A NEW SCENARIO DECOMPOSITION METHOD FOR LARGE-SCALE STOCHASTIC OPTIMIZATION

JOHN M. MULVEY

Princeton University, Princeton, New Jersey

ANDRZEJ RUSZCZYŃSKI

International Institute for Applied Systems Analysis, Laxenburg, Austria

(Received July 1992; revisions received February, May, August 1993; accepted August 1993)

A novel parallel decomposition algorithm is developed for large, multistage stochastic optimization problems. The method decomposes the problem into subproblems that correspond to scenarios. The subproblems are modified by separable quadratic terms to coordinate the scenario solutions. Convergence of the coordination procedure is proven for linear programs. Subproblems are solved using a nonlinear interior point algorithm. The approach adjusts the degree of decomposition to fit the available hardware environment. Initial testing on a distributed network of workstations shows that an optimal number of computers depends upon the work per subproblem and its relation to the communication capacities. The algorithm has promise for solving stochastic programs that lie outside current capabilities.

Multistage stochastic programs (MSPs) are among the most intractable in numerical computations. Not only does the size of the problem grow as a quadratic function of the number of scenarios—practical problems can easily exceed 100,000 constraints, but also the problem's structure is difficult to take advantage of due to numerical instability. As a consequence, few actual implementations of MSPs have occurred. Some recent exceptions are discussed in Wets (1988), Mulvey (1991) and Dempster and Ireland (1995).

It seems unlikely that direct solvers will be able to handle realistic-size MSPs in the foreseeable future. Decomposition is the only real alternative. But decomposition generally has an unpleasant characteristic—the number of iterations can become unmanageable, especially as the degree of decomposition increases and the subproblems become a smaller part of the original model. This argues against a massively parallel solution strategy.

Another pertinent issue involves computer technology and, in particular, the ratio of price to performance for various categories of computers. Today, the workstation has an edge over the large mainframes, PCs, and even supercomputers if one can mobilize 10–30 workstations in unison. But proliferating computers over a network becomes unmanageable beyond, say 30–50 computers, due to communication bottlenecks. While these numbers may change over time, it is safe to say that the algorithms we develop must be flexible to handle a variety of computer environments. Software and hardware must match.

The proposed algorithm attempts to overcome these difficulties through a careful choice of approximations. First, the difficult parts of the problem are temporarily relaxed. The nonnegativity constraints are placed in the objective as logarithmic barriers, similarly to the interior point algorithms. The nonanticipativity constraints are also temporarily ignored by means of an augmented Lagrangian. Then the total problem is subdivided in a general way through a diagonal quadratic approximation. Using this approach we break the problem into pieces based on the available computer hardware.

In Sections 1 and 2 we formulate the class of stochastic optimization problems under consideration and we present a new way of describing the nonanticipativity conditions. In Section 3 we discuss the application of the augmented Lagrangian approach to our problem. In Section 4 we introduce a new decomposition method for multistage stochastic programming problems (diagonal quadratic approximation, DQA), and we prove its convergence. Section 5 is devoted to a special interior point algorithm for solving the subproblems. Section 6 describes an experimental implementation of the method, and Section 7 presents results for real-life, large-scale stochastic optimization problems.

1. STOCHASTIC OPTIMIZATION MODELS

Uncertainty may be included in the model in many ways. One that has been used successfully in a variety of situations is to represent the uncertain quantities as random variables. In the case of decision-making problems, this leads to *stochastic optimization* models. In such a model there is some underlying probability space $(\Omega, \mathcal{B}, P)$, a measurable *objective function* f: $\Omega \times R^n \to R$, a measurable multivalued mapping X: $\Omega \to 2^{R^n}$ representing (event-dependent) constraints, a space $\mathcal{X}$ of measurable

Subject classifications: Programming, stochastic: scenario decomposition, parallel computation.
Area of review: OPTIMIZATION.

0030-364X/95/4303-0477 $01.25

decision rules x: $\Omega \to R^n$ and a subspace $\mathcal{M} \subset \mathcal{X}$ of *implementable* decision rules. For each elementary event $\omega \in \Omega$ we denote by X_ω, x_ω and $f_\omega(x_\omega)$ the corresponding constraint set, decision, and objective value. The problem is formulated as follows: Find a decision rule x: $\Omega \to R^n$ that minimizes

$$\int f_\omega(x_\omega)P(d\omega) \tag{1a}$$

subject to $x_\omega \in X_\omega$ with probability 1, (1b)

$$x \in \mathcal{M}. \tag{1c}$$

A particularly interesting and important case arises when the decision problem has a dynamic structure with time stages $t = 1, \ldots, T$ and

$$x_\omega = (x_\omega(1), x_\omega(2), \ldots, x_\omega(T)). \tag{2}$$

Then we can interpret elementary events $\omega \in \Omega$ as *scenarios* and we can use (1b) to represent the conditions that have to be satisfied by the decision sequence (2) for each scenario. Condition (1c) usually represents *nonanticipativity* constraints: For each t decisions $x_\omega(t)$ must be equal for all scenarios ω that have a common past and present. Formally, this can be stated as the condition of measurability of $x_\omega(t)$ with respect to some σ-subfield $\mathcal{B}(t) \subseteq \mathcal{B}$, where $\mathcal{B}(t)$, $t = 1, \ldots, T$ is an increasing sequence of σ subfields.

To be more specific, let us consider the linear case, which will be our main concern in this paper, because of its significance for both theory and applications. We will also restrict our attention to the case of a finite probability space Ω.

Let $D_\omega(t)$ and $H_\omega(t)$, $t = 1, \ldots, T$ be sequences of random $m_b \times m_x$ matrices and $b_\omega(t)$ and $c_\omega(t)$, $t = 1, \ldots, T$ be sequences of random vectors in R^{m_b} and R^{m_x}, respectively. We will call each sequence

$$s_\omega(t) = (D_\omega(t), H_\omega(t), b_\omega(t), c_\omega(t)) \quad t = 1, \ldots, T,$$

corresponding to some event $\omega \in \Omega$ a *scenario*. The problem is to find a collection $x_\omega(t)$, $t = 1, \ldots, T$, $\omega \in \Omega$ of random vectors in R^{m_x} (a *policy*), which minimizes the linear form

$$\sum_{\omega \in \Omega} p_\omega \sum_{t=1}^{T} \langle c_\omega(t), x_\omega(t) \rangle \tag{3a}$$

subject to the constraints

$$D_\omega(t)x_\omega(t-1) + H_\omega(t)x_\omega(t) = b_\omega(t) \quad t = 1, \ldots, T,$$

$$x_\omega(t) \geq 0, \quad t = 1, \ldots, T, \quad \omega \in \Omega, \tag{3b}$$

with $x(0) = x_0$ fixed. The *nonanticipativity constraint* can be formulated as follows: For all $\omega, \zeta \in \Omega$ and any $t \in \{1, \ldots, T\}$

$$x_\omega(t) = x_\zeta(t) \quad \text{if } s_\omega(\tau) = s_\zeta(\tau) \text{ for } \tau = 1, \ldots, t. \tag{3c}$$

In other words, decisions that correspond to scenarios which are indistinguishable up to time t should be equal.

We see a direct correspondence between (1) and (3), but the special case (3) is more informative: In the simplest possible way it reveals the most important features of multistage stochastic programming problems.

First, one has to note the remarkable size of (3). If the scenarios introduced to the model are to reflect uncertainties that occur at successive time stages, then the number S of scenarios grows exponentially with the increase of the time horizon T. Even for relatively small T the dimension of (3) may be so large that the whole problem will become intractable by direct solvers.

However, (3) has a special structure. All scenarios can be arranged into a tree: common data for $t = 1$, some subgroups for $t = 2$, sub-subgroups for $t = 3$, and so on. The nonanticipativity constraint (3c) requires our decisions to have a similar tree-like structure: The first-stage decisions must be the same for all scenarios, because the future is not yet known, second-stage decisions must be equal within some subgroups of scenarios which are indistinguishable on the basis of information available so far, and so on. An example of such a tree for an 8-scenario problem is shown in Figure 1. This creates a number of possibilities for exploiting the structure and developing special methods for solving (3).

Clearly, the development of specialized algorithms for solving any particular class of optimization problems depends upon three primary issues. First, is the problem class wide enough to warrant the design of highly specialized tools when general-purpose software is available? Second, what is the speedup that results when specialized algorithms are executed? Third, do these codes give rise to the solution of problems that would not otherwise be solvable? In the case of multistage stochastic programming each of these questions leads to an affirmative answer.

Existing computational methods for multistage stochastic programming problems can be divided into two main groups. First, there are versions of general-purpose algorithms in which special features of stochastic problems are used to improve data structures and solution strategies (Lustig, Mulvey and Carpenter 1991, Gondzio and Ruszczyński 1992). Second, we have a number of special *decomposition methods* which exploit the structure of the problem to split it into manageable pieces and coordinate their solution (Wets). One can distinguish two classes: *primal* decomposition methods that work with

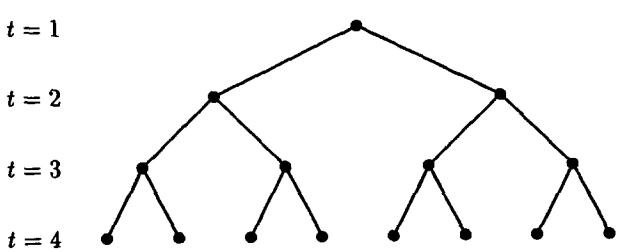

Figure 1. Scenario tree.

subproblems assigned to time stages (Birge 1985, Ruszczyński 1993) and *dual* methods, in which subproblems correspond to scenarios (Rockafellar and Wets 1991, Mulvey and Ruszczyński 1992).

We will develop the dual decomposition method proposed for multistage stochastic programs (and multicommodity networks) in Mulvey and Ruszczyński. We will show how to treat nonanticipativity in a compact way and we will develop a totally distributed version of the method in which there is no coordinating unit and communication between subproblems is restricted to nearest neighbors.

2. NONANTICIPATIVITY

Clearly, conditions (3) describe the linear subspace $\mathcal{M}$ in a massively redundant way. To develop an efficient numerical algorithm for solving multistage stochastic programming problems we need a description convenient for computation.

Rockafellar and Wets equate each decision $x_\omega(t)$ to the average in the *bundle* of scenarios indistinguishable up to time t:

$$x_\omega(t) = E\{x_\xi(t) | s_\xi(\tau) = s_\omega(\tau),\ \tau = 1, \dots, t\},$$
$$t = 1, \dots, T-1,\ \omega \in \Omega. \quad (4)$$

Equations (4) can be put together into

$$x = \Pi x, \quad (5)$$

where Π: $\mathcal{X} \to \mathcal{X}$ is an orthogonal projection on the nonanticipativity subspace $\mathcal{M}$. Conditions (4) have the advantage that they are applicable to arbitrary distributions. However, for a finite Ω conditions (4) define a rather dense set of equations: Each variable enters as many equations as there are scenarios in its bundle.

Another possibility, which we will follow, is to select a subset of (3) sufficient to describe $\mathcal{M}$ and easy to handle in computation. Although our approach is valid for any sufficiently rich subset, we will describe one particular selection, which we find useful for further presentation and in computation.

Let us define the *last common stage* of scenarios ω and ξ by

$$t^{max}(\xi, \omega) = \max\{t: s_\xi(\tau) = s_\omega(\tau),\ \tau = 1, \dots, t\}. \quad (6)$$

We now order scenarios in Ω by assigning to them numbers $i = 1, \dots, S$ in such a way, that for every i scenario $i+1$ has the largest last common stage with i among all scenarios $j > i$:

$$t^{max}(i, i+1) = \max\{t^{max}(i, j): j > i\}.$$

Scenarios in Figure 2 are ordered in this way. It is easy to observe that with such an ordering the bundles form connected subsets of $\{1, \dots, S\}$.

Next, for every scenario i and every time period t we define the *sibling* of i at t as

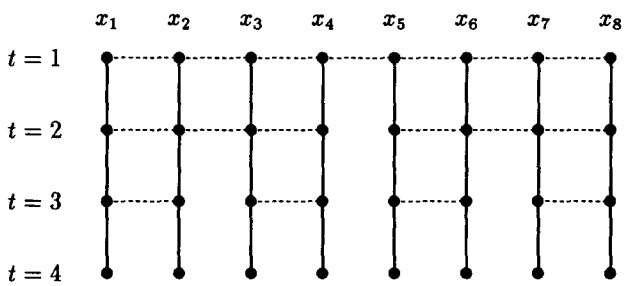

Figure 2. Sequences of decisions and nonanticipativity.

$$\nu(i, t) = \begin{cases} i+1 & \text{if } t^{max}(i, i+1) \geq t, \\ \min\{j: t^{max}(i, j) \geq t\} & \text{otherwise.} \end{cases} \quad (7)$$

Note that a scenario may have different siblings at different time stages. For the example of Figures 1 and 2 siblings of scenarios are shown in Table I.

For every t the mapping $\nu(i, t)$ defines a permutation of Ω, which maps bundles of indistinguishable scenarios onto themselves. It is easy to observe that $\nu(i, t) \neq i$, if the bundle of scenario i at stage t contains more than one member, while $\nu(i, T) = i$. The inverse permutation will be denoted by $\nu^{-1}(i, t)$.

Using the mapping $\nu(i, t)$ we can describe the nonanticipativity subspace $\mathcal{M}$ by the constraints:

$$x_i(t) = x_{\nu(i,t)}(t), \quad i = 1, \dots, S, \quad t = 1, \dots, T-1. \quad (8)$$

There is still some redundancy in this set (we can remove one equation for each bundle), but we keep all equations (8) for convenience.

Adding to (8) the trivial equations for $t = T$ we can write them together as

$$x = Ux \quad (9)$$

with the linear operator U: $\mathcal{X} \to \mathcal{X}$ defined by

$$[Ux]_i(t) = x_{\nu(i,t)}(t). \quad (10)$$

Since $\nu(i, t)$ is a permutation, U is orthogonal, $U^T = U^{-1}$, and there is $l > 1$ such that $U^l = U$. We call U the *scenario rotation* operator.

Equation (9) is formally similar to (5) and describes the same subspace. However, the operators used have a different nature: projection in (5) and rotation in (9).

Table I
Siblings of Scenarios

Time Stage	Scenario 1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8	1
2	2	3	4	1	6	7	8	5
3	2	1	4	3	6	5	8	7
4	1	2	3	4	5	6	7	8

3. THE AUGMENTED LAGRANGIAN

For each scenario i the feasible set defined by (3) has the form

$$X_i = \{x_i : A_i x_i = b_i, x_i \geq 0\}, \quad i = 1, \ldots, S, \tag{11}$$

with some staircase matrices A_i. The nonanticipativity constraints (3) can be put together into some joining constraint (9) (or (5)), which, although simple, is *nonseparable and multidimensional*. Thus, the whole problem has the structure:

$$\text{minimize} \sum_{i=1}^{S} \langle c_i, x_i \rangle \tag{12a}$$

$$\text{subject to} \quad x - Ux = 0, \tag{12b}$$

$$x_i \in X_i, \quad i = 1, \ldots, S. \tag{12c}$$

The well-known *Dantzig–Wolfe decomposition method* (Dantzig and Wolfe 1960) is the classical approach to these problems. It may be viewed as a dual method based on the *Lagrangian function*

$$\begin{aligned} L(x, \pi) &= \sum_{i=1}^{S} \langle c_i, x_i \rangle + \langle \pi, x - Ux \rangle \\ &= \sum_{i=1}^{S} \sum_{t=1}^{T} \langle c_i(t), x_i(t) \rangle \\ &\quad + \sum_{i=1}^{S} \sum_{t=1}^{T-1} \langle \pi_i(t), x_i(t) - x_{\nu(i,t)}(t) \rangle. \end{aligned} \tag{13}$$

The Lagrangian function is separable into terms dependent on x_i, $i = 1, \ldots, S$:

$$L(x, \pi) = \sum_{i=1}^{S} \langle \bar{c}_i, x_i \rangle$$

with

$$\bar{c}_i(t) = \begin{cases} c_i(t) + \pi_i(t) - \pi_{\nu^{-1}(i,t)}(t) & \text{if } t < T, \\ c_i(t) & \text{if } t = T, \end{cases}$$

and can be minimized subject to (12) independently for each x_i. However, updating multipliers π requires solution of a linear *master problem* which has the number of rows equal to the number of rows in (12b) and an unspecified number of columns. This makes the Dantzig–Wolfe method hard to implement for multistage stochastic programming problems.

An alternative dual approach to (12) is the use of the *augmented Lagrangian function*

$$\begin{aligned} \Lambda_r(x, \pi) &= \sum_{i=1}^{S} \langle c_i, x_i \rangle + \langle \pi, x - Ux \rangle + \frac{1}{2} r \|x - Ux\|^2 \\ &= \sum_{i=1}^{S} \sum_{t=1}^{T} \langle c_i(t) x_i(t) \rangle \\ &\quad + \sum_{i=1}^{S} \sum_{t=1}^{T-1} \langle \pi_i(t), x_i(t) - x_{\nu(i,t)}(t) \rangle \\ &\quad + \frac{1}{2} r \sum_{i=1}^{S} \sum_{t=1}^{T-1} \|x_i(t) - x_{\nu(i,t)}(t)\|^2. \end{aligned} \tag{15}$$

Here $r > 0$ is a penalty parameter. The *augmented Lagrangian method* (see, e.g., Bertsekas 1982) can be stated for (12) as follows.

Algorithm 1

STEP 1. For fixed multipliers π^k solve the problem

$$\text{minimize } \Lambda_r(x, \pi^k) \text{ subject to } x \in X_1 \times \ldots \times X_S. \tag{16}$$

Let $x^k = (x_1^k, x_2^k, \ldots, x_S^k)$ be the solution to (16).

STEP 2. If $x_i^k(t) = x_{\nu(i,t)}^k(t)$ for all i and t, then stop (optimal solution found); otherwise set for $i = 1, \ldots, S$ and $t = 1, \ldots, T - 1$

$$\pi_i^{k+1}(t) = \pi_i^k(t) + r(x_i^k(t) - x_{\nu(i,t)}^k(t)), \tag{17}$$

increase k by 1 and go to Step 1.

There are a number of general advantages of the augmented Lagrangian approach over usual dual methods: Simplicity and stability of multiplier iterations and the possibility of starting from arbitrary π^0 are among the most important ones. It is also well known that if (12) has a solution, then Algorithm 1 is finitely convergent.

However, a serious disadvantage of the augmented Lagrangian method is that (15) is not decomposable, so problem (16) cannot be split into S-independent subproblems.

One possibility to overcome this difficulty is the use of *alternating direction methods* (Fortin and Glowinski 1983). Introducing additional variables

$$z = (z_1, z_2, \ldots, z_S)$$

we can replace (12b) by a new set of conditions:

$$x_i - z_i = 0, \quad i = 1, \ldots, S, \tag{18}$$

$$z = Uz. \tag{19}$$

If we write augmented Lagrangian terms corresponding to (18) (with multipliers u_i) we see that in (16) minimization with respect to x is decomposable (for fixed z) into S subproblems

$$\text{minimize } \left\{ \langle c_i, x_i \rangle + \langle u_i, x_i \rangle + \frac{1}{2} r \|x_i - z_i\|^2 \right\}$$

$$\text{subject to } x_i \in X_i. \tag{20}$$

Minimization in z (for fixed x and subject to (19)) is obvious, so it is possible to develop a block-wise Gauss-Seidel method for solving (16) with alternating steps made in x and z. Multiplier updates (17) can be made after each alternating direction iteration. For multistage stochastic programming problems this approach has an interesting extension with some simplifications and deep probabilistic interpretation of successive steps (Rockafellar and Wets). The resulting progressive hedging algorithm (**PH**) is easily parallelizable (see Bertsekas and Tsitsiklis 1989 for a comprehensive analysis of similar parallel methods), but numerical experience shows (Vladimirou 1990, Helgason and Wallace 1991, Mulvey

and Vladimirou 1991) that in some cases convergence of the method can be slow.

Another method based on augmented Lagrangians has been suggested in Ruszczyński (1989). It has linear subproblems identical with the Dantzig–Wolfe method:

$$\text{minimize } \langle \bar{c}_i, x_i \rangle \text{ subject to } x_i \in X_i \quad \text{for } i = 1, \ldots, S \tag{21}$$

with $\bar{c}_i$ given by (14). The solutions x_i^j of (21) from previous iterations j enter the objective in a quadratic master program:

$$\begin{aligned}\text{minimize } & \Lambda_r(\lambda, \pi) \\ &= \sum_{i=1}^{S} \sum_{t=1}^{T} \langle c_i(t), \sum_{j \in J_i} \lambda_i^j x_i^j(t) \rangle \\ &\quad + \sum_{i=1}^{S} \sum_{t=1}^{T-1} \langle \pi_i(t), \sum_{j \in J_i} \lambda_i^j x_i^j(t) - \sum_{j \in J_{\nu(i,t)}} \lambda_{\nu(i,t)}^j x_{\nu(i,t)}^j(t) \rangle \\ &\quad + \frac{1}{2} r \sum_{i=1}^{S} \sum_{t=1}^{T-1} \| \sum_{j \in J_i} \lambda_i^j x_i^j(t) - \sum_{j \in J_{\nu(i,t)}} \lambda_{\nu(i,t)}^j x_{\nu(i,t)}^j(t) \|^2 \end{aligned} \tag{22a}$$

$$\text{subject to } \sum_{j \in J_i} \lambda_i^j = 1, \quad i = 1, \ldots, S, \tag{22b}$$

$$\lambda_i^j \geq 0, \quad i = 1, \ldots, S, \quad j \in J_i. \tag{22c}$$

In Ruszczyński (1989) detailed rules for updating the sets J_i and for changing multipliers by an analog of (17) are described. The method is still finitely convergent, but special algorithms for solving the master are needed to make the approach effective. We believe that it may have a potential for solving larger problems than the Dantzig–Wolfe method, because the multiplier iteration (17) is stable and insensitive to small errors in solving the master (22). The method is dual to the regularized decomposition method of Ruszczyński (1986), which proved efficient for two-stage stochastic programs.

In the next section we present another approach based on augmented Lagrangians, in which we distribute the master among the subproblems.

4. DIAGONAL QUADRATIC APPROXIMATION

Clearly, nonseparability of (15) is due to the quadratic terms

$$\| x_i(t) - x_{\nu(i,t)}(t) \|^2 \tag{23}$$

which contain cross-products $\langle x_i(t), x_{\nu(i,t)}(t) \rangle$. Suppose that x belongs to a neighborhood of some reference point $\bar{x}$. As noted in Mulvey and Ruszczyński, we can use the technique of Stephanopoulos and Westerberg (1975) and approximate the cross-products locally by

$$\begin{aligned}\langle x_i(t), x_{\nu(i,t)}(t) \rangle \approx & \langle x_i(t), \bar{x}_{\nu(i,t)}(t) \rangle + \langle \bar{x}_i(t), x_{\nu(i,t)}(t) \rangle \\ & - \langle \bar{x}_i(t), \bar{x}_{\nu(i,t)}(t) \rangle \end{aligned} \tag{24}$$

with an error on the order of $O(\|x - \bar{x}\|^2)$. Using (24) in (23) and then in (15) we see that problem (16) can be in the neighborhood of $\bar{x}$ approximated by S subproblems

$$\begin{aligned}\text{minimize } & \bar{\Lambda}_r^i(x_i, \pi; \bar{x}) \\ &= \sum_{t=1}^{T} \langle \bar{c}_i(t), x_i(t) \rangle + \frac{1}{2} r \sum_{t=1}^{T-1} \\ &\quad \{ \| x_i(t) - \bar{x}_{\nu(i,t)}(t) \|^2 + \| x_i(t) - \bar{x}_{\nu^{-1}(i,t)}(t) \|^2 \} \end{aligned} \tag{25a}$$

$$\text{subject to } x_i \in X_i, \tag{25b}$$

with $\bar{c}_i(t)$ defined by (14).

The approximation point $\bar{x}$ can be updated iteratively by the following algorithm. Note that k is the iteration counter of the outer loop (Algorithm 1) and stays fixed during Algorithm 2.

Algorithm 2

STEP 1. Set $\pi = \pi^k$, $\bar{x}^{k,m} = x^{k-1}$ and $m = 1$.

STEP 2. For $i = 1, \ldots, S$ solve (25) with $\bar{x} = \bar{x}^{k,m}$ obtaining new points $x_i^{k,m}$.

STEP 3. If $\|x_i(t) - \bar{x}_i(t)\| \leq \epsilon$ for all i and $t < T$, where $\epsilon > 0$ is some prescribed accuracy, then stop; otherwise set for $i = 1, \ldots, S$, $t = 1, \ldots, T$

$$\bar{x}_i^{k,m+1}(t) = \bar{x}_i^{k,m}(t) + \alpha(x_i^{k,m}(t) - \bar{x}_i^{k,m}(t)), \tag{26}$$

increase m by 1 and go to Step 2.

We prove that Algorithm 2 (with $\epsilon = 0$) generates a sequence of points $\{x^{k,m}\}_{m=1}^{\infty}$ whose accumulation points are solutions of (16), provided that $\alpha \in (0, \frac{1}{2})$.

Let us define the function

$$\begin{aligned}\bar{\Lambda}_r(x, \pi; \bar{x}) = & \sum_{i=1}^{S} \bar{\Lambda}_r^i(x_i, \pi; \bar{x}) \\ & + \frac{1}{2} r \sum_{i=1}^{S} \sum_{t=1}^{T-1} \| \bar{x}_i(t) - \bar{x}_{\nu(i,t)}(t) \|^2. \end{aligned} \tag{27}$$

Clearly, it is $\bar{\Lambda}_r(x, \pi; \bar{x})$ that is minimized at Step 2 of Algorithm 2 instead of the augmented Lagrangian (15). The error of the approximation of the augmented Lagrangian by $\bar{\Lambda}_r(x, \pi; \bar{x})$ can be estimated by the following lemma.

Lemma 1. *For all $\bar{x}$ and all x in X the inequality holds:*

$$|\Lambda_r(x, \pi) - \bar{\Lambda}_r(x, \pi; \bar{x})| \leq r \sum_{i=1}^{S} \sum_{t=1}^{T-1} \| x_i(t) - \bar{x}_i(t) \|^2. \tag{28}$$

Proof. Directly from the definition of Λ_r and $\bar{\Lambda}_r^i$ we have

$$\begin{aligned}& \Lambda_r(x, \pi) - \sum_{i=1}^{S} \bar{\Lambda}_r^i(x, \pi; \bar{x}) \\ &= \frac{1}{2} r \sum_{i=1}^{S} \sum_{t=1}^{T-1} \{ \| x_i(t) - x_{\nu(i,t)}(t) \|^2 - \| x_i(t) - \bar{x}_{\nu(i,t)}(t) \|^2 \\ &\qquad - \| x_i(t) - \bar{x}_{\nu^{-1}(i,t)}(t) \|^2 \}. \end{aligned}$$

Since $i = \nu(\nu^{-1}(i, t), t)$, by setting $j = \nu^{-1}(i, t)$ for every $t < T$ we get

$$\sum_{i=1}^{S} \| x_i(t) - \bar{x}_{\nu^{-1}(i,t)}(t) \|^2 = \sum_{j=1}^{S} \| \bar{x}_j(t) - x_{\nu(j,t)}(t) \|^2.$$

The last two equations yield

$$\Lambda_r(x, \pi) - \sum_{i=1}^{S} \tilde{\Lambda}_r^i(x, \pi; \bar{x})$$
$$= \frac{1}{2} r \sum_{i=1}^{S} \sum_{t=1}^{T-1} \{\|x_i(t) - x_{\nu(i,t)}(t)\|^2 - \|x_i(t) - \bar{x}_{\nu(i,t)}(t)\|^2$$
$$- \|\bar{x}_i(t) - x_{\nu(i,t)}(t)\|^2\}.$$

Using the identity

$$\|a - b\|^2 + \|\tilde{a} - \tilde{b}\|^2 + 2\langle a - \tilde{a}, b - \tilde{b}\rangle$$
$$= \|a - \tilde{b}\|^2 + \|\tilde{a} - b\|^2$$

with $a = x_i(t)$, $\tilde{a} = \bar{x}_i(t)$, $b = x_{\nu(i,t)}(t)$, $\tilde{b} = \bar{x}_{\nu(i,t)}(t)$ we arrive at the expression

$$\Lambda_r(x, \pi) - \sum_{i=1}^{S} \tilde{\Lambda}_r^i(x, \pi; \bar{x})$$
$$= -\frac{1}{2} r \sum_{i=1}^{S} \sum_{t=1}^{T-1} \{\|\bar{x}_i(t) - \bar{x}_{\nu(i,t)}(t)\|^2$$
$$+ 2\langle x_i(t) - \bar{x}_i(t), x_{\nu(i,t)}(t) - \bar{x}_{\nu(i,t)}(t)\rangle\}.$$

Combining it with (27) we conclude that

$$|\Lambda_r(x, \pi) - \tilde{\Lambda}_r(x, \pi; \bar{x})|$$
$$\leq r \sum_{t=1}^{T-1} \sum_{i=1}^{S} \|x_i(t) - \bar{x}_i(t)\| \|x_{\nu(i,t)}(t) - \bar{x}_{\nu(i,t)}(t)\|$$
$$\leq \frac{1}{2} r \sum_{t=1}^{T-1} \sum_{i=1}^{S} \{\|x_i(t) - \bar{x}_i(t)\|^2$$
$$+ \|x_{\nu(i,t)}(t) - \bar{x}_{\nu(i,t)}(t)\|^2\}.$$

Since for each t

$$\sum_{i=1}^{S} \|x_{\nu(i,t)}(t) - \bar{x}_{\nu(i,t)}(t)\|^2 = \sum_{i=1}^{S} \|x_i(t) - \bar{x}_i(t)\|^2,$$

we obtain (28). The proof is complete.

As an immediate conclusion from the above lemma we see that if $x_i(t) = \bar{x}_i(t)$, $i = 1, \ldots, S$, $t = 1, \ldots, T - 1$, then the values and the gradients of Λ_r and $\tilde{\Lambda}_r$ are equal.

Our next result is based on the observation that for x to be a minimizer of $\tilde{\Lambda}_r(y, \pi; \bar{x})$ over $y \in X$ it is necessary and sufficient that

$$\langle \nabla_x \tilde{\Lambda}_r(x, \pi; \bar{x}), y - x\rangle \geq 0 \tag{29}$$

for all $y \in X$.

Lemma 2. *If x is a minimizer of $\tilde{\Lambda}_r(y, \pi; \bar{x})$ over $y \in X$ then*

$$\tilde{\Lambda}(x, \pi; \bar{x}) - \tilde{\Lambda}(\bar{x}, \pi; \bar{x})$$
$$\leq -\frac{1}{2} r \sum_{i=1}^{S} \sum_{t=1}^{T-1} \|x_i(t) - \bar{x}_i(t)\|^2. \tag{30}$$

Proof. Let $x = (x_1, \ldots, x_S)$. Since each x_i minimizes $\tilde{\Lambda}_r^i(y_i, \pi; \bar{x})$ with respect to $y_i \in X_i$ we must have, by (29),

$$\langle \nabla_{x_i} \tilde{\Lambda}_r^i(x_i, \pi; \bar{x}), y_i - x_i\rangle \geq 0$$

for every $y_i \in X_i$. In particular, for $y_i(t) = \bar{x}_i(t)$ we obtain

$$\sum_{t=1}^{T} \langle \nabla_{x_i(t)} \tilde{\Lambda}_r^i(x_i, \pi; \bar{x}), \bar{x}_i(t) - x_i(t)\rangle \geq 0. \tag{31}$$

Next, by the definition of $\tilde{\Lambda}_r^i(x_i, \pi; \bar{x})$,

$$\nabla_{x_i(t)} \tilde{\Lambda}_r^i(x_i, \pi; \bar{x}) = \nabla_{x_i(t)} \tilde{\Lambda}_r^i(\bar{x}_i, \pi; \bar{x}) + r(x_i(t) - \bar{x}_i(t)),$$
$$t = 1, \ldots, T - 1, \tag{32a}$$
$$\nabla_{x_i(T)} \tilde{\Lambda}_r^i(x_i, \pi; \bar{x}) = \nabla_{x_i(T)} \tilde{\Lambda}_r^i(\bar{x}_i, \pi; \bar{x}) \tag{32b}$$

and

$$\tilde{\Lambda}_r^i(x_i, \pi; \bar{x})$$
$$= \tilde{\Lambda}_r^i(\bar{x}_i, \pi; \bar{x}) + \langle \nabla_{x_i} \tilde{\Lambda}_r^i(\bar{x}_i, \pi; \bar{x}), x_i - \bar{x}_i\rangle$$
$$+ \frac{1}{2} r \sum_{t=1}^{T-1} \|x_i(t) - \bar{x}_i(t)\|^2. \tag{33}$$

Combining (32) and (31) we see that

$$\langle \nabla_{x_i} \tilde{\Lambda}_r^i(\bar{x}_i, \pi; \bar{x}), x_i - \bar{x}_i\rangle \leq -r \sum_{t=1}^{T-1} \|x_i(t) - \bar{x}_i(t)\|^2.$$

Using the last inequality in (33) we get

$$\tilde{\Lambda}_r^i(x_i, \pi; \bar{x}) - \tilde{\Lambda}_r^i(\bar{x}_i, \pi; \bar{x})$$
$$\leq -\frac{1}{2} r \sum_{t=1}^{T-1} \|x_i(t) - \bar{x}_i(t)\|^2,$$

which yields (30).

We are now ready to prove our main result.

Theorem 1. *Assume that the sets X_i, $i = 1, \ldots, S$ are bounded. Then for every $0 < \alpha < \frac{1}{2}$:*

i. *each accumulation point of the sequence $\{x^{k,m}\}_{m=0}^{\infty}$ generated by Algorithm 2 is a solution of (16);*
ii. *for all $i = 1, \ldots, S$ and $t = 1, \ldots, T - 1$, $\|x_i^{k,m}(t) - \bar{x}_i^{k,m}(t)\| \to 0$ when $m \to \infty$.*

Proof. Let us skip the superscripts k and m for brevity. If x minimizes $\tilde{\Lambda}_r(y, \pi; \bar{x})$ with respect to $y \in X$, then by Lemma 1,

$$\Lambda_r(\bar{x} + \alpha(x - \bar{x}), \pi) - \tilde{\Lambda}_r(\bar{x} + \alpha(x - \bar{x}), \pi; \bar{x})$$
$$\leq \alpha^2 r \sum_{i=1}^{S} \sum_{t=1}^{T-1} \|x_i(t) - \bar{x}_i(t)\|^2.$$

Next, by the convexity of $\tilde{\Lambda}_r$ and Lemma 2,

$$\tilde{\Lambda}_r(\bar{x} + \alpha(x - \bar{x}), \pi; \bar{x}) - \tilde{\Lambda}_r(\bar{x}, \pi; \bar{x})$$
$$\leq \alpha[\tilde{\Lambda}_r(x, \pi; \bar{x}) - \tilde{\Lambda}_r(\bar{x}, \pi; \bar{x})]$$
$$\leq -\frac{1}{2} \alpha r \sum_{i=1}^{S} \sum_{t=1}^{T-1} \|x_i(t) - \bar{x}_i(t)\|^2.$$

Combining the last two inequalities we see that for $m = 1, 2, \ldots$

$$\Lambda_r(\bar{x}^{k,m+1}, \pi) - \Lambda_r(\bar{x}^{k,m}, \pi)$$
$$\leq -\alpha r\left(\frac{1}{2} - \alpha\right) \sum_{t=1}^{S} \sum_{t=1}^{T-1} \|x_t^{k,m}(t) - \bar{x}_t^{k,m}(t)\|^2.$$

Therefore, for $\alpha \in (0, \frac{1}{2})$ the sequence $\{\Lambda_r(\bar{x}^{k,m})\}_{m=1}^{\infty}$ is monotone, hence it is convergent. We also see that assertion ii is true. Suppose that $\bar{x}^k$ is a limit of a convergent subsequence of the sequence $\{x^{k,m}\}_{m=1}^{\infty}$. Then, by assertion ii, for the corresponding subsequence of $\{\bar{x}^{k,m}\}_{m=1}^{\infty}$ we have $\bar{x}_t^{k,m}(t) \to \bar{x}_t^k(t)$. With a view to (32), we can pass to the limit in (29) with $x \to \bar{x}^k$ and obtain

$$\langle \nabla_x \tilde{\Lambda}_r(\bar{x}^k, \pi; \bar{x}^k), y - \bar{x}^k \rangle \geq 0 \quad \text{for all } y \in X. \tag{34}$$

By Lemma 1

$$\nabla_x \tilde{\Lambda}_r(\bar{x}^k, \pi; \bar{x}^k) = \nabla_x \Lambda_r(\bar{x}^k, \pi),$$

and relation (34) implies optimality of $\bar{x}^k$ for problem (16). The proof is complete.

We can mention here that theoretical properties of DQA for stochastic programs follow from the sparsity of the linking constraints (8). They can be generalized to a broader class of convex optimization problems with sparse linking rows (Ruszczyński 1992).

5. INTERIOR POINT ITERATIONS

Although one can apply directly to (25) the primal-dual barrier method for convex separable programs of Carpenter et al. (1993), we still have another possibility: combining Algorithm 2 with the iterations of the barrier method. Interior point methods are based on successive approximations, so using our approximation (25) at each point seems promising.

Expanding the quadratic penalty term in (25) and neglecting terms that do not depend on x_t we arrive at the formulation

$$\text{minimize } \bar{c}_t^T x_t + \frac{1}{2} x_t^T H_t x_t \tag{35a}$$
$$\text{subject to } A_t x_t = b_t, \tag{35b}$$
$$x_t \geq 0, \tag{35c}$$

where

$$\bar{c}_t(t) = \begin{cases} c_t(t) + \pi_t(t) - \pi_{\nu^{-1}(t,t)}(t) - r\bar{x}_{\nu(t,t)}(t) \\ \qquad - r\bar{x}_{\nu^{-1}(t,t)}(t) & \text{if } t < T, \\ c_t(t) & \text{if } t = T, \end{cases} \tag{36}$$

and H_t is a diagonal matrix with the diagonal elements equal to $2r$ for variables which have siblings, and 0 otherwise.

Problem (35) is a separable quadratic programming problem, so identically as in Carpenter et al., we can formulate its dual and use logarithmic barrier functions for nonnegativity constraints. Then the first-order optimality conditions for both problems take on the form

$$A_t x_t = b_t, \tag{37a}$$
$$A_t^T y_t + z_t - H_t x_t = \bar{c}_t, \tag{37b}$$
$$X_t Z_t e = \mu e, \tag{37c}$$

where $\mu > 0$ is a barrier function coefficient, e is a vector of ones, y_t is the vector of multipliers associated with (35b), z_t is the vector of nonnegative slack variables in the dual problem, and X_t and Z_t are diagonal matrices with diagonals x_t and z_t, respectively.

When the solution is strictly interior, primal feasible (i.e., satisfies (37a)) and dual feasible (i.e., satisfies (37b)), Newton's method applied to (37) yields the search directions

$$\Delta z_t = -\rho(\mu) - X_t^{-1} Z_t \Delta x_t, \tag{38a}$$
$$\Delta x_t = \Theta(A_t^T \Delta y_t - \rho(\mu)), \tag{38b}$$
$$\Delta y_t = (A\Theta A^T)^{-1} A\Theta\rho(\mu), \tag{38c}$$

where

$$\Theta = (H_t + X_t^{-1} Z_t)^{-1}$$

and

$$\rho(\mu) = (Z_t - \mu X_t^{-1})e.$$

The directions (38) can be used to iteratively update primal and dual variables by

$$x_t' = x_t + \tau \Delta x_t, \tag{39a}$$
$$y_t' = y_t + \tau \Delta y_t, \tag{39b}$$
$$z_t' = z_t + \tau \Delta z_t, \tag{39c}$$

with step-size τ and barrier coefficient μ calculated by rules described by Carpenter et al. (1993), Lustig, Marsten and Shanno (1992), and Lustig, Mulvey and Carpenter (1991).

In this way, we can proceed until the solution of (35) is found. However, we may stop after a small number of iterations (or just one) and update the approximation point $\bar{x}$ by (26).

In general, the change of $\bar{x}$ results in the change of $\bar{c}_t$ given by (36) which, in turn, causes the loss of dual feasibility (37b). Primal feasibility (37a) is retained, so only minor changes in directions are necessary. Formulas (38) remain valid, but with

$$\rho(\mu) = (Z_t - \mu X_t^{-1})e - d_D,$$

where

$$d_D = A_t^T y + z - H_t x_t - \bar{c}_t.$$

Summing up, the iterative character of barrier methods and their ability to start from arbitrary interior points makes it possible to integrate the algorithm for solving (25) with the updates of the approximation point (26).

6. A PARALLEL DISTRIBUTED IMPLEMENTATION

The diagonal quadratic approximation (DQA) method is a flexible theoretical scheme which can be implemented in many ways, in particular, in a parallel distributed environment.

Let us assume that we have a sufficiently large number of processors and let us assign each scenario to a different processor. Then it is clear that iterations (39) can be carried out (for a fixed $\bar{x}$) independently in each processor. It is also true for the algorithm for updating $\bar{x}$ by (26).

Although the change of $\bar{x}$ causes changes in $\bar{c}$ in (35), it follows from (36) that the interactions occur only between siblings at a given stage: $\bar{x}_i(t)$ influences the cost $\bar{c}_j(t)$ in scenarios $j = \nu(i, t)$ and $j = \nu^{-1}(i, t)$, and vice versa. So, we derive a communication structure between processors, in which messages need to be passed between nearest neighbors. For the example of Figure 2 and Table I the communication structure is depicted in Figure 3. The number of lines joining subproblems in the figure denotes the number of stages at which the siblings need to exchange data.

The multiplier iteration (17) can also be carried out in a distributed fashion. Indeed, when Algorithm 2 stops, we have $x_i(t) = \bar{x}_i(t)$, so $x_i(t)$ is known to the sibling $\nu(i, t)$, while $x_{\nu(i,t)}$ is known to subproblem i. Thus, both i and $\nu(i, t)$ can calculate $\pi(i, t)$ by (17). Clearly, the same argument applies to the sibling $\nu^{-1}(i, t)$. Thus, all subproblems can update their costs (36) in a distributed fashion.

It is worth stressing that in this way we obtained a decomposition method without any coordinating unit. What once was a large master problem in the Dantzig–Wolfe method in our approach became a simple shift of costs (36) distributed among subproblems.

Assigning each scenario to a different processor is not the only possibility to distribute the DQA method. We can form larger subproblems to comprise multiple scenarios. Then, obviously, the nonanticipativity links between scenarios included in one subproblem should be treated directly as constraints in the subproblem solver. The only nonanticipativity constraints that need to be coordinated by the DQA method are those that link scenarios assigned across subproblems.

To describe it formally, let $I_1 = \{1, \ldots, i_1\}$, $I_2 = \{i_1 + 1, \ldots, i_2\}, \ldots, I_K = \{i_{K-1} + 1, \ldots, S\}$ be subsets of scenarios assigned to subproblems $1, 2, \ldots, K$. Let us define the *outgoing boundary* of subproblem k by

$$\partial^+ I_k = \{(i, t) \in I_k \times \{1, \ldots, T\}: \nu(i, t) \notin I_k\}.$$

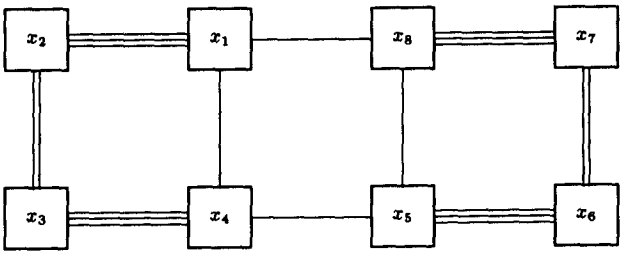

Figure 3. Communication between subproblems by full decomposition.

In a similar way we define the *incoming boundary*:

$$\partial^- I_k = \{(i, t) \in I_k \times \{1, \ldots, T\}: \nu^{-1}(i, t) \notin I_k\}.$$

Then we can formulate subproblem k as:

$$\text{minimize} \sum_{i \in I_k} \sum_{t=1}^{T} \langle \bar{c}_i(t), x_i(t) \rangle \tag{40a}$$

$$+ \frac{1}{2} r \sum_{(i,t) \in \partial^+ I_k} \|x_i(t)\|^2 + \frac{1}{2} r \sum_{(i,t) \in \partial^- I_k} \|x_i(t)\|^2$$

subject to

$$x_i \in X_i, \quad i \in I_k, \tag{40b}$$

$$x_i(t) = x_{\nu(i,t)}(t), \quad i \in I_k, \ \nu(i, t) \in I_k, \tag{40c}$$

with the modified costs defined by:

$$\bar{c}_i(t) = \begin{cases} c_i(t) + \pi_i(t) - r\bar{x}_{\nu(i,t)}(t) & \text{if } (i, t) \in \partial^+ I_k, \\ c_i(t) - \pi_{\nu^{-1}(i,t)}(t) - r\bar{x}_{\nu^{-1}(i,t)}(t) & \text{if } (i, t) \in \partial^- I_k, \\ c_i(t) & \text{otherwise.} \end{cases}$$

Since internal nonanticipativity constraints (40c) are treated directly, we only need to apply augmented Lagrangian terms to the links between boundaries of subproblems:

$$x_i(t) = x_{\nu(i,t)}(t), \quad (i, t) \in \partial^+ I_k, \ k = 1, \ldots, K,$$

and only for these (i, t) do we carry out iteration (17). So, similarly to the full decomposition scheme, exchange of information between subproblems occurs only along coordinated links $(i, t) \leftrightarrow (\nu(i, t), t)$ with $(i, t) \in \partial^+ I_k$, $k = 1, \ldots, K$.

In the example of Figure 2, assuming that subproblem 1 contains scenarios 1, 2, 3, and 4, subproblem 2 contains scenarios 5 and 6, subproblem 3 contains scenario 7, and subproblem 4 contains scenario 8, only the links shown in Table II need be coordinated. The resulting communication structure of subproblems is shown in Figure 4. It could also be obtained by clustering subgroups of nodes of Figure 3.

7. NUMERICAL RESULTS

Following the ideas presented in this paper an experimental C code has been developed on the network of Silicon Graphics IRIS 4D/70 workstations. For subproblem solution the code LOQO of Vanderbei (1991) was used. LOQO is a primal-dual interior point method with an alternative approach to the system of linear equations (39). Recent comparison with other LP solvers, such as MINOS, CPLEX, and OB1 indicates high efficiency and reliability of LOQO on a variety of large-scale linear programming problems.

Subproblems were generated by a specially designed generator and immediately sent to remote computer nodes. So, the large model has never been formed explicitly. To distribute the subproblems over the network and organize their communication we used LINDA—a special package for parallel network programming (Carriero and Gelernter 1989).

Table II
Coordinated Links by Partial Decomposition

Time	I_1					I_2	I_3	I_4
Stage	1	2	3	4	5	6	7	8
1	—	—	—	5	—	7	8	1
2	—	—	—	—	—	7	8	5
3	—	—	—	—	—	—	8	7
4	—	—	—	—	—	—	—	—

The principal goal of the implementation was to gain some insight into the capabilities of the new approach, so we used rather general and straightforward implementation techniques. In particular, all subproblems were scaled by geometrical means of rows and columns in a "basic" scenario. The fixed value $\rho = 1$ of the penalty parameter was used for the scaled problem. In Algorithm 2 the value $\alpha = 0.45$ was used.

Example 1

Our first example is a dynamic financial planning problem (see Vladimirou 1990, Mulvey and Vladimirou 1992) modeled as a network with node set $\mathcal{N}$ and arc set $\mathcal{A}$. The nodes represent assets at successive time stages, and the arcs model transactions at each time stage and transformations of the value of the asset over a time period. For every node $n \in \mathcal{N}$ we use d_n to denote (uncertain) supply or demand at node n.

There are multipliers r_a associated with arcs $a \in \mathcal{A}$: They represent transaction costs for arcs within a given period and uncertain returns on the arcs joining one period to the next.

Random demands and multipliers are modeled as a set of scenarios $i = 1, \ldots, S$; we will write d_{ni} and r_{ai} to denote data associated with scenario i. Scenario probabilities are denoted by p_i.

Decision variables are flows on the arcs. There are two groups of them: Flows z_a should be determined before the scenario is known, while flows y_{ai} may depend on scenario i.

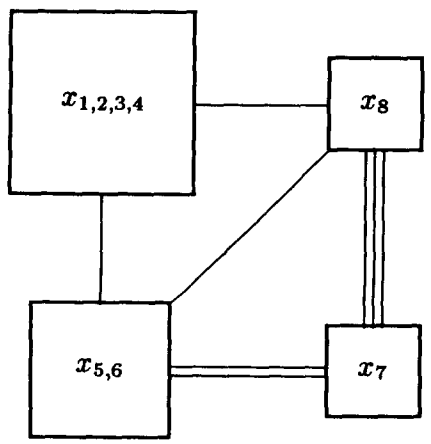

Figure 4. Communication between subproblems by partial decomposition.

Finally, we have a set of terminal arcs $\mathcal{T} \subset \mathcal{A}$ over which the final wealth is accumulated. The problem is to maximize the expected final wealth subject to flow conservation constraints and simple lower and upper bounds on flows.

Let us introduce the notation:

$\mathcal{A}_1$ = the subset of arcs representing first stage decisions for which all associated multipliers are deterministic;
$\mathcal{A}_2$ = the subset of arcs representing first stage decisions which have associated stochastic arc multipliers;
$\mathcal{A}_3$ = the subset of arcs representing second stage decisions;
$\mathcal{N}_1$ = the subset of nodes with deterministic flow balance equations;
$\mathcal{N}_2$ = the subset of nodes with flow balance equations involving scenario-dependent data or decisions;
Δ_i^+ = the set of outgoing arcs at node i;
Δ_i^- = the set of incoming arcs at node i.

With such notation the problem can be formally stated as:

$$\max \sum_{i=1}^{S} p_i \Big(\sum_{a \in \mathcal{A}_1 \cap \mathcal{T}} r_a z_a + \sum_{a \in \mathcal{A}_2 \cap \mathcal{T}} r_{ai} z_a + \sum_{a \in \mathcal{A}_3 \cap \mathcal{T}} r_{ai} y_{ai} \Big)$$

subject to

$$\sum_{a \in \Delta_n^+} z_a - \sum_{a \in \Delta_n^-} r_a z_a = b_n, \quad n \in \mathcal{N}_1,$$

$$\sum_{a \in \Delta_n^+ \cap \mathcal{A}_1} z_a - \sum_{a \in \Delta_n^- \cap \mathcal{A}_1} r_a z_a + \sum_{a \in \Delta_n^+ \cap \mathcal{A}_2} z_a - \sum_{a \in \Delta_n^- \cap \mathcal{A}_2} r_{ai} z_a + \sum_{a \in \Delta_n^+ \cap \mathcal{A}_3} y_{ai} - \sum_{a \in \Delta_n^- \cap \mathcal{A}_3} r_{ai} y_{ai} = b_{ni}, \quad n \in \mathcal{N}_2, \quad i = 1, \ldots, S.$$

$$l_a \leq z_a \leq u_a, \quad a \in \mathcal{A}_1 \cap \mathcal{A}_2,$$

$$l_a \leq y_{ai} \leq u_a, \quad a \in \mathcal{A}_3, \quad i = 1, \ldots, S.$$

We see that the problem is a two-stage stochastic programming problem with first-stage variables z_a, $a \in \mathcal{A}_1 \cup \mathcal{A}_2$, and second stage variables y_{ai}, $a \in \mathcal{A}_3$, $i = 1, \ldots, S$.

The DQA method was tested on such a problem with 40 scenarios and 334 variables per scenario, which in total made a large LP with 4,840 rows, 13,400 columns and 26,720 nonzeros of the constraint matrix. Table III presents results for the number of subproblems ranging from 1–14. Scenarios were assigned to subproblems evenly, if possible. When the number of subproblems was 1 we directly solved a large LP problem by the interior primal-dual method of LOQO.

A number of conclusions can be drawn from these results. Decomposing the problem into smaller subproblems increases the number of multiplier iterations and, consequently, the number of iterations of the direct solver. On the other hand, the size of subproblems and their complexity (caused by nonanticipativity constraints) decrease fast. As a result, iterations become much cheaper, maximum CPU time per node (column 3 of Table III) decreases fast, and the total work (the sum of the CPU times used at the nodes) does not grow significantly.

Parallel subproblem solution speeds up computation considerably. The maximum CPU per subproblem, which is the lower bound of the solution time, decreases linearly. The real elapsed time decreases too, although we can observe saturation for a larger number of nodes. This is due to the increase of the communication overhead; when the number of subproblems grows, the number of messages increases and more time is spent on communication than on computation.

Example 2

Our second example is a stochastic scheduling and transportation problem. We have to plan flights (sorties) over some routes in a network to ship cargo from some nodes to other ones. The amounts to be shipped are uncertain. Therefore, the load of an aircraft may be event dependent provided that it does not exceed the maximumcapacity. The sorties, however, must be determined before the actual demands will be known. As in the previous example, we represent uncertainty by scenarios $i = 1, 2, \ldots, S$ with probabilities p_i and we use i to index scenario-dependent data and decisions.

We introduce the following notation:

$\mathcal{N}$ = the set of nodes;
$b_i(m, n)$ = the amount of cargo to be shipped from nodes m to n in scenario i;
$f(m, n)$ = the minimum number of flights from m to n;
Π = the set of *routes*, i.e., selected sequences of nodes $n_1, n_2, \ldots, n_l$;
$l(\pi)$ = the number of nodes in route π;
$\nu(\pi, j)$ = the jth node in route π, $j = 1, \ldots, l(\pi)$;
$\sigma(n)$ = the maximum number of landings allowed in node n;
$\mathcal{A}$ = the set of aircraft types;
$d(a)$ = the maximum payload of an aircraft of type a;
$h^{\max}(a)$ = the maximum flying hours of aircrafts of type a;
$h^{\min}(a)$ = the minimum flying hours of aircrafts of type a;
$h(\pi, a)$ = the flying hours necessary for an aircraft of type a to pass route π;
$c(a)$ = the cost of an operating hour of an aircraft of type a;
q = the unit cost of cargo handling;
ρ = the penalty for each undelivered cargo unit.

Table III
Performance of the Distributed DQA Method for Example 1

Subproblems	Multiplier Iterations	CPU Time	Elapsed Time
1	1	426	503
2	2	342	352
3	3	312	326
4	3	202	212
5	3	169	180
6	3	113	138
8	3	81	104
10	3	64	90
12	3	58	97
14	3	56	126

Cargo can be moved either by direct delivery from m to n over some routes passing through m and n, or by transshipment. This leads to the following set of decision variables:

$x(\pi, a)$ = the number of sorties on route π by aircraft type a;
$d_i(\pi, m, n)$ = the amount of cargo delivered directly from nodes m to n over route π in scenario i;
$t_i(\pi, m, k, n)$ = the amount of cargo from nodes m to n to be moved to the transshipment node k over route π in scenario i;
$s_i(\pi, k, n)$ = the amount of transshipment cargo to be moved from the transshipment node k to node n over route π in scenario i;
$y_i(m, n)$ = the amount of undelivered cargo from nodes m to n in scenario i;
$z_i(\pi, j)$ = the unused capacity on leg j of route π, $j = 1, \ldots, l(\pi) - 1$ in scenario i.

The $x(\pi, a)$ are first-stage variables; the remaining ones are second-stage variables. All variables are restricted to nonnegative values.

We are now ready to formulate the problem as

$$\min\Bigg[\sum_{\pi\in\Pi}\sum_{a\in\mathcal{A}} c(a)h(\pi, a)x(\pi, a) + \sum_{i=1}^{S} p_i\Bigg(q\sum_{\pi\in\Pi}\sum_{(m,n)\in l(\pi)}(d_i(\pi, m, n)+s_i(\pi, m, n) + \sum_{k\in\mathcal{N}} t_i(\pi, m, n, k)) + \rho\sum_{m\in\mathcal{N}}\sum_{n\in\mathcal{N}} y_i(m, n)\Bigg)\Bigg] \tag{41}$$

subject to cargo balance conditions for every scenario $i = 1, \ldots, S$:

$$\sum_{\pi\in\Pi}\Big(d_i(\pi, m, n) + \sum_{k\in\mathcal{N}} t_i(\pi, m, k, n)\Big) + y_i(m, n) \geq b_i(m, n), \quad m, n \in \mathcal{N};$$

$$\sum_{\pi\in\Pi}\sum_{m\in\mathcal{N}} t_i(\pi, m, k, n) = \sum_{\pi\in\Pi} s_i(\pi, k, n), \quad k, n \in \mathcal{N};$$

and payload balance equations for every route $\pi \in \Pi$ and every scenario $i = 1, \ldots, S$:

$$\sum_{k\in\mathcal{N}} \Big(d_i(\pi, \nu(\pi, 1), k) + s_i(\pi, \nu(\pi, 1), k) + \sum_{n\in\mathcal{N}} t_i(\pi, \nu(\pi, 1), k, n)\Big) = \sum_{a\in\mathcal{A}} d(a)x(\pi, a) - z_i(\pi, 1);$$

$$\sum_{k\in\mathcal{N}} \Big(d_i(\pi, \nu(\pi, j), k) + s_i(\pi, \nu(\pi, j), k) + \sum_{n\in\mathcal{N}} t_i(\pi, \nu(\pi, j), k, n)\Big) - \sum_{k\in\mathcal{N}} \Big(d_i(\pi, k, \nu(\pi, j)) + s_i(\pi, k, \nu(\pi, j)) + \sum_{n\in\mathcal{N}} t_i(\pi, k, \nu(\pi, j), n)\Big) = z_i(\pi, j-1) - z_i(\pi, j); \quad j = 2, \ldots, l(\pi) - 1.$$

To formulate the remaining constraints of our problem let us define the sets

$$U(m, n) = \{\pi \in \Pi: m = \nu(\pi, j_1), n = \nu(\pi, j_2), j_1 < j_2\},$$
$$V_1(n) = \{\pi \in \Pi: n = \nu(\pi, 1)\},$$
$$V_l(n) = \{\pi \in \Pi: n = \nu(\pi, l(\pi))\}$$
$$W(n) = \{\pi \in \Pi: n = \nu(\pi, j) \text{ for some } j\}.$$

We can now write the frequency requirements

$$\sum_{a\in\mathcal{A}}\sum_{\pi\in U(m,n)} x(\pi, a) \geq f(m, n), \quad m, n \in \mathcal{N};$$

the landing and take-off balance

$$\sum_{\pi\in V_1(n)} x(\pi, a) = \sum_{\pi\in V_l(n)} x(\pi, a), \quad a \in \mathcal{A}, n \in \mathcal{N};$$

$$\sum_{a\in\mathcal{A}}\sum_{\pi\in W(n)} x(\pi, a) \leq \sigma(n), \quad n \in \mathcal{N};$$

and the balance of flying hours:

$$h^{\min}(a) \leq \sum_{\pi\in\Pi} x(\pi, a)h(\pi, a) \leq h^{\max}(a), \quad a \in \mathcal{A}.$$

The above formulation can be refined further by carrying out the summation over nonzero entries only, but we will not complicate the description here; our aim is just to indicate the application area.

We solved a version of the above problem with 585 rows, 1,380 variables, and 3,799 nonzeros for each scenario. There were 118 random right-hand side coefficients $b_i(m, n)$ and 121 first-stage variables $x(\pi, a)$. Scenarios were generated randomly from the independent uniform distributions within a ±20% range around the basic demand forecast. Sizes of the resulting multiscenario problems are given in Table IV.

The results of experiments are summarized in Table V. We see that the direct approach by the LP solver to the large-scale formulation suffers from rapid growth of computation time when the number of scenarios grows. In fact, one cannot solve problems with more than 15 scenarios on our SGI workstations. It should be stressed that our LP solver, LOQO of Vanderbei, is one of the most efficient and compact solvers available today. The growth of the costs is due to the increasing structural complication of the resulting LP problems and the resulting increase of the fill-in in the Cholesky factors. A similar observation was recently made in Lustig, Mulvey and Carpenter.

The distributed DQA approach allows for solving much larger problems. The final accuracy attainable is not high: The method was terminated when the relative error of nonanticipativity conditions was 10^{-3}, but the ability to treat problems which are intractable otherwise is decisive. We see from the results that for really large problems it is better to have as many subproblems as possible because the cost of the subproblem solution grows very fast (see the first four rows). For very large problems the elapsed time is much larger than the maximum CPU time per node (column 4 of Table V) due to communication delays in a distributed multiuser environment.

Since our example is a large-scale two-stage problem, it seems natural to compare the performance of the new method with highly specialized techniques for linear two-stage problems. The most successful methods found so far are based on a Benders-type decomposition suggested for stochastic programming in Van Slyke and Wets (1969). One of the most recent implementations of this idea is MSLiP (Gassmann 1990), which also allows for solving linear multistage problems in a nested formulation. A regularized version of the Benders decomposition method has been suggested in Ruszczyński (1986); a special code, QDECOM, based on this idea turned out to be rather fast and stable in manifold applications. However, experiments on problems having such large sizes have never been conducted.

Table IV
Dimension of LP Equivalent Formulations for Example 2

Scenarios	Rows	Columns	Nonzeros
1	585	1,380	3,799
5	3,530	6,900	20,205
10	7,060	13,800	40,410
15	10,590	20,700	60,615
50	35,300	69,000	202,050
100	70,600	138,000	404,100
200	141,200	276,000	808,200

Table V
Performance of the Distributed DQA Method for Example 2

Scenarios	Subproblems	Multiplier Iterations	CPU Time	Elapsed Time
1	1	0	9	11
5	1	0	222	225
10	1	0	1,584	1,625
15	1	0	5,205	9,077
20	1	0	[a]	[a]
50	10	5	3,802	4,252
50	13	5	2,486	5,551
50	17	6	1,094	4,929
100	10	2	8,799	28,754
100	15	3	5,884	24,870
100	20	5	4,175	15,562
200	20	1[b]	8,289	39,789

[a]LP solver crashed because of insufficient space.
[b]Final accuracy not obtained.

Table VI summarizes the computation times for the special code MSLiP obtained on a SUN SPARCstation 2. As a benchmark, we also provide computation times for the general LP code MINOS 5.3 (we owe the results in this table to the courtesy of Dr. Horand I. Gassmann). Table VII provides details of the performance of the regularized decomposition method of Ruszczyński (1986) for the same example on an identical computer.

A number of interesting conclusions can be drawn from these results. It is obvious that specialized decomposition methods favorably compare with direct LP solvers. Even though the straight Benders decomposition approach is slightly slower than MINOS for smaller problems, note that the growth of costs is slow for MSLiP and fast for MINOS, when the number of scenarios grows.

The regularized decomposition method of Ruszczyński (1986) proved to be particularly efficient for large two-stage problems; it came out as a clear winner in both the computation time and, far more important, in the size of problems solved. The 1,000-scenario problem solved by QDECOM had $7 \cdot 10^5$ rows, $1.4 \cdot 10^6$ columns, and $4 \cdot 10^6$ nonzeros in the equivalent LP formulation. The regularized decomposition method was the only special method that outperformed our new DQA method. Its success is due to the application of a number of ideas specially developed for linear problems with a dual angular structure:

- regularization of the master problem which stabilizes the procedure;
- a special numerical procedure for solving the master which takes advantage of the structure of stochastic problems and has a stable sparse QR factorization kernel;
- re-optimization of subproblems by the dual simplex method.

Table VI
Computation Time of the Benders Decomposition Method MSLiP and of MINOS 5.3 for Example 2

Scenarios	MSLIP	MINOS 5.3
1	701	95
2	2,179	265
5	3,502	1,606
10	6,401	5,344

It is clear that the regularized decomposition method, as a highly specialized technique for linear two-stage problems, should be best. However, the performance of DQA for these problems is still second best, and DQA has a potential of solving nonlinear and multistage problems as well.

As an illustration, Table VIII presents some preliminary results for a nonlinear two-stage problem derived from Example 2. The cost, instead of the linear formulation in (41), was defined as

$$F_2 = \sum_{i=1}^{S} p_i \Phi\Big(q \sum_{\pi \in \Pi} \sum_{(m,n) \in I(\pi)} \cdot (d_i(\pi, m, n) + s_i(\pi, m, n) + \sum_{k \in \mathcal{N}} t_i(\pi, m, n, k)) + \rho \sum_{m \in \mathcal{N}} \sum_{n \in \mathcal{N}} y_i(m, n)\Big)$$

with $\Phi(x) = \alpha \exp(\beta x)$.

The scenario subproblems were similar to (25b), but with a nonlinear objective part rather then linear. For their solution we used a version of the interior point method for convex separable problems described in Monteiro and Adler (1990). It was close to the one described in Section 5, but the Hessians H_i were point dependent rather than constant.

The results of Table VIII show that there is no significant difficulty introduced to the method by convex separable nonlinearities; performance is close to the linear case. Much remains to be done in the nonlinear case, but these preliminary results already indicate a potential of DQA.

An important feature of DQA is that it has subproblems that possess an identical structure. Scenarios give

Table VII
Performance of the Regularized Decomposition Method for Example 2

Scenarios	QDECOM Iterations	Dual Iterations in Subproblems	CPU Time
1	34	1,291	38
5	33	3,594	104
10	31	5,561	160
15	35	8,501	243
20	37	11,368	325
50	56	35,183	1,002
100	53	66,122	1,898
200	50	121,464	3,278
1,000	68	806,992	21,850

rise to submatrices with different data, but the same pattern of nonzeros. When an interior point method is used, as described in Section 5, the matrices to be factorized at each iteration have the same structure for every subproblem. Thus, we can vectorize the computations across the subproblems and solve, in parallel, a large collection of LP's with the same structure and different data. We can add here that the concept of vectorization does not offer much for other general methods with sparse matrix computations, such as the simplex method.

To test the potential of this approach some preliminary experiments were carried out on a Convex computer (3200) at Convex's headquarters in Richardson, Texas. The results of vectorizing the key loops in the LOQO code are shown in Table IX. Here, the size of vectors are displayed along with the accompanying scalar and vector speeds in (Megaflops). The results are quite impressive: Over 14 times speedup when the vectors are greater than 100, and 20 times speedup when the vectors are equal to 1,000. Since the size of the vector depends upon the number of subproblems, there is an incentive to solve MSPs with a large number of scenarios. This greatly reduces the difficulties with sparse matrix calculations for large stochastic optimization problems.

8. CONCLUSIONS

This paper has demonstrated that the DQA method can be implemented successfully on a distributed network of workstations, connected via an ethernet or on a vector-processing computer. In the former case, the only new item beyond the workstations and the NFS-Unix operating system is the network-LINDA software package or a similar system. If these concepts can be refined further, operations researchers may be able to achieve supercomputer performance for certain applications—without the need to purchase a dedicated supercomputer. Much work remains to be accomplished before a definitive statement can be made concerning the effectiveness of distributed optimization strategies. We have shown considerable speedup (almost eight-fold improvement) in the CPU time for some real-world stochastic optimization problems. We have also demonstrated the ability of DQA to treat very large problems which are beyond the reach of direct solvers.

Table VIII
Performance of DQA for a Nonlinear Problem

Scenarios	Sub-problems	Interior Point Iterations	Multiplier Iterations	CPU Time	Elapsed Time
1	1	35	1	12	13
2	1	35	1	33	34
5	1	36	1	142	143
10	1	37	1	504	515
15	1	37	1	996	1,028
10	2	198	7	577	1,170
15	3	131	7	379	1,159
20	4	227	7	680	2,730
25	5	236	7	716	3,622
30	6	231	7	713	4,436
35	7	311	9	897	13,783

Table IX
Performance of Vectorizing Key LOQO Loops

Problem Number	Vector Size	Scalar MFLOPs	Vector MFLOPs
1	1001	1.27	20.71
2	501	1.27	19.74
3	201	1.27	17.27
4	100	1.43	14.37
5	50	1.42	10.66
6	20	1.36	6.04

There are several interrelated issues to be resolved when a parallel system is to be implemented. For instance, the network-LINDA system does not take advantage of any special structure in the communication network, such as the ethernet broadcast capabilities. Thus, the optimal number of subproblems for this configuration (relatively slow SGI workstations and a general purpose network "operating system") appears to be in the neighborhood of 10–12 processors. Increasing the bandwidth and the reliability of the communication system would allow for a larger number of subproblems to be solved via the decomposition. Further testing is needed to determine the "optimal" number of processors for SIMD and MIMD vector machines. Likewise, the decomposition pattern as defined by the $I(1)$, $I(2)$, ... , $I(K)$ must be matched to the computer capabilities: More powerful processors assigned to a larger number of scenarios; and slower machines getting smaller subproblems.

Also, the degree of decomposition will have an impact on the number of outer iterations that are required to obtain a specified precision in the answer. In general, greater decomposition results in a larger number of steps. This favors a relatively small number of powerful processors. Anyway, synchronization issues are rather easy here, because for interior point methods computational effort per iteration is directly related to the number of scenarios in a subproblem. Due to its greater reliability, a direct solver that achieves acceptable accuracy is preferable to a decomposition strategy.

However, a disadvantage of the distributed DQA method at the current state of research is the difficulty of dynamic load balancing (if, e.g., the work load of one of the computers changes suddenly during the run and we would like to assign its subproblem to some other machine). This would require either migration of processes in the network, which seems rather difficult to implement, or a hot start of the method with a new assignment of subproblems to computers. The hot start solution appears to be more realistic, but still quite hard. Ideally, if a processor goes down, we need to have a new processor initiated into the network, a subproblem sent to it, and then have it read

the restarting information files (the last values of the coordination variables). Unfortunately, we have not yet discovered an easy approach to achieve this goal without either having a controlling master process which we strive to avoid, or performing a plethora of time-consuming checks during data transmission. Hopefully, progress in operating systems for multicomputers will make it easier.

In summary, the design of parallel optimization algorithms is complicated by the need to tailor software to hardware, and, perhaps, vice versa. For instance, a company may already possess a network of workstations, say for its salesforce. Then there would be little increase in computational costs for conducting the DQA method while the salesmen are using their computers for word processing—a common occurrence. On the other hand, a time sensitive calculation may not be able to wait for machines to become idle. These issues have existed in the past, certainly, but the problems are more critical (and complex) when put in the context of a parallel optimization solver.

REFERENCES

BERTSEKAS, D. P. 1982. *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press, New York.

BERTSEKAS, D. P., AND J. N. TSITSIKLIS. 1989. *Parallel and Distributed Computation*. Prentice-Hall, Englewood Cliffs, N.J.

BIRGE, J. R. 1985. Decomposition and Partitioning Methods for Multistage Stochastic Linear Programs. *Opns. Res.* **33,** 989–1007.

CARPENTER, T. J., I. J. LUSTIG, J. M. MULVEY AND D. F. SHANNO. 1993. Separable Quadratic Programming via a Primal-Dual Interior Point Method and Its Use in a Sequential Procedure. *ORSA J. Comput.* **5,** 182–191.

CARRIERO, N., AND D. GELERNTER. 1989. How to Write Parallel Programs: A Guide to the Perplexed. *ACM Comput. Surv.* **21,** 323–357.

DANTZIG, G. B., AND P. WOLFE. 1960. Decomposition Principle for Linear Programs. *Opns. Res.* **8,** 101–111.

FORTIN, M., AND R. GLOWINSKI. 1983. On Decomposition-Coordination Methods Using an Augmented Lagrangian. In *Augmented Lagrangian Methods: Applications to the Numerical Solution of Boundary-Value Problems*, M. Fortin and R. Glowinski (eds.). North-Holland, Amsterdam, 97–146.

DEMPSTER, M. A. H., AND A. M. IRELAND. 1995. Object-Oriented Model Integration in a Financial Decision-Support System. *Dec. Sup. Syst.* (to appear).

GASSMANN, H. I. 1990. MSLiP: A Computer Code for the Multistage Stochastic Linear Programming Problem. *Math. Prog.* **47,** 407–423.

GONDZIO, J., AND A. RUSZCZYNSKI. 1992. A Sensitivity Method for Basis Inverse Representation in Multistage Stochastic Linear Programming Problems. *J. Optim. Theory and Applic.* **74,** 221–242.

HELGASON, T., AND S. W. WALLACE. 1991. Approximate Scenario Solutions in the Progressive Hedging Algorithm. *Anns. Opns. Res.* **31,** 425–444.

LUSTIG, I. J., J. M. MULVEY AND T. J. CARPENTER. 1991. Formulating Stochastic Programs for Interior Point Methods. *Opns. Res.* **39,** 757–770.

LUSTIG, I. J., R. E. MARSTEN AND D. F. SHANNO. 1992. On Implementing Mehrotra's Predictor-Corrector Interior Point Method for Linear Programming. *SIAM J. Optim.* **2,** 435–449.

MONTEIRO, R. D. C., AND I. ADLER. 1990. An Extension of Karmarkar Type Algorithm to a Class of Convex Separable Programming Problems With Global Linear Rate of Convergence. *Math. O.R.* **15,** 408–422.

MULVEY, J. M. 1989. A Surplus Optimization Perspective. *Invest. Mgmt. Rev.* **3,** 31–39.

MULVEY, J. M. 1991. Incorporating Transaction Costs in Models for Asset Allocation. In: *Financial Optimization*, S. Zenios (ed.). Cambridge University Press, U.K.

MULVEY, J. M., AND A. RUSZCZYNSKI. 1992. A Diagonal Quadratic Approximation Method for Large Scale Linear Programs. *O.R. Letts.* **12,** 205–215.

MULVEY, J. M., AND H. VLADIMIROU. 1991. Solving Multistage Stochastic Networks: an Application of Scenario Aggregation. *Networks* **21,** 619–643.

MULVEY, J. M., AND H. VLADIMIROU. 1992. Stochastic Network Programming for Financial Planning Problems. *Mgmt. Sci.* **38,** 1642–1664.

ROCKAFELLAR, R. T., AND R. J.-B. WETS. 1991. Scenarios and Policy Aggregation in Optimization Under Uncertainty. *Math. O.R.* **16,** 1–23.

RUSZCZYNSKI, A. 1986. A Regularized Decomposition Method for Minimizing a Sum of Polyhedral Functions. *Math. Prog.* **35,** 309–333.

RUSZCZYNSKI, A. 1993. Parallel Decomposition of Multistage Stochastic Programs. *Math. Prog.* **58,** 201–228.

RUSZCZYNSKI, A. 1989. An Augmented Lagrangian Decomposition Method for Block Diagonal Linear Programming Problems. *O.R. Letts.* **8,** 287–294.

RUSZCZYNSKI, A. 1992. Augmented Lagrangian Decomposition for Sparse Convex Optimization. Math. O. R. (to appear).

STEPHANOPOULOS, G., AND W. WESTERBERG. 1975. The Use of Hestenes' Method of Multipliers to Resolve Dual Gaps in Engineering System Optimization. *J. Optim. Theory and Applic.* **15,** 285–309.

VAN SLYKE, R., AND R. J.-B. WETS. 1969. L-Shaped Linear Programs With Applications to Optimal Control and Stochastic Programming. *SIAM J. Appl. Math.* **17,** 638–663.

VANDERBEI, R. J. 1991. Symmetric Quasidefinite Matrices. Technical Report SOR 91-10, Department of Civil Engineering and Operations Research, Princeton University, Princeton, N.J.

VLADIMIROU, H. 1990. Stochastic Networks: Solution Methods and Applications in Financial Planning. PhD Thesis, Department of Civil Engineering and Operations Research, Princeton University, Princeton, N.J.

WETS, R. J.-B. 1988. Large Scale Linear Programming Techniques. In: *Numerical Methods in Stochastic Programming*, Yu Ermoliev and R. J.-B. Wets, (eds). Springer-Verlag, Berlin, 65–94.