SIAM J. Optim.
Vol. 21, No. 2, pp. 517–544

INEXACT BUNDLE METHODS FOR TWO-STAGE STOCHASTIC PROGRAMMING*

WELINGTON OLIVEIRA†, CLAUDIA SAGASTIZÁBAL‡, AND SUSANA SCHEIMBERG§

Abstract. Stochastic programming problems arise in many practical situations. In general, the deterministic equivalents of these problems can be very large and may not be solvable directly by general-purpose optimization approaches. For the particular case of two-stage stochastic programs, we consider decomposition approaches akin to a regularized L-shaped method that can handle inexactness in the subproblem solution. From a nonsmooth optimization perspective, these variants amount to applying a proximal bundle method to an oracle that gives inaccurate values for the objective function and a subgradient. Rather than forcing early termination of the subproblem optimization to define inexact oracles, we select a small subset of scenarios for which the subproblem solution is exact, and we replace the information for the remaining scenarios by a fast procedure that does not involve solving an optimization problem. The inaccurate oracle information creates inexact cuts in the master program, which are well handled by the recently introduced inexact bundle methods. The proposed approaches are validated by encouraging numerical results on several two-stage stochastic linear programs found in the literature.

Key words. two-stage stochastic linear programs, bundle methods, inexact cuts

AMS subject classifications. 90C15, 90C25

DOI. 10.1137/100808289

1. Introduction.

Solving real-life optimization problems often requires a trade-off between modeling and numerical tractability: the more details are brought into the model, the harder becomes the optimization problem. This is particularly true when modeling uncertainty, a crucial matter for decision makers wanting to hedge risk. When adopting a scenario representation for uncertainty, it is desirable to use as many scenarios as possible, to ensure an accurate representation of the underlying stochastic process. But having many scenarios leads to a combinatorial explosion, especially in the multistage case. For this reason, often a choice has to be made between two bad options: either uncertainty is represented roughly and the optimization problem is solved exactly; or a fine discretization is used for the stochastic process and the optimal solution is coarse.

In this work we focus on decomposition strategies for two-stage stochastic programs yielding a good compromise between accuracy in the uncertainty representation and resolvability. Frequently, such problems are large-scale, with block-diagonal structure suitable for decomposition techniques such as Dantzig–Wolfe or Benders decomposition, [3, Chap. 11.1]. The L-shaped method [31], in particular, can be seen as a Benders decomposition of two-stage stochastic linear programs. To derive our decomposition approach we adopt a nonsmooth optimization point of view and revisit the L-shaped

*Received by the editors September 13, 2010; accepted for publication (in revised form) March 28, 2011; published electronically June 29, 2011.

http://www.siam.org/journals/siopt/21-2/80828.html

†COPPE/Sistemas, Universidade Federal do Rio de Janeiro (UFRJ), 21941-972 Rio de Janeiro, Brazil (wlo@cos.ufrj.br). This author was partially supported by PESC, UFRJ's Ph.D. program on Systems Engineering and Computer Science, which made it possible for him to attend international meetings and conferences during his doctoral studies.

‡CEPEL, Electric Energy Research Center. On leave from INRIA Rocquencourt, France (sagastiz@impa.br). This author's research was supported by CNPq grant 383066/2004-2.

§Instituto de Matemática, COPPE/Sistemas, Universidade Federal do Rio de Janeiro (UFRJ) 21941-972 Rio de Janeiro, Brazil (susana@cos.ufrj.br).

method. In this setting, the L-shaped master program is nothing but one iteration of the cutting-plane method for minimizing a nonsmooth convex function f, obtained by summing the first-stage objective and the recourse function. Usually, the recourse function is difficult to evaluate, because it involves computing a multidimensional integral, and, hence, many second-stage optimization subproblems need to be solved.

The cutting-plane algorithm, introduced by [18], [4], defines new iterates on the basis of knowing, for each given point x, the function value $f(x)$ and a subgradient $g(x) \in \partial f(x)$. We refer to this knowledge as black-box or *oracle* information; see [3, Chap. 9.3]. Although more reliable than a subgradient algorithm, the cutting-plane method has two known drawbacks. First, it suffers from a tailing-off effect that makes calculations unstable as the iteration process progresses. Second, since there is no theoretical background allowing the removal of cuts, the optimization problem giving a new iterate has more and more constraints and, hence, gets more and more ill-conditioned and difficult to solve. These two handicaps are addressed by bundle methods [17, Vol. II], which can be seen as stabilized variants of the cutting-plane algorithm that remain convergent even after dropping cuts.

The regularized decomposition [27] exploits bundle-method ideas in a stochastic programming setting. Our method differs from the above in one important point: instead of basing iterations in exact cuts, that is, in linearizations of the form $f(x^k) + (\cdot - x^k)^\top g(x^k)$, we consider *inexact cuts* of the form $f_x^k + (\cdot - x^k)^\top g_x^k$. The values f_x^k and g_x^k are provided by an *inexact oracle*, estimating the exact function and subgradient evaluation with inaccuracy bounded by a constant $\epsilon > 0$. The inexact proximal bundle method introduced in [20] asymptotically finds points that are ϵ-optimal with a mechanism close to classical bundle methods, except for a *noise attenuation step*, detecting when the inaccuracy in the cuts becomes too cumbersome.

The interest of using a method capable of handling inexact oracles in stochastic programming lies in the possibility of reducing the time spent in solving the second-stage subproblems that define the recourse function. This feature was already exploited in [33] for Benders decomposition; but, being a cutting-plane method, the approach exhibits all the drawbacks mentioned above. In addition, the inexact oracle in [33] is defined by terminating the solution of the subproblems before optimality. A similar strategy was considered in [10] in the framework of a "level decomposition." This is a bundle-like method that estimates the recourse function by clustering scenarios into a smaller group of barycenters. Instead, we define oracles that are inexact because only a (small) subset of subproblems is solved. The missing oracle information is replaced by a fast procedure that does not involve an optimization problem and satisfies natural conditions, given in (3.3) below, required for convergence of the inexact proximal bundle method.

Inexact oracles have also been explored by the stochastic decomposition methods [16], starting with the inexact subgradient method in [1]. The regularized stochastic decomposition [15], [32] is a bundle-like algorithm using randomly generated observations to construct cuts. From the perspective considered in this paper, the regularized stochastic decomposition can be interpreted as a "randomized" bundle method using an inexact oracle that randomly selects one scenario for exact subproblem solution. Because randomization is the basis of stochastic decomposition, inexact cuts satisfy statistical estimates that, with probability one, epi-converge to the function f for a subsequence of certain *incumbent* points.

Although incumbent points are the serious iterates in bundle methods (cf. section 3), our proposal differs from the regularized stochastic decomposition from its very conception. Namely, rather than drawing scenarios from an infinite set, we consider that un-

certainty has a large, but finite, support, and we study different alternatives to build inexact cuts in a manner that inaccuracy in the solution can be kept controlled. Instead of randomly choosing scenarios for which subproblems will be solved exactly, we select them by means of some proximity criterion that allows us to group similar scenarios in a manner consistent with the convergence assumptions of the inexact bundle method. Since such assumptions are rather mild, many grouping strategies are possible. In section 4.1, a *collinearity* selection, specifically tailored for second-stage linear programs, is presented. Another strategy, of *scenario selection* (related to [7]; see also [13], [25]), suitable for second-stage convex subproblems, is given in section 4.2.

The remainder of the paper is organized as follows. Section 2 gives the two-stage stochastic linear programming setting considered throughout and fixes the notation and main assumptions (the approach can be applied to convex two-stage stochastic problems; we take a linear setting for simplicity). Section 3 lays the background of the proximal inexact bundle method used for solving the master program, including convergence results. Section 4, with the inexact oracles, describes the collinearity and scenario selection strategies, as well as their respective ways of replacing the missing oracle information. Section 5 reports some encouraging preliminary numerical results, obtained by running the algorithm on a collection of different stochastic programs found in the literature. The final section contains some concluding remarks.

2. General setting. We follow the notation in [29], mainly section 2.1 therein, devoted to two-stage stochastic linear programming problems. More precisely, given an $m_1 \times n_1$ matrix A and an m_1-dimensional vector b, there is a first-stage problem

$$\begin{cases} \min & c^\top x + \mathbb{E}[Q(x;\xi)] \\ \text{s.t.} & x \in X = \{x \in \mathfrak{R}^{n_1} \colon Ax = b, x \geq 0\}. \end{cases} \tag{2.1}$$

The expectation in the objective function is taken with respect to the probability distribution of random second-stage data over the optimal values of the second-stage problem

$$Q(x;\xi) = \begin{cases} \min_{y \in \mathfrak{R}^{n_2}} & q^\top y \\ \text{s.t.} & Tx + Wy = h, \quad y \geq 0, \end{cases} \tag{2.2}$$

depending on given x and on the random data ξ, which is shortened by $\xi := (q, T, W, h) \in \mathfrak{R}^{n_\xi}$. In the second-stage feasible set, matrices T and W are $m_2 \times n_1$ and $m_2 \times n_2$, respectively, while h is an m_2-dimensional vector.

The second-stage problem (2.2) is a linear program, with dual problem given by

$$\begin{cases} \max_u & u^\top (h - Tx) \\ \text{s.t.} & W^\top u \leq q. \end{cases} \tag{2.3}$$

The function Q is measurable and, by [29, Prop. 2.1], for any given ξ the convex function $Q(\cdot;\xi)$ is polyhedral if the primal and dual feasible sets (in (2.2) and (2.3), respectively) are nonempty. It is also shown in [29, Prop. 2.2] that, for any given x^0 and ξ such that $Q(x^0;\xi)$ is finite, the function $Q(\cdot;\xi)$ is subdifferentiable at x^0 with

$$\partial Q(x^0;\xi) = -T^\top \mathcal{D}(x^0;\xi), \quad \text{where } \mathcal{D}(x^0;\xi) = \arg\max\{u^\top (h - Tx^0) \colon W^\top u \leq q\} \tag{2.4}$$

is the set of solutions to the dual problem (2.3).

The probability distribution of the random vector ξ can be either continuous or discrete. In the former case, a discrete approximation is needed; we suppose that there are finitely many realizations ξ_i, each one with probability p_i for $i = 1, \ldots, N$. Thus, the expectation in (2.1) has the expression

$$\mathcal{Q}(x) := \mathbb{E}[Q(x;\xi)] = \sum_{i=1}^{N} p_i Q(x;\xi_i).$$

Since each scenario ξ_i corresponds to some (q_i, T_i, W_i, h_i), the recourse function above is separable along scenarios:

$$\mathcal{Q}(x) = \sum_{i=1}^{N} p_i Q_i(x) \text{ for } Q_i(x) := Q(x;\xi_i) = \begin{cases} \min\limits_{y \in \mathfrak{R}^{n_2}} & q_i^\top y \\ \text{s.t.} & T_i x + W_i y = h_i, \\ & y \geq 0. \end{cases} \tag{2.5}$$

As a result, to compute the expected recourse function $\mathcal{Q}(x)$ for a given $x \in X$, it is necessary to solve N linear programs, which may become too time consuming for large N. Like the function Q, the expected recourse function $\mathcal{Q}$ is convex and has a finite value at a given x if for each $i = 1, \ldots, N$ the problems above have nonempty primal and dual feasible sets. As a result, whenever $\mathrm{dom}\mathcal{Q}$ is nonempty, the function is polyhedral and subdifferentiable with $\partial\mathcal{Q}(x) = \sum_{i=1}^{N} p_i \partial Q(x;\xi_i)$; see [29, Prop. 2.3]. Together with (2.4), a solution $\bar{x}$ to (2.1)–(2.2) is characterized by the inclusion

$$0 \in c - \sum_{i=1}^{N} p_i T_i^\top \mathcal{D}_i(\bar{x}) + \mathcal{N}_X(\bar{x}),$$

where we used the short notation $\mathcal{D}_i(\bar{x}) = \mathcal{D}(\bar{x};\xi_i)$ for the set of dual solutions defined in (2.4) and where $\mathcal{N}_X(\bar{x})$ is the normal cone of convex analysis.

As shown by Example 2.5 in [29], discretizing the support of the random variable ξ into a finite number of scenarios may lead to instability and ill-posedness if the recourse matrix W is random. For this reason, we focus on problems with *fixed recourse*, i.e., such that for any ξ_i, the recourse matrix is fixed: $W_i = W$. In addition, we suppose that the property of *relatively complete recourse* is satisfied: for all $x \in X$ the second-stage feasible sets in (2.2) are nonempty with probability 1. By [29, sect. 2.1.3], when $W_i = W$ the relatively complete recourse property is equivalent to $Q(x;\xi) < +\infty$ for all ξ defined in its support set Ξ. If, in addition, each component of the random vector ξ has finite variance, by Proposition 2.7 in [29], the expected recourse function $\mathcal{Q}(x)$ is well defined, proper, convex, lower semicontinuous, and Lipschitz continuous on its domain with $\mathcal{Q}(x) > -\infty$ for all $x \in \mathfrak{R}^{n_1}$.

In view of the above, in order to avoid dealing with feasibility cuts, duality gaps, and/or infinite values, throughout we assume the following.

Assumption 1.

— The feasible set of the first-stage problem (2.1) is nonempty;
— The two-stage problem (2.1)–(2.2) has fixed recourse;
— The recourse is relatively complete; and
— The random vector $\xi = (q, T, h)$ has finite variance.

The first-stage problem (2.1) has a nonsmooth convex objective function and affine constraints. In this setting, an algorithm dealing with noncontinuous derivatives should be applied. This is the subject of the next section.

3. Inexact bundle methods. Bundle methods are well known by their robustness and reliability for solving nonsmooth convex problems on the basis of an *oracle* that, for each given point x, computes the value of the function and one subgradient. For the case of interest, the nonsmooth objective function in (2.1) is

$$f(x) \coloneqq c^\top x + \mathcal{Q}(x) \quad \text{with } \mathcal{Q}(x) \text{ from (2.5).} \tag{3.1}$$

Therefore, to obtain each value function $Q_i(x)$, the oracle needs to solve, for each $i = 1, \ldots, N$, the linear program in (2.5) (or its dual problem). These computations also give a subgradient $-T_i^\top u_i$ with $u_i \in \mathcal{D}_i(x) = \arg\max\{u^\top(h_i - T_i x)\colon W_i^\top u \le q_i\}$. Then, for each given $x \in \operatorname{dom}\mathcal{Q} \cap X$,

$$\text{the oracle gives} \begin{cases} \text{the function} & f(x) = c^\top x + \sum_{i=1}^N p_i Q_i(x), \\ \text{the subgradient} & g(x) = c - \sum_{i=1}^N p_i T_i^\top u_i. \end{cases} \tag{3.2}$$

Since we are interested in considering a large number of scenarios, the number of terms in the summations above is large and, hence, many linear programs need to be solved. In order to save computational time, for some scenarios one can either solve (2.5) only approximately (as in [33] and [10]), or just skip the solution of (2.5), replacing the missing information by some sound value. From a nonsmooth point of view, these modifications amount to having an *inexact* evaluation of the function and the subgradient. Essentially, instead of (3.2), the available information can be inaccurate, in the sense that

$$\text{the oracle gives} \begin{cases} \text{a function estimate} & f_x \in [f(x) - \epsilon_f, f(x) + \epsilon_g], \\ \text{a subgradient estimate} & g_x \in \partial_{\epsilon_f + \epsilon_g} f(x), \end{cases} \tag{3.3}$$

and where the errors $\epsilon_f \ge 0$ and $\epsilon_g \ge 0$ are unknown, but bounded.

The method in [33] stops the optimization process solving the linear programs (2.5) when some accuracy, say ϵ_{acc}, is reached. In this case, $f_x \le f(x)$, and (3.3) holds with $\epsilon_f = \epsilon_{\text{acc}}$ and $\epsilon_g = 0$. This is not the only alternative; we will see in section 4 different inexact oracles, obtained by exploiting structural properties of problems (2.5), for which (3.3) holds with $\epsilon_g > 0$.

We now give the main elements of a bundle method capable of handling inaccuracies, introduced in [20], for the unconstrained minimization of the function $f + i_X$, where the second term is the indicator function of the set X.

The method generates a sequence of *trial* points $\{z^k\} \subset X$ and uses the information given by an inexact oracle satisfying (3.3) to build *approximate linearizations* of the form

$$f_k(\cdot) \coloneqq f_z^k + (\cdot - z^k)^\top g^k \quad \text{with} \quad f_z^k \coloneqq f_{z^k} \quad \text{and} \quad g^k \coloneqq g_{z^k}.$$

These linearizations define at iteration k a polyhedral *cutting-plane model* of f:

$$\check{f}_k(\cdot) \coloneqq \max_{j \in J^k} f_j(\cdot) \quad \text{with} \quad J^k \subseteq \{1, \ldots, k\}.$$

Recalling that $\partial_\epsilon f(x) \coloneqq \{g\colon f(\cdot) \ge f(x) + (\cdot - x)^\top g - \epsilon\}$, from (3.3) we see that $f_k \le f + \epsilon_f + 2\epsilon_g$ and, hence,

$$\check{f}_k(\cdot) \le f(\cdot) + \varepsilon + \epsilon_g, \quad \text{where we defined} \quad \varepsilon := \epsilon_f + \epsilon_g. \tag{3.4}$$

The cutting-plane model, together with the *serious-point* x^k, corresponding to some "good" past iterate, is used to define the next trial point:

$$z^{k+1} := \arg\ \min \boldsymbol{\phi}_k(z) \quad \text{with} \quad \boldsymbol{\phi}_k(\cdot) := \check{f}_k(\cdot) + i_X(\cdot) + \frac{1}{2t_k}\|\cdot - x^k\|^2. \tag{3.5}$$

Since the set X is polyhedral, the quadratic programming problem (3.5) is easy to solve. The *stepsize* t_k, controlling the weight of the quadratic term in the objective function, is adjusted along iterations, for example, using the procedure given in [19, sect. 2], noting that [19, eq. (2.17)] is replaced by [20, eq. (4.5)] written with $R = 1$.

Due to inaccuracies, the model no longer approximates the function from below. In particular, even though $\boldsymbol{\phi}_k(z^{k+1}) \le \check{f}_k(x^k)$ by (3.5), we can have

$$f_x^k - \boldsymbol{\phi}_k(z^{k+1}) < 0. \tag{3.6}$$

Due to the fact that f is convex, such behavior can only come from the oracle inexactness, and the method checks satisfaction of (3.6) to detect when inaccuracy is becoming cumbersome. In this case, increasing the stepsize t_k makes the next trial point get closer to the serious-point x^k and, eventually, decreases the value of $\boldsymbol{\phi}_k(z^{k+1})$. The stepsize is increased until obtaining $f_x^k - \boldsymbol{\phi}_k(z^{k+1}) \ge 0$, after which the iteration proceeds as in a standard bundle method. This procedure is called *noise attenuation*.

A trial point is declared a *serious-point* ($x^{k+1} := z^{k+1}$) when

$$f_z^{k+1} \le f_x^k - \boldsymbol{\kappa} v_k \quad \text{with} \quad v_k := f_x^k - \check{f}_k(z^{k+1}) \quad \text{for a fixed } \boldsymbol{\kappa} \in (0,1).$$

Otherwise, the iteration is declared *null*.

The cardinality of sets J^k, defining the bundle size, can be kept bounded as follows. At every iteration, the trial point oracle information enters the bundle: $J^{k+1} \subset \{k+1\}$. In addition, by the optimality condition for (3.5),

$$z^{k+1} = x^k - t_k(p_f^k + p_X^k) \quad \text{with} \quad \begin{cases} p_f^k := \sum_{j\in J^k} \nu_j^k g^j & \in \partial \check{f}_k(z^{k+1}), \\ p_X^k := -\frac{z^{k+1}-x^k}{t_k} - p_f^k & \in \partial i_X(z^{k+1}). \end{cases} \tag{3.7}$$

The simplicial multiplier ν^k satisfies the relations

$$\sum_{j\in J^k} \nu_j^k = 1, \qquad \nu_j^k \ge 0, \quad \text{and} \quad \nu_j^k[\check{f}_k(z^{k+1}) - f_j(z^{k+1})] = 0 \quad \text{for all } j \in J^k,$$

and it can be used to save storage without impairing convergence. More precisely, "inactive" indices, corresponding $\nu_j^k = 0$, can be dropped: $J^{k+1} \subset \{j \in J^k\colon \nu_j^k \ne 0\}$.

Finally, by (3.4), the *aggregate subgradient* and the *aggregate linearization error*

$$p^k := p_f^k + p_X^k \quad \text{and} \quad \alpha_k := v_k - t_k\|p_k\|^2 \tag{3.8}$$

give an *optimality estimate*:

$$f_x^k \le f(x) + \epsilon_f + 2\epsilon_g + \|p^k\|\|x - x^k\| + \alpha_k \quad \text{for all } x \in X,$$

or, equivalently, $f_x^k \le f(x) + \epsilon_f + 2\epsilon_g + V_k(\|x - x^k\| + 1)$ for all $x \in X$, where

$$V_k := \max\{\|p^k\|, \alpha_k\} \tag{3.9}$$

is an *optimality measure*.

As a result, when the optimality measure V_k is equal to zero, x^k can be considered 2ε-optimal. More precisely, if $f_* := \inf_X f$ denotes the optimal value in (2.1), then $f(x^k) - f_* \le 2\varepsilon = 2(\epsilon_f + \epsilon_g)$.

Algorithm 1 (inexact bundle method).

Step 0 (initialization). Select $x^1 \in X$, a stopping tolerance $\epsilon_V \ge 0$, a descent parameter $\kappa \in (0,1)$, a stepsize bound $\tau_1 > 0$, and a stepsize $t_1 \in (0, \tau_1)$. Set $z^1 := x^1$, $f_x^1 := f_z^1 := f_{z^1}$, $g^1 := g_{z^1}$, $J^1 := \{1\}$, $i_t^1 := 0$, $k := k(0) := 1$, $l := 0$ ($k(l) - 1$ will denote the iteration of the lth descent step).

Step 1 (trial point finding). Find z^{k+1} solving (3.5), and let ν^k denote the corresponding optimal simplicial multiplier. Compute V_k, v_k, and α_k.

Step 2 (stopping criterion). If $V_k \le \epsilon_V$, stop.

Step 3 (inaccuracy detection). If $v_k < -\alpha_k$, set $t_k := 10t_k$, $\tau_k := \max\{\tau_k, t_k\}$, $i_t^k := k$ and loop back to step 1; else set $\tau_{k+1} = \tau_k$.

Step 4 (oracle call and descent test). Call the inexact oracle (3.3) to compute f_z^{k+1} and g^{k+1}. If the descent test

$$f_z^{k+1} \le f_x^k - \kappa v_k$$

holds, then declare the iterate serious: set $x^{k+1} := z^{k+1}$, $f_x^{k+1} := f_z^{k+1}$, $i_t^{k+1} := 0$, $k(l+1) := k+1$ and increase l by 1. Otherwise, declare the iterate null: set $x^{k+1} := x^k$, $f_x^{k+1} := f_x^k$, and $i_t^{k+1} := i_t^k$.

Step 5 (bundle management). Choose $J^{k+1} \supseteq \{j \in J^k : \nu_j^k \ne 0\} \cup \{k+1\}$.

Step 6 (stepsize updating and loop). If the iterate was declared serious, select $t_{k+1} \in [t_k, \tau_{k+1}]$. If the iterate was declared null, either set $t_{k+1} := t_k$, or choose $t_{k+1} \in [0.1t_k, t_k]$ if $i_t^{k+1} = 0$ and $V_k \le f_x^k - f_{k+1}(x^k)$.

In all cases increase k by 1 and go to Step 1.

The inaccuracy detection inequality $v_k < -\alpha_k$ at Step 3 implies (3.6). This can be seen by (3.7) and (3.8) together with the definitions of the predicted descent v_k.

For Algorithm 1 to be convergent, the boundedness assumption of the oracle inaccuracy is crucial, as shown by the result below.

Lemma 3.1 (consequences of bounded inaccuracy in the oracle). *For an inexact oracle* (3.3) *and the definitions above, the following holds:*

(i) *At each iteration the linearization error is bounded below by the inaccuracy:*

$$\alpha_k = v_k - t_k\|p_k\|^2 \ge -2\varepsilon = -2(\epsilon_f + \epsilon_g).$$

(ii) *For each iteration k detecting noise at Step* 3*, the optimality measure satisfies*

$$0 \le V_k = \max\{[(v_k - \alpha_k)/t_k]^{1/2}, \alpha_k\} < (-2\alpha_k/t_k)^{1/2} \le (4\varepsilon/t_k)^{1/2}.$$

Proof. To show item (i), first note that, for the current serious-point x^k and the iterate z^{k+1} given by (3.7),

$$\alpha_k = v_k - t_k p_k^\top p_k = v_k - p_k^\top (x^k - z^{k+1}) = f_x^k - \check{f}_k(z^{k+1}) - p_k^\top (x^k - z^{k+1}).$$

Also from (3.7), $p_k \in \partial\{\check{f}_k(z^{k+1}) + i_X(z^{k+1})\}$, so $\check{f}_k(x^k) \geq \check{f}_k(z^{k+1}) + p_k^\top (x^k - z^{k+1})$, because both x^k and $z^{k+1} \in X$. This means that $\alpha_k \geq f_x^k - \check{f}_k(x^k)$ and, together with (3.3) and (3.4), we see that

$$\alpha_k \geq f_x^k - \check{f}_k(x^k) \geq f_x^k - f(x^k) - \epsilon_f - 2\epsilon_g \geq f(x^k) - \epsilon_f - f(x^k) - \epsilon_f - 2\epsilon_g = -2\varepsilon,$$

as desired. To show the second item, notice that each iteration k detecting noise at Step 3 satisfies $v_k < -\alpha_k$. Since, by (3.8), $v_k \geq \alpha_k$, we conclude that $\alpha_k \leq 0$. Together with (3.8), (3.9), and item (i), the result follows. ∎

When the tolerance ϵ_V is set to zero, either the algorithm terminates having found an approximate minimizer (because $V_k = 0$), or it loops forever. If at some iteration k an infinite loop occurs between Steps 1–3 to cope with inaccuracy, then x^k is already 2ε-optimal. More precisely, since an infinite loop between Steps 1 and 3 drives both τ_k and t_k to infinity, when ε is bounded, we have that $V_k \to 0$ by item (ii) in Lemma 3.1.

The remaining case refers to Algorithm 1 generating infinitely many oracle values f_z^{k+1}, g^{k+1} in Step 4 (i.e., infinitely many serious and/or null iterates), and it is dealt with by the following result.

Theorem 3.9 in [20]. Let $f_x^\infty := \lim_k f_x^k$ and $f_* := \inf_{x\in X} f(x)$. If Algorithm 1 generates infinitely many serious- and/or null-points, then $f_x^k \downarrow f_x^\infty \leq f_* + \epsilon_f + 2\epsilon_g$. Moreover, $\limsup f(x^k) \leq f_* + 2\varepsilon$ for $\varepsilon = \epsilon_f + \epsilon_g$, so each cluster point x^* of $\{x^k\}$ (if any) satisfies $x^* \in X$ and $f(x^*) \leq f_* + 2\varepsilon$.

This convergence result does not show existence of cluster points. When there is a last serious-point followed by infinitely many null-points, the sequence $\{x^k\}$ is finite and, hence, bounded. If there are infinitely many serious-points, one can suppose that the feasible set X is bounded to guarantee the existence of cluster points, or that the function f is coercive. For more details on the algorithm, we refer the reader to [20]; see also [21], [8].

4. Defining inexact oracles. We now consider three different inexact oracles, satisfying (3.3), devised according to the following strategy:

- By some suitable criterion, a subset of scenarios I_{LP}, with cardinality $|I_{\mathrm{LP}}|$, is selected. For each $i \in I_{\mathrm{LP}}$ a linear programming problem related to (2.5) is solved.
- For the $N - |I_{\mathrm{LP}}|$ remaining scenarios, the missing oracle information is replaced by a fast procedure satisfying (3.3), without solving an optimization problem.

Each iteration in Algorithm 1 calls the inexact oracle at Step 4. For this reason, the set I_{LP} above may vary with k. In particular, if $|I_{\mathrm{LP}}^k| \to N$ as $k \to \infty$, the inexact oracle would become asymptotically exact.

4.1. Collinearity selection. Instead of (2.5), we consider the dual form (2.3) and denote by u_i an optimal solution, recalling that $Q_i(x) = Q_i(x; \xi_i)$ with $\xi_i = (q_i, T_i, h_i)$:

$$\begin{cases} \max_u & u^\top (h_i - T_i x) \\ \text{s.t.} & W^\top u \leq q_i \end{cases} = (h_i - T_i x)^\top u_i. \tag{4.1}$$

Since, in addition, $-T_i^\top u_i \in \partial Q_i(x)$, for the inexact oracle to provide approximate function and subgradient values, it is enough to define an estimate, say ϑ_j, for each dual solution u_j that is not computed.

We distinguish two cases, depending on whether the primal cost is fixed ($q_i = q$ for all $i \le N$) or uncertain.

4.1.1. Second-stage linear problem with fixed cost. When the primal cost is deterministic, $q_i = q$, and the feasible sets in (4.1) are identical for all scenarios. In these circumstances, the oracle for two scenarios i and j will be identical if the corresponding dual cost vectors in (4.1), $h_i - T_i x$ and $h_j - T_j x$, are collinear. Indeed, whenever the angle between these vectors satisfies

$$\cos\,\theta_{ij} = \frac{(h_i - T_i x)^\top (h_j - T_j x)}{\|h_i - T_i x\| \|h_j - T_j x\|} = 1,$$

both problems (4.1), written respectively for i and j, have the same optimal solution: $u_i = u_j$.

The inexact oracle proceeds as follows, detecting pairs of vectors that are approximately collinear.

Inexact Oracle 1 (collinearity strategy, fixed primal cost).

Step 0 (initialization). Let $\epsilon_{\cos} \in (0, 1)$ be a collinearity parameter. For a fixed x, select a nonempty set $I_{\mathrm{LP}} \subseteq \{1, \ldots, N\}$ such that $i,\ l \in I_{\mathrm{LP}} \Rightarrow \cos\,\theta_{il} \le 1 - \epsilon_{\cos}$.

Step 1 (exact calculations). Define $\mathcal{D}_{\mathrm{LP}} = \emptyset$. For each $i \in I_{\mathrm{LP}}$ find u_i solving (4.1) and add u_i to $\mathcal{D}_{\mathrm{LP}}$.

Step 2 (collinearity detection and estimates). For each $j \notin I_{\mathrm{LP}}$: compute $\vartheta_j := \arg\max_{u \in \mathcal{D}_{\mathrm{LP}}} u^\top (h_j - T_j x)$.

Instead of (3.2), the Inexact Oracle 1 provides the estimates

$$\begin{cases} f_x = c^\top x + \sum_{i \in I_{\mathrm{LP}}} p_i u_i^\top (h_i - T_i x) + \sum_{j \notin I_{\mathrm{LP}}} p_j \vartheta_j^\top (h_j - T_j x), \\ g_x = c - \sum_{i \in I_{\mathrm{LP}}} p_i T_i^\top u_i - \sum_{j \notin I_{\mathrm{LP}}} p_j T_j^\top \vartheta_j. \end{cases}$$

To show that the inaccuracy of the Inexact Oracle 1 is bounded, we first give a general bound on the function estimates, which will be useful in what follows.

Lemma 4.1. *Given the nonempty convex polyhedron* $\Pi(q) := \{u:\ W^\top u \le q\}$, *consider the support function*

$$s_q(d) := \max\{u^\top d:\ u \in \Pi(q)\}.$$

There exists a constant $K = K(q, W)$ *such that, for every basic feasible point* $v \in \Pi(q)$, *the relations*

$$s_q(d) - K(q, W)\|d\| \le v^\top d \le s_q(d)$$

hold for any $d \in \Pi := \{d \in \mathbb{R}^{m_2}:\ d^\top u \le 0\ for\ all\ u \in \Pi(0)\}$.

Proof. The domain of the support function is Π, the cone polar to $\Pi(0)$; see, for example, [29, pp. 28–29]. For $d \in \Pi$, the rightmost inequality trivially holds, because $v \in \Pi(q)$. Let $u_d \in \Pi(q)$ be an optimal basic feasible solution so that $s_q(d) = u_d^\top d$. Then,

$$0 \le s_q(d) - v^\top d \le \|u_d - v\| \|d\|.$$

Since both v and u_d are basic feasible points, there exist primal feasible bases B_v and B_u of the convex polyhedron $\Pi(q)$ such that $v = B_v^{-1}q$ and $u_d = B_u^{-1}q$. Since the set $\Pi(q)$ is nonempty, there is a finite number of such bases, and the result follows. ∎

We now show that the inaccuracy of the Inexact Oracle 1 satisfies (3.3).

PROPOSITION 4.1. *Let* (2.1)–(2.2) *be such that the cost q is fixed for all scenarios and Assumption* 1 *holds. Suppose that the linear programming solver used in Step* 1 *of the Inexact Oracle* 1 *finds basic optimal solutions.*

If X is bounded, so is the inaccuracy, and the inexact oracle satisfies (3.3) *with $\epsilon_g = 0$.*

Proof. Let $x \in X$ and $j \in \{1, \ldots, N\} \setminus I_{\mathrm{LP}}$ be given. Since the recourse is relatively complete, the functional value $Q_j(x) = s_q(h_j - T_j x)$ is finite. The assumption on the linear programming solver implies that $v = \vartheta_j$ is a basic feasible point in $\Pi(q)$. Therefore, letting $d = h_j - T_j x$ in Lemma 4.1, we have that

$$(0 \leq)\epsilon_j := Q_j(x) - \vartheta_j^\top (h_j - T_j x) \leq K(W, q)\|h_j - T_j x\|.$$

The assumption of fixed recourse implies that $K = K(W, q)$ does not depend on j, and the finite variance condition in Assumption 1 gives a uniform bound for $\|h_j - T_j x\|$. Since N is finite and X is bounded, $\epsilon_j \leq \epsilon_f$ for some constant ϵ_f.

To show that $g_x \in \partial_{\epsilon_f} f(x)$ and that (3.3) holds with $\epsilon_g = 0$, consider problem (4.1) written with x replaced by $z \in X$, and let u_j' be an optimal solution of this problem. Since both the recourse and the cost of the second-stage problem are fixed, we obtain that

$$\begin{aligned}
Q_j(z) &= (h_j - T_j z)^\top u_j' \geq (h_j - T_j z)^\top \vartheta_j \\
&= Q_j(x) - z^\top T_j^\top \vartheta_j + x^\top T_j^\top \vartheta_j - [Q_j(x) + x^\top T_j^\top \vartheta_j - h_j^\top \vartheta_j] \\
&= Q_j(x) - (z - x)^\top T_j^\top \vartheta_j - [Q_j(x) - (h_j - T_j x)^\top \vartheta_j] \\
&= Q_j(x) - (z - x)^\top T_j^\top \vartheta_j - \epsilon_j.
\end{aligned}$$

By (2.4), $-u_i^\top T_i \in \partial Q_i(z)$ for $i \in I_{\mathrm{LP}}$ and, hence,

$$\begin{aligned}
f(z) &= c^\top z + \sum_{i=1}^{N} p_i Q_i(z) \\
&= c^\top z + \sum_{i \in I_{\mathrm{LP}}} p_i Q_i(z) + \sum_{j \notin I_{\mathrm{LP}}} p_j Q_j(z) \\
&\geq c^\top z + \sum_{i \in I_{\mathrm{LP}}} p_i [Q_i(x) - u_i^\top T_i (z - x)] + \sum_{j \notin I_{\mathrm{LP}}} p_j [Q_j(x) - \vartheta_j^\top T_j (z - x) - \epsilon_j] \\
&= c^\top x + \sum_{i=1}^{N} p_i Q_i(x) - \sum_{j \notin I_{\mathrm{LP}}} p_j \epsilon_j + \left[c^\top - \sum_{i \in I_{\mathrm{LP}}} p_i u_i^\top T_i - \sum_{j \notin I_{\mathrm{LP}}} p_j \vartheta_j^\top T_j \right] (z - x) \\
&= f(x) - \sum_{j \notin I_{\mathrm{LP}}} p_j \epsilon_j + g_x^\top (z - x).
\end{aligned}$$

Therefore, $f(z) \geq f(x) - \epsilon_f + g_x^\top (z - x)$, as desired. Finally, the algebraic manipulations

$$\begin{aligned} f(x) &= c^\top x + \sum_{i \in I_{\mathrm{LP}}} p_i Q_i(x) + \sum_{j \notin I_{\mathrm{LP}}} p_j Q_j(x) \\ &= c^\top x + \sum_{i \in I_{\mathrm{LP}}} p_i Q_i(x) + \sum_{j \notin I_{\mathrm{LP}}} p_j \vartheta_j^\top (h_j - T_j x) + \sum_{j \notin I_{\mathrm{LP}}} p_j [Q_j(x) - \vartheta_j^\top (h_j - T_j x)] \\ &= f_x + \sum_{j \notin I_{\mathrm{LP}}} p_j \epsilon_j \end{aligned}$$

imply that $f_x \in [f(x) - \epsilon_f, f(x)]$, and the proof is complete. ∎

4.1.2. Second-stage linear problem with uncertain cost. When the vector q is also random, the feasible set in (4.1) varies with each scenario i.

In order to save computational time, no exact calculation is done. Instead, for each $i \in I_{\mathrm{LP}}$, we group almost collinear cost vectors $h_j - T_j x$ in a set $\mathcal{J}_i$ and consider a common feasible set, defined by replacing q_j by the average over the set $\mathcal{J}_i$. Since $i \in \mathcal{J}_i$, we solve a linear programming problem with cost vector $h_i - T_i x$ over the average set and use the corresponding solution as a proxy for the remaining scenarios in the group.

Inexact Oracle 2 (collinearity strategy, uncertain primal cost).
Step 0 (initialization). Let $\epsilon_{\cos} \in (0, 1)$ be a collinearity parameter.

For a fixed x, select a set $I_{\mathrm{LP}} \subseteq \{1, \ldots, N\}$ such that $i, l \in I_{\mathrm{LP}} \Rightarrow \cos \theta_{il} \leq 1 - \epsilon_{\cos}$.
Step 1 (collinearity grouping). For each $i \in I_{\mathrm{LP}}$: set $\mathcal{J}_i := \{i\} \cup \{j \notin I_{\mathrm{LP}}: \cos \theta_{ij} > 1 - \epsilon_{\cos}\}$.
(Step 2) (estimates). For each $i \in I_{\mathrm{LP}}$: find $\bar{u}_i$ solving

$$\begin{cases} \max_u & u^\top (h_i - T_i x) \\ \text{s.t.} & W^\top u \leq \bar{q}_i := \dfrac{\sum_{j \in \mathcal{J}_i} p_j q_j}{\sum_{\ell \in \mathcal{J}_i} p_\ell}. \end{cases}$$

For each $j \in \mathcal{J}_i$, set $\vartheta_j = \bar{u}_i$.

The Inexact Oracle 2 provides the estimates

$$\begin{bmatrix} f_x = c^\top x + \sum_{i \in I_{\mathrm{LP}}} p_i \bar{u}_i^\top (h_i - T_i x) + \sum_{j \notin I_{\mathrm{LP}}} p_j \vartheta_j^\top (h_j - T_j x), \\ g_x = c - \sum_{i \in I_{\mathrm{LP}}} p_i T_i^\top \bar{u}_i - \sum_{j \notin I_{\mathrm{LP}}} p_j T_j^\top \vartheta_j. \end{bmatrix}$$

When the costs are deterministic ($q = q_i$), the estimates above for $i \in I_{\mathrm{LP}}$ are exact and coincide with the solutions computed in Step 2 of the Inexact Oracle 1. Such is not the case for $j \notin I_{\mathrm{LP}}$, because there is no search in the set of previously computed vertices $\mathcal{D}_{\mathrm{LP}}$. For this reason, even when $q = q_i$, this inexact oracle does not subsume the Inexact Oracle 1.

We now show that (3.3) is satisfied.

Proposition 4.2. *Let* (2.1)–(2.2) *satisfy Assumption* 1 *and suppose the Inexact Oracle* 2 *is employed. Suppose that the linear programming solver used at Step* 2 *of the Inexact Oracle* 2 *finds basic optimal solutions.*

If X is bounded, so is the inaccuracy, and (3.3) *holds with $\epsilon_f = \epsilon_g > 0$.*

Proof. In a manner similar to Lemma 4.1, we bound the difference between the value functions $Q_j(x)$ and their estimates. More precisely, letting $d_j := h_j - T_j x$, by construction,

if $j \in I_{\mathrm{LP}}$, $Q_j(x)$ is replaced by $s_{\bar{q}_j}(d_j)$;
if $j \notin I_{\mathrm{LP}}$, then $j \in \mathcal{J}_i$ and $Q_j(x)$ is replaced by $\bar{u}_i^\top d_j$, where $s_{\bar{q}_j}(d_i) = \bar{u}_i^\top d_i$.

Since $Q_j(x) = s_{q_j}(d_j)$, the inaccuracy in the function estimate is given by

$$\epsilon_j = \begin{cases} s_{q_j}(d_j) - s_{\bar{q}_j}(d_j) & \text{if } j \in I_{\mathrm{LP}}, \\ s_{q_j}(d_j) - \bar{u}_i^\top d_j & \text{if } j \notin I_{\mathrm{LP}}, \quad j \in \mathcal{J}_i. \end{cases}$$

Since the recourse is relatively complete, the terms above are all finite. By [23, Thm. 2.4], the feasible set of a linear program is Lipschitzian with respect to right-hand side perturbations, so there is a constant $L(d_j, W)$ such that

$$\epsilon_j \le |s_{q_j}(d_j) - s_{\bar{q}_j}(d_j)| \le L(d_j, W)\|q_j - \bar{q}_j\| \quad \text{if } j \in I_{\mathrm{LP}}. \tag{4.2}$$

When $j \notin I_{\mathrm{LP}}$, $j \in \mathcal{J}_i$, we start by writing

$$\epsilon_j = s_{q_j}(d_j) - \bar{u}_i^\top d_j = s_{q_j}(d_j) - s_{\bar{q}_i}(d_j) + s_{\bar{q}_i}(d_j) - \bar{u}_i^\top d_j =: \Delta_1 + \Delta_2.$$

As in (4.2), $|\Delta_1| \le L(d_j, W)\|q_j - \bar{q}_i\|$. To bound the term Δ_2, note that $d_j \in \Pi^\circ$ because the recourse is relatively complete, and note that $u_i \in \Pi(\bar{q}_i)$ is a vertex, by assumption. As a result, applying Lemma 4.1 together with (4.2), written with $(q, d, v) = (\bar{q}_i, d_j, \bar{u}_i)$,

$$\epsilon_j \le L(d_j, W)\|q_j - \bar{q}_i\| + K(\bar{q}_i, W)\|d_j\| \quad \text{if } j \notin I_{\mathrm{LP}}, \qquad j \in \mathcal{J}_i. \tag{4.3}$$

By boundedness of X and finiteness of both N and the variance of $\xi = (q, T, h)$, there are uniform bounds L, K, and M_d for $L(d_j, W)$, $K(\bar{q}_i, W)$, and $\|d_j\|$, respectively. Letting M_q be a bound for $\|q_j - \bar{q}_j\|$ and $\|q_j - \bar{q}_i\|$, the result follows by taking $\epsilon_f = \epsilon_g = 2LM_q + KM_d$. ∎

The collinearity strategy presented in this section preserves the initial probability distribution of the uncertainty. This is not the case for the two strategies below, based on scenario selection techniques.

4.2. Scenario selection. We consider two strategies based on the *scenario optimal reduction* (SOR) technique developed in [7]. In one variant, the corresponding inexact oracles are progressively more and more accurate as the inexact bundle method progresses along iterations.

The SOR technique makes use of the Fortet–Mourier distance between two probability measures. These measures correspond, respectively, to all the N scenarios and to the fewer scenarios in the set I_{LP}, for which oracle calculations will be exact. Namely, reducing the size of the scenario tree entails redistributing the initial set of probabilities $P := \{p_i, i = 1, \ldots, N\}$ into a reduced set

$$\tilde{P} := \{\tilde{p}_i, i = 1, \ldots, N\} \quad \text{with} \begin{cases} \tilde{p}_i = 0 \quad \text{for } i \notin I_{\mathrm{LP}}, \\ \sum_{i \in I_{\mathrm{LP}}} \tilde{p}_i = 1. \end{cases}$$

The best redistribution from P to $\tilde{P}$ is determined by, first, fixing the cardinality desired for I_{LP}, say n_{LP}, and then solving an optimization problem for finding the best set with such cardinality.

Using linear programming duality arguments, [7, Thm. 2] shows that given a pseudonorm $d(\cdot, \cdot)$ and a set $I \subseteq \{1, \ldots, N\}$, the Monge–Kantorovich functional

$$c(I) = \sum_{j \notin I} p_j \min_{i \in I} d(\xi_j, \xi_i)$$

is an upper bound for the distance between P and the new probability measure $\tilde{P}$, defined by,

$$\text{for each } i \in I, \quad \tilde{p}_i = p_i + \sum_{j \in J_i} p_j, \quad \text{where } J_i := \{j \notin I\colon\ i \in \arg\min_{l \in I} d(\xi_j, \xi_l)\}. \tag{4.4}$$

When $d(\cdot,\cdot)$ is a norm, the Monge–Kantorovich functional coincides with the Fortet–Mourier distance.

By [7, eq. (4) and Thm. 2],

$$\left| \sum_{i=1}^{N} p_i Q(x;\xi_i) - \sum_{i \in I} \tilde{p}_i Q(x;\xi_i) \right| \le c(I). \tag{4.5}$$

As a result, smaller values of $c(I)$ correspond to better sets I. The best set is the one minimizing the function $c(\cdot)$ over all the subsets of $\{1, \ldots, N\}$ with cardinality n_{LP}. But this is a combinatorial (set-covering) problem, so instead we determine I_{LP} by a heuristic method from [12, Alg. 2.4], called fast forward selection.

For more information on the SOR technique, we refer to [7],[11], [12] for two-stage problems and to [22], [26], [13] for the multistage setting.

Inexact Oracle 3 (scenario selection strategy).

Step 0 (initialization). A point x is given, as well as a working set $I_w \subseteq \{1, \ldots, N\}$, a parameter $\epsilon_{\mathrm{MK}} \ge 0$ (or alternatively $n_{\mathrm{LP}} \le N$), and a pseudonorm $d(\cdot,\cdot)$. If $I_w = \emptyset$, compute the pseudodistances $d_{jl} = d(\xi_j, \xi_l)$ for $j, l \in \{1, \ldots, N\}$. Set $z_{l_m} = +\infty$.

Step 1 (fast forward selection). While $z_{l_m} > \epsilon_{\mathrm{MK}}$ (or alternatively $|I_w| < n_{\mathrm{LP}}$): for each $l \notin I_w$

$$\text{compute } z_l = \sum_{j \notin I_w, j \ne l} p_j d_{jl} \quad \text{and select} \quad l_m \in \arg\min_{l \notin I_w} z_l.$$

Set $I_w = I_w \cup \{l_m\}$. For each j, $l \notin I_{\mathrm{LP}}$ recompute the distances $d_{jl} = \min(d_{jl}, d_{jl_m})$.

Step 2 (best set determination and exact calculations). Set $I_{\mathrm{LP}} := I_w$. For each $i \in I_{\mathrm{LP}}$ find u_i solving (4.1).

Step 3 (redistribution and new distances). Compute the new probabilities from (4.4) written with I therein replaced by I_{LP}.

Since N is finite, the best set I_{LP} is always found after a finite number of inner iterations in Step 1. The Inexact Oracle 3 provides the estimates

$$\begin{bmatrix} f_x = c^\top x + \sum_{i \in I_{\mathrm{LP}}} \tilde{p}_i u_i^\top (h_i - T_i x), \\ g_x = c - \sum_{i \in I_{\mathrm{LP}}} \tilde{p}_i T_i^\top u_i. \end{bmatrix}$$

We now show that relation (4.5) ensures satisfaction of (3.3) for this oracle.

PROPOSITION 4.3. *Let* (2.1)–(2.2) *satisfy Assumption* 1*, and suppose the Inexact Oracle* 3 *is employed. Then the inaccuracy is bounded and* (3.3) *holds with* $\epsilon_f = \epsilon_g = c(I_{\mathrm{LP}})$.

Proof. Let u_i be a solution for the problem (2.3) for $\xi = \xi_i$. By Assumption 1, for all $x \in X$ and $i \in I_{\mathrm{LP}}$ the value $Q_i(x) = u_i^\top (h_i - T_i x)$ is finite. In [12, Thm. 2.5] it is shown that $z_{lm} = c(I_w)$, so from (4.5) it follows that $f_x \in [f(x) - c(I_{\mathrm{LP}}), f(x) + c(I_{\mathrm{LP}})]$. To show that $g_x \in \partial_{2c(I_{\mathrm{LP}})} f(x)$ first note that, by definition,

$$f_x + g_x^\top (z - x) = c^\top z + \sum_{i \in I_{\mathrm{LP}}} \tilde{p}_i u_i^\top (h_i - T_i z).$$

Since $W^\top u_i \le q_i$, from (4.1) written with x replaced by z, we see that $u_i^\top (h_i - T_i z) \le Q(z, \xi_i)$ and, hence,

$$f_x + g_x^\top (z - x) \le c^\top z + \sum_{i \in I_{\mathrm{LP}}} \tilde{p}_i Q(z, \xi_i).$$

Relation (4.5) written with $x = z$ implies that the right-hand side is bounded above by $f(z) + c(I_{\mathrm{LP}})$, and the desired relation follows because $f_x \ge f(x) - c(I_{\mathrm{LP}})$. ∎

We comment on two alternatives employed in the numerical results. The oracle call enters the inexact bundle method at Step 4 of Algorithm 1 to compute estimates at the point z^{k+1}. Accordingly, we let the Inexact Oracle 3 parameters vary with the iterate index k. In particular,

$$\text{the stopping tolerance } \epsilon_{\mathrm{MK}}^{k+1} \begin{cases} \text{is kept constant at null-points and} \\ \text{strictly decreases after a serious-point,} \end{cases}$$

knowing that ϵ_{MK}^1 is set at the initialization Step 0. As for the working set, we let $I_w^1 = I_{\mathrm{LP}}^1 = \emptyset$ at Step 0 of Algorithm 1 and consider two variants:

$$I_w^{k+1} := \begin{cases} I_{\mathrm{LP}}^k \quad \text{and} \quad \epsilon_{\mathrm{MK}}^{k+1} < \epsilon_{\mathrm{MK}}^k \ (\text{or } n_{\mathrm{LP}}^{k+1} > n_{\mathrm{LP}}^k) & \text{if } k \text{ gave a serious-point} \\ I_{\mathrm{LP}}^k \quad \text{and} \quad \epsilon_{\mathrm{MK}}^{k+1} = \epsilon_{\mathrm{MK}}^k \ (\text{or } n_{\mathrm{LP}}^{k+1} = n_{\mathrm{LP}}^k) & \text{if } k \text{ gave a null-point; or} \end{cases} \tag{4.6}$$

$$I_w^{k+1} := \emptyset \text{ at all iterations } k. \tag{4.7}$$

With the variant (4.6), the cardinality of the working sets and, hence, of I_{LP}^k, increases as $k \to \infty$. Since, in addition, $\epsilon_f^k = \epsilon_g^k := c(I_{\mathrm{LP}}^k) \le \epsilon_{\mathrm{MK}}^k$ strictly decreases at serious-points, when Algorithm 1 makes an infinite number of serious-points, it is asymptotically exact and can be seen as an incremental bundle method; see [8]. Concerning variant (4.7), there is no initial working set to keep memory of the past. For this reason, it is sound to let the pseudonorm $d(\cdot, \cdot)$ incorporate additional information along iterations, as explained below.

Choosing the pseudonorm. The pseudonorm $d(\xi_i, \xi_j)$ somehow measures the distance between the scenarios ξ_i and ξ_j. A first, natural, choice sets $d(\xi_i, \xi_j) := \|\xi_i - \xi_j\|$ for some norm $\|\cdot\|$. Another possibility takes $d(\xi_i, \xi_j) := g(\xi_i, \xi_j)\|\xi_i - \xi_j\|$, where g is a nonnegative problem-dependent function. For instance, in [26], g depends on a scaling factor that estimates the ratio between the theoretical and sample variance of scenarios (of a certain generalized periodic autoregressive model). With this choice, the variance is not reduced after discarding scenarios, allowing extreme events to be sufficiently represented in the smaller subset I_{LP}.

In our runs, we consider a more general form for $d(\cdot,\cdot)$, inspired from [25], where proximity between scenarios is measured through their impact in the value functions $Q(x;\cdot)$ from (2.5). More precisely, in [25], the authors use the values $Q(x^1;\xi_i)$, $i=1,\ldots,N$, to discard scenarios that appear as "close" ($x^1 \in X$ is an initial point chosen carefully, and the scenario selection is done once, before the optimization process). However, since different scenarios may have the same value function, the function from [25] is *not* a pseudonorm, and the theory from [7] no longer holds. To overcome this handicap, we consider the following modification:

$$d_\lambda(\xi_i,\xi_j) := \lambda\|\xi_i-\xi_j\| + (1-\lambda)|\mathtt{Q}_i - \mathtt{Q}_j| \tag{4.8}$$

for an estimate $\mathtt{Q}_i$ of $Q(x;\xi_i)$ with $i=1,\ldots,N$ and a threshold $\lambda \in (0,1]$.

When $\lambda = 1$, the classical norm $\|\xi_i-\xi_j\|$ is used; when $\lambda < 1$, the computation of the values $\mathtt{Q}_i$ involves the following modification in the Inexact Oracle 3:

— In **Step 0**, compute

$$d_{jl} = \begin{cases} \|\xi_i-\xi_j\| & \text{if } k=1,\\ \lambda\|\xi_i-\xi_j\| + (1-\lambda)|\mathtt{Q}_i-\mathtt{Q}_j| & \text{otherwise.}\end{cases}$$

— **Step 3′ (redistribution and pseudonorm estimates)**. For each $i \in I_{\mathrm{LP}}$ compute the new probabilities: $\tilde{p}_i = p_i + \sum_{j\in J_i} p_j$, where $J_i := \{j \notin I_{\mathrm{LP}}\colon i \in \arg\min_{l\in I_{\mathrm{LP}}} d_{jl}\}$. Set

$$\mathtt{Q}_l = \begin{cases} Q(x;\xi_l) = u_l^\top(h_l - T_l x) & \text{if } l \in I_{\mathrm{LP}},\\ u_i^\top(h_l - T_l x) & \text{if } l \notin I_{\mathrm{LP}}, \quad \text{where } i \in I_{\mathrm{LP}} \text{ is such that } l \in J_i.\end{cases}$$

When $\lambda = 0$, the Inexact Oracle 3 estimates coincide with those of the Inexact Oracle 1 if the second-stage problems have deterministic cost. However, unlike the Inexact Oracles 1 and 2, the scenario selection strategy above can be applied to convex two-stage stochastic programs (keeping in mind that if $\lambda < 1$, when $l \notin I_{\mathrm{LP}}$, suitable proxies $\mathtt{Q}_l$ for $Q(x;\xi_l)$ have to be defined).

5. Numerical assessment. In this section we report some results obtained for different two-stage stochastic linear programing programs. The battery comprises 10 families of problems, with uncertainty dimension ranging from 2 to 200. For each family, 11 instances, corresponding to

$$N \in \{100, 200, 300, 500, 800, 1000, 1200, 1500, 1800, 2000, 2500\}$$

scenarios, were considered to benchmark 5 different solvers. Comparisons are done in terms of accuracy and CPU time, as well as solution quality of primal and dual variables. All the tests were run using MATLAB 7.8.0 (R2009a) on an AMD Athlon II X2 240 computer with 2800 MHz, 2 GB RAM, with Ubuntu OS, using the MOSEK optimization toolbox for MATLAB; see http://www.mosek.com.

5.1. Main features of the benchmark.

5.1.1. Solvers. In addition to the inexact bundle method combined with the three inexact oracles in section 4, we consider a somewhat opposite approach. More precisely, we first select a reduced set of n_{LP} scenarios using the pseudonorm d_λ and then solve the

resulting optimization problem by applying a cutting-plane or bundle method using exact oracles. The mnemonic device for the solvers is as follows:

d_λ-ECP—exact cutting-plane method over a reduced and fixed set with n_{LP} scenarios;

d_λ-EBM—exact bundle method over a reduced and fixed set with n_{LP} scenarios;

IBM-cos—inexact bundle method with Inexact Oracle 1 or 2, based on collinearity strategies;

IBM-d_1—inexact bundle method with the Inexact Oracle 3 with $\lambda = 1$, as in [7];

IBM-d_λ—inexact bundle method with the Inexact Oracle 3 with $\lambda \in (0,1)$, selecting scenarios with the (pseudo) norm in (4.8).

For IBM-d_1, we applied the rule (4.6), and for IBM-d_λ, we used (4.7). We sometimes refer to the first two solvers as "static" ones, in opposition to the last three, which are "dynamic," in the sense that they change the considered scenarios along iterations. In particular, note that, even if IBM-cos used a fixed tolerance $\epsilon_{\cos} \in (0,1)$, the sets I_{LP} would change along iterations, because the values $\cos \theta_{ij}$ depend on each iterate z^k. In order to define the set I_{LP} in our numerical experiments with IBM-cos, we employed the following procedures.

For the Inexact Oracle 1, Step 0 includes the following:

Given $z^k \in X$ and $\epsilon_{\cos} \in (0,1)$, set $I_{\mathrm{LP}} \coloneqq \{1 \ldots, N\}$
FOR $i = 1$ to N
IF $i \in I_{\mathrm{LP}}$ THEN
FOR $j = i + 1$ to N
IF $j \in I_{\mathrm{LP}}$ THEN
compute $\cos \theta_{ij} = \frac{(h_i - T_i z^k)^\top (h_j - T_j z^k)}{\|h_i - T_i z^k\| \|h_j - T_j z^k\|}$
IF $(1 - \cos \theta_{ij}) < \epsilon_{\cos}$ THEN
do $I_{\mathrm{LP}} \coloneqq I_{\mathrm{LP}} \setminus \{j\}$
END IF
END IF
END FOR
END IF
END FOR

For the Inexact Oracle 2, Steps 0 and 1 incorporate the following:

Given $z^k \in X$ and $\epsilon_{\cos} \in (0,1)$, define $I_{\mathrm{LP}} \coloneqq \{1 \ldots, N\}$
FOR $i = 1$ to N
IF $i \in I_{\mathrm{LP}}$ THEN
define $\mathcal{J}_i \coloneqq \{i\}$
FOR $j = i + 1$ to N
IF $j \in I_{\mathrm{LP}}$ THEN
compute $\cos \theta_{ij} = \frac{(h_i - T_i z^k)^\top (h_j - T_j z^k)}{\|h_i - T_i z^k\| \|h_j - T_j z^k\|}$
IF $(1 - \cos \theta_{ij}) < \epsilon_{\cos}$ THEN
do $I_{\mathrm{LP}} \coloneqq I_{\mathrm{LP}} \setminus \{j\}$
do $\mathcal{J}_i \coloneqq \mathcal{J}_i \cup \{j\}$
END IF
END IF
END FOR
END IF
END FOR

Finally, for the Inexact Oracle 3, the set I_{LP} was built by applying the SOR procedure given in [7], as explained in Step 1 of the Inexact Oracle 3. We refer to [10] for more numerical experiments on two-stage stochastic programs, where the level decomposition was applied.

5.1.2. Stopping tolerance and nonsmooth parameters. Recalling that $p^k \in \mathfrak{R}^{n_1}$, as proposed in [21], both inexact and exact bundle algorithms stop when $\|p^k\| \le 5 \cdot 10^{-4}\sqrt{n_1}$ and either $v_k \le 10^{-5}(1+|f_x^k|)$ with $-\alpha_k \le v_k$, or $\|p^k\| + \alpha_k \le 10^{-5}(1+|f_x^k|)$. For declaring a serious-point, we fixed $\kappa = 10^{-1}$, and the initial prox-parameter was set to $t_1 = \min\{\|g^1\|^{-1}, \tau_1\}$ for $\tau_1 = 10$. The cutting-plane method stops when the difference of the functional value and its model at the current iterate $(f(z^{k+1}) - \check{f}_k(z^{k+1}))$ is below the tolerance 10^{-5}.

5.1.3. Errors. For comparing the quality of the ϵ-solutions obtained by the different solvers we use the relative error

$$e\% := 100\frac{|f_N(x_N^*) - f_N(x_N^\epsilon)|}{1+|f_N(x_N^*)|},$$

where $x_N^* := \arg\min_{x\in X}\{f_N(x) := c^T x + \sum_{i=1}^N p_i Q(x;\xi_i)\}$, and x_N^ϵ is the approximate solution found by the solver. To define x_N^*, we use the equivalent deterministic formulation "EquiDet," or, if the complexity of the problem does not make EquiDet directly solvable, the problem is decomposed and solved by the proximal bundle method, denoted by "EBM." Similarly, CPU% measures the relative decrease in computational time when compared to the reference time. For this measure, we do not take absolute values, so a negative value for CPU% should be understood as a reduction of CPU time.

5.1.4. Metric and starting point. In (4.8) we use the Euclidean norm and set λ as follows. The initial point x^1 is the *expected value solution* $\bar{x}$ of the *expected value problem* [2, sect. 4.2],

$$EV := \min_{x\in X}\{\bar{f}_N(x) := c^\top x + Q(x;\mathbb{E}[\xi])\}.$$

Since $Q(\cdot;\xi)$ is a convex function in variable $\xi = (h, T)$ by [2, Thm. 5], we have that $EV \le f_N(\bar{x})$. If $EV = f_N(\bar{x})$, the random parameter ξ does not give any additional information about the optimization problem. Thus, we can use the relation between EV and $f_N(\bar{x})$ to set the parameter λ for d_λ in (4.8) by taking

$$\lambda := 1 - \frac{EV}{f_N(\bar{x})}.$$

For IBM-d_λ we update λ along iterations by replacing $f_N(\bar{x})$ with f_x^k in the above relation. Finally, the safeguard $\lambda \in [0.05, 0.95]$ ensures that d_λ is a norm, different from the Euclidean norm employed by d_1.

5.1.5. Test problems. The name of the problems considered are SH10, InvestmentRisk, SH10Risk, SH31, SH25, AirCraft, ProductMix, ProjectSchedule, ExpTerm3, and ExpTerm23. Their description and corresponding results are reported below.

5.2. Results for SH10. The first numerical test is an artificial problem presented in [30] with the right-hand side constraint h independently and normally distributed with support $\Xi \subset \mathfrak{R}^{10}$. It is denoted by SH10, and its formulation is presented in [30]; see also [5] for two misprint corrections. Table 5.1 gives some parameters for the solvers, where $n_{\rm LP}^k$ stands for the maximum cardinality allowed for the working sets I_w, when applying fast forward selection at Step 1 of the Inexact Oracle 3.

The optimal values and CPU time (seconds) used for reference for each tested instance are reported in Table 5.2. The two rightmost columns in the table contain the number of variables and constraints of the equivalent deterministic formulation.

Table 5.3 reports the results of the 5 solvers for the 11 instances. We observe a good quality of the ϵ-solutions with important reductions in CPU time. As expected, as the scenario number N increases, the variant IBM-d_λ slows down, due to the complexity of the scenario selection process. Such is not the case for IBM-cos, which remains fast for large N.

By solving 20 independent instances, each with 200 scenarios, and applying the technique presented in [29, sect. 5.1.2], we computed a lower bound $L = 15.12496$ (with 95% confidence) for SH10. To compute an upper bound, we use x^ϵ_{2500}, a solution obtained with the inexact methods for an instance with $N = 2500$ scenarios:

$$U \coloneqq f_{8000}(x^\epsilon_{2500}) + 1.96\frac{\sigma_x}{\sqrt{8000}} \quad \text{with } \sigma_x^2 \coloneqq \frac{1}{7999}\sum_{i=1}^{8000}\left(Q(x^\epsilon_{2500};\xi_i) - \mathbb{E}[Q(x^\epsilon_{2500};\xi)]\right)^2.$$

Table 5.4 gives the bounds obtained using x^*_{2500} (a solution obtained with EBM for an instance with $N = 2500$ scenarios) and x^ϵ_{2500} for the different solvers and the bounds obtained using the three different approximate solutions given in [5, Table 3]. We observe a good fitness for all solvers, with a slight advantage of the dynamic methods over the static ones. The upper bounds found with the approximate solutions are very close to the one obtained with x^*_{2500}. A similar behavior is observed in Figure 5.1 for the primal variables. More precisely, Figure 5.1 shows the 10 components of the first-stage approximate solution, with a confidence interval from [30]. Since in some applications the dual second-stage variables are important, due to their economical interpretation as shadow prices, we report in Table 5.5 the corresponding expected values, which appear as more solver-dependent than the primal variables.

5.3. Results for risk-averse problems. The introduction of a risk measure akin to the one given in [28] makes the recourse matrix W random when the second-stage cost q is also random, an interesting feature to test the Inexact Oracle 2. Since in this case the collinearity strategy computes an average feasible set, IBM-cos is expected to perform less well. We consider problem InvestmentRisk, derived from [24] with 200 normally

TABLE 5.1
Solver parameters.

Method	$n_{\rm LP}^1$	$n_{\rm LP}^k$ with $k > 1$	$\epsilon_{\cos}$
EBM	—	—	—
d_λ-ECP	$\min\{0.5N, 350\}$	—	—
d_λ-EBM	$\min\{0.5N, 350\}$	—	—
IBM-d_1	$\min\{0.4N, 300\}$	$\min\{1.1n_{\rm LP}^{k-1}, 0.6N\}$	—
IBM-d_λ	$\min\{0.4N, 300\}$	$\min\{0.4N, 300\}$	—
IBM-cos	—	—	$2 \cdot 10^{-3}$

TABLE 5.2
*Optimal value and CPU time for SH*10.

	EquiDet		EBM		Size	
N	$f_N(x_N^*)$	CPU	$f_N(x_N^*)$	CPU	Nvar	Ncons
100	15.064	1.443	15.064	7.02	1510	1005
200	15.216	5.960	15.216	16.00	3010	2005
300	15.129	18.868	15.129	27.82	4510	3005
500	15.140	85.106	15.140	39.46	7510	5005
800	15.155	357.551	15.155	68.85	12010	8005
1000	—	—	15.129	73.30	15010	10005
1200	—	—	15.130	88.34	18010	12005
1500	—	—	15.164	129.85	22510	15005
1800	—	—	15.160	167.52	27010	18005
2000	—	—	15.159	158.37	30010	20005
2500	—	—	15.149	210.93	37510	25005

TABLE 5.3
*Solution quality and CPU time for SH*10.

N	d_λ-ECP		d_λ-EBM		IBM-d_1		IBM-d_λ		IBM-cos	
	e%	CPU%	e%	CPU%	e%	CPU%	e%	CPU%	e%	CPU%
100	0.05	−10.57	0.05	−24.52	0.01	−60.84	0.06	−77.17	0.00	−44.18
200	0.01	−10.22	0.01	−32.93	0.07	−66.22	0.07	−80.12	0.00	−50.27
300	0.01	−26.37	0.01	−44.37	0.01	−74.84	0.01	−78.42	0.00	−49.14
500	0.01	11.17	0.01	−46.01	0.01	−71.37	0.01	−68.66	0.00	−36.40
800	0.01	−12.17	0.01	−38.57	0.00	−49.28	0.00	−59.84	0.00	−71.41
1000	0.00	−1.93	0.00	−25.42	0.01	−62.28	0.01	−48.06	0.00	−68.32
1200	0.01	−11.12	0.01	−36.63	0.03	−59.96	0.03	−40.41	0.00	−70.09
1500	0.02	−34.90	0.02	−33.99	0.01	−61.88	0.01	−42.98	0.00	−69.77
1800	0.05	−38.60	0.05	−42.41	0.00	−58.44	0.00	−39.61	0.00	−73.63
2000	0.03	−23.33	0.03	−30.57	0.01	−45.67	0.01	−21.79	0.00	−72.19
2500	0.02	−20.84	0.02	−30.02	0.01	−42.15	0.01	−9.22	0.00	−74.95
min.	0.00	−38.60	0.00	−46.01	0.00	−74.84	0.00	−80.12	0.00	−74.95
max.	0.05	11.17	0.05	−24.52	0.07	−42.15	0.07	−9.22	0.00	−36.40
aver.	0.02	−16.26	0.02	−35.04	0.02	−59.36	0.02	−51.48	0.00	−61.85

distributed variables ($h, q \in \mathfrak{R}^{100}$). In order to fit our framework, the risk measure needs to appear only at the second stage. Accordingly, we replaced the risk measure in [24] by the one in [28]. We note that the risk measure makes the recourse matrix W random, somewhat escaping our initial setting (the second item in Assumption 1 is not satisfied).

We also introduced in SH10 the risk measure given in [28], to create problem SH10Risk (for this problem q is fixed). Except for $\epsilon_{\cos} = 10^{-3}$, the solver parameters coincide with those in Table 5.1.

Tables 5.6 and 5.7 report the results obtained for the two problems with all the solvers. In both cases we observe a good performance in CPU times for IBM-cos, with a slight reduction in accuracy for the family InvestmentRisk.

TABLE 5.4
*Optimal value bounds for SH*10.

Method	U	$U-L$	$100(U-L)/U$	$f_{8000}(x^\epsilon)$	$\sigma_x/\sqrt{8000}$
d_λ-EBM	15.18512	0.06016	0.39618	15.16965	0.00940
d_λ-ECM	15.18515	0.06019	0.39640	15.16968	0.00941
IBM-d_1	15.18394	0.05898	0.38846	15.16862	0.00932
IBM-d_λ	15.18394	0.05898	0.38846	15.16862	0.00932
IBM-cos	15.18254	0.05758	0.37927	15.16755	0.00911
EBM	15.18253	0.05757	0.37918	15.16754	0.00911
[5]^1	15.18500	0.06004	0.39537	15.16996	0.00914
[5]^2	15.18531	0.06035	0.39740	15.17039	0.00907
[5]^3	15.18495	0.05999	0.39505	15.17008	0.00904

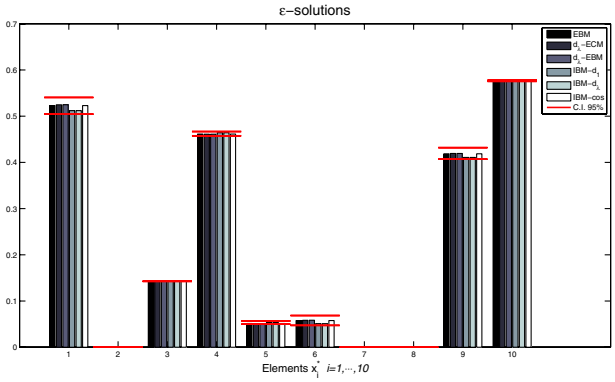

FIG. 5.1. *Confidence interval for SH*10 *first-stage solution.*

5.4. Assessing performance of solvers. *The performance profile* [6] is a convenient tool to present and compare results for a set of different solvers and problems. To compare accuracy and CPU time reduction of our techniques, we used the MATLAB's routine given in [14, sect. 22.4]. As the input data of this routine must be positive, we added the values 1 to the percentage error $e\%$ and 100 to the CPU reduction. In this manner, higher values in the performance profile always indicate a better performance of the solver.

In addition to SH10, SH10Risk, and InvestmentRisk, we consider the following problems:

— SH31—based in SH10, multiplying by 10 the standard deviation, and with each nonzero element of the technology matrix T random, with independent normal distribution with mean $T(i,j)$ and variance $(T(i,j)/10)^2$, yielding 31 random variables;

— SH25—based in SH10. This version considers independent normal costs with mean q (given in [30, sec. 6.3]) and variance $q/2$ so that there are 25 random variables;

— AirCraft—the aircraft allocation problem in [9, pp. 544–546] with the following modifications: each one of the five components of h (passenger demand) is independently and normally distributed with mean and variance computed using the data given in [9];

TABLE 5.5
*Dual second-stage variable for SH*10.

Method	$\mathbb{E}[u_1]$	$\mathbb{E}[u_2]$	$\mathbb{E}[u_3]$	$\mathbb{E}[u_4]$	$\mathbb{E}[u_5]$	$\mathbb{E}[u_6]$	$\mathbb{E}[u_7]$	$\mathbb{E}[u_8]$	$\mathbb{E}[u_9]$	$\mathbb{E}[u_{10}]$
d_λ-EBM	−0.1981	−0.0833	−0.1955	0	−0.1073	−0.0109	−0.6595	1.2074	0.6447	0.2146
d_λ-ECM	−0.1989	−0.0832	−0.1955	0	−0.1060	−0.0111	−0.6597	1.2076	0.6498	0.2154
IBM-d_1	−0.1857	−0.0748	−0.1955	0	−0.1315	−0.0040	−0.6521	1.1879	0.6245	0.2011
IBM-d_λ	−0.1857	−0.0748	−0.1955	0	−0.1315	−0.0040	−0.6521	1.1879	0.6245	0.2011
IBM-cos	−0.1571	−0.0494	−0.1955	0	−0.1900	0.0139	−0.6329	1.1348	0.6121	0.1702
EBM	−0.1573	−0.0494	−0.1955	0	−0.1897	0.0138	−0.6330	1.1349	0.6133	0.1704
[5]^1	−0.1672	−0.0475	−0.1955	0	−0.1736	0.0117	−0.6352	1.1371	0.6826	0.1811
[5]^2	−0.1496	−0.0455	−0.1955	0	−0.2040	0.0176	−0.6290	1.1249	0.5921	0.1620
[5]^3	−0.1479	−0.0375	−0.1955	0	−0.2099	0.0211	−0.6252	1.1119	0.6311	0.1603

TABLE 5.6
Solution quality and CPU time for InvestmentRisk.

N	d_λ-ECP		d_λ-EBM		IBM-d_1		IBM-d_λ		IBM-cos	
	e%	CPU%	e%	CPU%	e%	CPU%	e%	CPU%	e%	CPU%
100	0.67	−43.41	0.67	−52.84	0.65	−49.19	1.88	−58.32	1.89	−62.08
200	1.01	−62.19	1.01	−63.91	0.10	−60.79	0.43	−74.89	1.46	−69.09
300	0.47	−46.91	0.47	−57.51	0.03	−50.52	0.28	−59.64	1.78	−65.30
500	0.04	−45.81	0.04	−55.37	0.02	−44.32	0.39	−65.56	1.80	−69.74
800	0.14	−45.15	0.14	−63.60	0.05	−57.27	0.03	−62.82	2.18	−78.34
1000	0.22	−47.48	0.23	−58.61	0.10	−37.65	0.18	−60.23	1.11	−78.12
1200	0.06	−56.61	0.06	−61.06	0.36	−55.96	0.08	−62.29	2.25	−75.60
1500	0.01	−62.03	0.01	−70.17	0.05	−58.89	0.08	−69.15	1.60	−72.51
1800	0.07	−68.36	0.06	−74.68	0.14	−63.47	0.14	−66.66	1.73	−77.65
2000	0.07	−70.62	0.07	−71.62	0.02	−63.90	0.54	−66.13	1.81	−80.88
2500	0.09	−74.87	0.06	−78.37	0.14	−65.53	0.08	−66.98	1.62	−85.36
min.	0.01	−74.87	0.01	−78.37	0.02	−65.53	0.03	−74.89	1.11	−85.36
max.	1.01	−43.41	1.01	−52.84	0.65	−37.65	1.88	−58.32	2.25	−62.08
aver.	0.26	−56.68	0.26	−64.34	0.15	−55.23	0.37	−64.79	1.75	−74.06

— ProductMix—the product mix problem given in [9, pp. 554–555]. It has 10 random variables, T is uniformly distributed, and $h \in \mathfrak{R}^2$ is normally distributed;
— ProjectSchedule—a project scheduling problem also given in [9, pp. 549–552]. The time for completion of each one of 25 projects was modified from [9], considering an integer uniform distribution in the interval $[2, 25]$;
— ExpTerm3—an expansion thermal power capacity problem based in [2, sect. 1.3] with 10 power plants. The demand for electricity $h \in \mathfrak{R}^3$ is normally distributed;
— ExpTerm23—based on ExpTerm3, but with random generation costs, yielding 23 random variables.

TABLE 5.7
*Solution quality and CPU time for SH*10*Risk.*

N	d_λ-ECP		d_λ-EBM		IBM-d_1		IBM-d_λ		IBM-cos	
	e%	CPU%	e%	CPU%	e%	CPU%	e%	CPU%	e%	CPU%
100	0.30	−34.19	0.30	−43.45	0.03	−37.14	0.71	−63.82	0.00	−54.64
200	0.28	−46.32	0.28	−51.28	0.28	−37.44	0.29	−64.75	0.02	−72.54
300	0.06	−39.21	0.06	−43.51	0.06	−34.01	0.08	−61.92	0.00	−66.75
500	0.01	−37.50	0.01	−41.49	0.10	−44.90	0.09	−55.73	0.07	−80.86
800	0.15	−30.89	0.16	−40.34	0.03	−27.94	0.14	−43.09	0.00	−73.70
1000	0.03	−50.65	0.03	−50.17	0.11	−29.73	0.47	−53.87	0.01	−81.77
1200	0.04	−54.77	0.04	−60.94	0.04	−55.98	0.03	−48.69	0.00	−79.83
1500	0.17	−72.55	0.17	−74.28	0.16	−61.01	0.18	−66.76	0.00	−84.02
1800	0.10	−61.34	0.09	−64.48	0.11	−49.23	0.49	−46.01	0.00	−81.25
2000	0.33	−68.95	0.32	−68.74	0.40	−48.81	0.44	−52.00	0.00	−83.63
2500	0.26	−67.84	0.26	−63.97	0.31	−52.95	0.29	−42.00	0.00	−83.24
min.	0.01	−72.55	0.01	−74.28	0.03	−61.01	0.03	−66.76	0.00	−84.02
max.	0.33	−30.89	0.32	−40.34	0.40	−27.94	0.71	−42.00	0.07	−54.64
aver.	0.16	−51.29	0.16	−54.79	0.15	−43.56	0.29	−54.42	0.01	−76.57

TABLE 5.8
Solver parameters.

Method	n_{LP}^1	n_{LP}^k with $k > 1$
ECP	—	—
d_λ-ECP	$\min\{0.5N, 200\}$	—
d_λ-EBM	$\min\{0.5N, 200\}$	—
IBM-d_1	$\min\{0.4N, 150\}$	$\min\{1.1n_{\mathrm{LP}}^{k-1}, 0.6N\}$
IBM-d_λ	$\min\{0.4N, 150\}$	$\min\{0.4N, 150\}$

For each one of the 10 problems above, we solve all 11 instances (corresponding to different scenario numbers) using the parameters given in Table 5.8. The tolerance $\epsilon_{\cos}$ is problem-dependent, with values ranging in the interval [0.0001, 0.2].

Table 5.9 reports the average (over the 11 instances) in both accuracy and CPU time reduction, obtained for each problem and solver. In this table, the symbol † is used to indicate that the second-stage cost q is deterministic for the corresponding family. We note that IBM-cos is the more accurate technique for such problems.

Table 5.10 reports the average iteration number. Both IBM-d_λ and IBM-d_1 exhibit the lowest number of iterations to converge.

Our first performance profiles are given in Figures 5.2 and 5.3. Each line in a performance profile can be understood as a cumulative probability distribution of a resource of interest (accuracy, number of iterations, CPU time). We compared three different resources: accuracy, CPU time reduction, and a measure combining accuracy and CPU time, which can be thought of as a compromise between accuracy and computational work.

In Figure 5.2 the top left and right graphs give, respectively, the profile for accuracy and CPU time reduction. In terms of accuracy, d_λ-EBM and d_λ-ECP have identical profiles (the dashed black and solid gray lines in the top left graph coincide), with IBM-cos being the most accurate solver in 63% of the runs (the highest value at $\theta = 1$). The second best solvers in terms of accuracy are d_λ-EBM and d_λ-ECP, which can be considered comparable within a factor 1.1 to the most accurate technique, IBM-cos (such

TABLE 5.9
Average solution quality and CPU time for the 10 *problems (over* 11 *instances).*

Problem	d_λ-ECP		d_λ-EBM		IBM-d_1		IBM-d_λ		IBM-cos	
	e%	CPU%	e%	CPU%	e%	CPU%	e%	CPU%	e%	CPU%
AirCraft⊂†	0.02	−62.15	0.02	−67.86	0.03	−68.96	0.24	−68.15	0.00	−67.91
ExpTerm3⊂†	0.00	−44.96	0.00	−44.29	0.36	−20.70	0.75	−52.53	0.02	−75.54
ExpTerm23	0.01	−14.64	0.01	−11.34	0.84	−41.57	1.43	−51.32	0.72	−49.47
ProductMix⊂†	0.03	−62.26	0.03	−63.12	0.07	−72.99	0.30	−62.13	0.00	−59.18
ProjectSchedule⊂†	0.00	−20.35	0.00	−30.46	0.04	−51.76	0.08	−50.87	0.00	−42.08
SH10⊂†	0.02	−39.22	0.02	−48.93	0.02	−73.39	0.02	−70.63	0.00	−61.85
SH31⊂†	0.08	−52.08	0.08	−57.54	0.48	−78.56	0.54	−71.44	0.00	−53.07
SH25	0.02	−31.79	0.02	−46.16	0.03	−71.35	0.03	−64.29	0.05	−51.45
SHRisk⊂†	0.01	−67.50	0.01	−68.08	0.33	−59.11	1.02	−68.19	0.01	−76.57
InvestmentRisk	0.29	−71.05	0.29	−74.83	0.18	−72.44	0.28	−77.89	1.75	−74.06
Average	0.05	−46.60	0.05	−50.16	0.24	−61.08	0.47	−63.74	0.25	−61.12

TABLE 5.10
Average iteration number for the 10 *problems (over* 11 *instances).*

Problem	d_λ-ECP	d_λ-EBM	IBM-d_λ	IBM-d_1	IBM-cos	EBM
AirCraft$\subset\dagger$	41	33	34	27	34	37
ExpTerm3$\subset\dagger$	6	9	6	10	8	8
ExpTerm23	7	8	3	8	5	8
ProductMix$\subset\dagger$	17	19	16	13	21	21
ProjectSchedule$\subset\dagger$	9	7	4	6	6	7
SH10$\subset\dagger$	17	14	5	7	13	12
SH31$\subset\dagger$	14	13	6	7	13	14
SH25	18	13	6	7	6	12
SHRisk$\subset\dagger$	36	39	38	35	37	40
InvestmentRisk	36	30	30	25	22	37
Average	20	19	15	15	17	20

factor is given by the abscissa of intersection between the solid black, the dashed black, and solid gray lines). Nevertheless, since $1 - \phi(\theta)$ is the fraction of problems that a solver cannot solve within a factor θ of the best solver, looking at the difference between the dashed black/solid gray lines and the solid black line at the abscissa 2 in the top left graph, we see that IBM-cos could not solve 8% of the cases with half the accuracy obtained by either d_λ-ECP or d_λ-EBM (this percentage roughly corresponds to problems with variable second-stage cost.)

In terms of CPU time, d_λ-ECP is systematically slower than d_λ-EBM, a known fact in nonsmooth optimization, due to cutting-plane methods suffering from tailing-off phenomena. Since in the top right graph $\phi(4) = 0.99$ for d_λ-ECP, the cutting-plane static solver failed to solve 1% of the problems in less than 4 times the CPU time needed by the fastest solver (IBM-d_1) to solve all the problems. As for CPU time reduction, IBM-d_1 (resp., IBM-d_λ) is the fastest solver in 37% (resp., 22%) of the cases, but the left picture shows that IBM-d_1 (resp., IBM-d_λ) was the most accurate solver for 12% (resp., 4.5%) of cases only.

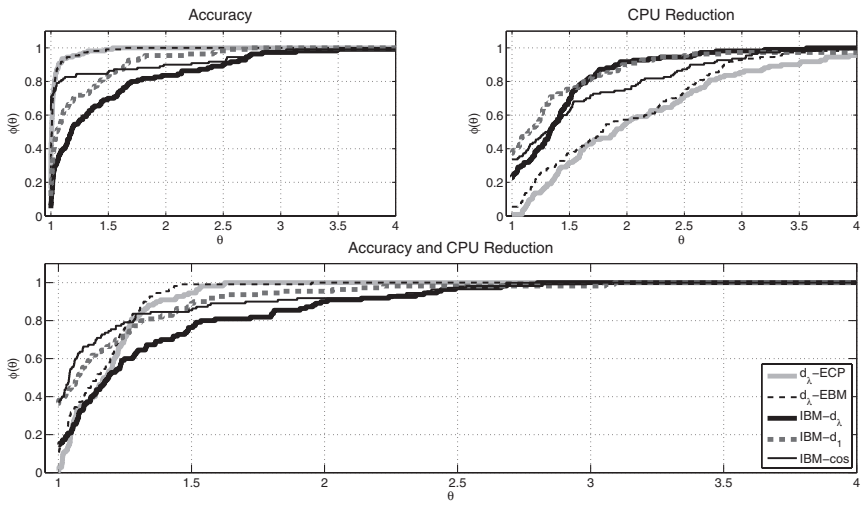

FIG. 5.2. *Performance profiles for all problems in the battery and all instances (*110 *runs).*

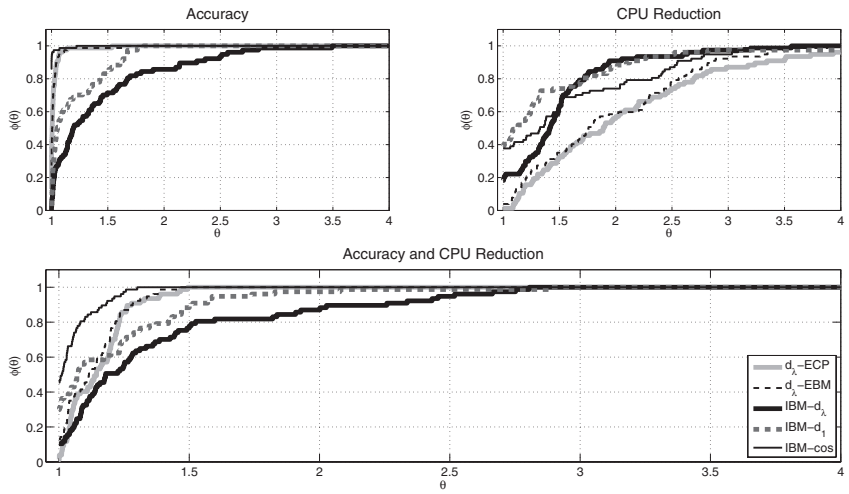

FIG. 5.3. *Performance profiles for problems with fixed recourse and fixed second-stage cost and all instances (77 runs).*

Finally, the bottom graph in Figure 5.2 shows the combined measure with weight 0.5 for both accuracy and time reduction. The best solver combining accuracy and speed is IBM-cos, followed by IBM-d_1. Both ECP-d_λ and EBM-d_λ are comparable to the best technique, within a factor of 1.2 in about 70–80% of the cases.

In the three graphs in Figure 5.2, at the abscissa value equal to 4 all solvers reached a value close to 1. In this sense, if we chose being within a factor of 4 of the best solver as the scope of our interest, then any solver would suffice. But the performance profiles show that the probability that IBM-cos can solve a job within a factor 4 of the most accurate (resp., fastest) solver is about 63%. In view of these figures, to determine the impact on IBM-cos performances for problems with uncertain second-stage cost (q_i) and variable recourse (W_i), we made a new set of graphs, given in Figure 5.3, excluding from the battery the three families with variable recourse and/or uncertain second-stage cost: InvestmentRisk, SH25, and ExpTerm23. The situation changes dramatically, with IBM-cos being as accurate as the static solvers, with an impressive advantage in CPU times. This feature is consistent with the fact that, to save computational time, IBM-cos makes no exact calculation for problems with random second-stage cost.

On the basis of our numerical experience, the collinearity oracle appears to be the best strategy in terms of both accuracy and speed. If CPU time is not a serious concern, the static variant using an exact bundle method could be a good option (keeping in mind that it was systematically slower than the collinearity oracle). If a static variant is to be used, then the method of choice should be the exact bundle method over the L-shaped (cutting-plane) method. A similar experiment using a level decomposition was presented in [10] for problems with only h random.

We give in Figure 5.4 the performance profile of the number of iterations, noting that the lower the line is, the higher the number of iterations is. We note that IBM-cos gives a number of iterations similar to the number given by EBM. However, each iteration of IBM-cos requires less effort and, for this reason, this variant is faster than EBM. Figure 5.4 shows that IBM-d_1 needs more iterations to stop than IBM-d_λ, but Figure 5.3 (top right) shows that overall IBM-d_1 is faster. The reason is that IBM-d_1 uses variant

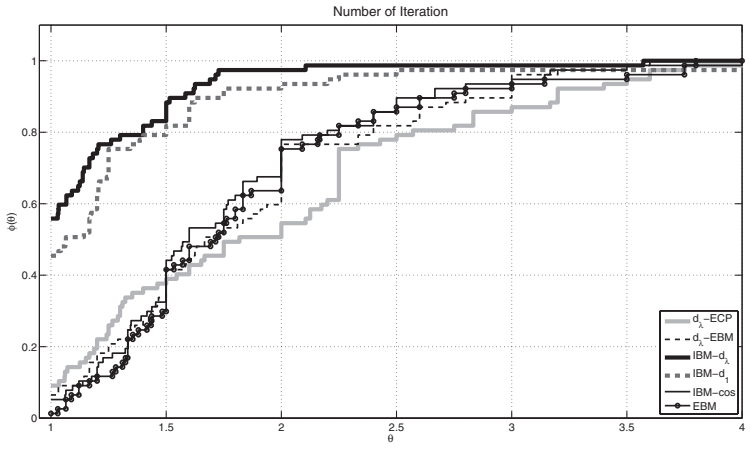

FIG. 5.4. *Performance profiles for problems with fixed recourse and fixed second-stage cost and all instances (77 runs).*

(4.7), while IBM-d_λ uses (4.6), which requires the computationally heavy distance calculations d_{ij} at Step 0 of the Inexact Oracle 3 for each iteration k.

For the Inexact Oracle 3, IBM converges in fewer iterations than IBM-cos because of how accuracy is handled. When the noise attenuation process at Step 3 of Algorithm 1 is accessed often, the measure V_k is driven to zero, triggering an early iteration process with an approximate solution (see comments after Lemma 3.1). Differently from [10], which explicitly controls the oracle inaccuracy, for our inexact bundle method a rough approximation does not necessarily increase the number of iterations. However, a too coarse solution can be produced, as illustrated by Figure 5.5.

For testing the impact of the collinearity parameter $\epsilon_{\cos}$ on the computational effort, Figure 5.5 reports on results obtained for the IBM-cos with the Inexact Oracle 1, when applied to problem SHRisk for $N = 200$ scenarios. The problem was solved 100 times for values of $\epsilon_{\cos}$ ranging between 0 and 0.1.

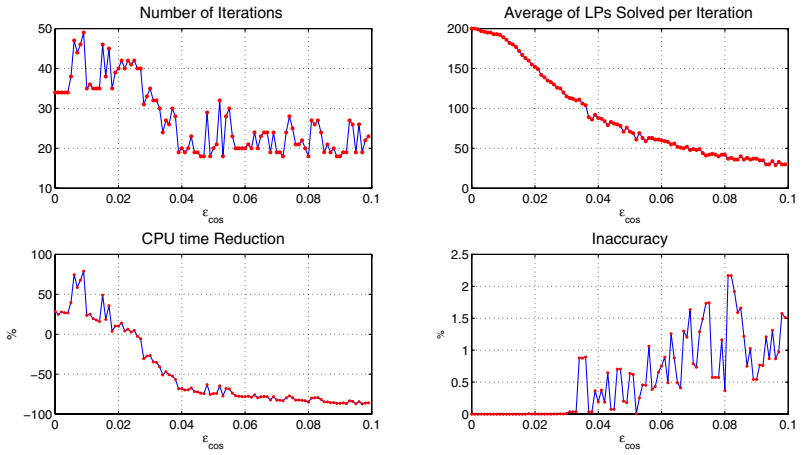

FIG. 5.5. *Sensitivity analysis for the Inexact Oracle* 1.

For $0 \leq \epsilon_{\cos} \leq 0.025$, the CPU time required for solving SHRisk is the highest. This is due to the complexity of computing $\cos\theta$ by the Inexact Oracle 1 and the growth of iteration number. For $0.025 \leq \epsilon_{\cos} \leq 0.033$, the CPU time can be reduced (the negative values in the bottom left graph in Figure 5.5 mean time decreases), and solutions obtained by IBM-cos were optimal. However, for $0.04 \leq \epsilon_{\cos} \leq 0.1$, IBM-cos provided only rough approximate solutions, with the iteration number having an oscillatory behavior and the CPU time systematically decreasing. As the average number of linear programs (LPs) solved per iteration (top right graph in Figure 5.5) decreases, a single iteration of IBM-cos is less expensive when the parameter $\epsilon_{\cos}$ increases. In fact, the average number of LPs solved per iteration represents the average cardinality of the set I_{LP}, defined at Step 0 of the Inexact Oracle 1.

Our conclusions are parameter-dependent, and should be taken as an indication, rather than as affirmation of superiority of IBM-cos over the other techniques. Clearly, setting $n_{\mathrm{LP}} = \min\{0.5N, 200\}$ for IBM-d_1 and IBM-d_λ would increase the accuracy of these techniques. But without doubt, they would also become the slowest ones, because the selection scenario strategy applied at each serious-point of the inexact bundle method would substantially increase the computational effort.

6. Concluding remarks. Our new decomposition approach can be considered a variant of the regularized decomposition [27] that is capable of handling inexact cuts. Inexact cuts can be defined not only by forcing early termination of the subproblem optimization, but also by incorporating scenario selection techniques. This versatility is an important feature of our method, because it makes it possible to fully exploit structural properties of the problem. In particular, incremental methods, with asymptotically vanishing errors for serious iterates [8], [21], could also be employed.

The approach naturally applies to convex two-stage programs, as in [27], introducing feasibility cuts if the recourse is not relatively complete. Also, extending our work to deal with multicuts is straightforward. We note that, contrary to [27], the stepsize in (3.7) is allowed to vary along iterations, a crucial matter for computational efficiency.

Finally, somewhat similarly to [33], since Benders and Dantzig–Wolfe decompositions are dual to each other, it should be possible to mimic our proposal from a Dantzig–Wolfe perspective.

Acknowledgment. We are grateful to the referees for constructive comments and suggestions.

REFERENCES

[1] K. T. Au, J. L. Higle, and S. Sen, *Inexact subgradient methods with applications in stochastic programming*, Math. Programming, 63 (1994), pp. 65–82.

[2] J. R. Birge and F. Louveaux, *Introduction to Stochastic Programming*, Springer-Verlag, New York, 1997.

[3] J. F. Bonnans, J. Ch. Gilbert, C. Lemaréchal, and C. Sagastizábal, *Numerical Optimization. Theoretical and Practical Aspects*, 2nd ed., Universitext, Springer-Verlag, Berlin, 2006.

[4] E. Cheney and A. Goldstein, *Newton's method for convex programming and Tchebycheff approximations*, Numer. Math., 1 (1959), pp. 253–268.

[5] I. Deak, *Two-stage stochastic problems with correlated normal variables: Computational experiences*, Ann. Oper. Res., 142 (2006), pp. 79–97.

[6] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Math. Program., 91 (2002), pp. 201–213.

[7] J. Dupacová, N. Gröwe-Kuska, and W. Römisch, *Scenario reduction in stochastic programming: An approach using probability metrics*, Math. Program., 95 (2003), pp. 493–511.

[8] G. Emiel and C. Sagastizábal, *Incremental-like bundle methods with application to energy planning*, Comput. Optim. Appl., 46 (2010), pp. 305–332.
[9] A. J. King, *Stochastic programming problems: Examples from the literature*, in Numerical Techniques for Stochastic Optimization, Springer Ser. Comput. Math. 10, Y. Ermoliev and R. J.-B. Wets, eds., Springer-Verlag, Berlin, 1988, pp. 543–567.
[10] C. Fábián and Z. Szőke, *Solving two-stage stochastic programming problems with level decomposition*, Comput. Manag. Sci., 4 (2007), pp. 313–353.
[11] N. Gröwe-Kuska, H. Heitsch, and W. Römisch, *Scenario reduction and scenario tree construction for power management problems*, in Power Management Problems, IEEE Bologna Power Tech Proceedings, Vol. 3, A. Borghetti, C. A. Nucci, and M. Paolone, eds., 2003, p. 1–7.
[12] H. Heitsch and W. Römisch, *Scenario reduction algorithms in stochastic programming*, Comput. Optim. Appl., 24 (2003), pp. 187–206.
[13] H. Heitsch and W. Römisch, *Scenario tree reduction for multistage stochastic programs*, Comput. Manag. Sci., 6 (2009), pp. 117–133.
[14] D. J. Higham and N. J. Higham, *MATLAB Guide*, SIAM, Philadelphia, 2000.
[15] J. L. Higle and S. Sen, *Finite master programs in regularized stochastic decomposition*, Math. Program., 67 (1994), pp. 143–168.
[16] J. L. Higle and S. Sen, *Stochastic decomposition: A statistical method for large scale stochastic linear programming*, Nonconvex Optim. Appl. 8, Kluwer Academic Publishers, Dordrecht, 1996.
[17] J.-B. Hiriart-Urruty and C. Lemaréchal, *Convex Analysis and Minimization Algorithms*, I–II, Grundlehren Math. Wiss. 305–306, Springer-Verlag, Berlin, 1993.
[18] J. E. Kelley, *The cutting plane method for solving convex programs*, J. Soc. Indust. Appl. Math., 8 (1960), pp. 703–712.
[19] K. C. Kiwiel, *Proximity control in bundle methods for convex nondifferentiable minimization*, Math. Programming, 46 (1990), pp. 105–122.
[20] K. C. Kiwiel, *A proximal bundle method with approximate subgradient linearizations*, SIAM J. Optim., 16 (2006), pp. 1007–1023.
[21] K. C. Kiwiel, *Bundle Methods for Convex Minimization with Partially Inexact Oracles*, Technical report, 2009.
[22] C. Küchler, *On stability of multistage stochastic programs*, SIAM J. Optim., 19 (2008), pp. 952–968.
[23] O. L. Mangasarian and T. H. Shiau, *Lipschitz continuity of solutions of linear inequalities, programs and complementarity problems*, SIAM J. Control Optim., 25 (1987), pp. 583–595.
[24] N. Miller and A. Ruszczynski, *Risk-averse two-stage stochastic linear programming: Modeling and decomposition*, Oper. Res., 59 (2011), pp. 125–132.
[25] J. Morales, S. Pineda, A. Conejo, and M. Carrión, *Scenario reduction for futures market trading in electricity markets*, IEEE Trans. Power Syst., 24 (2009), pp. 878–888.
[26] W. L. Oliveira, C. Sagastizábal, D. D. J. Penna, M. E. P. Maceira, and J. M. Damázio, *Optimal scenario tree reduction for stochastic streamflows in power generation planning problems*, Optim. Methods Softw., 25 (2010), pp. 917–936.
[27] A. Ruszczyński, *A regularized decomposition method for minimizing a sum of polyhedral functions*, Math. Programming, 35 (1986), pp. 309–333.
[28] A. Shapiro, *Analysis of stochastic dual dynamic programming method*, Eur. J. Oper. Res., 209 (2011), pp. 63–72.
[29] A. Shapiro, D. Dentcheva, and A. Ruszczyński, *Lectures on Stochastic Programming: Modeling and Theory*, MPS-SIAM Ser. Optim., 9, SIAM, Philadelphia, 2009.
[30] A. Shapiro and T. Homem-de-Mello, *A simulation-based approach to two-stage stochastic programming with recourse*, Math. Programming, 81 (1998), pp. 301–325.
[31] R. M. Van Slyke and R. J.-B. Wets, *L-shaped linear programs with applications to optimal control and stochastic programming*, SIAM J. Appl. Math., 17 (1969), pp. 638–663.
[32] D. S. Yakowitz, *A regularized stochastic decomposition algorithm for two-stage stochastic linear programs*, Comput. Optim. Appl., 3 (1994), pp. 59–81.
[33] G. Zakeri, A. B. Philpott, and D. M. Ryan, *Inexact cuts in Benders decomposition*, SIAM J. Optim., 10 (2000), pp. 643–657.