP algorithm, the PH method. In this sense, the work in this paper adopts Maarten's view (MISP), although most of the second author's other work may be looked upon as Stochastic Mixed-Integer Programming (SMIP). For some, this distinction may not mean much, but we believe that the difference meant something to Maarten. We wish to honor his vision with this paper.

Appendix A SSLP Formulation

We provide the formulation of the stochastic server location problems (SSLPs) of Ntaimo and Sen (2005), which we use in §5. The problem considers a set of locations $\mathcal{J}$, zones $\mathcal{Z}$, and clients $\mathcal{I}$. The goal is to obtain (i) a cost-efficient allocation of servers into locations specified in $\mathcal{J}$, and (ii) a cost-efficient assignment of clients to these servers under random client availability. We list the decision variables of the problem below.

z_j:	1 if server is located at site j, 0 otherwise
y_{ij}^s:	1 if client i is served by server at location j under scenario s, 0 otherwise
y_{j0}^s:	amount of demand overflow due to server capacity limitations.

The variable y_{j0}^s must not take large values, therefore is penalized using the penalty function ϕ_j^s. Each problem instance takes the following input parameters:

c_j:	cost of locating a server at location $j \in \mathcal{J}$
q_{ij}:	revenue from client $i \in \mathcal{I}$ when served by server at location j
d_{ij}:	demand of client i from server at location j
u:	server capacity
v:	upper bound on the total number of located servers
w_z:	minimum number of servers to be located in zone $z \in \mathcal{Z}$
$\mathcal{J}_z$:	subset of server locations that belong to zone z
γ_i:	random availability of client i
γ_i^s:	realization of γ_i under scenario s (1 if client i is present in scenario s, 0 otherwise).

The first- and the second-stage subproblems are respectively given below.

$$\begin{aligned} \min \quad & \sum_{j \in \mathcal{J}} c_j z_j + \mathbb{E}[h(z,\gamma)] \\ \text{s.t.} \quad & \sum_{j \in \mathcal{J}} z_j \leq v \qquad \text{(13a)} \\ & \sum_{j \in \mathcal{J}_z} z_j \geq w_z, \ \forall z \in \mathcal{Z} \qquad \text{(13b)} \end{aligned}$$

$$\begin{aligned} h(z,\gamma^s) = \min \quad & -\left(\sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} q_{ij}^s y_{ij}^s - \sum_{j \in \mathcal{J}} \phi_j^s \left(y_{j0}^s\right) \right) \\ \text{s.t.} \quad & \sum_{i \in \mathcal{I}} d_{ij} y_{ij}^s - y_{j0}^s \leq u z_j, \ \forall i \in \mathcal{I}, j \in \mathcal{J} \qquad \text{(13c)} \\ & \sum_{j \in \mathcal{J}} y_{ij}^s = \gamma_i^s, \ \forall i \in \mathcal{I} \qquad \text{(13d)} \end{aligned}$$

$$z_j \in \mathbb{B},\ \forall z \in \mathcal{Z}, \qquad y^s_{ij} \in \mathbb{B},\ \forall i \in \mathcal{I},\ j \in \mathcal{J}$$
$$y^s_{j0} \geq 0,\ \forall j \in \mathcal{J}.$$

Constraints (13a) limit the maximum number of server allocations; (13b) ensures each zone receives the minimum number of required servers; (13c) models the server capacity restrictions; (13d) ensures that each client is served by a single server.

In the original SSLP instances of Ntaimo and Sen (2005), the penalty function is assumed to have a linear form $\phi^s_j(y^s_{j0}) = q^s_{j0}y^s_{j0}$, where q^s_{j0} is a large coefficient. The quadratic extension of the problem upgrades this penalty function as $\phi^s_j(y^s_{j0}) = q^s_{j0}\left(y^s_{j0}\right)^2$. In order to increase the significance of the new quadratic penalty function, the server capacity parameter u is adjusted to be smaller in the new instances.

References

Ahmed, S. (2013). A scenario decomposition algorithm for 0-1 stochastic programs. *Operations Research Letters*, 41(6):565–569.

Ahmed, S., Garcia, R., Kong, N., Ntaimo, L., Parija, G., Qiu, F., and Sen, S. (2015). SIPLIB: A stochastic integer programming test problem library. `http://www.isye.gatech.edu/~sahmed/siplib`.

Angulo, G., Ahmed, S., and Dey, S. S. (2014). Improving the integer L-shaped method. Technical report, School of Industrial & Systems Engineering, Georgia Institute of Technology.

Barnett, J., Watson, J.-P., and Woodruff, D. L. (2017). Bbph: Using progressive hedging within branch and bound to solve multi-stage stochastic mixed integer programs. *Operations Research Letters*, 45(1):34 – 39.

Birge, J. R. (1985). Decomposition and partitioning methods for multistage stochastic linear programs. *Operations Research*, 33(5):989–1007.

Bixby, R. E. (2002). Solving real-world linear programs: A decade and more of progress. *Operations Research*, 50(1):3–15.

Boland, N., Christiansen, J., Dandurand, B., Eberhard, A., Linderoth, J., Luedtke, J., and Oliveira, F. (2016). Combining progressive hedging with a Frank-Wolfe method to compute Lagrangian dual bounds in stochastic mixed-integer programming. Technical report, Optimization Online.

Boland, N. L. and Eberhard, A. C. (2014). On the augmented Lagrangian dual for integer programming. *Mathematical Programming*, 150(2):491–509.

Boyd, S. (2010). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122.

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

Carøe, C. C. and Tind, J. (1998). L-shaped decomposition of two-stage stochastic programs with integer recourse. *Mathematical Programming*, 83:451–464.

Escudero, L. F., Garín, A., Merino, M., and Pérez, G. (2009). BFC-MSMIP: An exact branch-and-fix coordination approach for solving multistage stochastic mixed 0-1 problems. *Top*, 17:96–122.

Feizollahi, M. J., Ahmed, S., and Sun, A. (2015). Exact augmented Lagrangian duality for mixed integer linear programming. Technical report, School of Industrial & Systems Engineering, Georgia Institute of Technology.

Gade, D., Hackebeil, G., Ryan, S. M., Roger, J.-P. W., and David, J. W. (2016). Obtaining lower bounds from the progressive hedging algorithm for stochastic mixed-integer programs. *Mathematical Programming*, 157(1):47–67.

Gade, D., Küçükyavuz, S., and Sen, S. (2014). Decomposition algorithms with parametric Gomory cuts for two-stage stochastic integer programs. *Mathematical Programming*, 144(1-2):39–64.

Guignard, M. and Kim, S. (1987). Lagrangean decomposition: A model yielding stronger Lagrangean bounds. *Mathematical Programming*, 39(2):215–228.

Guo, G., Hackebeil, G., Ryan, S. M., Watson, J.-P., and Woodruff, D. L. (2015). Integration of progressive hedging and dual decomposition in stochastic integer programs. *Operations Research Letters*, 43(3):311 – 316.

Helmberg, C. (2000). *Semidefinite programming for combinatorial optimization*. Konrad-Zuse-Zentrum für Informationstechnik Berlin.

Hemmecke, R. and Schultz, R. (2003). Decomposition of test sets in stochastic integer programming. *Mathematical Programming*, 94(2–3):323–341.

Higle, J. L. and Sen, S. (2006). Multistage stochastic convex programs: Duality and its implications. *Annals of Operations Research*, 142(1):129–146.

Horst, R. and Tuy, H. (2013). *Global optimization: Deterministic approaches*. Springer Science & Business Media.

Jörnsten, K., Näsberg, M., and Smeds, P. (1985). Variable splitting: A new Lagrangean relaxation approach to some mathematical programming models. Technical report, University of Linköping, Department of Mathematics.

Laporte, G. and Louveaux, F. V. (1993). The integer L-shaped method for stochastic integer programs with complete recourse. *Operations Research Letters*, 13(3):133–142.

Løkketangen, A. and Woodruff, D. L. (1996). Progressive hedging and tabu search applied to mixed integer (0, 1) multistage stochastic programming. *Journal of Heuristics*, 2(2):111–128.

Louveaux, F. V. (1980). A solution method for multistage stochastic programs with recourse with application to an energy investment problem. *Operations Research*, 28(4):889–902.

Lubin, M., Martin, K., Petra, C. G., and Sandıkçı, B. (2013). On parallelizing dual decomposition in stochastic integer programming. *Operations Research Letters*, 41(3):252 – 258.

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

Mulvey, J. M. and Vladimirou, H. (1991). Applying the progressive hedging algorithm to stochastic generalized networks. *Annals of Operations Research*, 31(1):399–424.

Mulvey, J. R. and Ruszczyński, A. (1995). A new scenario decomposition method for large-scale stochastic optimization. *Operations Research*, 43(3):477–490.

Ntaimo, L. and Sen, S. (2005). The million-variable "march" for stochastic combinatorial optimization. *Journal of Global Optimization*, 32(3):385–400.

Pages-Bernausa, A., Pérez-Valdés, G., and Tomasgard, A. (2015). A parallelised distributed implementation of a branch and fix coordination algorithm. *European journal of operational research*, 244:77–85.

Qi, Y. and Sen, S. (2016). The ancestral Benders' cutting plane algorithm with multi-term disjunctions for mixed-integer recourse decisions in stochastic programming. *Mathematical Programming*.

Ralphs, T. and Hassanzadeh, A. (2014). A generalization of Benders' algorithm for two-stage stochastic optimization problems with mixed integer recourse. Technical report, Laboratory for Computational Optimization at Lehigh (COR@L), Department of Industrial and Systems Engineering, Lehigh University.

Rockafellar, R. T. (1976). Monotone operators and the proximal point algorithm. *SIAM Journal on Control and Optimization*, 14(5):877–898.

Rockafellar, R. T. and Wets, R. J.-B. (1991). Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research*, 16(1):119–147.

Ruszczyński (2002). Probabilistic programming with discrete distributions and precedence constrained knapsack polyhedra. *Mathematical Programming*, 93:195–215.

Ryan, K., Ahmed, S., Dey, S. S., and Rajan, D. (2016). Optimization driven scenario grouping. Technical report, School of Industrial & Systems Engineering, Georgia Institute of Technology.

Ryan, S. M., Wets, R. J. B., Woodruff, D. L., Silva-Monroy, C., and Watson, J. P. (2013). Toward scalable, parallel progressive hedging for stochastic unit commitment. In *Power and Energy Society General Meeting (PES), 2013 IEEE*, pages 1–5.

Sen, S. and Higle, J. L. (2005). The C3 theorem and a D2 algorithm for large scale stochastic mixed-integer programming: Set convexification. *Mathematical Programming*, 104(1):1–20.

Sen, S. and Sherali, H. D. (2006). Decomposition with branch-and-cut approaches for two-stage stochastic mixed-integer programming. *Mathematical Programming*, 106(2):203–223.

Trapp, A. C., Prokopyev, O. A., and Schaefer, A. J. (2013). On a level-set characterization of the value function of an integer program and its application to stochastic programming. *Operations Research*, 61(2):498–511.

Van Slyke, R. M. and Wets, R. (1969). L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics*, 17(4):638–663.

Watson, J.-P. and Woodruff, D. L. (2010). Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems. *Computational Management Science*, 8(4):355–370.

Zhang, M. and Küçükyavuz, S. (2014). Finitely convergent decomposition algorithms for two-stage stochastic pure integer programs. *SIAM Journal on Optimization*, 24(4):1933–1951.