Working Paper

On the Regularized Decomposition Method for Two Stage Stochastic Linear Problems

Andrzej Ruszczyński
Artur Świętanowski

WP-96-014
February 1996

IIASA International Institute for Applied Systems Analysis □ A-2361 Laxenburg □ Austria
Telephone: +43 2236 807 □ Fax: +43 2236 71313 □ E-Mail: info@iiasa.ac.at

On the Regularized Decomposition Method for Two Stage Stochastic Linear Problems

Andrzej Ruszczyński
Artur Świętanowski

WP-96-014
February 1996

International Institute for Applied Systems Analysis ◻ A-2361 Laxenburg ◻ Austria
Telephone: +43 2236 807 ◻ Fax: +43 2236 71313 ◻ E-Mail: info@iiasa.ac.at

Abstract

A new approach to the regularized decomposition (RD) algorithm for two stage stochastic problems is presented. The RD method combines the ideas of the Dantzig–Wolfe decomposition principle and modern nonsmooth optimization methods.

A new subproblem solution method using the primal simplex algorithm for linear programming is proposed and then tested on a number of large scale problems. The new approach makes it possible to use a more general problem formulation and thus allows considerably more freedom when creating the model. The computational results are highly encouraging.

Keywords: stochastic programming, decomposition, nonsmooth optimization.

On the Regularized Decomposition Method for Two Stage Stochastic Linear Problems

Andrzej Ruszczyński
*Artur Świętanowski**

1 Introduction

Let A be an $m_1 \times n_1$ matrix, $c \in \mathbb{R}^{n_1}$ and $b \in \mathbb{R}^{m_1}$. Next, let T_ω be an $m_2 \times n_1$ random matrix and $d_\omega \in \mathbb{R}^{m_2}$, $l^y_\omega \in \mathbb{R}^{n_2}$, $u^y_\omega \in \mathbb{R}^{n_2}$ and $q_\omega \in \mathbb{R}^{n_2}$ be random vectors over a probability space $(\Omega, \mathcal{B}, P)$; ω denotes an elementary event in this space. We consider the following *two-stage stochastic programming problem*:

$$\min \left[c^T x + \int q^T_\omega y_\omega P(d\omega) \right] \tag{1.1}$$

subject to

$$Ax = b, \tag{1.2}$$

$$l^x \le x \le u^x, \tag{1.3}$$

and, for P-almost all $\omega \in \Omega$,

$$T_\omega x + W y_\omega = d_\omega, \tag{1.4}$$

$$l^y_\omega \le y_\omega \le u^y_\omega. \tag{1.5}$$

Here $x \in \mathbb{R}^{n_1}$ denotes the deterministic *first stage decision*, while $y_\omega \in \mathbb{R}^{n_2}$ is the *recourse decision*, which may depend on the elementary event $\omega \in \Omega$.

There are many practical problems that lead to models of form (1.1)-(1.5). For example, in the linear programming problem

$$\min \ c^T x$$

$$Mx = r,$$

$$l^x \le x \le u^x,$$

some coefficients of the resource/demand vector r or some entries of the matrix M may be uncertain. They can be modeled as random variables $r(\omega)$ and $M(\omega)$, but then the constraints

$$M(\omega)x = r(\omega), \quad \omega \in \Omega,$$

*This research was partially sponsored by a grant of the Scientific Research Committee of Poland no. 3 P403 018 06.

become prohibitively restrictive and usually impossible to satisfy for all realizations of the random entries. By splitting the constraints into the deterministic and the random parts:

$$M(\omega) = \begin{bmatrix} A \\ T_\omega \end{bmatrix}, \; r(\omega) = \begin{bmatrix} b \\ d_\omega \end{bmatrix},$$

and introducing corrective activities $y_\omega \in \mathbb{R}^{n_2}$ to take care of the random shortage/surplus vector $d_\omega - T_\omega x$, we arrive to (1.1)-(1.5).

It should be stressed that the two stage stochastic programming problem is not just a formal trick to correctly pose linear programs with uncertainty. It can be used to model decision problems in which two types of decisions can be distinguished: deterministic ones, which have to be determined before the uncertain quantitities are known, and "on-line" decisions that can be dependent on the observation of uncertain data.

While there seems to be no doubt as to theoretical advantages of using models of form (1.1)-(1.5), their solution is much more difficult than for underlying deterministic models. We shall focus our attention on problems with discrete distributions; approximation of general distributions by discrete ones in stochastic programming is discussed in [2] and [11].

Let Ω be finite, $\Omega = \{1, 2, \ldots, L\}$, and let the realizations (T_ω, d_ω), $\omega \in \Omega$, be attained with probabilities $p_\omega > 0$, $(\sum_{\omega \in \Omega} p_\omega = 1)$. Then (1.1)-(1.5) can be rewritten as a large linear programming problem

$$\begin{array}{lccccccccccc}
\min & c^T x & + & p_1 q_1^T y_1 & + & p_2 q_2^T y_2 & + & \ldots & + & p_L q_L^T y_L & & \\
& Ax & & & & & & & & & = & b, \\
& T_1 x & + & W y_1 & & & & & & & = & d_1, \\
& T_2 x & & & + & W y_2 & & & & & = & d_2, \\
& \vdots & & & & & & \ddots & & & & \vdots \\
& T_L x & & & & & & & + & W y_L & = & d_L,
\end{array} \tag{1.6}$$

$$l^x \leq x \leq u^x,$$

$$l^y_\omega \leq y_\omega \leq u^y_\omega, \; \omega = 1, 2, ..., L.$$

There are several reasons for studying (1.6) thoroughly.

First of all, it is the remarkable size that makes this problem difficult from the practical point of view. Stochastic programs are usually extensions of deterministic linear models, so we should think of T_ω having size of a constraint matrix in a typical linear program, and this size is multiplied in (1.6) by the number L of realizations of (T_ω, d_ω). For nontrivial problems with many independent random factors causing the stochasticity of the entries of T_ω and d_ω, L must be sufficiently large to reflect this randomness in our model. As a result, the dimension of (1.6) may go in hundreds of thousands.

Another difficulty is the possibility of ill-conditioning of (1.6). If the number of first stage activities x in the optimal basis exceeds $m_1 + m_2$, then the similarity of the realizations T_ω, $\omega \in \Omega$, implies that the columns corresponding to these activities are close to being linearly dependent (for $T_\omega \equiv T$ singularity would occur).

A very rich literature is devoted to solution methods for problems of form (1.6) or their duals. The first group of methods are variants of the simplex method which take advantage

of the structure of the constraint matrix of (1.6) to construct compact representations of the basis inverse and to improve pivotal strategies (cf, e.g., [23]). The second group are linear decomposition methods coming down from the famous decomposition principle of Dantzig and Wolfe [7] (see [5, 21]). Finally, there is a possibility of reformulating (1.6) as a nonsmooth optimization problem and applying to it general non-differentiable optimization algorithms (see, e.g., [9]).

In the regularized decomposition (RD) method, proposed for general large scale structured linear programming problems in [15], we combine the last two approaches: the problem is stated as a nonsmooth optimization problem, but for the purpose of solving it we modify the general bundle method introduced in [13] (and further refined in [12] and many other works, see [9]) by taking full advantage of problem's structure. As a result, a finitely convergent non-simplex method for large structured linear programs can be obtained.

The main purpose of our paper is to specialize the regularized decomposition method to stochastic problems with recourse. We present various techniques developed for exploiting specific structural properties of stochastic programs and for mitigating the computational effort. In particular, a new subproblem solution technique is presented, which takes advantage of the similarity of the subproblems. Finally, results of some computational tests are described, which show that the method is capable of solving stochastic programs of considerable size.

2 The RD method

2.1 Outline

It can be readily seen that if x is fixed in (1.6) then minimization with respect to $y_1, y_2, .., y_L$ can be carried out separately. This leads to the following two-stage formulation

$$\min \left[F(x) = c^T x + \sum_{\omega \in \Omega} p_\omega f_\omega(x) \right] \tag{2.1}$$

subject to

$$x \in X_0 = \{x : \ Ax = b, \ l^x \leq x \leq u^x\}, \tag{2.2}$$

$$x \in X_\omega \quad \text{for} \quad \omega \in \Omega, \tag{2.3}$$

with $f_\omega(x)$ defined as the optimal value of the second stage problem:

$$f_\omega(x) = \min \left\{ q_\omega^T y \mid Wy = d_\omega - T_\omega x, \ l_\omega^y \leq y \leq u_\omega^y \right\}, \tag{2.4}$$

and with

$$X_\omega = \{x : \ f_\omega(x) < \infty\}.$$

We introduce condition (2.3) explicitly to the problem formulation, because we are going to use separate approximations of f_ω and of their domains X_ω.

The functions f_ω are convex and polyhedral and the sets X_ω are convex closed polyhedra [22]. Thus (2.1)–(2.3) can in principle be solved by a method for piecewise linear

problems or by a general algorithm for constrained nonsmooth optimization. Although the pieces of f_ω and the facets of X_ω are not given explicitly, it is possible to extract from the subproblems (2.4) at successive points $x^k,\ k = 1, 2, \ldots,$ information about the piece of f_ω or facet of X_ω active at x^k. If $f_\omega(x^k) < \infty$ an *objective cut* can be obtained:

$$f_\omega(x) \geq \alpha_\omega^k + (g_\omega^k)^T x. \tag{2.5}$$

If the subproblem is infeasible, one can obtain information about a constraint $(\bar{\alpha}_\omega^j, \bar{g}_\omega^j)$ defining X_ω and violated at x^k: a *feasibility cut*

$$\bar{\alpha}_\omega^k + (\bar{g}_\omega^k)^T x \leq 0. \tag{2.6}$$

The pieces (cuts) collected so far can be used to construct lower approximations of the functions f_ω,

$$f_\omega(x) \geq \bar{f}_\omega(x) = \max\{\alpha_\omega^j + (g_\omega^j)^T x,\ j \in J_\omega\},$$

and outer approximations of the sets X_ω

$$X_\omega \subset \bar{X}_\omega = \{x :\ \bar{\alpha}_\omega^j + (\bar{g}_\omega^j)^T x \leq 0,\ j \in \bar{J}_\omega\};$$

J_ω and $\bar{J}_\omega$ are some selected subsets of $\{1, 2 \ldots, k\}$.

Crucial questions that arise in this respect are the following:

- how are the successive points x^k generated?
- how are the cuts at x^k constructed?
- how are the approximations $\bar{f}_\omega$ and $\bar{X}_\omega$ updated?

The most natural method for generating successive points x^k is to solve the linear approximation of (2.1)-(2.3) constructed on the basis of currently available information:

$$\min_{x \in \bar{X}} \left[\bar{F}(x) = c^T x + \sum_{\omega \in \Omega} p_\omega \bar{f}_\omega(x) \right], \tag{2.7}$$

where

$$\bar{X} = X_0 \cap \left(\bigcap_{\omega \in \Omega} \bar{X}_\omega \right).$$

After solving (2.7) we obtain cuts at the current solution, add them to the sets of cuts used previously, solve (2.7) again, etc..

The cutting-plane approach, however, has well-known drawbacks. Initial iterations can be inefficient. The number of cuts increases after each iteration and there is no reliable rule for deleting them. The master problem (2.7) is sensitive with respect to changes in the set of cuts and its conditioning is getting worse when approaching the solution.

For these reasons, in the RD method the linear master (2.7) is modified by adding to its objective a quadratic regularizing term:

$$\min_{x \in \bar{X}} \left[\eta(x) = \frac{1}{2\sigma} \|x - \xi^k\|^2 + c^T x + \sum_{\omega \in \Omega} p_\omega \bar{f}_\omega(x) \right]. \tag{2.8}$$

Here ξ^k is a certain reference point, and σ is a positive parameter. This modification stabilizes the master and makes it possible to delete inactive cuts so that the size of (2.8) is limited. It also facilitates using advanced starting points.

Instead of constructing separate approximations for all f_ω in (2.7), we can also work with a piecewise linear approximation of their weighed sum $f(x) = \sum_{\omega\in\Omega} p_\omega f_\omega(x)$, as it was originally suggested in the L-shaped method of Van Slyke and Wets [21] and general bundle algorithms (see [9] and [12]). This would mean constructing objective cuts for f by averaging (with the weights p_ω) the objective cuts for f_ω. We use here more complicated separate approximations, because aggregation of cuts may slow down convergence of the method, as it was observed in [15] (this idea was also analyzed in [1]). We shall show that it is possible to efficiently process separate approximations for each f_ω by exploiting the structural properties of (2.8).

2.2 Logic

The method generates two sequences: a sequence ξ^k of reference points and a sequence x^k of trial points. Each iteration of the method consists in updating and solving the regularized master problem (2.8), which can be equivalently stated as follows. We introduce variables v_ω , $\omega \in \Omega$, to represent $\bar{f}_\omega(x)$ by inequalities involving objective cuts:

$$(g^j_\omega)^T x + \alpha^j_\omega \le v_\omega,\ j \in J^k_\omega,\ \omega \in \Omega.$$

Using explicit formulations of feasibility cuts (2.6) and putting all the cuts together we can rewrite the master (2.8) in a more compact form

$$\min\ \left[\frac{1}{2\sigma}\|x-\xi^k\|^2 + c^T x + p^T v\right] \tag{2.9}$$

subject to

$$(G^k)^T x + a^k \le (E^k)^T v. \tag{2.10}$$

The set of constraints (2.10) (so called *bundle*) comprises in general three groups of cuts:

(a) selected direct constraints from (1.2)-(1.3);

(b) selected feasibility cuts (2.6) collected at some previously generated trial points $x^j,\ j \in \bar{J}^k_\omega \subset \{0,1,..,k\},\ \omega \in \Omega$;

(c) selected objective cuts (2.5) collected at some previously generated trial points $x^j,\ j \in J^k_\omega \subset \{0,1,..,k\},\ \omega \in \Omega$.

Thus each column of the matrix E^k in (2.10) is either a null vector, if the cut is of class (a) or (b), or the l-th unit vector if the cut belongs to class (c) and approximates $f_l(x)$. There are never more than $n + 2L$ cuts in the bundle.

When solving (2.9)-(2.10), we shall always ensure that the constraints corresponding to positive Lagrange multipliers are linearly independent; they will be called *active* constraints.

There are two phases of the method. At Phase 1 we seek a point which satisfies (2.2)-(2.3). It serves then as a starting point for Phase 2, where we aim at solving (2.1)-(2.3).

Since the Phase 1 algorithm is in fact a special case of the main Phase 2 method, we shall now describe in detail the latter.

Let ξ^0 be a starting reference point satisfying (2.2)-(2.3) and let the initial bundle be given by

$$G^0 = \left[\begin{array}{ccc} g_1 & \cdots & g_L \end{array} \right], \; a^0 = \left[\begin{array}{c} \alpha_1 \\ \vdots \\ \alpha_L \end{array} \right], \; E^0 = I = \left[\begin{array}{ccc} 1 & & \\ & \ddots & \\ & & 1 \end{array} \right],$$

with $(g_\omega, \alpha_\omega)$ describing objective cuts at ξ^0 for $\omega \in \Omega$. The bundle may (but need not) contain also some constraints from (2.2) and some feasibility cuts of form (2.6) inherited from Phase 1.

RD ALGORITHM

Step 1. Solve the master at ξ^k getting a trial point x^k and objective estimates v^k and calculate $\hat{F}^k = c^T x^k + p^T v^k$. If $\hat{F}^k = F(\xi^k)$, then stop (optimal solution found); otherwise continue.

Step 2. Delete from the bundle some members inactive at (x^k, v^k) so that no more than $n_1 + L$ members remain.

Step 3. If x^k satisfies (2.2) then go to Step 4. Otherwise add to the bundle no more than L violated constraints, set $\xi^{k+1} = \xi^k$, increase k by 1 and go to Step 1.

Step 4. For $\omega \in \Omega$ solve (2.4) at x^k.

(a) If the constraints of (2.4) are inconsistent then append to the bundle the feasibility cut (2.6);

(b) else if $f_\omega(x^k) > v^k_\omega$ then append to the bundle the objective cut (2.5).

Step 5. If all subproblems were solvable then go to Step 6, else set $\xi^{k+1} = \xi^k$ and go to Step 7.

Step 6. If $F(x^k) = \hat{F}^k$ or $F(x^k) < F(\xi^k)$ and exactly $n + L$ members were active at (x^k, v^k) then set $\xi^{k+1} = x^k$; otherwise set $\xi^{k+1} = \xi^k$.

Step 7. Increase k by 1 and go to Step 1.

If the starting reference point is not feasible, we can put into the starting bundle artificial cuts $v_\omega \geq -C$, where C is a very large constant, for all the functions $f_\omega(x)$ for which objective cuts are not yet available, and set $F(\xi) = +\infty$.

It follows from the theory developed in [15] that after finitely many steps the method either discovers inconsistency in (2.1)-(2.3) or finds an optimal solution. Our proof for the case $p = [\, 1 \; 1 \; \ldots \; 1 \,]$, $\sigma = 1$ can be trivially extended to arbitrary $p > 0$, $\sigma > 0$. It is worth mentioning that the finite convergence property does not require any additional non-degeneracy assumptions typical for general cutting plane methods and bundle methods (see [20, 12]).

Few comments concerning implementation of Algorithm 1 are in order. The number of bundle members may vary between L and $n_1 + 2L$, but in fact only cuts generated at Step 4 need be stored. If the number of linear constraints in (2.2) is large, various strategies can be used at Step 3, similarly to pricing strategies in linear programming. Finally, one can control the penalty coefficient σ on line, increasing it whenever steps are too short, and decreasing σ when $F(x^k) > F(\xi^k)$ (see section 5).

2.3 Critical scenarios

To solve the regularized master problem an active set strategy can be used [15]. At each iteration we select a subset of constraints (2.10), defined by some submatrices G, a and E of G^k, a^k and E^k, such that E is of full row rank (at least one cut for each f_ω) and $\begin{bmatrix} G \\ E \end{bmatrix}$ is of full column rank. We treat these cuts as equalities and solve the resulting equality constrained subproblem by solving the system of its necessary and sufficient conditions of optimality

$$E\lambda = p, \tag{2.11}$$

$$E^T v + \sigma G^T G\lambda = G^T(\xi - \sigma c) + a \tag{2.12}$$

(for simplicity we drop the superscript k). The solution is given by

$$x = \xi - \sigma(c + G\lambda). \tag{2.13}$$

In the method we alter the active set by adding or deleting cuts until the solution is optimal for (2.9)-(2.10).

Let us look closer at the structure of the set of active constraints in the problem (2.9)-(2.10), and in particular - at the numbers $\mu(\omega)$ of active objective and feasibility cuts for each scenario $\omega \in \Omega$. We define the *degree of criticality* $\delta(\omega)$ of scenario $\omega \in \Omega$ as the number its active cuts decremented by one,

$$\delta(\omega) = \mu(\omega) - 1.$$

Clearly, the degree of criticality of each scenario is non-negative, because there must be at least one objective cut for each $\omega \in \Omega$ in the active set. A scenario $\omega \in \Omega$ is called a *critical scenario* if it has a positive degree of criticality.

Critical scenarios play a special role in the method. Indeed, if a scenario ω is not critical, its recourse cost $f_\omega(\cdot)$ is represented in the active set by a singe objective cut

$$v_\omega \geq (g_{B\omega})^T x + \alpha_{B\omega} = \bar{f}_\omega(x).$$

Thus, at this point it is sufficiently well approximated by a linear function $\bar{f}_\omega(\cdot)$. This allows for an analytical elimination of the scenario ω from the master problem; just the linear part c of the objective function needs to be modified. Consequently, only critical scenarios substantially contribute to the complexity of the master problem.

The number of critical scenarios and the sum of their degrees of criticality can be easily bounded by a number which does not depend on the total number of scenarios L. Indeed, the number of active constraints in (2.9)-(2.10) does not exceed $n_1 + L$ and there

must be at least one active feasibility cut for each $\omega \in \Omega$. Therefore the total degree of criticality of all scenarios

$$\Delta = \sum_{\omega \in \Omega} \delta(\omega)$$

does not exceed $n_1 - \mu_0$, where $\mu_0 \geq 0$ be the number of active direct constraints from (2.2).

Clearly, Δ is also an upper bound on the number of critical scenarios, because degrees of criticality are non-negative. It is worth noting once again that the bound Δ does not grow with L; if $L \gg n_1 - \mu_0$ then the vast majority of scenarios are non-critical. Although we do not know which scenarios are critical (at the current point), but the knowledge, that such a set exists and is small, facilitates the solution. Roughly speaking, we try to guess the critical set and iteratively update it until the correct set is found.

Formally, we select for each scenario one active cut and call it a *basic cut*. The basic cuts form the system

$$G_B^T x + a_B = v \tag{2.14}$$

of dimension L. Other active cuts, which occur only for critical scenarios will be called *nonbasic*. Rearranging the order of cuts so that the basic cuts appear first we shall have $E = \left[\begin{array}{cc} I & N \end{array} \right]$. The nonbasic cuts form the system

$$G_N^T x + a_N = N^T v \tag{2.15}$$

of dimension m. Subtracting (2.14) multiplied by N^T from (2.15) yields *reduced cuts*:

$$\hat{G}^T x + \hat{a} = 0, \tag{2.16}$$

where $\hat{G} = G_N - G_B N$ and $\hat{a} = a_N - N^T a_B$. In other words, each critical scenario is represented by the differences between its nonbasic cuts and its basic cut.

Next, partitioning λ into $\left[\begin{array}{c} \lambda_B \\ \lambda_N \end{array} \right]$, we can use (2.14)–(2.16) to eliminate v and λ_B from (2.11)–(2.12), which yields

$$\sigma \hat{G}^T \hat{G} \lambda_N = \hat{G}^T x_B + \hat{a}, \tag{2.17}$$

where $x_B = \xi - \sigma(c + G_B p)$ is the solution implied by basic cuts alone. The other unknowns in (2.11)-(2.12) are defined by $\lambda_B = p - N\lambda_N$ and $x = x_B - \sigma \hat{G} \lambda_N$.

In this way the large system of necessary and sufficient conditions of optimality has been reduced to a relatively small system (2.17) whose order m never exceeds the number of first stage variables n_1, independently of the number of realizations taken into account. This is a substantial improvement over the linear programming formulation (1.6).

Our approach is close in spirit to the generalized upper bounding (GUB) technique applied to the dual of the master problem (2.9)-(2.10) (see [6]), because (2.11) is a collection of GUB constraints. Using this connection, we can interpret non-critical scenarios as those for which the matrix W in the extended form (1.6) can be replaced by some locally stable basis matrix B_ω, which reduces the block ω to a system of equations. Critical scenarios still have to be represented by (reduced) linear programs.

The system (2.17) can be solved by QR factorization (cf. e.g. [4]). We can also update the solution each time the active set is revised or ξ is changed by Algorithm 1. Since the set

of critical scenarios may change, a number of special transformations need be developed to take full advantage of our reduced formulation. They are specialized versions of general techniques for updating the QR factorization (see [16] for the details).

3 Solving subproblems

3.1 A penalty-based primal simplex method

The general form of a subproblem is

$$\min q^T y \tag{3.1}$$

$$\begin{aligned} Wy &= h \\ l \le y &\le u, \end{aligned} \tag{3.2}$$

where q, W, l, u and h assume for $\omega \in \Omega$ the values q_ω, W_ω, l^y_ω, u^y_ω and $d_\omega - T_\omega x^k$. Let us observe that only the recourse matrix W is fixed, when the scenario ω or the upper level solution x^k change.

A modified version of the primal simplex algorithm has been devised in order to make it possible to solve a long sequence of such problems in an efficient way. In this section we shall highlight those features that are most important for the regularized decomposition scheme (see [19] for the details).

The method uses an exact penalty approach to define a penalty problem

$$\min q^T y + M \sum_{i=1}^{m_2} |t_i| \tag{3.3}$$

s. t.

$$\begin{aligned} Wy + It &= h, \\ l \le y &\le u, \end{aligned} \tag{3.4}$$

in which a vector t of artificial variables is introduced, I denotes an identity matrix and M is a positive penalty factor.

The method starts the calculations from an arbitrary starting point y^0, $l \le y^0 \le u$. Without loss of generality we assume here that $t^0 = h - Wy^0$ is nonnegative. Then problem (3.3)–(3.4) is transformed into a linear program

$$\min \left[q^T y + M e^T t \right] \tag{3.5}$$

s. t.

$$\begin{aligned} Wy + It &= h, \\ l \le y &\le u, \\ t &\ge 0, \end{aligned} \tag{3.6}$$

where $e^T = [1\ 1\ \ldots\ 1]$.

At each iteration the current state of the method is defined by the following objects: a quadratic nonsingular basic matrix $B = [W_B\ I_B]$ of dimension m_2, and a primal solution (y, t) which satisfies the constraints (3.6). It is not required that the values of the nonbasic

variables y_N and t_N (corresponding to the parts W_N and I_N of the constraint matrix, which are not included into the basis matrix) are at their bounds.

For each solution we define the dual vector π as

$$\pi^T = [q_B^T \; Me_B^T]B^{-1}, \tag{3.7}$$

where q_B and e_B denote the subvectors of q and e corresponding to the columns of the basis matrix.

The method proceeds in a usual way, by determining reduced costs for the structural variables $\lambda = q - W^T\pi$ and for the artificials $\mu = Me - \pi$. They are used to select a nonbasic variable to be moved and its direction of change. Then the direction of changes of the basic variables is determined. If the seleted nonbasic variable hits its bound, a new nonbasic variable is selected, etc. If a basic variable hits its bound first, the basis is changed. An artificial variable hitting zero is fixed there by making its both bounds equal to zero.

We need to notice a few relations between problems (3.1)–(3.2) and (3.5)–(3.6):

1. if (3.5)–(3.6) has an optimal solution in which $t = 0$, then (y, π) constitutes an optimal primal-dual solution pair for (3.1)–(3.2),
2. if (3.1)–(3.2) is infeasible then any solution to (3.5)–(3.6) will have $t \geq 0$, $t \neq 0$,
3. if (3.1)–(3.2) is dual infeasible, so is (3.5)–(3.6).

Additionally, there are two potential problems that may only occur when M is too small:

1. problem (3.1)–(3.2) is feasible but the optimal solution to (3.5)–(3.6) has $t \geq 0$, $t \neq 0$,
2. problem (3.1)–(3.2) is dual feasible but (3.5)–(3.6) is not.

In both cases a penalty M_0 exists such that for any $M > M_0$ neither of those problems will occur (see, e.g., [19]). The task of finding such M_0, or rather some M known to be greater than or equal to M_0 is solved as follows.

If the simplex algorithm terminates by indicating unboundedness at iteration at which variable y_j is selected to move, we try to find such value of M that the corresponding reduced cost λ_j changes its sign (and thus y_j ceases to be an attractive candidate). To achieve this we divide the basic cost vector into two parts: one that depends on the penalty, $M \begin{bmatrix} 0 \\ e_B \end{bmatrix}$, and one that does not, $\begin{bmatrix} q_B \\ 0 \end{bmatrix}$. By (3.7), the reduced cost λ_j is then expressed as

$$\lambda_j = q_j - \left(M \begin{bmatrix} 0 & e_B^T \end{bmatrix} + \begin{bmatrix} q_B & 0 \end{bmatrix}\right) B^{-1}w_j, \tag{3.8}$$

where B is the current basis matrix. Determining whether a penalty M_0 exists for which λ_j changes sign is now a matter of simple arithmetics. If it doesn't exist (i.e., λ_j does not depend on M) the problem is declared unbounded. For an artificial variable t_i in an analogous way one can determine for which values of the penalty coefficient the reduced cost

$$\mu_i = M - \left(M \begin{bmatrix} 0 & e_B^T \end{bmatrix} + \begin{bmatrix} q_B & 0 \end{bmatrix}\right) B^{-1}e_i \tag{3.9}$$

becomes non-positive.

Similarly, if the simplex algorithm terminates and produces an optimal solution $(\hat{y}, \hat{t})$ in which $\hat{t} \neq 0$, we determine whether an increase of M will revive the method. We apply the formulae (3.8) and (3.9) again, but this time all non-basic variables are considered. If $\hat{t} \neq 0$ and no profitable reduced cost can be produced by increasing M, the original problem is declared infeasible.

Additionally, for reasons that will be made clear in Section 3.4, if the problem is found to be infeasible, all non-zero artificial variables are forcibly moved into the basis. It is done by performing a number of degenerate iterations.

Owing to the dynamic penalty control criteria described here, the penalty can be initially set to a relatively small value. Experience shows that it needs to be adjusted very rarely (see [19] for detailed analysis and numerical results). This practically eliminates the main drawback of the penalty approach: numerical difficulties due to the introduction of very large numbers.

3.2 Restart procedure

The subproblem solver can restart from numerical values $\hat{y}$ after arbitrary changes to some or all of the vectors q, l, u and h associated with the subproblem. It needs to be stressed that the method can restart from any combination of a non-singular basis and an initial solution.

The restart procedure has three phases. First, the current solution is made feasible with respect to the simple bounds l and u by the projection

$$y_j^0 = \begin{cases} \hat{y}_j & \text{if } l_j \leq \hat{y}_j \leq u_j \\ l_j & \text{if } \hat{y}_j < l_j \\ u_j & \text{if } \hat{y}_j > u_j \end{cases} \qquad j = 1, \ldots, n.$$

In the second phase the new value of the artificial variable vector $t^0 = h - Wy^0$ is computed. The last phase, which is not necessary, but often helps, consists of shifting the variables y_j^0 within the intervals $[l_j, u_j]$ so as to decrease the Euclidean norm $\|t^0\|$. A one dimensional optimization problem

$$\min_{l_j \leq y_j \leq u_j} \|t^0 - w_j(y_j - y_j^0)\|^2$$

is solved for each column w_j, $j = 1, \ldots, n$ of W, followed by possible updates of y_j^0 and t^0. The values of y^0 and t^0 thus obtained are passed to the simplex method as the starting point.

3.3 The initial solution

The regularized decomposition method requires that at each major iteration L subproblems be solved. At the first major iteration one has no choice but restart a subproblem ω_k from the final (optimal or infeasible) solution to some other subproblem ω_l, $l < k$. Exactly one subproblem needs to be solved from a "cold start", while all others may be restarted.

However, from the second major iteration on two possibilities are available.

1. Repeat the abovementioned scheme (as in [16]).

2. For each subproblem use its own solution from the previous iteration as a new starting point.

This is an important choice to make. A fundamental difference is such that in case 1 one always has to deal with possible changes of l, u, q and h while in case 2 only the right hand side h changes between the major iterations (due to the variation of the first stage variables x transferred to the subproblems by the technology matrix T_{ω_k}). It is easy to see that the scenario-dependent differences of l, u, q and d remain constant throughout the computation process. But as the method converges, the differences between the consecutive trial points x^k decrease and so do the differences between solutions to the same subproblem. Indeed, as we recall from the interpretation of non-critical scenarios, the majority of subproblems have locally stable bases and need little or no updating of their solutions.

Additionally, it is clear that case 2 is convenient for truly coarse-grain parallel computations. Each subproblem may be solved and re-solved independently of others, possibly in a separate processing node and with negligible communication overhead (a vector of first stage variables must be broadcast at each major iteration and a cut is sent back from each subproblem afterwards).

3.4 Generating cuts

The coordination algorithm constructs an outer approximation of an epigraph of each subproblem's objective function $f_\omega(x)$.

Optimality cuts

If $f_\omega(x^k) < \infty$ then the algorithm for determining the penalty coefficient M guarantees that $\hat{t} = 0$ at the solution of (3.5)–(3.6). Therefore

$$\begin{aligned} f_\omega(x^k) &= q^T\hat{y} + Me^T\hat{t} \\ &= q_B^T\hat{y}_B + Me_B^T\hat{t}_B + q_N^T\hat{y}_N \\ &= [q_B^T\ Me_B^T]B^{-1}\left(d_\omega - T_\omega x^k - N\hat{y}_N\right) + q_N^T\hat{y}_N \\ &= \hat{\pi}^T\left(d_\omega - T_\omega x^k - N\hat{y}_N\right) + q_N^T\hat{y}_N. \end{aligned}$$

Since the optimal basis is dual feasible, at any other point x

$$f_\omega(x^k) \geq \hat{\pi}^T\left(d_\omega - T_\omega x - N\hat{y}_N\right) + q_N^T\hat{y}_N. \tag{3.10}$$

Indeed, if the bounds on y_B and t_B were removed from problem's formulation, equality in (3.10) would occur.

Consequently, to obtain the standard form of the optimality cut (2.5) we compute

$$g_\omega^k = -T_\omega^T\hat{\pi} \tag{3.11}$$

and

$$\alpha_\omega^k = f_\omega(x^k) - (g_\omega^k)^T x^k. \tag{3.12}$$

Feasibility cuts

Let us assume that subproblem ω has been found infeasible. In such case all non-zero artificial variables are in the final basis, i.e. $\hat{t}_N = 0$. Thus

$$B \begin{bmatrix} \hat{y}_B \\ \hat{t}_B \end{bmatrix} + N\hat{y}_N = d_\omega - T_\omega x^k. \tag{3.13}$$

Let $\hat{t}_i$, $\hat{t}_i > 0$ be the artificial variable with the largest value. Multiplying (3.13) by the inverse of the basis matrix and by the ith unit vector e_i we obtain

$$\hat{t}_i = e_i^T B^{-1} \left(d_\omega - T_\omega x^k - N y_N \right).$$

A change $\Delta x = x^k - x$ of the trial point x^k has to be such that t_i is reduced to zero. Therefore we require that

$$\hat{t}_i - e_i^T B^{-1} T_\omega x^k + e_i^T B^{-1} T_\omega x \leq 0.$$

The above formula is a standard form of a feasibility cut (2.6) in which

$$(\bar{g}_\omega^k)^T = e_i^T B^{-1} T_\omega$$

and

$$\bar{\alpha}_\omega^k = t_i - (\bar{g}_\omega^k)^T x^k.$$

4 Crash

The crash algorithm attempts to find easily and cheaply an approximate solution to the stochastic problem. This solution then becomes the starting point for the regularized decomposition.

Benders (L-shaped) decomposition techniques [21] can hardly take much advantage of an advanced starting point. The initial iterations of those methods may escape far away from any starting point which might be given to them. In case of the RD method the objective of the quadratic master problem is equipped with a regularizing term (in this context calling it a proximal term might be more appropriate). The value of the penalty coefficient $(1/2\sigma)$ from the formula (2.8) allows us to control the steplength. Normally, when no starting information is available, it is initialized with a relatively small value. Thus the method may reach a neighborhood of the optimum in only a few steps. However, if approximate values of the first stage variables are available, we may start the method with a much larger penalty and thus tell it to search the neighborhood of the initial solution first.

Note, that this means that the RD algorithm is capable of restarting after some changes to the stochastic model, e.g., refinement of discretization of the random variable distributions, addition of new constraints, generation of more scenarios, etc.

The crashing method is very simple indeed. A deterministic two stage linear program

$$\min c^T x + q_D^T y \tag{4.1}$$

s. t.

$$\begin{aligned} Ax &= b \\ T_D x + Wy &= d_D \\ l^x \le x &\le u^x \\ l^y_D \le y &\le u^y_D \end{aligned} \tag{4.2}$$

is solved with the revised simplex method. In (4.1)–(4.2) the matrix T_D and vectors q_D, d_D, l^y_D and u^y_D may represent the expected values of the random variables, may be a randomly chosen realization or may correspond to the most likely scenario. The starting point chosen in this way does not have to be feasible; the RD method accepts infeasible starting points.

5 Numerical results

The development of a specialized implementation of the algorithms, ideas and techniques was accompanied by extensive computational experiments aimed at assessment of the usefulness of these techniques. A series of numerical tests have been performed on a number of small, medium and large problems. Table 5 presents a summary of the two-stage problems used.

Problem called `snn` represents a network planning model with random demand [18]. It has 85 independent random variables, each of which has five to ten possible realizations. The total number of scenarios is far too huge to even consider taking them all into account (around 10^{70}). As it is commonly done in such cases, we have decided to take random samples of 10, 50, 100, 500 and 1000 scenarios.

Problem `storm` is a stochastic aircraft scheduling and transportation problem [14] with 118 independent random variables having five possible realizations each. It has a total of $3 \cdot 10^{82}$ scenarios.

Problems `fxm2`, `pltexpA2`, `stormG2` come from a collection of stochastic linear test problems of [10], where further details about their background can be found.

All the experiments with the RD code were done on a CRAY 6400 SuperServer multiprocessor. The code is so far entirely sequential. It was compiled with a non-parallelizing GNU C++ compiler and was always run on a single computation node equipped with a Sparc RISC processor. All times are given in seconds of sequential CPU work as reported by the Solaris operating system function `times()`.

The default version of the method contained the following features:

- Crash procedure based on the expected value problem.
- Restart of each subproblem from its basis and its solution at the previous iteration.
- Initial penalty coefficient in the master $1/\sigma = 1$.
- Update of the penalty parameter σ according to the rules ($\gamma = 0.9$):
  - if $F(x^k) > \gamma F(\xi^k) + (1-\gamma)\hat{F}^k$ ("null step") then decrease σ;
  - if $F(x^k) < (1-\gamma) F(\xi^k) + \gamma \hat{F}^k$ ("exact serious step") then increase σ;

– otherwise ("approximate serious step") keep σ unchanged.

Coefficients 0.5 and 2 were used to decrease/increase σ.

The primal and dual accuracy in the subproblem solver was 10^{-8}, and the master accuracy (in terms of the errors of the objective value predicted by the cuts) was 10^{-8}, too.

The results are collected in Table 2. We see that the number of master iterations practically does not grow when the number of scenarios increases, and that our penalty control mechanism preserves reasonable proportions between serious and null steps. For problems `fxm2` and `pltexpA2` the crash procedure found the optimal solution, and a series of null steps was necessary to collect cuts sufficient to prove optimality. The restart procedure in subproblems seems to be rather successful, too. Many simplex iterations were simple steps in which the values of variables were updated without changes in the basis. The number of basis updates per subproblem per iteration was very small in all cases (the largest proportion was 7 in `ssn` with 1000 scenarios). Finally, it is worth noting that the total solution time was far below the costs of solving these problems by earlier methods, especially for large problems.

In Table 3 the final values of the penalty coefficient $1/\sigma$ are shown. We see that our rules for updating the penalty successfully adapt it to the shape of the function considered. In `ssn` the objective function is very flat and therefore the penalty converged to a relatively small number.

Figure 1 illustrates (in the logarithmic scale) the objective error at succesive iterations for the most difficult problem `ssn` with 1000 scenarios in the default case. Note that at null steps the objective value increases and that the last step was a null step. It is worth observing that the method behaves in a rather regular way.

Figure 2 shows the number of critical scenarios in succesive iterations for the most difficult problem `ssn` with 1000 scenarios. We see that it is very small compared to the total number of scenarios and to the theoretical sizes of the master predicted by the theory. This feature is one of the main reasons for the good performance of the RD method for large problems and virtually negligible costs incurred in the master itself. Since the master is the only non-parallelizable part of the RD method, there are reasons to expect good performance of our approach in truly parallel computing environments. In other problems the number of critical scenarios was even smaller and never exceeded 10.

The next series of experiments aimed at assessing the value of various new techniques introduced into the method.

Table 4 shows the performance of other versions of the method relative to the default version. Threee modified versions were tested.

1. The penalty coefficient $1/\sigma = 0.01$.

2. Each subproblems restarted from the optimal solution of the previous subproblem, instead of its own optimal solution at the previous upper level iteration.

3. Crash option disabled.

Name	Scena-	Stage 1		Stage 2		Deterministic LP		Master	
	rios	Rows	Columns	Rows	Columns	Rows	Columns	Rows	Columns
fxm2	6	92	114	238	343	1520	2172	104	126
	16					3900	5602	124	146
pltexpA2	6	62	188	104	272	686	1820	74	200
	16					1726	4540	94	220
ssn	10	1	89	175	706	1751	7149	21	109
	50					8751	35389	101	189
	100					17501	70689	201	289
	500					87501	353089	1001	1089
	1000					175001	706089	2001	2089
storm	10	1290	121	526	1259	6550	12711	1310	141
	50					27590	63071	1390	221
	100					53890	126021	1490	321
	500					264290	629621	2290	1121
	1000					527290	1259121	3290	2121
stormG2	8	187	121	526	1259	4395	10193	203	137
	27					14389	34114	241	175
	125					65937	157496	437	371
	1000					526187	1259121	2187	2121

Table 1: Two stage test problems – the summary.

Problem	Scenarios	Master Iterations			Simplex Iterations			Time
		Total	Null	Serious	Total	Simple	Full	
fxm2	6	10	10	0	814	482	332	7.6
	16	11	11	0	1462	686	776	14.1
pltexpA2	6	7	7	0	941	469	472	5.3
	16	7	7	0	2490	1220	1270	7.1
ssn	10	21	1	20	9789	2900	6889	19.1
	50	41	3	38	118546	29934	88612	216.3
	100	34	2	32	234637	63618	171019	427.4
	500	95	21	74	2344477	354073	1990404	3837.0
	1000	110	34	76	5388854	755128	4633726	8575.3
storm	10	18	9	9	4212	2477	1735	30.8
	50	33	13	20	24389	11193	13196	171.2
	100	33	11	22	47427	20193	27234	326.2
	500	42	18	24	250992	87377	163615	1851.7
	1000	43	19	24	506109	176875	329234	3834.9
stormG2	8	21	7	14	3850	2274	1576	26.4
	27	25	10	15	12724	7294	5430	83.2
	125	36	16	20	61955	29180	32775	443.4
	1000	37	14	23	489147	171734	317413	3389.2

Table 2: Performance of the default version of the method.

Problem	Scenarios	Final Penalty
fxm2	6	1.600E+01
	16	3.200E+01
pltexpA2	6	1.000E+00
	16	1.000E+00
ssn	10	9.766E-04
	50	9.766E-04
	100	1.953E-03
	500	3.906E-03
	1000	1.953E-03
storm	10	5.000E-01
	50	3.125E-02
	100	7.813E-03
	500	6.250E-02
	1000	7.813E-03
stormG2	8	5.000E-01
	27	1.250E-01
	125	6.250E-02
	1000	1.250E-01

Table 3: Final penalty.

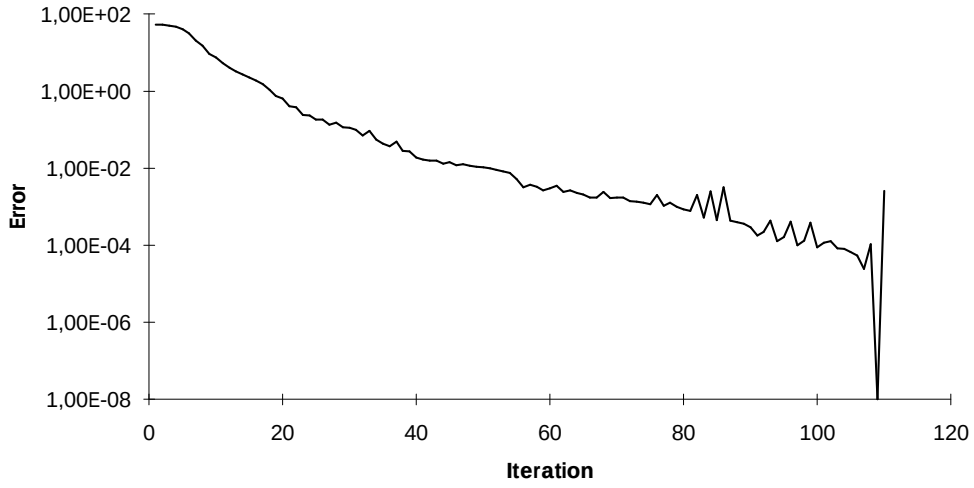

Figure 1: Objective error in the 1000-scenario version of ssn. Non-monotonicity occurs at null steps.

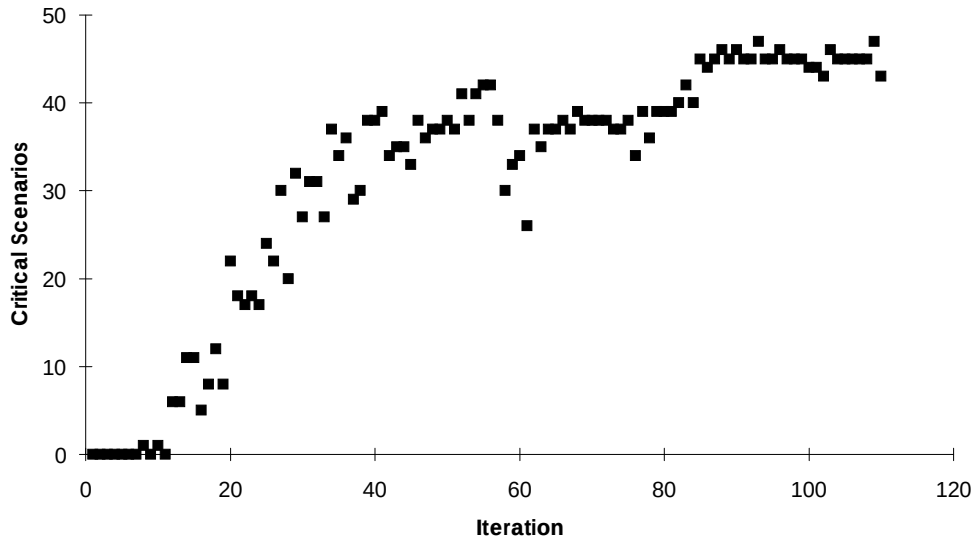

Figure 2: Critical scenarios in the 1000-scenario version of `ssn`.

Problem	Scena-	Small Penalty			Cyclic Restart			No Crash		
	rios	Master	Simplex	Time	Master	Simplex	Time	Master	Simplex	Time
`fxm2`	6	1.00	1.24	1.09	1.00	0.56	0.86	6.30	5.52	4.95
	16	1.27	1.88	1.45	1.00	0.38	0.83	7.00	9.95	6.11
`pltexpA2`	6	1.00	1.00	1.02	1.00	0.89	1.17	26.86	33.15	10.43
	16	1.00	1.00	1.01	1.29	0.96	1.30	45.29	39.90	26.00
`ssn`	10	0.52	0.88	0.93	1.00	3.49	5.03	0.71	1.35	1.55
	50	0.83	0.77	0.83	1.00	2.95	4.84	0.83	0.88	1.07
	100	0.83	0.77	0.83	1.29	3.35	6.29	1.03	1.02	1.08
	500	0.91	0.86	0.87	1.04	3.83	8.03	0.97	0.93	1.06
	1000	1.15	1.07	1.06	0.95	3.55	6.83	1.57	1.32	1.35
`storm`	10	1.56	3.05	2.43	1.39	7.34	2.99	2.06	2.77	2.33
	50	0.94	2.41	1.90	0.85	7.15	2.69	0.88	2.20	1.84
	100	1.27	2.50	2.12	1.15	9.84	3.79	1.03	2.51	2.11
	500	1.02	2.33	1.86	1.10	11.01	3.85	1.12	2.31	1.91
	1000	1.16	2.38	1.94	0.91	9.17	3.05	1.30	2.45	2.06
`stormG2`	8	1.38	2.45	2.14	1.10	5.27	2.75	0.95	1.97	1.67
	27	1.16	2.54	2.13	1.08	4.51	2.16	0.76	2.17	1.89
	125	0.86	2.28	2.62	0.81	3.46	1.32	0.78	2.09	1.78
	1000	0.86	2.26	2.10	1.16	4.38	1.60	0.81	2.21	1.77

Table 4: Performance of other versions of the method relative to the default.

In the method with small penalty our penalty adjustment mechanism incrased it after some time, but the costs incurred in this transition period were substantial: the total time was ca. 50% higher than in the default case. It is worth noting that the costs were mainly located in the simplex method; a smaller penalty in the master implied longer trial steps and larger differences in the subproblems in successive iterations. Smaller penalty helped in `ssn`, where the objective is extremely flat (cf. Table 3).

In the version without self restarts the solution time was on average 3 times higher, and the increase of the costs was again concentrated in the subproblem solver. The reason for this is simple: when a subproblem is restarted from its own optimal solution at the previous steps, the number of simplex steps decreases, because the first stage decision are convergent to the solution, so the differences between the successive versions of the same subproblem diminish. On the other hand, the differences between distinct subproblems remains, even if x does not change at all.

The method with the crash option disabled is about 4 times worse than the default version. The increase of the costs follows mainly from the increase of the number of master iterations, which implies more subproblem calls, and more simplex iterations, especially at the initial iterations.

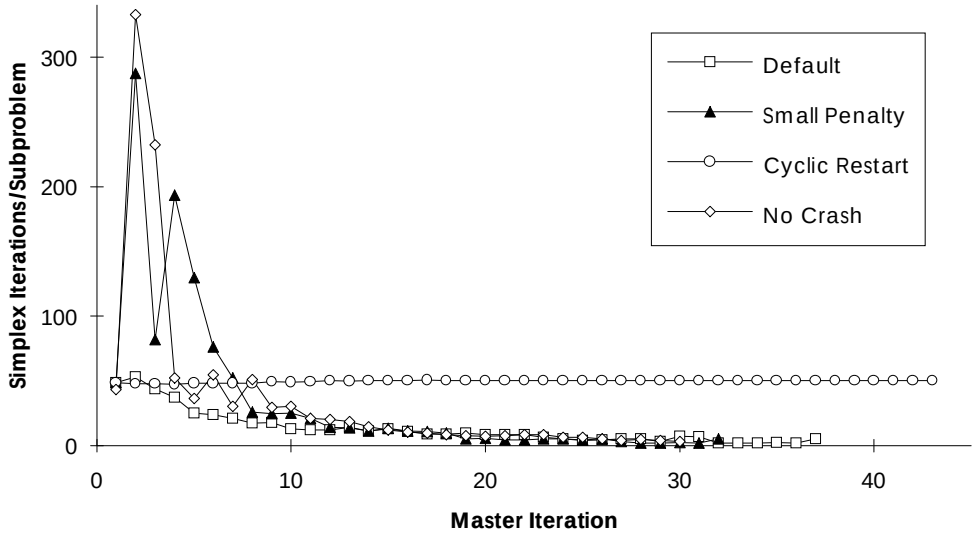

Figure 3: Simplex iterations in the 1000-scenario version of `stormG2`.

All these effects are best seen on the 1000-scenario problem `stormG2`, as illustrated in Figure 3.

Summing up, the seems to be little doubt as to the advantages offered by the enhancements discussed in this paper.

6 Conclusions

The regularized decomposition method appears to be a rather efficient tool for solving large scale two-stage stochastic programming problems. Its efficiency is due to the following features.

1. The quadratic regularizing term stabilizes the master problem and facilitates the accumulation of information about the shape of the function around the current point. It also allows to use advanced starting points generated by a crash procedure.

2. The use of separate approximations for scenario subproblems instead of aggregate (averaged) cuts speeds up convergence owing to the better description of the recourse function.

3. The special algorithm for solving the master problem based on dynamic selection of critical scenarios reduces it to a small numerical core whose size does not depend on the number of scenarios.

4. The penalty-based simplex method allows quick hot starts and reoptimization of the subproblems. In non-critical scenarios the re-optimization is quickly reduced to simple updates of the values of variables without the changes of the basis.

The numerical experience gathered so far indicates that the regularized decomposition method has a potential for solving very large stochastic programming problems. It can also be easily parallelized.

References

[1] J. R. Birge and F. V. Louveaux. A multicut algorithm for two-stage stochastic linear programs. *European Journal of Operations Research*, 34:384–392, 1988.

[2] J. R. Birge and Roger J.-B. Wets. Designing approximation schemes for stochastic approximation problems, in particular for stochastic programs with recourse. *Mathematical Programming Study*, 27:54–102, 1986.

[3] CPLEX Optimization, Incline Village. *Using the CPLEX Callable Library and CPLEX Mixed Integer Library*, 1993.

[4] J. W. Daniel et al. Reorthogonalization and stable algorithms for updating the Gram–Schmidt QR factorization. *Mathematics of Computation*, 30:772–795, 1976.

[5] G. Dantzig and A. Madansky. On the solution of two-stage linear programs under uncertainty. In *Proceedings of the 4th Berkeley Symposium on Mathematical Statistics and Probability*, pages 165–176, Berkeley, 1961. University of California Press.

[6] G Dantzig and R. Van Slyke. Generalized upper bounding techniques. *Journal of Computer System Science*, 1:213–226, 1967.

[7] G. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

[8] H.I. Gassmann. MSLiP: A computer code for the multistage stochastic linear programming problem. *Mathematical Programming* 47 (1990) 407-423.

[9] J.-B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms*. Springer-Verlag, Berlin, 1993.

[10] D. Holmes. A collection of stochastic programming problems. Technical Report 94-11, Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor 1994.

[11] P. Kall, A. Ruszczyński, and K. Frauendorfer. Approximation techniques in stochastic programming. In Y. Ermoliev and R. J.-B. Wets, editors, *Numerical Techniques for Stochastic Optimization*, pages 33–64. Springer–Verlag, Berlin, 1988.

[12] Krzysztof C. Kiwiel. *Methods of Descent for Nondifferentiable Optimization.* Springer–Verlag, 1985.

[13] C. Lemaréchal. Nonsmooth optimization and descent methods. Research Report 4–78, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1978.

[14] J.M. Mulvey and A. Ruszczyński. A new scenario decomposition method for large-scale stochastic optimization. *Operations Research* 43(1995) 477-490.

[15] A. Ruszczyński. A regularized decomposition method for minimizing a sum of polyhedral functions. *Mathematical Programming*, 35:309–333, 1986.

[16] A. Ruszczyński. Regularized decomposition of stochastic programs: algorithmic techniques and numerical results. Working Paper WP-93-21, International Institute for Applied Systems Analysis, Laxenburg, 1993.

[17] A. Ruszczyński. On the regularized decomposition method for stochastic programming problems, in: *Stochastic Programming: Numerical Techniques and Engineering Applications*. K. Marti and P. Kall (eds.), *Lecture Notes in Control and Information Sciences* 423(1995), *Springer-Verlag*, Berlin 1995, pp. 93-108.

[18] S. Sen, R.D. Doverspike and S. Cosares. Network planning with random demand, technical report, Department of Systems and Industrial Engineering, University of Arizona, Tucson, 1992.

[19] A. Świętanowski. A penalty based simplex method for linear programming. Working Paper WP-95-005, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1995.

[20] J. M. Topkis. A cutting plane algorithm with linear and geometric rates of convergence. *Journal of Optimization Theory and Applications*, 36:1–22, 1982.

[21] R. Van Slyke and R. J.-B. Wets. L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics*, 17:638–663, 1969.

[22] R. J.-B. Wets. Stochastic programs with fixed recourse: the equivalent deterministic program. *SIAM Review*, 16:309–339, 1974.

[23] R. J.-B. Wets. Large scale linear programming techniques in stochastic programming. In Y. Ermoliev and R. J.-B. Wets, editors, *Numerical Techniques for Stochastic Optimization*, pages 65–93. Springer–Verlag, Berlin, 1988.