Scalable Heuristics for Stochastic Programming with Scenario Selection

Jean-Paul Watson
Discrete Math and Complex Systems Department, Sandia National Laboratories
Albuquerque, NM 87185-1318
jwatson@sandia.gov

Roger J-B Wets Department of Mathematics, University of California, Davis
Davis, CA 95616-8633
rjbwets@ucdavis.edu

David L. Woodruff
Graduate School of Management, University of California, Davis
Davis, CA 95616-8609
DLWoodruff@UCDavis.edu

July 3, 2008

Abstract

We describe computational procedures to solve a wide-ranging class of stochastic programs with chance constraints where the random components of the problem are discretely distributed. Our procedures are based on a combination of Lagrangian relaxation and scenario decomposition, which we solve using a novel variant of Rockafellar and Wets' progressive hedging algorithm. Experiments demonstrate the ability of the proposed algorithm to quickly find near-optimal solutions – where verifiable – to both difficult and very large chance constrained stochastic programs using scenario decomposition. The algorithm requires orders of magnitude less time on most test instances than existing exact algorithms, and exhibits stronger scalability in terms of final solution quality on large-scale instances.

1 Introduction

Reliability considerations are always an intrinsic component of decision making under uncertainty and the design of good optimization models must necessarily include them. One way to do this is via a "penalty function" that measures the discrepancies between the decisions' output and the potential future events, i.e., scenarios. This leads to stochastic programs with recourse where the recourse costs play the role of the penalty function. Another possibility is to model non-compliance via a risk measure and optimize to minimize this risk measure; value-at-risk is a popular choice (Holton 2003). One could also ask explicitly for compliance by including in the model a probabilistic constraint. This leads to the formulation of a stochastic program with chance constraints, i.e., certain constraints will have to be satisfied with a high probability. Unfortunately, such problems are inherently difficult to solve except when they fall in a very narrow family. Stochastic programs are usually solved by relying on an approximating problem obtained via discretization of the probability space. Stochastic programs with recourse preserve convexity under discretization but, in general, that is not the case for stochastic programs with chance constraints; note however that Nemirovski and Shapiro, by relying on "Bernstein approximations" are able to build convex approximations for a relatively significant class of chance constrained programs (Nemirovski and Shapiro 2006).

In this article, we are concerned with computational procedures to solve a wide ranging class of stochastic programs with chance constraints where the random components of the problem are discretely distributed, i.e., with finite support. We shall not be concerned if the original problem was already of this type, if it resulted from a discretization of the underlying probability distributions, or is obtained via a sampling scheme such as in the approach suggested by Luedtke and Ahmed (2007); cf. also Salinetti (1983), who deals with convergence issues. We can thus formulate our decision model as a two stage stochastic program where some portion of the scenarios can be ignored. There are many real-world situations where this is an appropriate model; a commonly discussed instance is the so-called 100-year disaster planning problem where one percent (or less) of the scenarios do not have to be included. Another example involves supply chain design, where it is typically not cost-effective to satisfy performance constraints in all realizable scenarios. In general, the use of chance constraints gives the modeler a chance to express the idea that constraints need not be satisfied or costs minimized across every conceivable eventuality.

For notational convenience, we will write our formulations and algorithms for the situation where the scenarios are equally likely and the cost function does not depend on the scenario. Both restrictions can be relaxed immediately. A formal statement of such a problem is as follows. Given a scenario index set $\mathcal{S}$ of size $|\mathcal{S}|$, n decision variables, a cost function $c : \Re^n \to \Re$, and a scalar $0 < \alpha \leq 1$, find a vector $x \in \Re^n$ and binary vector d of length $|\mathcal{S}|$ to

$$\begin{array}{lc} \text{minimize} & c(x) \\ \text{subject to} & x \in \mathcal{Q}(s), \;\; \forall s \in \{\mathcal{S} : d_s = 1\} \\ & \sum_{s \in \mathcal{S}} d_s \geq (1-\alpha)|\mathcal{S}| \\ & d_s \in \{0,1\}, \;\; \forall s \in \mathcal{S} \end{array} \qquad \text{(E)}$$

where $\mathcal{Q}(s)$ summarizes the problem constraints and d_s represents the binary decision to select scenario $s \in \mathcal{S}$. Examples of $\mathcal{Q}(s)$ are provided in §5, where computational experiments concerning problems with both linear and mixed-integer constraints are described. Regardless of the nature of the constraints on x, the interesting aspect of this formulation is the ability to select which scenarios are to be considered and which can be ignored. We assume that once we have found a good d vector, obtaining an appropriate, corresponding x vector is relatively easy (even given instances in which the remainder of Problem (E) is non-convex, e.g., due to integer x components), at least in comparison to finding d.

Apart from the presence of the d vector, this is a standard stochastic programming formulation (Birge and Louveaux 1997) where the use of the decision vector x that does not depend on the scenario implicitly implements the *non-anticipativity* constraints that avoid allowing the decisions to depend on the scenario. Note that Problem (E) is non-convex due to the integrality of the d vector. This formulation is similar to one given by Ruszczyński (2002), who provided cutting planes and an exact algorithm useful for the case where the constraints are linear and x is continuous. However, our interest here is in algorithms for more general (including non-convex) and computationally difficult forms, and for problem instances that are too large to be solved directly.

The remainder of this paper is organized as follows. In §2, we discuss specific relaxations and decompositions of Problem (E). In §3, we introduce simple candidate algorithms based on scenario decomposition to solve instances of Problem (E). A more sophisticated algorithm is then introduced in §4. Computational experiments comparing the various algorithmic alternatives are presented in §5. We conclude by summarizing our contributions in §6.

2 Relaxation and Decomposition

Relaxation and decomposition are standard computational techniques for addressing situations involving both very large-scale and non-convex (or both) problem instances. First, we consider a Lagrangian relaxation of Problem (E), as follows:

$$\begin{array}{rc} \text{minimize} & c(x) - \lambda\left(\sum_{s\in\mathcal{S}} d_s - (1-\alpha)|\mathcal{S}|\right) \\ \text{subject to} & x \in \mathcal{Q}(s),\ \ \forall s \in \{\mathcal{S} : d_s = 1\} \\ & d_s \in \{0,1\},\ \ \forall s \in \mathcal{S} \end{array} \qquad \text{(L)}$$

For any fixed λ, the objective value of Problem (L) is a lower bound on (E) as one would expect for the Lagrangian relaxation.

Building on Problem (L), we further wish to exploit scenario decomposition, which in turn facilitates decomposition of x and d. The decomposition of Problem (E) by scenarios results (temporarily ignoring the coupling constraint) in a problem of obtaining a solution $x(s)$ for each scenario $s \in \mathcal{S}$, as follows:

$$\begin{array}{rc} \text{minimize} & c(x(s)) \\ \text{subject to} & x(s) \in \mathcal{Q}(s) \end{array}$$

When we decompose (E) by scenarios, we are implicitly relaxing the non-anticipativity constraint present in (E). As a result, we obtain a set of problems that are, even collectively, much easier to solve than the full Problem (E). Depending on the structure of $\mathcal{Q}(s)$, the subproblems can be tractable when the full problem is computationally prohibitive. This situation is illustrated in our computational experiments described in §5.

For our purposes, a key aspect of scenario-based decomposition is that d appears only in the coupling constraints. Thus, once the x variables for the scenario subproblems have been optimized, the optimal assignment of d_s variables for Problem (L) is immediate. Let $x(s)$ be the optimal value for a scenario subproblem of (L), i.e.,

$$\begin{array}{rc} x(s) := \underset{x}{\operatorname{argmin}} & c(x) - \lambda\left(d_s - (1-\alpha)|\mathcal{S}|\right) \\ \text{subject to} & x \in \mathcal{Q}(s) \end{array}$$

Clearly, if the optimal $x(s)$ values would result in a reduction in the value of the objective function, then $d_s = 1$; otherwise it is optimal to set d_s to zero for that scenario. This observation is formalized as follows:

Remark 1 *When non-anticipativity is Problem (L) is relaxed, if $c(x(s)) < \lambda$, then $d_s = 1$ is optimal, otherwise $d_s = 0$ is optimal.*

In order to exploit this decomposition, we need to deal with the fact that the decision vector x cannot depend on s. If we could obtain an optimal or nearly optimal x^* such that for all $s \in \mathcal{S}$, $x(s) = x^*$, we could immediately set the d values and terminate the search. We accomplish this using algorithms described next in §3. Alternatively, a very simple greedy algorithm is obtained simply by setting $d_s := 1$ for the $(1-\alpha)|\mathcal{S}|$ scenarios for which $c(x(s))$ is lowest. This greedy algorithm is used as a computational baseline in §5.

3 Applications of Progressive Hedging for Scenario Selection

The progressive hedging algorithm proposed by Rockafellar and Wets (1991) provides a mechanism for combining scenario sub-problem solutions and enforcing non-anticipitivity. Progressive hedging (PH) is sometimes referred to as a horizontal decomposition method because it decomposes the problem by scenarios rather than by time stages.

In this section, we outline two ways to use PH to address the solution of Problem (E). The first approach serves as a straightforward introduction to the PH algorithm, and can solve Problem (E) when the d vector is given. This algorithm is used effectively as a post-processor in §5. The second approach is an implementation for solving Problem (L) which can then be embedded in a search for the smallest λ that results in $\sum_{s\in\mathcal{S}} d_s \geq (1-\alpha)|\mathcal{S}|$. This algorithm is used as a comparative baseline in §5.2. A more sophisticated algorithm is presented in §4 that simultaneously embeds the search for λ and d in the progressive hedging algorithm.

3.1 PH for Problem (E) Given d

Given a particular d vector, PH for the solution of Problem (E) reduces to the basic PH algorithm, which takes a perturbation vector $\rho > 0$ of length n and a convergence tolerance ϵ as input parameters. Pseudo-code for PH in this context is given as follows:

```
1. $k := 0$
2. For all $s \in \mathcal{S}$, $x^{(k)}(s) := \text{argmin}_x\, c(x) : x \in \mathcal{Q}(s)$
3. $\bar{x}^k := (\sum_{s\in\mathcal{S}} d_s x^{(k)}(s))/\sum_{s\in\mathcal{S}} d_s$
4. For all $s \in \mathcal{S}$, $w^{(k)}(s) := \rho(x^{(k)}(s) - \bar{x}^{(k)})$
5. $k := k+1$
6. For all $s \in \mathcal{S}$, $x^{(k)}(s) := \text{argmin}_x\, c(x) + w^{(k-1)}(s)x + \rho/2\, \left\|x - \bar{x}^{(k-1)}\right\|^2$
                          $: x \in \mathcal{Q}(s)$
7. $\bar{x}^{(k)} := (\sum_{s\in\mathcal{S}} d_s x^{(k)}(s))/\sum_{s\in\mathcal{S}} d_s$
8. For all $s \in \mathcal{S}$, $w^{(k)}(s) := w^{(k-1)}(s) + \rho\left(x^{(k)}(s) - \bar{x}^{(k)}\right)$
9. $g^{(k)} := \frac{1}{|\mathcal{S}|}\sum_{x\in\mathcal{S}} \left\|x^{(k)} - \bar{x}^{(k)}\right\|$
10. If $g^{(k)} < \epsilon$, then go to step 5. Otherwise, terminate.
```

In the pseudocode, $x^{(k)}(s)$ denotes the value of $x(s)$ for scenario $s \in \mathcal{S}$ at iteration k of PH, while $\bar{x}^{(k)}$ denotes the corresponding average $x^{(k)}(s)$ over all scenarios s with $d_s = 1$. The $w^{(k)}$ are PH-specific, per-scenario vectors of length n, and serve as the mechanism through which the $x^{(k)}(s)$ are eventually forced to agreement. PH is terminated once the scenario sub-problems are sufficiently homogeneous, as quantified by $g^{(k)}$ and thresholded by ϵ. If Problem (E) is convex given a fixed d, PH is guaranteed to locate optimal solutions given appropriate values of ρ and ϵ. Otherwise, the use of PH is as a heuristic, with the objective of quickly locating high-quality solutions.

3.2 PH for Problem (L) Given λ

Remark 1 enables modification of the PH algorithm given above in §3.1 by adding the logic

If $c(x(s)^{(k)}) \leq \lambda$ then $d_s := 1$ else $d_s := 0$

to Steps 2 and 6. The result is that for a given λ, a straightforward PH algorithm for Problem (L) can be stated as follows, again taking $\rho > 0$ and ϵ as input parameters:

```
1. $k := 0$
2. For all $s \in \mathcal{S}$, $x^{(k)}(s) := \text{argmin}_x\, c(x) : x \in \mathcal{Q}(s)$
       If $c(x(s)^{(k)}) \leq \lambda$ then $d_s := 1$ else $d_s := 0$
3. $\bar{x}^k := \sum_{s\in\mathcal{S}} d_s x^{(k)}(s) / \sum_{s\in\mathcal{S}} d_s$
4. For all $s \in \mathcal{S}$, $w^{(k)}(s) := \rho(x^{(k)}(s) - \bar{x}^{(k)})$
5. $k := k + 1$
6. For all $s \in \mathcal{S}$, $x^{(k)}(s) := \text{argmin}_x\, c(x) + w^{(k-1)}(s)x + \rho/2 \left\| x - \bar{x}^{(k-1)} \right\|^2$
                             $: x \in \mathcal{Q}(s)$
       If $c(x(s)^{(k)}) \leq \lambda$ then $d_s := 1$ else $d_s := 0$
7. $\bar{x}^{(k)} := \sum_{s\in\mathcal{S}} d_s x^{(k)}(s) / \sum_{s\in\mathcal{S}} d_s$
8. For all $s \in \mathcal{S}$, $w^{(k)}(s) := w^{(k-1)}(s) + \rho\left(x^{(k)}(s) - \bar{x}^{(k)}\right)$
9. $g^{(k)} := \frac{1}{|\mathcal{S}|} \sum_{x\in\mathcal{S}} \left\| x^{(k)} - \bar{x}^{(k)} \right\|$
10. If $g^{(k)} < \epsilon$, then go to step 5. Otherwise, terminate.
```

In Steps 2 and 6, the search for optimal x and d values are separated exploiting Remark 1. As a practical matter, Steps 3 and 7 must include a test for the possibility that $\sum_{s\in\mathcal{S}} d_s = 0$; in this case the algorithm terminates and reports that λ is too small. Due to non-convexity, PH for solution of Problem (L) is again a heuristic solution technique.

4 Progressive Hedging for Simultaneous Determination of λ, d, and x

In order to simultaneously determine λ, d, and x, we use progressive hedging to find values of $x(s)$ that progressively satisfy the non-anticipativity requirement. In the process, we determine a minimal λ that would result in $\sum_{s\in\mathcal{S}} d_s \geq (1-\alpha)|\mathcal{S}|$ and then set the d_s values accordingly. In order to improve the stability of the $\bar{x}$ estimates (and therefore the w estimates), we relax the strict binary constraints on the d_s variables, and instead progressively bias the values of d_s toward either 0 or 1 as the algorithm converges, as discussed in 4.1. The details of the enhanced PH algorithm are provided in Section 4.2.

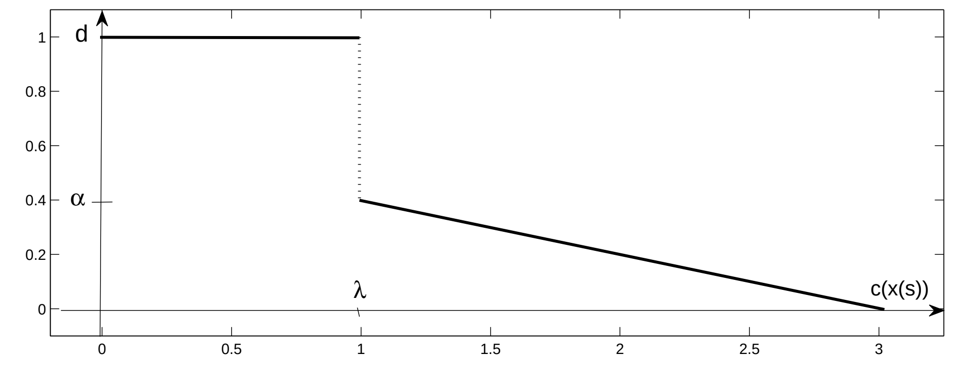

Figure 1: A piecewise linear bias function for relaxing the binary constraints on the d_s scenario selection variables as a function of the PH convergence gap α and λ.

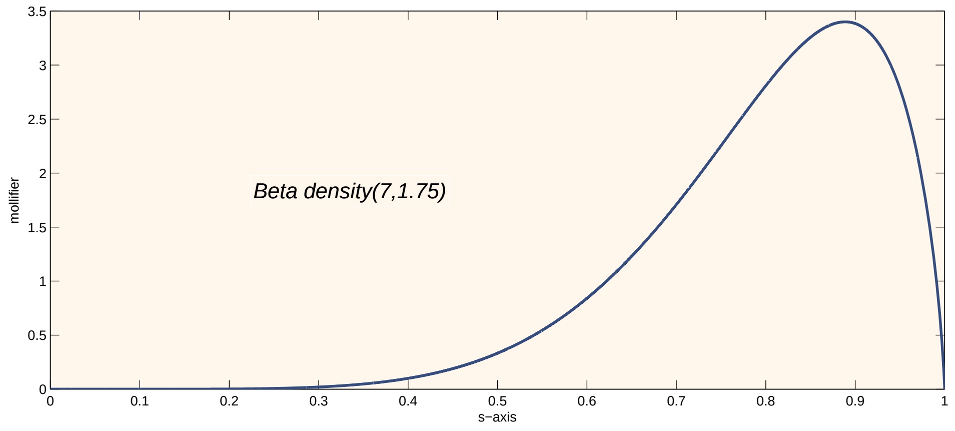

Figure 2: A mollifier function based on the Beta density function when $\beta = 1$.

4.1 Techniques for Biasing the d_s Variables

A straightforward approach to relaxing the binary restrictions on the d_s variables is via an augmentation function. Assuming the existence of an upper bound on the cost is known and represented by $\lambda_{\max}$. The mechanisms for obtaining or computing this upper bound depend on the underlying problem. One possibility is to impose a simple piecewise linear bias : if $c(x(s)) \leq \lambda$ then $d_s := 1$ else $d_s := \alpha \cdot \max(0, (\lambda_{\max} - c(x(s)))/(\lambda_{\max} - \lambda))$ where $\alpha = g^{(k)}/g^{(0)}$ and g is the PH convergence gap. Consequently, α returns one at PH iteration 0, and approaches zero (i.e., the d_s become binary) as PH converges. The augmentation function equals 1 for inputs $\leq \lambda$, and linearly decreases between α and 0 for inputs ranging between λ and the upper bound on cost, $\lambda_{\max}$. This function is shown in Figure 1.

A more effective and mathematically motivated augmentation function, and the one that we ultimately implemented, relies on the composition of the piecewise step function with a sequence of *mollifiers* that serve to smooth the step augmentation functions; cf.

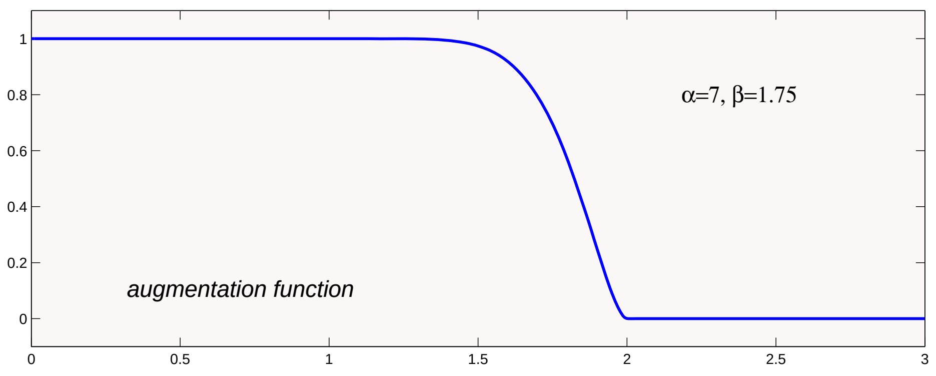

Figure 3: The Beta mollifier-based augmentation function given $\beta = 1$.

Ermoliev et al. (1995) for a related approach when dealing with the minimization of discontinuous functions. Our "limiting" step function, with argument $\tau = c(x(s))$, is given by:

$$f(\tau, \lambda) = \begin{cases} 1 & \text{when } 0 \leq \tau < \lambda, \\ 0 & \text{for } \lambda \leq \tau \leq \lambda_{\max} \end{cases}$$

Our collection of mollifiers are Beta density functions defined on the intervals $[0, \beta]$ with $\beta \searrow 0$ as the PH algorithm converges:

$$\phi^{1/\beta}(x) = \begin{cases} \frac{1/\beta}{B(7,1.75)}(x/\beta)^6(1 - x/\beta)^{0.75} & \text{when } x \in [0, \beta], \\ 0 & \text{everywhere else} \end{cases}$$

where the Beta function $B(\alpha, \beta) = \int_0^1 x^{\alpha-1}(1-x)^{\beta-1}\,dx$. Here, $\beta = \alpha = g^{(k)}/g^0$, i.e., the PH convergence gap. We use distinct symbols for the PH gap to clarify the context – the original piecewise step function or the mollified variant. A graphical depiction of our particular parameterization $B(7, 1.75)$ is provided in Figure 2, given $\beta = 1$.

Given the family of mollifier functions parameterized on β, the corresponding augmentation functions are obtained via the convolutions

$$f^{1/\beta}(x, \lambda, \lambda_{\max}) = \int_0^\beta f(x - z, \lambda)\phi^{1/\beta}(z)\,dz, \qquad x \in [0, \lambda_{\max}].$$

A graphical depiction of the mollifier-based augmentation function is provided in Figure 3, given $\beta = 1$.

4.2 Progressive Hedging for Scenario Selection

Given the family of augmentation functions $f^{1/\beta}$, a PH algorithm for scenario selection can be stated as follows:

1. $k := 0$ and $\beta := 1$.
2. For all $s \in \mathcal{S}$, $x^{(k)}(s) := \operatorname{argmin}_x c(x) : x \in \mathcal{Q}(s)$
3. Assign an upper bound on the cost function to λ_{max}

4. $(\lambda, d) := \text{argmin}_{\lambda \leq \lambda_{\max}}$ such that for all $s \in \mathcal{S}$,

$$d_s := f^{1/\beta}(c(x^0(s)), \lambda, \lambda_{\max}) \text{ and } \sum_{s \in \mathcal{S}} d_s \geq (1-\alpha)|\mathcal{S}|$$

5. $\bar{x}^0 := \sum_{s\in\mathcal{S}} d_s x^{(0)}(s) / \sum_{s\in\mathcal{S}} d_s$
6. For all $s \in \mathcal{S}$, $w^{(0)}(s) := \rho(x^{(0)}(s) - \bar{x})$
7. $k := k+1$
8. For all $s \in \mathcal{S}$, $\begin{array}{l} x^{(k)}(s) := \text{argmin}_x \, c(x) + w^{(k-1)}(s)x + \rho/2 \left\| x - \bar{x}^{(k-1)} \right\|^2 \\ : x \in \mathcal{Q}(s) \end{array}$
9. $(\lambda, d) := \text{argmin}_{\lambda \leq \lambda_{\max}}$ such that for all $s \in \mathcal{S}$,

$$d_s := f^{1/\beta}(c(x^k(s)), \lambda, \lambda_{\max}) \text{ and } \sum_{s \in \mathcal{S}} d_s \geq (1-\alpha)|\mathcal{S}|$$

10. $\bar{x}^{(k)} := \sum_{s\in\mathcal{S}} d_s x^{(k)}(s) / \sum_{s\in\mathcal{S}} d_s$
11. For all $s \in \mathcal{S}$, $w^{(k)}(s) := w^{(k-1)}(s) + \rho\left(x^{(k)}(s) - \bar{x}^{(k)}\right)$
12. $g^{(k)} := \frac{1}{\sum_{s\in\mathcal{S}} d_s} \sum_{i \in [1..n]} ((x_i^{(k)} - \bar{x}_i^{(k)})/\bar{x}_i^{(k)})$ and $\beta := g^{(k)}/g^{(0)}$
13. If $g^{(k)} < \epsilon$, then go to step 7. Otherwise, terminate.

We denote the above algorithm simply by SSPH. Recall that n is the number of decision variables in Problem (E). In Steps 4 and 9, the condition for achieving the target number of scenarios is based on non-binary (relaxed) d_s. As SSPH converges, certain d_s are driven toward 0, and in practice once a d_s is "sufficiently" close to 0, it is highly unlikely that the same $d_s = 1$ in a converged solution. Thus, we further accelerate SSPH convergence by truncating all $d_s \leq 0.1$ to $d_s = 0$. In many situations (e.g., as in §5), the solution of Problem (E) given a specific d is computationally efficient. Thus, *given* binary d_s for all $s \in \mathcal{S}$ – achieved as SSPH converges – one can exit SSPH after Step 9 once $\sum_{s\in\mathcal{S}} d_s \geq (1-\alpha)|\mathcal{S}|$, and solve the remainder of Problem (E) using algorithms not targeted toward scenario selection. We refer to this technique as a "quick exit" strategy.

Steps 4 and 9 determine the value of λ that will result in the desired sum over the d vector. As can be seen from Figures 1 and 3, the values of d_s are monotonic in λ so the minimization can be done easily.

5 Computational Experiments

We now examine the performance of the proposed PH algorithms for scenario selection in stochastic programming, considering two test cases: a small-scale, yet difficult "laboratory" problem and a large-scale, real-world problem. The advantage of the laboratory problem is that performance can be assessed relative to both establish algorithms in the literature and extensive form solutions obtained via commercial mixed-integer programming solvers. In contrast, the extensive form of the real-world problem is not solvable with commercial solvers, and competing approaches have yet to be introduced. However, the size of the real-world problem instances serves to demonstrate the scalability and applicability of the proposed algorithms.

Both examples are of a form that facilitates easy computation of an upper bound on cost. For our examples, Step 3 in the combined PH algorithm is replaced by $\forall i \in [1..n], x_i^{\max} := \max_{s\in\mathcal{S}} x_i^{(k)}(s)$ and $\lambda_{\max} := c(x^{\max})$

5.1 Ruszczyński's Network Flow Model

Ruszczyński makes use of a chance constrained network flow problem to illustrate his method for solving the scenario selection problem (E) (Ruszczyński 2002). We adopt and and also extend this problem for use in our experiments. For the purpose of describing the formulation, we stay as close as possible to his notation, deviating only when necessary to avoid conflicts with our own. Ruszczyński's notation allows for scenarios with different probabilities, but the example he used has equal probabilities, so we introduce the model formulation without loss of generality in the context of that assumption.

5.1.1 The Basic Model.

We begin with a node set $\mathcal{V}$ and an arc set $\mathcal{A} \subseteq \mathcal{V} \times \mathcal{V}$. For each scenario $s \in \mathcal{S}$ a quantity $D_{kl}(s)$ must be shipped between each distinct pair of nodes $(k, l) \in \mathcal{V} \times \mathcal{V}$, $k \neq l$. The optimization objective is to determine arc capacities $x(a), a \in \mathcal{A}$, that minimize the total cost $\sum_{a \in \mathcal{A}} c(a)x(a)$ while enabling the shipping requirements (i.e., demands) to be met with probability $1 - \alpha$. The $c(a)$ represent capacity cost parameters associated with each arc $a \in \mathcal{A}$. Variables $y_{kl}(a, s)$ are introduced to represent the flow from k to l passing through arc $a \in \mathcal{A}$ in scenario $s \in \mathcal{S}$. The problem formulation, which we denote (NF), is then given as:

$$\begin{array}{ll}
\text{minimize} & \sum_{a \in \mathcal{A}} c(a)x(a) \\
\text{subject to} & \sum_{a \in \mathcal{A}^+(\nu)} y_{kl}(a, s) - \sum_{a \in \mathcal{A}^-(\nu)} y_{kl}(a, s) = \begin{cases} -D_{kl}(s) & \text{if } \nu = k \\ D_{kl}(s) & \text{if } \nu = l \\ 0 & \text{otherwise} \end{cases} \\
& \qquad\qquad\qquad\qquad \forall \nu, k, l \in \mathcal{V}, \forall s \in \mathcal{S} \\
& \sum_{s \in \mathcal{S}} d_s \geq (1 - \alpha)|\mathcal{S}| \\
& x(a) \geq \sum_{k,l \in \mathcal{V}} y_{kl}(a, s), \qquad \forall a \in \mathcal{A},\ \forall s \in \{\mathcal{S} : d_s = 1\} \\
& x \geq 0,\ y \geq 0,\ d_s \in \{0, 1\}, \qquad \forall s \in \mathcal{S}
\end{array}$$

where the notations $\mathcal{A}^+(\nu)$ and $\mathcal{A}^-(\nu)$ respectively indicate the set of arcs into and out of node ν. In practice, α and $|\mathcal{S}|$ are selected such that $(1 - \alpha)|\mathcal{S}|$ is integer.

The particular test instance of (NF) used by Ruszczyński is a small example with 5 nodes and 7 arcs and is intended to demonstrate his method rather than to be the basis of extensive computational experiments (see Ruszczyński (2002) for a full description of the data used, including the specific network configuration and arc capacity costs). In this instance all arcs are bi-directional and the capacities are assumed to be symmetric across an arc, i.e., $a_1 = (i, j) \wedge a_2 = (j, i) \Rightarrow x(a_1) = x(a_2)$. The target $\alpha = 0.1$ was used in all experiments.

Building on this example, we generated ten random 100-scenario and five random 400-scenario (NF) instances using the procedure described in Ruszczyński (2002). In particular, the $D_{kl}(s)$ for a specific scenario $s \in \mathcal{S}$ is given by $D_{kl}(s) = 0.1D_s + \epsilon_{kl}$, where D_s represents the aggregate flow volume for scenario s, sampled from a normal distribution $\mathcal{N}(30, 5)$; the ϵ_{kl} are independent normal variables with mean 0 and standard deviation 0.25.

For each instance, we compute the optimal solution of the (NF) extensive form using both CPLEX 10.1 (ILOG 1997) and Ruszczyński's exact method; we are grateful to Ruszczyński for sharing his AMPL code for solving (NF). We then compare the performance of our SSPH algorithm against these baselines. In particular, we consider two variants of SSPH: one variant in which the "quick-exit" logic described in Section 4 is utilized, and another in which the algorithm is allowed to run to convergence. In all

	CPLEX		Ruszczyński		Greedy		SSPH(Quick)		SSPH(Full)	
Instance	Obj.	T.	Obj.	T.	Obj.	T.	Obj.	T.	Obj.	T.
1	27264.1	58.5	27264.1	9.5	27454.4	0	27264.1	0	27264.1	3.5
2	26993.7	107	26993.7	9	26993.7	0	26993.7	0	26993.7	3
3	27318.6	29	27318.6	3	27473.9	0	27318.6	0	27318.6	3.5
4	28190.7	38	28190.7	4.5	28190.7	0	28255.2	0.5	28255.2	3
5	27461.2	66	27461.2	3.5	27461.2	0	27461.2	0	27461.2	2.5
6	28457.2	53	28457.2	1	28457.2	0	28457.2	0.5	28457.2	3
7	26313.3	54.5	26313.3	3.5	26377.9	0	26377.9	0.5	26313.3	3
8	26541.7	45.5	26541.7	1.5	26563.2	0	26563.2	0	26541.7	3
9	28547.1	42	28547.1	2.5	28547.1	0	28547.1	0	28547.1	3
10	28151.1	44	28151.1	6	28224.1	0	28224.1	0	28224.1	3
% Gap	0.0		0.0		0.185		0.081		0.049	
Time		53.75		4.4		0		0.15		3.05

Table 1: Performance results for various scenario selection algorithms on 100-scenario network flow problems. For each algorithm, the total cost (Obj. labeled columns) and run time (T. labeled columns) are reported. Run-times are reported in minutes, rounded to the nearest half minute increment. The final two rows record the average percentage above optimal solution cost (row labeled "Gap") and the average run time (row labeled "Time").

SSPH runs, we define for each variable $x(a)$, $a \in \mathcal{A}$, the PH parameter $\rho_a = c(a)$; this particular choice was based on our prior experience with variable-dependent ρ values (Watson et al. 2007). In all runs, $\epsilon = 0.01$.

At termination, both variants of SSPH have identified a candidate set of scenarios, as defined by the resulting d_s variables. The d_s are then fixed in Problem (NF), yielding a straightforward linear program which is then solved using CPLEX. Finally, for an additional baseline, we compare the performance of SSPH variants relative to that of a simple greedy approach that first selects the most expensive $(1-\alpha)|\mathcal{S}|$ scenarios, as described in §2 As with SSPH, the obtained d_s are then fixed in the (NF) problem formulation, and the resulting linear program is solved via CPLEX. The times reported for the SSPH and greedy algorithms include all overhead processing and final CPLEX linear program solve times. All execution times are rounded to the nearest half minute increment. All experiments were executed on a 2.2GHz AMD Athlon architecture running Linux, with 64GB of RAM.

We first consider the results for the 100-scenario instances, which are summarized in Table 1. Consistent with the statements made in Ruszczyński (2002), Ruszczyński's method significantly outperforms CPLEX, obtaining optimal solutions in roughly an order of magnitude less time. This is despite the advances in CPLEX solver technology since Ruszczyński (2002) appeared. The baseline greedy scenario selection approach performs remarkably well, achieving solutions on average only 0.185% above optimal in a few seconds of runtime; in half of the instances, the greedy solution is optimal. This result is likely due to the sampling procedure used to construct the problem instances; the ϵ_{kl} are sampled from a relatively tight distribution, such that there is no significant overlap in demands $D_{lk}(s)$ across scenarios $s \in \mathcal{S}$. On average, SSPH with the quick exit strategy enabled yields improvements over greedy scenario selection with only slight increases in run-time. SSPH with full convergence requires relatively longer run-times, but they remain lower than those obtained by Ruszczyński's method; solutions were

	CPLEX		Ruszczyński		Greedy		SSPH(Quick)		SSPH(Full)	
Instance	Obj.	T.	Obj.	T.	Obj.	T.	Obj.	T.	Obj.	T.
1	27931.9	1440	27708.6	5600	27908.3	0	27908.3	1	27724.8	13
2	28560.6	1440	27434.1	3529	27739.3	0	27595.7	1	27512.5	13
3	30184.7	1440	27524	2240.5	27784.6	0	27742.7	1	27609.7	13.5
4	29082.9	1440	27955.8	2889.5	28111.3	0	28111.3	1	28019.6	15.5
5	28995.9	1440	27613.2	2358	27700.6	0	27700.6	3.5	27690.7	12.5
% Better	0.0		4.441		3.744		3.872		4.220	
Time		1440		3323		0		1.5		13.5

Table 2: Performance results for various scenario selection algorithms on 400-scenario network flow problems. For each algorithm, the total cost (Obj. labeled columns) and run time (T. labeled columns) are reported. Run-times are reported in minutes, rounded to the nearest half minute increment. The final two rows record the average percentage quality improvement over the CPLEX solution (row labeled "% Better") and the average run time (row labeled "Time").

only 0.0488% above optimal on average, and optimal solutions were identified in eight instances.

Overall, the heuristic scenario selection algorithms were able to obtain very high-quality solutions in minimal run-times, relative to the extensive form solves via CPLEX. Clearly, Ruszczyński's exact method is the preferred algorithm for these particular small instances. Rather, the results reported in Table 1 serve as a preliminary demonstration of the potential effectiveness of SSPH.

Next, we consider the results for the 400-scenario instances, which are summarized in Table 2. Here, we limited the run-time of CPLEX on each instance to 1440 minutes, i.e., 1 day. In no case did CPLEX locate an optimal solution, and the optimality gaps at termination were significant. Despite the apparent simplicity of the (NF) problem formulation, obtaining high-quality solutions for moderate numbers of scenarios appears problematic. This is confirmed by the results we obtained for Ruszczyński's method; within the allocation of 1440 minutes of run time, the method was only able to complete roughly 13-14 iterations on any given problem instance, with some individual iterations requiring greater than half of the time allocation. In no case were we able to identify a solution within the time limit. We did eventually allow runs of Ruszczyński's method to complete, which required several days of computation; the results are reported in Table 2. These extended-duration runs allow us to obtain optimal solutions, which we use to assess the absolute performance of our heuristics. In contrast, CPLEX was unable to identify optimal solutions on any instance given a week of run-time.

Surprisingly, even greedy scenario selection obtains solutions over 3.7% better than those achieved by CPLEX, in less than 10 seconds for all instances. SSPH with the quick exit strategy yields slight improvements over greedy (to nearly 4% better than CPLEX) in reasonable run times ($\approx$ 1.5 minutes on average). Executing SSPH to full convergence results in further improvements, albeit at the expense of increased (but still reasonable) run times. In absolute terms, the SSPH algorithm with full convergence is able to locate very near-optimal solutions in orders-of-magnitude less time than Ruszczyński's method, and is able to locate higher quality solutions than CPLEX, again in a fraction of the run-time.

Overall, both SSPH algorithm variants (quick-exit and full convergence) yield improvements relative to the greedy algorithm, while still executing in reasonable run-

	CPLEX		Greedy		SSPH(Quick)		SSPH(Full)	
Instance	Obj.	T.	Obj.	T.	Obj.	T.	Obj.	T.
1	29209.9	1440	29250.4	32	29258.4	2	29209.9	18.5
2	29408.8	1440	29079.8	16	29079.8	5	29105.9	78
3	29541.3	1440	29748.1	165	29608.3	2	29372.8	222
4	32173.8	1440	30480	2	30480	2	30224.2	128.5
5	29819.6	1440	30604.4	270	29741	2	29741	8
6	30481.3	1440	30709	110	30709	6.5	30675.7	13.5
7	28311.4	1440	28448.5	2	28171.5	14	28171.5	34
8	28857.5	1440	28763.7	22.5	28637.4	2.5	28637.4	113.5
9	31121.3	1440	31121.3	76.5	30777.4	2.5	30777.4	11
10	30422.9	1440	30472.6	21	30422.9	1440	30436.5	17.5
% Better			0.184		0.787		0.960	
Time		1440		71.7		147.85		64.45

Table 3: Performance results for various scenario selection algorithms on 100-scenario budget network flow problems. For each algorithm, the total cost (Obj. labeled columns) and run time (T. labeled columns) are reported. Run-times are reported in minutes, rounded to the nearest half minute increment. The final two rows record the average objective percentage improvement over the CPLEX solution (row labeled "% Better") and the average run time (row labeled "Time").

times. In contrast to both exact methods, SSPH is conclusively more scalable; even at 400 scenarios on a 5-node instance, the exact methods are encountering serious performance issues.

5.1.2 An Extended Model With Arc Budgets.

To further investigate the scalability of SSPH relative to direct solution of the extensive form via CPLEX, we consider a slightly more complicated version of Problem (NF). To increase computational complexity, we add a per-scenario budget for the number of arcs that can be used in a solution. For each scenario $s \in \mathcal{S}$ and arc $a \in A$, we introduce a binary variable $b(a, s)$. In each scenario $s \in \mathcal{S}$, an arc $a \in A$ can be made active, subject to a total budget on the number of active arcs $B(s)$. Assuming the availability of a large constant M (e.g., one plus the sum of all demands), we add the following constraints to the (NF) problem formulation:

$$\begin{aligned} y_{kl}(a,s) \le M \cdot b(a,s), \qquad & \forall k,l \in \mathcal{V}, \forall a \in \mathcal{A}, \forall s \in \mathcal{S} \\ \textstyle\sum_{a\in\mathcal{A}} b(a,s) \le B(s), \qquad & \forall s \in \mathcal{S} \\ b(s,a) \in \{0,1\}, \qquad & \forall a \in \mathcal{A},\ \forall s \in \mathcal{S} \end{aligned}$$

To maintain consistency with the arc symmetry budget described in §5.1.1, we additionally impose a corresponding budget symmetry constraint, i.e., $a_1 = (i,j) \wedge a_2 = (j,i) \Rightarrow b(a_1,s) = b(a_2,s), \forall s \in \mathcal{S}$. We denote the resulting formulation by (BNF), which is an acronym for Budget Network Flow.

We mirror the experimental methodology described in Section 5.1.1 to generate random instances of Problem (BNF). The $B(s)$ for each problem instance are sampled as

	CPLEX		Greedy		SSPH(Quick)		SSPH(Full)	
Instance	Obj.	T.	Obj.	T.	Obj.	T.	Obj.	T.
1	30422.9	1440	39030.2	5	31105	10	29708.6	29708.6
2	45226	1440	38512.8	5	29822.3	9.5	29664.6	29664.6
3	46081.8	1440	36528	5	31223.8	10	29742.8	29742.8
4	74587.8	1440	40430	5	31127.6	9	29918.8	29918.8
5	47891.5	1440	38372.8	5	29955.8	9.5	29893	29893
% Better	-		14.59		31.96		63.86	
Time		1440		5		9.6		174.6

Table 4: Performance results for various scenario selection algorithms on 400-scenario budget network flow problems. For each algorithm, the total cost (Obj. labeled columns) and run time (T. labeled columns) are reported. Run-times are reported in minutes, rounded to the nearest half minute. The final two rows record the average objective percentage improvement over the CPLEX solution (row labeled "% Better") and the average run time (row labeled "Time").

follows: with probability 0.8, B(s) = 12; otherwise, B(s) = 10. Due to the arc symmetry constraints, this sampling implies that either 5 or 6 of the total 7 (symmetric) arcs may be activated. We again generate ten and five 100 and 400 scenario instances, respectively. With the exceptions noted below, we replicate the experimental methodology described in Section 5.1.1. In the case of Problem (BNF), our sole comparative baseline is CPLEX for solving the extensive form. SSPH parameters are identical to those described for the experiments presented in §5.1.1.

First, we consider the results for the 100-scenario (BNF) instances, as reported in Table 3. In contrast to the (NF) results, CPLEX is unable to identify optimal solutions within the allocated run-time budget of 1440 minutes; at termination, the optimality gap is again significant. We assess the performance of our heuristic scenario selection algorithms relative to the CPLEX baseline. The greedy algorithm obtains better solutions than CPLEX on average (a slight improvement of 0.184%), in substantially less run-time ($\approx$ 71 minutes), although CPLEX does outperform the greedy algorithm on half of the instances. Nearly all of the run-time associated with the greedy algorithm is attributable to the cost of solving the mixed-integer program resulting from fixing of the d_s scenario selection variables. These models can be very difficult, in contrast to the linear programs resulting from fixing of the d_s in the (NF) formulation. SSPH with the quick exit strategy enabled yields further improvements in solution quality (0.787% better than CPLEX). On average, we observe an increased run-time relative to greedy, but this is due to the specific results for instance 10. For instances 1–9, SSPH with quick exit enabled requires less run-time than the greedy algorithm (due to improved initial incumbents being provided to CPLEX); for instance 10, the initial incumbent yielded an optimality gap of 0.02%, and CPLEX failed to prove optimality within the 1440 minute time allocation. Finally, the fully converged SSPH algorithm obtains the best overall performance (a 0.96% improvement over CPLEX), in roughly an hour on average.

Finally, we consider the results for the 400-scenario (BNF) instances, as reported in Table 4. For these instances, the mixed-integer programs resulting from fixing of the d_s variables are extremely difficult, with CPLEX typically failing to improve the upper bound within the time budget in most cases. Consequently, we restrict the use of CPLEX to the greedy algorithm, in which we first obtain the d_s and allow CPLEX to locate the first incumbent, at which point search is terminated. The SSPH algorithms yield complete solutions upon termination. The results exhibit similar patterns to those obtained for the (NF) and 100-scenario (BNF) runs: (1) greedy outperforms CPLEX and is in turn outperformed by both SSPH variants and (2) improved performance of the heuristics comes with the expected increase in computational cost. The primary difference is the degree of improvement, which ranges from a remarkable 14% to 64% in the case of 400-scenario (BNF) instances.

5.2 An Aircraft Sustainability Planning Model

Our work on scenario selection heuristics for stochastic programming was originally motivated by a real-world application involving the allocation of spare parts and part repair-related resources to a large aircraft maintenance operation. The objective in allowing a certain proportion of infeasible scenarios in this application is to facilitate investigation into the risk-reward trade-off: the maintenance contractor may accept a certain probability α of failing to meet target performance criteria (expressed as the average percentage of time that aircraft in the fleet are available to fly) in exchange for a reduced capital expenditure in the deployment of the associated supply and repair chains. Monetary penalties are imposed if performance targets are not achieved.

This aircraft sustainability problem can be formulated as a stochastic mixed-integer program with the following characteristics. Given a cost vector $c \geq 0$ of length n, a scenario set $\mathcal{S}$ of size $|\mathcal{S}|$, a scalar $0 < \alpha \leq 1$, matrices $A(s) \geq 0$ each of which is of dimension m by n, and $b(s)$ vectors each of length m for all $s \in \mathcal{S}$, find vector a x of length n and vector d of length $|\mathcal{S}|$ to:

$$\begin{array}{rcr}
\text{minimize} & cx & \text{(ASP)} \\
\text{subject to} & A(s)x >= b(s)d_s, \quad \forall s \in \mathcal{S} & \\
& \sum_{s\in\mathcal{S}} d_s \geq (1-\alpha)|\mathcal{S}| & \\
& d_s \in \{0,1\}, \quad \forall s \in \mathcal{S} & \\
& x \in (\mathcal{Z}^+)^n &
\end{array}$$

In this formulation, which we denote (ASP), the $A(s)$ and $b(s)$ associated with a given scenario $s \in \mathcal{S}$ represent constraints that model a discrete event simulation of the sustainability operation, subject to a given realization of aircraft part failures (Savage et al. 2005). The variables x encode inventory policy parameters, resource levels, and time-indexed state tracking variables. The latter represent a dominant fraction of the total number of variables, which reaches over a million for the largest instances we consider below. For any scenario $s \in \mathcal{S}$, the number of constraints reaches over a million, with the corresponding number of non-zeros in $A(s)$ reaching 10 million. Details of the (ASP) formulation, a scenario solver, and an enhanced PH algorithm are given in (Watson et al. 2007). In particular, we describe efficient mechanisms for solving scenario subproblems heuristically, setting and varying the PH parameter ρ, accelerating convergence, and economically terminating PH.

Clearly, the defining characteristic of Problem (ASP) is its size, which allows us to investigate the run-time scalability of our scenario selection heuristics. On the other hand, the scale and novelty of the formulation prevents us from obtaining a comparative performance baseline; CPLEX requires tens of gigabytes of RAM to store the extensive form of our largest problem instances, and solution of even the linear relaxation of (ASP) can require several days. However, the problem is representative of industrial scales, for which solutions must be obtained despite the lack of a strict comparative baseline. Thus, our main interest here is in demonstrating run-time scalability of scenario selection heuristics for stochastic programming, specifically in comparing the following algorithms:

- Greedy: Solve all scenario subproblems independently, select the resulting $(1-\alpha)|\mathcal{S}|$ lowest-cost scenarios (setting $d_s = 1$ for the lowest-cost scenarios and $d_s = 0$ otherwise), and use PH to solve the related stochastic mixed-integer program consisting strictly of scenarios $s \in \mathcal{S}$ with $d_s = 1$.
- SSPH: Use the SSPH algorithm described in §4 to simultaneously determine λ, a scenario vector d, and a solution vector x.
- SSPH+: First, execute SSPH as above to obtain a d vector. Then, form a stochastic mixed-integer program consisting strictly of scenarios $s \in \mathcal{S}$ where $d_s = 1$. Finally, use PH to solve the resulting program.

In the case of Greedy and SSPH+, the PH algorithm is being used in a "touch-up" role, analogous to our use of CPLEX in conjunction with scenario selection heuristics as described in Section 5.1. As discussed above, the computational difficulty and scale of Problem (ASP) prevents exact solution of even the extensive form when the scenario selection constraint is relaxed. In contrast to our experiments with Problems (NF) and (BNF), we do not use the quick-exit logic in SSPH; accelerator techniques designed to exploit the general structure of Problem (ASP) mitigate the need for such a strategy. We defer to Watson et al. (2007) for a detailed description of these techniques.

We execute each heuristic algorithm on each of three synthetic (ASP) test instances, of varying size. For all trials, we use $\alpha = 0.2$. The computational platform is identical to that described above in §5.1. For each instance and algorithm combination, we record the overall run-time and the cost of the best solution obtained. The time and cost units are respectively minutes and USD. As previously noted, only a small fraction of the variables in the x vector require blending by progressive hedging to enforce non-anticipativity; the exact number ranges from 144 to 528 for the smallest and largest instances, respectively. All instances consist of 30 scenarios, which is deemed sufficient by the end-user given multi-year planning horizons and comparatively frequent aircraft part failures.

The obtained solution costs are reported in Table 5. The instances are characterized by the first three columns of each row and the remaining columns record the solution costs for each algorithm. For reference, we also provide two additional columns of results. The column labeled "All $d = 1$" records the solution cost obtained by solving Problem (ASP) with $\alpha = 1$ using PH. The column labeled "λ Search" records the solution cost obtained by embedding the PH algorithm for a given λ as given in §3.2 in a binary search for an optimal λ. The initial lower and upper bounds λ_l and λ_u are respectively given as (1) the minimum solution cost for an individual scenario identified in PH iteration 0 and (2) the cost of the solution obtained by taking an element-wise maximum of the $x(s)$ vectors obtained for all scenarios in PH iteration 0. The binary search is terminated once $(\lambda_h - \lambda_l)/\lambda_l \cdot 100 \leq 0.01$.

Necessarily, allowing infeasibility in a proportion $\alpha = 0.2$ of scenarios yields significant reductions in overall solution cost. While no clear "winner" exists among the scenario selection heuristics, we observe that SSPH+ dominates SSPH, which in turn dominates the greedy algorithm. SSPH dominates λ search on all but the medium-sized instance, while λ search underperforms the greedy algorithm on the largest instance. Although we cannot draw general conclusions given the small number of test instances, the

n	m	$\lvert\mathcal{S}\rvert$	All $d = 1$	Greedy	λ Search	SSPH	SSPH+
5M	4M	30	60,264,200	58,330,300	58,076,200	57,878,700	57,287,650
11M	8M	30	133,909,900	126,140,650	125,291,550	126,021,300	125,448,300
22M	16M	30	359,249,300	345,419,550	346,937,100	345,000,400	344,800,320

Table 5: Costs of solutions obtained using various scenario selection heuristics (see text), on a range of aircraft sustainability planning problem instances. The n and m values respectively represent the number of variables and variables in the extensive form of the (ASP) stochastic mixed-integer program.

n	m	$\lvert\mathcal{S}\rvert$	All $d = 1$	Greedy	λ Search	SSPH	SSPH+
5M	4M	30	94	64	669	85	85+58
11M	8M	30	365	304	3949	377	377+386
22M	16M	30	3715	3470	28069	3356	3356+3287

Table 6: Algorithm run-times in minutes on a 2.2GHz AMD Athlon Linux workstation, for the results reported in Table 5
.

preliminary evidence suggests that SSPH and SSPH+ provide consistent, high-quality solutions.

Next, we consider the relative run-times of the various algorithms, as reported in Table 6. The computational difficulty of Problem (ASP) is illustrated by the absolute run-times for the baseline PH algorithm with $d_s = 1$ for all $s \in \mathcal{S}$. The key observation is that SSPH requires no more time to compute solutions than the baseline PH algorithm, i.e., the additional complexity associated with determination of the d vector does not translate into increased computational costs. The run-time cost of the greedy algorithm is dominated by the PH post-processing run; the analogous post-processing in SSPH+ acts to roughly double the base SSPH run-time, with the benefit of lower-cost solutions. Finally, while λ search can locate solutions competitive with SSPH and SSPH+, the run-time cost is prohibitive; typically 10–15 outer loop iterations are required to achieve convergence.

6 Discussion and Conclusions

We have described computational procedures based on scenario decomposition to heuristically solve a wide ranging class of stochastic programs with chance constraints where the random components of the problem are discretely distributed, with or without integer constraints. Given their well-known difficulty even for small instances, an effective and scalable method for such problems is an important addition to the suite of tools available for optimization under uncertainty.

Computational experiments demonstrate the ability of the proposed heuristics to find near-optimal solutions to small, yet very difficult laboratory test examples, in orders of magnitude less run-time than exact algorithms. On larger instances of laboratory test examples, the heuristics yield higher-quality solutions than commercial solvers, again in orders-of-magnitude lower run-times. We also report tests for a on a very large real-world example, which demonstrate that scenario selection analysis can be performed with modest additional computational effort relative to solving the corresponding stochastic programs in which the chance constraints are relaxed.

The use of chance constraints gives the modeler a chance to express the idea that constraints need not be satisfied or costs minimized across every conceivable eventuality. Efficient and scalable heuristics for scenario selection allow end-users to quickly explore risk-reward trade-offs. Further, such heuristics may be effectively leveraged to significantly accelerate exact algorithms, by providing initial near-optimal incumbents.

Acknowledgement

Sandia is a multipurpose laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

References

Birge, J.R., F. Louveaux. 1997. *Introduction to Stochastic Programming*. Springer.

Ermoliev, Y., V. Norkin, R.J-B. Wets. 1995. The minimization of discontinuous functions via averaged functions. *SIAM Journal on Control and Optimization* **33** 149–167.

Holton, G.A. 2003. *Value-at-Risk: Theory and Practice*. Academic Press.

ILOG. 1997. ILOG CPLEX 10.1 solver engine. www.ilog.com/products/cplex.

Luedtke, J., S. Ahmed. 2007. A sample approximation approach for optimization with probabilistic constraints. Tech. rep., Georgia Institute of Technology.

Nemirovski, A., A. Shapiro. 2006. Convex approximations of chance constrained programs. *SIAM Journal on Optimization* **17** 969–996.

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

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

Salinetti, G. 1983. Approximations for chance-constrained programming problems. *Stochastics* **10** 157–179.

Savage, E.L., L.W. Schruben, E. Yucesan. 2005. On the generality of event-graph models. *INFORMS Journal on Computing* **17**(3–9).

Watson, J-P., D.L. Woodruff, D.R. Strip. 2007. Progressive hedging innovations for a stochastic spare parts support enterprise problem. Tech. Rep. SAND-2007-3722J, Sandia National Laboratories, Albuquerque, New Mexico.