INFORMS Journal on Computing
Vol. 19, No. 2, Spring 2007, pp. 239–247
ISSN 1091-9856 | EISSN 1526-5528 | 07 | 1902 | 0239

DOI 10.1287/ijoc.1050.0163

Computational Assessment of Nested Benders and Augmented Lagrangian Decomposition for Mean-Variance Multistage Stochastic Problems

Panos Parpas, Berç Rustem
Department of Computing, Imperial College, London SW7 2AZ, United Kingdom
{pp500@doc.ic.ac.uk, br@doc.ic.ac.uk}

We consider decomposition approaches for the solution of multistage stochastic programs that appear in financial applications. In particular, we discuss the performance of two algorithms that we test on the mean-variance portfolio optimization problem. The first algorithm is based on a regularized version of Benders decomposition, and we discuss its extension to the quadratic case. The second algorithm is an augmented lagrangian method. Our results indicate that the algorithm based on regularized Benders decomposition is more efficient, which is in line with similar studies performed in the linear setting.

Key words: stochastic programming; large-scale optimization; quadratic programming; mean-variance optimization
History: Accepted by William J. Cook, Area Editor for Decision and Analysis of Algorithms; received June 2003; revised March 2004, June 2005, August 2005; accepted September 2005.

1. Introduction

We consider decomposition algorithms for multistage stochastic programs with a convex quadratic cost function. In particular we examine efficient solution algorithms for the mean-variance portfolio optimization problem. Stochastic programming (SP) is becoming an increasingly popular tool for modeling decisions under uncertainty because of the flexible way uncertain events can be modeled, and real-world constraints can be imposed with relative ease. SP also injects robustness to the optimization process. Consider the following standard "deterministic" quadratic program:

$$\begin{aligned} \min_x \quad & \tfrac{1}{2}x'Hx + c'x \\ \text{s.t.} \quad & Ax = b \\ & x^l \le x \le x^u. \end{aligned} \tag{1}$$

It is not always possible to know the exact values of the problem data of (1) given by H, A, c, and b. Instead, we may have some estimations in the form of data gathered either empirically or known to be approximated well by a probability distribution. The SP framework allows us to solve problems where the data of the problem are represented as functions of the randomness, yielding results that are more robust to deviations.

The power and flexibility of SP does, however, come at a cost. Realistic models include many possible events distributed across several periods, and the end result is a large-scale optimization problem with hundreds of thousands of variables and constraints. Models of this scale cannot be handled by general-purpose optimization algorithms, so special-purpose algorithms attempt to take advantage of the specific structure of SP models. We examine two decomposition algorithms that had encouraging results reported in linear SP; the first is based on the regularized version of Benders decomposition developed by Ruszczynski (1986), and the second on an augmented-lagrangian-based scheme developed by Bertsekas and Tsitsiklis (1989).

Others (Blomvall and Lindberg 2002, Salinger and Rockafellar 2003, Steinbach 1998) formulated multistage SP as a problem in optimal control, where the current stage variables depend on the parent node variables, and used techniques from optimal control theory to solve the resulting problem. Where inequality constraints were present in the model, Blomvall and Lindberg (2002) used a logarithmic barrier function to incorporate them into the objective and solved the resulting approximate problem. Another related method is the approximation algorithm by Frauendorfer (1996) where a sequence of scenario trees is generated whose solution produces lower and upper bounds on the solution of the true problem.

Our approach differs from the above formulations in that we decompose the problem so that each node is represented by a subproblem. We compensate for the "loss" of information of the decomposition by using cuts, in the case of Benders decomposition, or

a penalty term in the case of augmented lagrangian decomposition. Similar studies between the two proposed algorithms have been done for the linear case by Vladimirou (1998) and Ruszczynski (1993); the contribution of this paper is to study the algorithms for quadratic problems.

In Section 2, we describe the problem in greater detail. In Section 3 we discuss the two decomposition algorithms. In Section 4, we detail the results from our numerical experiments, and we conclude in Section 5.

2. Problem Statement

We consider a quadratic multistage SP. In the linear case, SP was first proposed independently by Dantzig (1955) and Beale (1955); for a more recent description see Birge and Louveaux (1997) and Kall and Wallace (1994). For two stages, the problem is

$$\min_x \ \tfrac{1}{2}x'Hx + c'x + \mathcal{Q}(x) \tag{2a}$$

$$\text{s.t.} \ \ Ax = b \tag{2b}$$

$$x^l \le x \le x^u. \tag{2c}$$

We use $'$ to denote the transpose of a vector or a matrix. c and $x^{u,l}$ are known vectors in $\Re^{n_1}$. Let A and H be known matrices in $\Re^{m_1 \times n_1}$ and $\Re^{n_1 \times n_1}$. These quantities represent the state of the world that is known. We assume that H is positive semidefinite. To relate this information to the portfolio-optimization problem, the decision vector $x \in \Re^{n_1}$ is the weight representing the commitment of the investor to different assets; constraints (2b) and (2c) impose various restrictions such as normalization of the weights, upper and lower bounds on the decision variables, etc. The first two terms in the objective function (2a) model the goals of the decision maker that do not depend on uncertain events. $\mathcal{Q}(x)$ represents the expected value of the second-stage objective function:

$$\mathcal{Q}(x) = E_\xi[Q(x, \xi(\omega))],$$

where

$$Q(x, \xi(\omega)) = \min_y \ \tfrac{1}{2}\alpha y'(\omega)H(\omega)y(\omega) - (1-\alpha)c'(\omega)y(\omega) \tag{3}$$

$$\text{s.t.} \ \ W(\omega)y(\omega) = h(\omega) - T(\omega)x \tag{4}$$

$$y(\omega)^l \le y(\omega) \le y(\omega)^u. \tag{5}$$

Let Ω be the set of all random events, and $\omega \in \Omega$ be the particular realization of an event so that when ω is known the random events are aggregated in the vector $\xi(\omega) = [y(\omega), H(\omega), W(\omega), h(\omega), T(\omega), y^{u,l}(\omega)]$, and let Ξ be the support of ξ. The uncertainty of the second stage is represented by the random data $H(\omega)$, $W(\omega)$, and $T(\omega)$, which are matrices in $\Re^{n_2 \times n_2}$, $\Re^{m_2 \times n_2}$, and $\Re^{m_2 \times n_1}$ respectively. The vectors $c(\omega)$, $h(\omega)$, and $y^{l,u}(\omega)$ are random vectors in $\Re^{n_2}$, $\Re^{m_2}$, and $\Re^{n_2}$ respectively. We assume that the number of possible realizations of ω is finite. Under this assumption, $\xi(\omega)$ is taken to mean that for different ω's the data of the problem change. The dependence of y on uncertainty is depicted as $y(\omega) \in \Re^{n_2}$. The vector $y(\omega)$ is still the decision variable but this notation is used to stress the point that for different realizations of ω we must have a different y. In the objective function (3), the quadratic term represents the risk of the investment measured by variance, while the linear term represents the returns for particular realizations. The scalar $\alpha \in [0, 1]$ is used in (3) to describe the trade-off between risk and return. Solving the problem for various α's will generate points on the efficient frontier.

The constraints facilitate transactions, i.e, buying and selling assets (with transaction costs) and re-balancing constraints; they obviously depend on the previous stage. The decision variable $y(\omega)$ is actually composed of three sub-vectors $y(\omega)' = [w(\omega)\ b(\omega)\ s(\omega)]$, where w, b, and s are weights, buy, and sell variables, respectively. For ease of exposition they are aggregated into a single vector. A more detailed description of the model is in Gulpinar et al. (2003).

Deriving the multi-stage problem from the two-stage formulation is just a matter of applying the ideas described above recursively to attain the required number of stages. In this context, stages can be interpreted as time periods where the investor has the opportunity to change the composition of the portfolio.

For the multistage problem with T_s periods, the first-stage decision remains the same but for $t = 2, \ldots, T_s$ we have

$$\mathcal{Q}_t(x_{t-1}) = E_{\xi_t}[Q_t(x_{t-1}, \xi_t(\omega))], \tag{6}$$

where

$$Q_t(x_{t-1}, \xi_t(\omega)) = \min_y \ \alpha\tfrac{1}{2}y_t'(\omega)H_t(\omega)y_t(\omega) - \gamma(1-\alpha)c_t'y_t(\omega) + \mathcal{Q}_{t+1}(y_t(\omega))$$

$$\text{s.t.} \ \ W_t(\omega)y_t(\omega) = h_t(\omega) - T_{t-1}(\omega)x_{t-1}$$

$$y_t(\omega)^l \le y_t(\omega) \le y_t(\omega)^u \tag{7}$$

$$\gamma = \begin{cases} 1 & \text{if } t = T_s \\ 0 & \text{otherwise.} \end{cases}$$

Note that γ appears in the objective function of (7) to enact our goal to maximize returns at the end of the planning period; it is possible or even desirable to have different values for γ in the intermediate planning periods. For the last time period $t = T_s$, the recourse function $\mathcal{Q}_{T_s+1}$ is zero.

Our principal concern involves decomposition algorithms for (7). For more insight into the properties

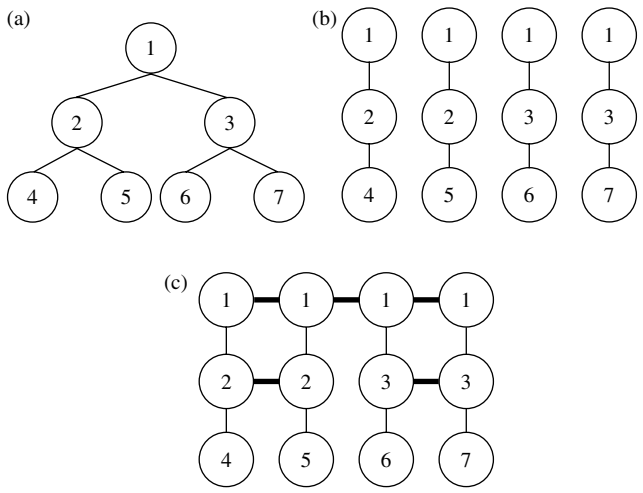

Figure 1 Different Views on Nonanticipativity

of stochastic quadratic problems the reader is referred to Lau and Womersley (2001), and Louveaux (1978). Before we delve into decomposition algorithms, we introduce some terminology that will be used in the next section.

The dynamic programming model (7) is usually referred to as *nonanticipative*. This property means that decisions are based on the past and not the future. There are two ways this concept can be represented, namely compact and split-view formulations (see, e.g, Rockafellar and Wets 1991).

The compact variable formulation can be mapped directly onto a tree structure known as the *scenario tree*; see Figure 1(a). The root of the tree represents the state of the world that is deterministic. As we move down the scenario tree, different events represent different realizations of ω, each level of the tree represents a different time period, and the path from the root to a leaf node is known as a *scenario*. We use $\nu = (t, k)$ to denote the kth node in period t, $a(\nu)$ the ancestor node, and $d(\nu)$ the descendant nodes. Benders decomposition, to be introduced in the next section, assumes such a structure and the result is a decomposition of the large scale problem into several subproblems, each representing a node in the tree.

In a split-variable formulation for each scenario, from the set of possible scenarios, new decision variables are introduced so that the large-scale problem is decomposed into n subproblems, where n is the number of scenarios. Conceptually, using this approach, the nonanticipative constraints are completely relaxed; see Figure 1(b). To enforce these constraints, new constraints are introduced that "rebuild" the links between subproblems, usually through some penalty function (see Figure 1(c)).

3. Decomposition Algorithms

The importance of decomposition algorithms in SP was recognized early on, as results in the theory of stochastic programs are closely linked with their solution algorithms. The two algorithms described in this section represent two very promising approaches in decomposition of SPs. Thus, a study of the two algorithms is necessary to gain insight about which method is more powerful for financial modeling. Decomposition algorithms are not, however, the only approach to tackle the state explosion from which SPs suffer; approximation algorithms and stochastic methods are just two examples of other methods where research is very active (Birge 1997). In this study, we are concerned only with decomposition.

3.1. Nested Benders Decomposition (NBD)

Benders decomposition was first proposed in Benders (1962), and it has been applied to SP by Slyke and Wets (1969); it is usually referred to as the *L-shaped method* due to the structure of the constraint matrix. The extension to the nonlinear convex case has been done by Geoffrion (1972), and the extension to the general convex SP appears in Birge and Rosa (1996). The algorithm has also been widely studied for multistage problems in a parallel environment by Birge et al. (1996), and more recent studies appear in Nielsen and Zenios (1997). Louveaux (1978) has also studied the algorithm for the quadratic case.

It can easily be seen that (2) is equivalent to

$$\begin{aligned} \min_{x,\,\theta} \quad & \tfrac{1}{2}x'Hx + c'x + e'\theta \\ \text{s.t.} \quad & Ax = b \\ & \theta \le p_\omega Q(x, \xi(\omega)) \\ & x^l \le x \le x^u, \end{aligned} \tag{8}$$

where e is a vector of ones. The dimension of the latter vector is equal to the number of nodes in the next period. The expression $p_\omega Q(x, \xi(\omega))$ represents the value of the next stage decision if event ω occurs (with probability p_ω). The dimensions of the rest of the data are the same as in (2). Even though it is possible to aggregate the θ vector to a single variable, computational studies (Birge 1997, Birge et al. 1996) have shown that the reduction of variables did not enhance performance, possibly due to loss of information.

To represent the recourse function in (8), we construct an approximation using outer linearizations. This is achieved by computing cuts (cutting planes). There are two types of cuts: optimality and feasibility. Instead of solving the large-scale problem (8) we solve the relaxed version

$$\begin{aligned} \min_{x} \quad & \tfrac{1}{2}x'Hx + c'x + e'\theta \\ \text{s.t.} \quad & Ax = b \\ & Dx \ge d \qquad (9a) \\ & \theta \ge Gx + g \qquad (9b) \\ & x^l \le x \le x^u, \end{aligned}$$

where (9a) and (9b) represent feasibility and optimality cuts, respectively. The aim of these constraints is to approximate the feasible region of (8). Their construction is detailed in the following two propositions. Feasibility cuts are identical to the linear case and optimality cuts follow from the general convex case; they are stated here to clarify their use in the algorithm.

PROPOSITION 1. *The constraints* (9a) *are supporting hyperplanes to the feasible region of the original objective function* $Q(x)$.

PROOF. Assuming that $t = T$ and for a fixed $\hat{\omega}$, the νth problem in (7) takes the form

$$\begin{aligned} Q(x) = \min_{y} \ & \frac{\alpha}{2} y'Hy - (1-\alpha)c'y \\ \text{s.t.} \ & Wy = h - Tx_{a(\nu)} \qquad (10) \\ & y^l \le y \le y^u. \end{aligned}$$

Assume that this problem is infeasible due to the vector $x_{a(\nu)}$ generated in a subproblem of a previous stage. Consider

$$\begin{aligned} P(y, x_{a(\nu)}) = \min_{y} \ & e'y^+ + e'y^- \\ \text{s.t.} \ & Wy + y^+ - y^- = h - Tx_{a(\nu)} \qquad (11) \\ & y^l \le y \le y^u \qquad (12) \\ & y^{+,-} \ge 0. \end{aligned}$$

Then since the original problem was infeasible due to $x_{a(\nu)}$ we must have $P(\cdot, x_{a(\nu)}) > 0$. Let λ be the Lagrange multiplier of the constraint in (11). By duality, we must also have $\lambda'(h - Tx_{a(\nu)}) \le 0$. Set $D = \lambda'T$ and $d = \lambda'h$ to obtain (9a), a supporting hyperplane to $Q(x)$. To apply this result when $t \ne T$ just note that the same procedure is recursively applied by taking under consideration the additional constraints from cuts of other subproblems. □

PROPOSITION 2. *The constraints* (9b) *are supporting hyperplanes to the original objective function* $Q(x)$.

PROOF. Again, we start with the problem in (10). Let x_k be the solution vector of a subproblem in the previous stage. By the gradient inequality we must have (ω is dropped since it is clear from context)

$$\begin{aligned} Q(x) &\ge Q(x_k) + \nabla Q(x_k)(x - x_k) \\ Q(x_k) &= \frac{\alpha}{2} y'Hy - (1-\alpha)c'y + \lambda'(Wy - h + Tx_k) \\ \nabla Q(x_k) &= \lambda'T. \end{aligned}$$

Thus $Q(x) \ge \lambda'Tx + (\alpha/2)y'Hy - (1-\alpha)c'y + \lambda' \cdot (Wy - h)$. Set $\theta = Q(x)$, $G = \lambda'T$, and $g = (\alpha/2)y'Hy - (1-\alpha)c'y + \lambda(Wy - h)$ to obtain (9b). Since we require a lower support for the expected value, we then multiply G and g by the probability of ω taking the particular realization of $\hat{\omega}$ for two-stage problems and the conditional probability for multistage problems. The application of optimality cuts when $t \ne T$ is again developed recursively just by taking into account the additional variables and constraints. □

The algorithm proceeds by solving the relaxed problem (9) to obtain a solution vector, known as the *proposal vector*. The latter is then used to solve the subproblems in (10). If a subproblem is feasible then an optimality cut is appended to the constraint set of the ancestor problem (also called the *master problem*). Otherwise, only a feasibility cut is appended.

In the linear case, there are some well known drawbacks to the algorithmic framework developed above (Ruszczynski 1986, Birge 1997). We expect issues similar to the following to manifest themselves in the quadratic case:

- The algorithm tends to be inefficient in early iterations due to the poor description of the original objective function provided by the cuts. Moreover, if a good warm-start is used, the algorithm may deviate significantly from this point, so any efficiency achieved by a good starting point is lost.
- The number of cuts for master problems may increase substantially, adding considerable computational burden to their solution.

For these reasons Ruszczynski (1986) proposed a regularized version of the algorithm; see also Ruszczynski (1993, 1995) for the multistage version. Ruszczynski's results, as well as a study performed by Vladimirou (1998), indicate that the regularized version outperforms the original algorithm.

The basic idea is to add a quadratic term $\rho\|x - \hat{x}\|_2^2$ in the objective function, where $\hat{x}$ is chosen as the "best" current point, in a way to be made precise, and ρ is a penalty parameter. For a high value of ρ the algorithm is penalized from deviating from the current point. In Ruszczynski (1986), the convergence of the algorithm for $\rho = \frac{1}{2}$ was established for the convex case. The regularizing term stabilizes the behavior of the algorithm between iterations, enables valid deletion schemes of the cuts, and avoids degenerate iterations that would otherwise be possible.

The original problem is now decomposed into three types of subproblems. The first type is for the root node. The following problem is solved at each iteration:

$$\begin{aligned} \min_{x,\theta} \ & \frac{1}{2}\alpha x'Hx - (1-\alpha)c'x + e'\theta + \frac{\rho}{2}\|x - \hat{x}\|_2^2 \\ \text{s.t.} \ & Ax = b \\ & Gx \ge \theta + g \\ & Dx \ge d \\ & x^l \le x \le x^u. \end{aligned}$$

The second type is for nonterminal nodes. The following subproblem needs to be considered:

$$\begin{aligned}\min_{y_\nu,\,\theta_\nu}\ & \frac{a}{2}y_\nu' H_\nu y_\nu - (1-\alpha)c_\nu' y_\nu + e'\theta_\nu + \frac{\rho_\nu}{2}\|y_\nu - \hat{y}_\nu\|_2^2 \\ \text{s.t.}\ & W y_\nu = h_\nu - T_{a(\nu)}x_{a(\nu)} \\ & G_\nu y_\nu \geq \theta_\nu + g_\nu \\ & D_\nu y_\nu \geq d_\nu \\ & y_\nu^l \leq y_\nu \leq y_\nu^u.\end{aligned} \tag{13}$$

The third type is for terminal nodes. This type of subproblem is identical to (13) without, of course, the cuts in the constraint set and the regularizing term in the objective function.

The way cuts are recursively defined and the way subproblems are nested in each other has led this to be referred to as *nested Benders decomposition* (NBD). The algorithm can now be stated as follows:

Step 1. Set the iteration counter $i = 0$ and $t = k = 0$, and let $\hat{x}$ be a feasible point.

Step 2. Construct and solve $\nu(t, k)$ to find the solution vector x_ν^i.

2.1. If the problem is infeasible and $t = 0$ then STOP: the problem is infeasible.

2.2. If the problem is infeasible and $t > 0$, generate an optimality cut (9a) and append it to the constraint set of $a(\nu)$.

2.3. If the problem was optimal and $t > 0$, generate an optimality cut (9b) and append it to the constraint set of $a(\nu)$.

Step 3. Compute

$$\hat{F}(x_\nu^i) = \tfrac{1}{2}x'Hx + c'x + \sum_{j=d(\nu)}\theta_j$$

$$F(\hat{x}_\nu^i) = \tfrac{1}{2}x'Hx + c'x + \sum_{j=d(\nu)}Q_j(x).$$

If $\hat{F} = F$ and $t = k = 0$ then STOP: $\hat{x}$ is optimal;
Else go to Step 4

Step 4. Update the regularizing term:

4.1. If a subproblem returned a feasibility cut then $\hat{x}_\nu^{i+1} = \hat{x}_\nu^i$.

4.2. If $F(x_\nu^i) > F(\hat{x}_\nu^i)$ or $F(x_\nu^i) > \tau F(\hat{x}_\nu^i) + (1-\tau)\hat{F}$, then set $\hat{x}_\nu^{i+1} = \hat{x}_\nu^i$, and increase ρ.

4.3. If $F(x_\nu^i) < F(\hat{x}_\nu^i)$ or $F(x_\nu^i) < \tau F(\hat{x}_\nu^i) + (1-\tau)\hat{F}$, then set $\hat{x}_\nu^{i+1} = x_\nu^i$ and decrease ρ.

4.4. If $F(x_\nu^i) = \hat{F}$, then set $\hat{x}_\nu^{i+1} = x_\nu^i$, and decrease ρ.

Step 5. Set $i = i + 1$, find the next subproblem to solve (see below), and go to Step 2.

There are a few points worth noting about the algorithm. These are (i) the updating scheme for the regularizing term in Step 4, and (ii) the selection of the next subproblem to solve in Step 5.

The updating scheme for the penalty parameter is similar to the one deployed in other studies (Vladimirou 1998, Ruszczynski 1986). Our numerical experiments indicate that this scheme performs well as long as the penalty parameter is never allowed to exceed a certain threshold.

We give some motivation behind the selection of the proximal point (see also Ruszczynski 1986). In Step 4.1, we observe that a subproblem in the next period returned a feasibility cut, so one can assume that the regularizing point $\hat{x}_i$ is "better" than the new point x_i. The reason for this is that the x_i caused infeasibility while the regularizing point $\hat{x}_i$ is always chosen to be feasible. Moreover, x_i will also be infeasible in the next master iteration. Indeed, suppose that the νth subproblem is infeasible due to x_i. The optimal objective-function value of the Phase-I problem given by

$$G_\nu = \min\{y^+ + y^- \mid Wy + y^+ - y^- = h - Tx_i\} \tag{14}$$

must be strictly positive, i.e, $G_\nu > 0$. Let λ be the simplex multiplier of the feasibility problem in (14). Then, by duality, we must have $\lambda^T(h - Tx_i) = G_\nu > 0$. From the proof of Proposition 1 we have that in the next iteration the constraint $\lambda^T(h - Tx) \leq 0$ will be added to the master problem. Consequently, x_i will not satisfy the feasibility constraint above.

A similar situation arises for Step 4.2. Intuitively, since the newly generated point is not "better" (in a descent sense) than the regularizing point, it will be inappropriate to update the regularizing point. Moreover, x_i will again be infeasible in the next iteration. Similarly, for Step 4.2 to hold, the approximation θ of the objective function $Q(x)$ must, for at least one subproblem (say, ν), satisfy $Q(x_i) > \theta_\nu$, assuming that $\tau = 0$. A similar argument can be used for $\tau = 1$, and since the problem is convex, it is also true for $\tau \in [0, 1]$.

Let (y, λ) be fixed as the optimal KKT pair associated with the νth problem, given by (10). By complementary slackness, we have $(\alpha/2)y'Hy - (1-\alpha)\cdot c'y + \lambda'(Wy - h + Tx_i) = Q(x_i)$ (where, for clarity, we dropped the box constraints). The optimality cut for this subproblem is defined as $\theta \geq Q(x_{i+1})$ (see Proposition 2). We already have that at x_i, $Q(x_i) > \theta$ (which is the reason for introducing the cut). So x_i will be infeasible in the next iteration. The same argument can be used for Step 4.3, where the opposite course of action is taken.

In Step 4.4, we note that the approximation and the true objective function are identical. This implies that no new optimality cuts can be generated around the point $\hat{x}_\nu^i$. The description around this point is complete, and consequently the penalty parameter ρ is reduced in order to move away from the current point, and avoid cycling.

The next point that needs clarification is Step 5, where the next problem to be solved is selected. For

this step we used the Wittrock (1983) fast-forward-fast-back (FFFB) scheme. Birge et al. (1996) compared this scheme with alternatives such as forward-first (FF) and back-first (BF), and FFFB was generally best; this was confirmed by Gassmann (1990). In the FF scheme the idea is to go back to a problem in stage $t-1$ only if all the nodes from t to T are optimal. In the BF scheme the transition to the previous time period is always made unless no new cuts exist for that stage. In the FFFB scheme the same direction is maintained as long as possible.

3.2. Augmented Lagrangian Decomposition (ALD)

An alternative algorithm to Benders decomposition described in the previous section is based on the augmented lagrangian and the method of multipliers (Bertsekas 1999). The fundamental difference between NBD and ALD is the way the two algorithms attack nonanticipativity constraints. NBD handles these constraints by having a master problem generating proposals to the subproblems further down the event tree; proposal vectors are affected by "future" nodes by feasibility and optimality cuts. In ALD a different approach is taken: Nonanticipativity constraints are relaxed by expressing the large-scale problem in terms of smaller subproblems that are discouraged from violating the original constraints. The algorithm we use was developed by Bertsekas and Tsitsiklis (1989), so here we only sketch the main idea. ALD was developed and applied to the stochastic quadratic programming setting by Settergren (2001) with encouraging results. Similar algorithms to ALD have been developed for linear stochastic programs (Mulvey and Ruszczynski 1992, 1995).

The expectation in (6) for a given time period can also be written as

$$\begin{aligned} \min_{y} \ & \sum_{i=1}^{m} p_i \left(\frac{\alpha}{2} y_i' H_i y_i - (1-\alpha) c_i' y_i \right) \\ \text{s.t.} \ & W_j y_i = h_j - T_j x_{a(i)} \quad j = 1, \ldots, r \qquad (15) \\ & y_i^l \le y_i \le y_i^u. \end{aligned}$$

The problem in (15) is to be interpreted as follows: At the current time period there are m scenarios, each having different realizations for H, W, c, etc. There are r linking constraints (15) that are linked by the vector $x_{a(i)}$. Bertsekas and Tsitsiklis (1989) decompose this problem by introducing a new variable z as follows

$$\begin{aligned} \min_{y,z} \ & \sum_{i=1}^{m} p_i \left(\frac{\alpha}{2} y_i' H_i y_i - (1-\alpha) c_i' y_i \right) \\ \text{s.t.} \ & W_{ji} y_i = z_{ij} \quad j = 1, \ldots, r;\ i \in I(j) \\ & z_{ij} = h_j - T_j x_{a(i)} \quad j = 1, \ldots, r;\ i \in I(j) \qquad (16) \\ & y_i^l \le y_i \le y_i^u, \end{aligned}$$

where $I(j)$ contains the indices of the subproblems that the jth constraint "crosses," i.e., $I(j) = \{i \mid w_{ji} \neq 0\}$. "Crosses" means that a constraint contains data from more than one subproblem. It is obvious that (15) and (16) are exactly the same problem, but the structure of (16) facilitates a decomposition algorithm via the relaxation of the constraints of (16). Bertsekas and Tsitsiklis (1989) use the method of multipliers for the general problem $\min\{f(x) \mid Ax = b\}$. Let $L_c(x, \lambda)$ denote the associated augmented lagrangian defined by $L_c(x, \lambda) = f(x) + \lambda'(Ax - b) + (c/2)\|Ax - b\|_2^2$, where λ is the vector of multipliers. The general algorithmic framework of the method of multipliers can be described as follows:

Step 1. Initialization: Set the iteration counter $k = 0$, and set $c(0) > 0$. Set $x(0)$, and $\lambda(0)$ as the starting point for the decision variables, and lagrange multipliers, respectively.

Step 2. Compute the next point

$$x(k+1) = \arg\min L_c(x, \lambda(k)).$$

Step 3. Update the Lagrange multiplier vector

$$\lambda(k+1) = \lambda(k) + c(k)(Ax(k+1) - b).$$

Step 4. Update the penalty parameter $c(k)$, and set $k = k+1$. If some convergence criterion is not satisfied go to Step 2.

Applying this general algorithmic framework to (16), the problem is decomposed into m subproblems and the nonanticipativity constraints are enforced through the penalty term in the augmented lagrangian.

The computation for the solution of (16) involves keeping z fixed in order to compute the next incumbent for y, and then keeping y fixed in order to compute the next incumbent for z. Thus, at the kth iteration the following subproblems are solved:

$$\begin{aligned} y_i(k+1) = \arg\min_{\xi} \Bigg\{ & p_i \left(\frac{\alpha}{2} \xi_i' H_i \xi_i - (1-\alpha) c_i' \xi_i \right) \\ & + \sum_{\{j \mid i \in I(j)\}} \left(\lambda_{ji}'(k) W_{ji} \xi_i + \frac{c(k)}{2} (W_{ji}\xi_i - z_{ji})^2 \right) \Bigg\} \\ & \forall\, i = 1, \ldots, n \end{aligned}$$

$$\begin{aligned} z_{ji}(k+1) = \arg\min_{\zeta_{ji}} \Bigg\{ & - \sum_{i \in I(j)} \lambda_{ji}'(k) \zeta_{ji} + \frac{c(k)}{2} \\ & \cdot \sum_{i \in I(j)} (W_{ji}\xi_i - \zeta_{ji})^2 \Bigg\} \quad \forall\, j = 1, \ldots, r \\ \text{s.t.} \quad & \zeta_{ji} = h_j - T_j x_{a(i)} \quad i \in I(j), \end{aligned}$$

followed by an update of the lagrange-multiplier vector $\lambda_{ji}(k+1) = \lambda_{ji}(k) + c(k)(W_{ji} y_i(k+1) - z_{ji}(k+1))$. From a computational point of view the above iterative framework is inefficient because of the alternate minimizations required, making this algorithm

unsuitable for a parallel environment. In our implementation we used the more efficient iteration proposed in Bertsekas and Tsitsiklis (1989):

$$y_i(k+1) = \arg\min_{\xi} \left\{ p_i \left(\frac{\alpha}{2} \xi_i' H_i \xi_i - (1-\alpha) c_i' \xi_i \right) + \sum_{\{j \mid i \in I(j)\}} \left(\lambda_j'(k) W_{ji} \xi_i + \frac{c(k)}{2} \cdot (W_{ji}(\xi_i - y_i(k)) + w_j)^2 \right) \right\}, \quad (17)$$

where $w_j = (1/m_j)(W_j y_i - h_j + T_j x_{a(i)})$, $\lambda_j(k+1) = \lambda_j(k) + (c(k)/m_j)(W_j y_i - h_j + T_j x_{a(i)})$, and m_j denotes the cardinality of $I(j)$. The derivation of this iteration is discussed in Bertsekas and Tsitsiklis (1989, p. 249). The expression in (17) forms the main iteration of the ALD algorithm. In order to have a complete description of the algorithm we need to specify how one can perform the updates of the penalty parameter $c(k)$ and how we tested for convergence.

The obvious convergence criteria for ALD are a test for feasibility and small changes in the objective function. However, it is possible, due to a poor selection of updates for $c(k)$, to reach a suboptimal solution. For this reason, it is vital to check the KKT conditions of the problem in addition to any other stopping criteria. If the KKT conditions are not satisfied while the change in the objective function is small (10^{-6} in our implementation), the update strategy for the penalty parameter appears to have been inappropriate. We performed various experiments with different update strategies for this penalty parameter and found that the strategy that works best on most problems is to start with a small value (0.001) and increase it at every iteration by another small factor (1.05); being more aggressive with the update of this parameter caused the algorithm to terminate prematurely. Note that an arbitrary starting point can be used to start the algorithm. If a feasible solution or the solution from a previous run is available it may be beneficial to start with a higher penalty term.

4. Numerical Experiments

The algorithms were implemented in C++ and integrated with state-of-the-art solvers. The BPMPD solver (Meszaros 1999), a primal-dual interior point method, was used to solve the LP/QP subproblems that arise in the decomposition. NBD is also integrated with the IBM Corporation (1991) OSL library for solution of the phase-I problem in computation of the feasibility cuts.

We tested the algorithms on the problems given in Table 1. Problems prefixed with "wat" are from the Watson family of problems (Finance Research Group 2000). These problems forecast, to a horizon of ten years, asset classes, liabilities, and riskless assets (bank deposits and borrowing). In providing the solvers with the data, the same scenario tree structure was kept, and data that were not used in the model were discarded. The last two problems, prefixed with "sim," were generated using a simulation algorithm (Gulpinar et al. 2004). The latter algorithm takes as input a tree data structure. The basic component of this structure, the tree node, is constructed as the centroid of a cluster of scenarios. Each iteration consists of generating a random sequence, followed by the computation of the centroids. The final scenario tree consists of the centroid of each node.

Table 1 Problem Data

Problem	Scen.	Stages	Col.	Rows
wat16C	16	10	1,443	3,552
wat32C	32	10	2,483	6,112
wat64C	64	10	4,147	10,208
wat128C	128	10	6,643	16,352
wat256C	256	10	9,971	24,544
wat512C	512	10	13,299	32,736
wat768C	768	10	19,942	49,088
wat1024C	1,024	10	26,855	65,440
wat1152C	1,152	10	29,887	73,568
wat1536C	1,536	10	39,847	98,080
wat1920C	1,920	10	49,803	122,592
wat2304C	2,304	10	59,761	147,104
wat2688C	2,688	10	69,719	171,616
sim6250	6,250	6	589,806	156,306
sim10000	10,000	7	753,571	1,995,721

The chosen test set has no specific structure, (e.g., sparsity pattern) and we expect this sample to be representative of the class of quadratic multistage SPs. All problems were solved on a Linux machine with an Intel Xeon 2.6 GHz CPU and 3 GB of RAM.

The solution times for the problems are given in Table 2. Column 1 is for the "original" Benders decomposition, i.e., the same as NBD but without the additional quadratic term. For brevity, we refer to this version of the algorithm as ONBD. The number of iterations for each algorithm are in parentheses. For

Table 2 CPU-Time (secs), (#iter)

Problem	ONBD	NBD	ALD
wat16C	2.87 (11)	2.01 (11)	50.43 (56)
wat32C	4.96 (11)	3.48 (12)	97.89 (62)
wat64C	8.34 (12)	7.82 (12)	151.34 (57)
wat128C	15.53 (12)	10.38 (12)	263.43 (65)
wat256C	23.34 (12)	15.86 (12)	396.96 (68)
wat512C	30.92 (14)	21.01 (13)	76.23 (52)
wat768C	58.08 (12)	40.82 (12)	456.82 (72)
wat1024C	71.23 (13)	51.43 (14)	155.86 (54)
wat1152C	68.98 (12)	48.23 (13)	179.17 (61)
wat1536C	94.25 (13)	63.77 (14)	245.01 (71)
wat1920C	131.54 (14)	89.07 (13)	265.91 (82)
wat2304C	179.96 (13)	117.7 (11)	344.35 (77)
wat2688C	182.27 (14)	125.49 (15)	472.77 (84)
sim6250	1,028.93 (22)	824.99 (19)	–
sim10000	1,338.06 (21)	951.37 (20)	–

Benders decomposition, we counted each visit to the root node as one iteration, while in ALD an iteration occurs every time the decision vector is updated. It is therefore difficult to draw conclusions from the number of iterations. The "−" entry in Table 2 means that the particular problem could not be solved to tolerance. For the solution of the subproblems with BPMPD, we used the same relative duality gap tolerance, $(\|f(x_{k+1}) - f(x_k)\|^2)/(\|f(x_k)\|^2) \leq 10^{-7}$, as the convergence criterion. We made extensive use of the warm-start facilities of BPMPD and restarted the subproblems at each iteration from the solution of the previous iteration. This strategy worked well. An attractive alternative would be to use the solution vector from peer nodes, allowing warm-starting of subproblems with the newest information available; but this strategy would be expensive to generalize in a parallel environment. Whenever an increase in the penalty parameter of NBD (as detailed in Step 4) was required we multiplied ρ by 2. When a decrease was required we multiplied the ρ parameter by 0.8. During the numerical experiments we found that, if this penalty was allowed to exceed certain thresholds, the algorithm may terminate early. For this reason, we set an upper bound of 10 for this parameter. From Figure 2, it is clear that the regularized version of Benders decomposition is the most efficient of the algorithms we considered. This result is in line with similar studies performed in the linear setting. One possible explanation is that the NBD algorithm takes advantage of the constraint structure of multistage stochastic programming problems more effectively. Note that the ALD algorithm can be applied to separable convex problems with a more general constraint structure, while NBD will need to be modified to be applicable to other types of separable problems. SP problems are one of the most frequently occurring class of large-scale problems, so it is important to know whether cutting-plane-type algorithms or lagrangian-based algorithms take advantage of this structure more effectively. Based on the results of our experiments, it seems that the NBD algorithm is substantially better. Furthermore, we found that the penalty parameter often caused notable changes to the convergence times of both NBD and ALD. Finding an update scheme that works for all problems is a difficult task. In ALD, the penalty parameter has two goals. One is forcing feasibility, and the other is keeping iterations close to each other. Thus, a "suboptimal" penalty update scheme may be more damaging than in NBD, which may give some insight to the difference in performance of the two algorithms.

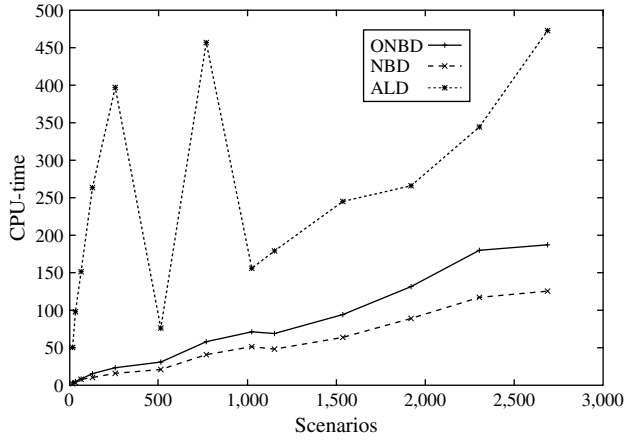

Figure 2 Solution Times vs. Number of Scenarios

The second experiment involved computation of the efficient frontier, i.e., first set α in (7) to zero to maximize the expected returns and then successively increase α until $\alpha = 1$, where the variance of the portfolio is minimized. The purpose of this experiment was to test whether the algorithms can take advantage of a potentially useful starting point. We compared the speed of computation of the efficient frontier when (i) starting with $\alpha = 1$ and then gradually reducing this parameter, with (ii) starting with $\alpha = 0$ and then gradually increasing this parameter. We found that starting from the minimization of variance ($\alpha = 1$) the efficient frontier was computed faster with both versions of Benders decomposition. If we start by optimizing the linear part ($\alpha = 0$) the computation of the efficient frontier took more time. This can be attributed to the fact that, when minimizing the linear part of the problem, useful information about the objective function is not computed until the subproblems including the leaf nodes are solved. Consequently, the first iterations are inefficient. As for the speedup between successive computations of the points on the efficient frontier, the regularized version outperformed the original algorithm but the speedup was disappointing. One possible explanation is that the updating scheme of the penalty term did not allow dramatic speed-ups. Research on how to update this parameter is far from over.

As mentioned earlier, the algorithm at initial iterations may generate points that do not induce cuts that sufficiently describe the recourse function. Since finding the exact *proposal vector* may be counterproductive, Gondzio and Sarkissian (1996) propose solving master problems using an infeasible primal-dual interior-point method with reduced tolerances. Instead of solving master problems to optimality, we solve them until an "acceptable" solution is found. We experimented with this approach by starting with a "large" duality gap tolerance of 10^{-1} and then decreasing this parameter by a factor of 10 at every master iteration. The results where very similar to those of NBD, indicating promise for this approach.

We also observed that more inactive cuts were generated by the algorithm when using suboptimal proposal vectors, and since these cuts are dropped, this could help explain our lack of computational gain. More research is needed to decide how the tolerances of the algorithm should be managed.

5. Conclusions

We have studied performance of two decomposition algorithms to solve large-scale optimization problems that arise in financial applications. The regularized decomposition method, known to perform well in linear stochastic programming, was applied to the quadratic case and tested against the more general augmented lagrangian algorithm. Our results indicate that the regularized decomposition scheme outperforms the other algorithms we implemented. We also came to realize that more work is needed on the penalty parameter update strategy. An intriguing direction would be to find an effective way to reuse cuts between successive solutions of problems with different α's, as well as using inexact proposal vectors.

Acknowledgments

The work of the first author was partially supported by an ORS grant and subsequently EPSRC Grant GR/T02560/01. The authors also wish to acknowledge helpful comments and criticisms of two anonymous referees, the Associate Editor, and the Editor-in-Chief.

References

Beale, E. M. L. 1955. On minimizing a convex function subject to linear inequalities. *J. Roy. Statist. Soc.* **17** 173–184.

Benders, J. F. 1962. Partitioning procedures for solving mixed-variables problems. *Numerische Mathematik* **4** 238–252.

Bertsekas, D. P. 1999. *Nonlinear Programming*, 2nd ed. Athena Scientific, Belmont, MA.

Bertsekas D. P., J. N. Tsitsiklis. 1989. *Parallel, and Distributed Computation*. Prentice-Hall, Englewood Cliffs, NJ.

Birge, J. R. 1997. Stochastic programming computation and applications. *INFORMS J. Comput.* **9** 111–133.

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

Birge, J. R., C. H. Rosa. 1996. Parallel decomposition of large-scale stochastic nonlinear programs. *Ann. Oper. Res.* **64** 39–65.

Birge, J. R., C. J. Donohue, D. F. Holmes, O. G. Svintsitski. 1996. A parallel implementation of the nested decomposition algorithm for multistage stochastic linear programs. *Math. Programming* **75** 327–352.

Blomvall, J., P. Lindberg. 2002. A Riccati-based primal interior point solver for multistage stochastic programming. *Eur. J. Oper. Res.* **143** 452–461.

Dantzig, G. B. 1955. Linear programming under uncertainty. *Management Sci.* **1** 197–206.

Finance Research Group. 2000. Watson pension fund management problem. Technical report, Judge Institute of Management Studies, Cambridge University, Cambridge, UK.

Frauendorfer, K. 1996. Barycentric scenario trees in convex multistage stochastic programming. *Math. Programming* **75** 277–293.

Gassmann, H. 1990. MSLIP, a computer code for the multistage stochastic linear programming problem. *Math. Programming* **47** 407–423.

Geoffrion, A. M. 1972. Generalized Benders decomposition. *J. Optim. Theory Appl.* **10** 237–260.

Gondzio, J., R. Sarkissian. 1996. Column generation with a primal-dual method. Logilab Technical Report 96.6, Department of Management Studies, University of Geneva, Geneva, Switzerland.

Gulpinar, N., B. Rustem, R. Settergren. 2003. Multistage stochastic mean-variance portfolio analysis with transaction costs. A. Nagurney, ed. *Innovations in Financial and Economic Networks*, Vol. 3. Edward Elgar Publishers Ltd., UK, 46–63.

Gulpinar, N., B. Rustem, R. Settergren. 2004. Simulation and optimization approaches to scenario tree generation. *J. Econom. Dynam. Control* **28** 1291–1315.

IBM Corporation. 1991. *Optimization Subroutine Library* (*OSL*)*: Guide and References*. Manual, IBM Corporation, Kingston, NY.

Kall, P., S. W. Wallace. 1994. *Stochastic Programming*. Wiley, Chichester, UK.

Lau, K., R. S. Womersley. 2001. Multistage quadratic stochastic programming. *J. Comput. Appl. Math.* **129** 105–138.

Louveaux. F. 1978. Piecewise convex programs. *Math. Programming* **15** 53–62.

Meszaros, C. 1999. The BPMPD interior solver for convex quadratic problems. Technical report, Department of Computing, Imperial College, London, UK.

Mulvey, J. M., A. Ruszczynski. 1992. A diagonal quadratic approximation method for linear multistage stochastic programming problems. *System Modeling and Optimization* (*Zurich, 1991*), *Lecture Notes in Control and Inform. Sci.*, Vol. 180. Springer, Berlin, Germany, 588–597.

Mulvey, J. M., A. Ruszczynski. 1995. A new scenario decomposition method for large-scale stochastic optimization. *Oper. Res.* **43** 477–490.

Nielsen, S. S., S. A. Zenios. 1997. Scalable parallel Benders decomposition for stochastic linear programming. *Parallel Comput.* **23** 1069–1088.

Rockafellar, T., R. Wets. 1991. Scenarios and policy aggregation in optimization under uncertainty. *Math. Oper. Res.* **16** 119–147.

Ruszczynski, A. 1986. A regularized decomposition method for minimizing a sum of polyhedral functions. *Math. Programming* **35** 309–333.

Ruszczynski, A. 1993. Parallel decomposition of multistage stochastic programming problems. *Math. Programming* **58** 201–228.

Ruszczynski, A. 1995. On the regularized decomposition method for stochastic programming problems. *Stochastic Programming* (*Neubiberg/München, 1993*), *Lecture Notes in Economics and Mathematics Systems*, Vol. 423. Springer, Berlin, Germany, 93–108.

Salinger, D. H. T. Rockafellar. 2003. Dynamic splitting: An algorithm for deterministic and stochastic multiperiod optimization. *Stochastic Programming E-Print Series* (*SPEPS*), http://www.speps.info/.

Settergren, R. 2001. Decomposition of financial engineering problems. Technical report, Department of Computing, Imperial College, London, UK.

Slyke, R. Van, R. J-B. Wets. 1969. L-shaped linear programs with applications to control and stochastic programming. *SIAM J. Appl. Math.* **17** 638–663.

Steinbach, M. C. 1998. Recursive direct algorithms for multistage stochastic programs in financial engineering. P. Kall, H. J. Luthi, eds. *Papers Internat. Conf. Oper. Res.* Springer, New York, 241–250.

Vladimirou, H. 1998. Computational assessment of distributed decomposition methods for stochastic linear programs. *Eur. J. Oper. Res.* **108** 653–670.

Wittrock, R. J. 1983. Advances in a nested decomposition algorithm for solving staircase linear programs. Technical Report SOL 83-2, Department of Operations Research, Stanford University, Stanford, CA.