Solving Multistage Stochastic Programs Using Tree Dissection

A.J. Berger
J.M. Mulvey
E. Rothberg
R.J. Vanderbei

Statistics and Operations Research
Technical Report SOR-95-07

Princeton University
School of Engineering and Applied Science
Department of Civil Engineering and Operations Research

June 1995

each σ-algebra there exists a finite partition of Ω such that every set in the σ-algebra is a union of sets in this finite partition.

A random variable, say Z, is simply a function defined on Ω: $Z = \{Z(\omega) : \omega \in \Omega\}$. If the function takes its values in a vector space, then it is called a *random vector*. Similarly, if it takes its values in a space of matrices, then it is called a *random matrix*. As is customary, given a random variable Z and a σ-algebra, say $\mathcal{F}_t$, we write $Z \in \mathcal{F}_t$ to indicate that Z is $\mathcal{F}_t$-measurable. Since $\mathcal{F}_t$ is finitely generated, Z is $\mathcal{F}_t$-measurable if and only if it is constant on each of the sets in the partition that generates $\mathcal{F}$.

Since Ω is finite, it is natural to introduce the "master" σ-algebra consisting of the collection of all subsets of Ω (generated by the partition consisting of the individual points of Ω). We assume that the sample space together with this master σ-algebra is also endowed with a probability measure which we denote by $\mathbf{P}$. The corresponding expectation operator shall be denoted, as usual, by $\mathbf{E}$. For each $\omega \in \Omega$, we let $p(\omega)$ denote the probability of the one-point set consisting of ω alone:

$$p(\omega) = \mathbf{P}(\{\omega\}).$$

The function p so defined is called the *probability mass function* associated with $\mathbf{P}$. Expectations are easily written in terms of this function:

$$\mathbf{E}Z = \sum_{\omega \in \Omega} Z(\omega)p(\omega).$$

Without loss of generality, we shall always assume that the probability mass function is strictly positive on Ω (otherwise points from Ω could simply be deleted).

Next, we fix an integer m and, for each $t = 0, 1, \ldots, T$, assume that we are given a random matrix A_t having m rows and n_t columns. Also given is a random m-vector b.[1] The set of *decision variables* for the stochastic program consists of a sequence of random vectors of the appropriate dimension:

$$x = \{x_t \in \mathbb{R}^{n_t} : t = 0, 1, \ldots, T\}.$$

Let $n = n_0 + n_1 + \cdots + n_T$ and let f be a concave function defined on $\mathbb{R}^n$. The function f defines our *objective function*. It is taken to be concave to allow for the modeling of risk averse behavior. The *multistage nonlinear stochastic*

[1] To be consistent with notational conventions in the optimization community, we allow lower case letters to denote random variables even though such a convention is rare in the stochastic processes community.

program can now be defined as:

(1.1)

$$\begin{array}{lll} \text{maximize} & \mathbf{E}f(x) & \\ \text{subject to} & \sum_{t=0}^{T} A_t(\omega)x_t(\omega) = b(\omega), & \omega \in \Omega, \\ & x_t(\omega) \geq 0, \quad t = 0, 1, \ldots, T, & \omega \in \Omega, \\ & x_t \in \mathcal{F}_t, \quad t = 0, 1, \ldots, T. & \end{array}$$

Constraints from the last group in (1.1) are called *non-anticipativity* constraints. These constraints have the obvious interpretation that decisions made today cannot depend on information received tomorrow (or any day thereafter).

A wide variety of real-world planning problems fit the multistage nonlinear stochastic program defined by (1.1). Some typical examples include: financial planning to maximize investor wealth over time [8, 15, 19, 24, 29], design of telecommunication facilities to achieve a robust system in the face of stochastic demands and possible equipment failure [1, 34], aircraft scheduling to meet uncertain demand [27], energy planning [16, 30], and assignment of production levels to global facilities in order to minimize uncertain currency costs. Also see the review paper by Dupačová [9] for additional applications. In each case, the model quickly becomes large as the number of decision stages T and the cardinality of the sample space Ω increase. As such, multi-stage stochastic programs often require supercomputer resources in order to find an optimal solution.

As mentioned above, for each t, the σ-algebra $\mathcal{F}_t$ is finitely generated which means that there is a partition, call it F_t, of Ω into sets $F_t^1, F_t^2, \ldots, F_t^{J_t}$ such that each set in $\mathcal{F}_t$ is a union of sets from this partition. Furthermore, a random variable is $\mathcal{F}_t$ measurable if and only if it is constant on each of the generating sets. Hence, the non-anticipativity constraints in (1.1) can be rewritten as

(1.2) $$x_t \text{ is constant on } F_t^j, \quad j = 1, 2, \ldots, J_t, \quad t = 0, 1, \ldots, T.$$

Since the σ-algebras in a filtration are increasing, it follows that each F_t^j is contained in some F_{t-1}^k. We say that F_t^j is a *descendent* of F_{t-1}^k if $F_t^j \subset F_{t-1}^k$. As shown in Figure 1, whenever $\mathcal{F}_0$ is trivial (i.e., has just one generator: $F_0^1 = \Omega$), this notion of descendency defines a tree structure on the generating sets (in general it defines a forest). This tree is called a *scenario tree.*

The constraint that x_t is constant on F_t^j can be expressed many ways. The most efficient is to enumerate the sample points belonging to F_t^j, say $\{\omega_1, \omega_2, \ldots, \omega_{K_{t,j}}\}$, and simply to equate the values of x_t from one point to the next:

$$x_t(\omega_1) = x_t(\omega_2) = \cdots = x_t(\omega_{K_{t,j}}).$$

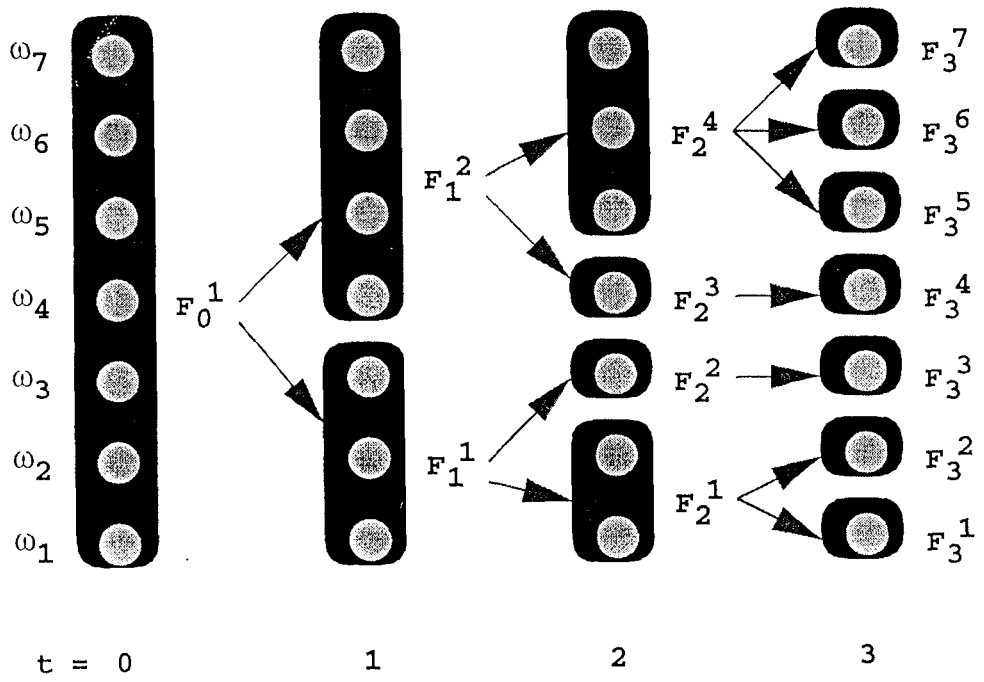

FIGURE 1. A typical scenario tree. Each filled circle represents a decision variable(s). Those decision variables enveloped within one darkened area must be equal to each other.

For example, in Figure 1 the constraint that x_2 is constant on F_2^4 is expressed as $x_2(\omega_5) = x_2(\omega_6) = x_2(\omega_7)$.

1.1. Compact vs. Expanded Representation. The problem defined by (1.1) is called the *expanded representation.* The so-called *compact representation* is obtained by introducing just a single decision vector for each generator set F_t^j, call it x_t^j, and then dropping the nonanticipativity constraints:

$$\begin{aligned} &\text{maximize} && \textstyle\sum_{\omega\in\Omega} p(\omega) f(x(\omega)) \\ &\text{subject to} && \textstyle\sum_{t=0}^{T} A_t(\omega) x_t^{j_t(\omega)} = b(\omega), \ \omega \in \Omega, \\ &&& x_t^j \geq 0, \quad j = 1, 2, \ldots, J_t, \\ &&&\qquad\quad t = 0, 1, \ldots, T, \end{aligned} \tag{1.3}$$

where $j_t(\omega)$ denotes that index j for which $\omega \in F_t^j$ and

$$x(\omega) = (x_0^{j_0(\omega)}, x_1^{j_1(\omega)}, \ldots, x_T^{j_T(\omega)}).$$

As its name implies, the compact representation possesses many fewer variables and constraints than the expanded representation (See references [18, 32, 40, 41]). It is well known, however, that for large sparse optimization problems,

structure determines algorithm efficiency more than overall size parameters such as numbers of variables and constraints. Such is the case in the present context: interior-point methods (described in the next section) applied to these problems perform just as well on the expanded representation as on the compact one (in some cases even better; see, e.g., [23, 42]). Hence, we consider the expanded representation from this point onward.

1.2. Expression in Matrix Form. For the Figure 1 example, equality constraints can be written in the following block matrix form:

$$\begin{bmatrix}
A_0\,A_1\,A_2\,A_3 & & & & & & & \\
 & A_0\,A_1\,A_2\,A_3 & & & & & & \\
 & & A_0\,A_1\,A_2\,A_3 & & & & & \\
 & & & A_0\,A_1\,A_2\,A_3 & & & & \\
 & & & & A_0\,A_1\,A_2\,A_3 & & & \\
 & & & & & A_0\,A_1\,A_2\,A_3 & & \\
 & & & & & & A_0\,A_1\,A_2\,A_3 & \\
\hline
I & -I & & & & & & \\
 & & & & I & -I & & \\
 & & & & & I & -I & \\
\hline
I & -I & & & & & & \\
 & I & -I & & & & & \\
 & & & I & -I & & & \\
 & & & & I & -I & & \\
 & & & & & I & -I & \\
\hline
I & -I & & & & & & \\
 & I & -I & & & & & \\
 & & I & -I & & & & \\
 & & & I & -I & & & \\
 & & & & I & -I & & \\
 & & & & & I & -I &
\end{bmatrix}
\begin{bmatrix}
x_0 \\ x_1 \\ x_2 \\ x_3 \\ \hline
x_0 \\ x_1 \\ x_2 \\ x_3 \\ \hline
x_0 \\ x_1 \\ x_2 \\ x_3 \\ \hline
x_0 \\ x_1 \\ x_2 \\ x_3 \\ \hline
x_0 \\ x_1 \\ x_2 \\ x_3
\end{bmatrix}
=
\begin{bmatrix}
b \\ b \\ b \\ b \\ b \\ b \\ b \\ \hline
0 \\ 0 \\ 0 \\ \hline
0 \\ 0 \\ 0 \\ 0 \\ \hline
0 \\ 0 \\ 0 \\ 0 \\ \hline
0 \\ 0 \\ 0 \\ 0
\end{bmatrix}.$$

The matrices A_t, the vectors x_t, and the right-hand side b are all random variables and their multiple appearances in the above equations represent evaluations at the different sample points. For example, in the first row of constraints each of these random variables are evaluated at ω_1, in the second row they are evaluated at ω_2, etc. In order to define an algorithm for solving the stochastic program, we shall need notations for these block matrices and vectors. Hence, we let $\boldsymbol{A}$ denote the constraint matrix, $\boldsymbol{b}$ the right-hand side vector, and $\boldsymbol{x}$ the vector of decision variables.

Viewed simply as a function of the vector of decision variables, $\boldsymbol{x}$, we write the objective function as

$$\phi(\boldsymbol{x}) = \mathbf{E}f(x) = \sum_{\omega\in\Omega} p(\omega)f(x_0(\omega), \ldots, x_T(\omega).$$

The objective function ϕ is partially separable. This structure will be exploited when developing efficient implementations, but can for the moment be forgotten.

The multistage nonlinear stochastic program can now be written succinctly as:

$$\begin{array}{ll} \text{maximize} & \phi(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0}. \end{array} \tag{1.4}$$

In the next section, we present an interior-point method for solving (1.4) and indicate that an efficient implementation of this method hinges on being able to efficiently solve a large system of equations called the reduced KKT system. For numerical stability, it is generally felt that direct methods provide the best approach to solving this system of equations. That is, some permutation of the system is to be factored and then solved using forward and backward substitution. The efficiency of direct methods depends critically on finding a good ordering heuristic to determine a row/column permutation which offers very little fill-in in the factorization process. These issues are taken up in section 3, where we describe a new ordering heuristic called nested-dissection. We refer to it as *tree dissection*. Last, section 4 contains our computational results.

2. An Interior-Point Method.

Our implementation employs the LOQO software system [36]. The algorithm is a one-phase primal-dual path-following method. It operates directly on quadratic programming problems presented in the following general form:

$$\begin{array}{ll} \text{minimize} & f + \boldsymbol{c}^T\boldsymbol{x} + \frac{1}{2}\boldsymbol{x}^T\boldsymbol{H}\boldsymbol{x} \\ \text{subject to} & \boldsymbol{b} \leq \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} + \boldsymbol{r} \\ & \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}. \end{array} \tag{2.1}$$

Here, $\boldsymbol{A}$ is an $m \times n$ matrix of coefficients, $\boldsymbol{b}$ is called the *right-hand side* (even though it appears on the left), $\boldsymbol{r}$ is the vector of *ranges* on the constraints, and $\boldsymbol{u}$ and $\boldsymbol{l}$ are the vectors of *upper bounds* and *lower bounds*, respectively, on the variables. The objective function, $f + \boldsymbol{c}^T\boldsymbol{x} + \frac{1}{2}\boldsymbol{x}^T\boldsymbol{H}\boldsymbol{x}$, is a quadratic function. Matrix $\boldsymbol{H}$ appearing in the quadratic term is assumed to be positive semidefinite so that the objective function is convex.

For the multi-stage nonlinear stochastic program given in (1.4), the lower bounds and ranges are zero and the upper bounds are infinite. Hence, we describe LOQO only as it applies to such cases. Also, the objective in (1.4) is a maximization. (To convert to minimization we simply negate the objective

function.) More crucially, the objective function in (1.4) is not necessarily quadratic. We handle general convex objective functions as follows. LOQO starts with an initial guess at the solution and iteratively updates this guess until it arrives at an essentially optimal solution. At each iteration, we approximate the general objective function by the first few terms of its Taylor series expansion to get a quadratic approximation. LOQO has built-in hooks that make it easy to modify the problem within the solution process; using these hooks we are able to handle quite easily (smooth) convex objective functions.

Before presenting the LOQO implementation, we must first introduce artificial variables as appropriate. In the present context of vanishing lower bounds and ranges and infinite upper bounds, this amounts to rewriting the problem as follows:

$$\begin{array}{lrcl} \text{minimize} & f + \boldsymbol{c}^T\boldsymbol{x} + \frac{1}{2}\boldsymbol{x}^T\boldsymbol{H}\boldsymbol{x} & & \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} - \boldsymbol{w} & = & \boldsymbol{b} \\ & \boldsymbol{w} + \boldsymbol{p} & = & \boldsymbol{0} \\ & \boldsymbol{x}, \boldsymbol{w}, \boldsymbol{p} & \geq & \boldsymbol{0}. \end{array} \tag{2.2}$$

The reader may be wondering why we have included the second constraint. Since $\boldsymbol{w}$ and $\boldsymbol{p}$ are assumed to be nonnegative, the second constraint implies that both $\boldsymbol{w}$ and $\boldsymbol{p}$ must be zero. But, as we shall see shortly, our method is an infeasible method. Thus equality constraints are not enforced at the start – they only take effect as the algorithm approaches an optimal solution. In fact, both vectors $\boldsymbol{w}$ and $\boldsymbol{p}$ are initialized to be strictly positive; the iterations of the algorithm eventually push them close to zero (they never get quite there). By putting these positive variables into the formulation we arrive at an algorithm that is much more stable than what one would get without them. As far as we know, LOQO is the only implementation of an interior-point method that exploits this observation.

The dual of (2.2) is:

$$\begin{array}{lrcl} \text{maximize} & f + \boldsymbol{b}^T\boldsymbol{y} - \frac{1}{2}\boldsymbol{x}^T\boldsymbol{H}\boldsymbol{x} & & \\ \text{subject to} & \boldsymbol{A}^T\boldsymbol{y} + \boldsymbol{z} - \boldsymbol{H}\boldsymbol{x} & = & \boldsymbol{c} \\ & \boldsymbol{y} + \boldsymbol{q} - \boldsymbol{v} & = & \boldsymbol{0} \\ & \boldsymbol{z}, \boldsymbol{v}, \boldsymbol{q} & \geq & \boldsymbol{0} \\ & \boldsymbol{y} \text{ free.} & & \end{array} \tag{2.3}$$

2.1. Central Path. A critical feature of interior-point methods is the primal-dual *central path.* We parametrize this path by a positive real parameter μ. Indeed, for each $\mu > 0$, we define the associated central-path point in primal-dual space as the unique point that simultaneously satisfies the conditions of

primal feasibility, dual feasibility, and μ-complementarity. Ignoring nonnegativity (which is enforced separately), these conditions are:

$$
\begin{aligned}
Ax - w &= b \\
w + p &= 0 \\
\\
A^T y + z - Hx &= c \\
y + q - v &= 0 \\
\\
XZe &= \mu e \\
VWe &= \mu e \\
PQe &= \mu e.
\end{aligned}
\tag{2.4}
$$

The last three equations are the μ-complementarity conditions. As usual, each upper case letter that appears on the left in these equations denotes the diagonal matrix having the components of the corresponding lower-case vector on its diagonal. The nonlinear system (2.4) consists of $2n + 5m$ equations in $2n + 5m$ unknowns. It has a unique solution in the strict interior of the appropriate orthant in primal-dual space:

$$
\{(x, w, p, y, z, v, q) : x, w, p, z, v, q \geq 0\}. \tag{2.5}
$$

This fact can be seen by noting that these equations are the first order optimality conditions for an associated strictly convex barrier problem (see, e.g. [39]).

As μ approaches to zero, the central path converges to the optimal solution to both the primal and dual problems. A *primal-dual path-following algorithm* presents an iterative process that starts from a point in the strict interior of (2.5) and at each iteration estimates a value of μ representing a point on the central path that is in some sense closer to the optimal solution than the current point and then attempts to step toward this central-path point making sure that the new point remains in the strict interior of the appropriate orthant.

2.2. Apply Newton's Method to get Step Directions. Suppose for the moment that we have already decided on the target value for μ and let $(x, \ldots, q)$ denote the current point in the strict interior of (2.5). Applying Newton's method to find a solution to (2.4), we get the following equations

for the step directions $(\Delta x, \ldots, \Delta q)$:

(2.6)
$$\begin{aligned}
A\Delta x - \Delta w &= b - Ax + w &&=: \rho \\
\Delta w + \Delta p &= -w - p &&=: \alpha \\
A^T\Delta y + \Delta z - H\Delta x &= c - A^T y - z + Hx &&=: \sigma \\
-\Delta y - \Delta q + \Delta v &= y + q - v &&=: \beta \\
X^{-1}Z\Delta x + \Delta z &= \mu X^{-1}e - z &&=: \gamma_z \\
V^{-1}W\Delta v + \Delta w &= \mu V^{-1}e - w &&=: \gamma_w \\
P^{-1}Q\Delta p + \Delta q &= \mu P^{-1}e - q &&=: \gamma_q,
\end{aligned}$$

where we have introduced notations $\rho, \ldots, \gamma_q$ as shorthands for the right-hand side expressions. This system can be mostly solved by inspection yielding a much smaller system to solve. The details are given in [36] so we just summarize the result here. First, we must solve

(2.7)
$$\left[\begin{array}{c|c} -(H + \bar{D}) & A^T \\ \hline A & E \end{array}\right] \left[\begin{array}{c} \Delta x \\ \hline \Delta y \end{array}\right] = \left[\begin{array}{c} \sigma - X^{-1}Z\hat{\nu} \\ \hline \rho - E(\hat{\beta} - P^{-1}Q\hat{\alpha}) \end{array}\right],$$

where E and $\bar{D}$ are diagonal matrices given by

$$E = \left(VW^{-1} + P^{-1}Q\right)^{-1}$$

and

(2.8)
$$\bar{D} = X^{-1}Z,$$

and $\hat{\alpha}$, $\hat{\beta}$, and $\hat{\nu}$ are given by

$$\begin{aligned}
\hat{\alpha} &= \alpha - PQ^{-1}\gamma_q \\
\hat{\beta} &= \beta - VW^{-1}\gamma_w \\
\hat{\nu} &= XZ^{-1}\gamma_z.
\end{aligned}$$

Then, compute the remaining step directions sequentially as follows:

$$\begin{aligned}
\Delta w &= -E(\hat{\beta} - P^{-1}Q\hat{\alpha} + \Delta y) \\
\Delta q &= P^{-1}Q(\Delta w - \hat{\alpha}) \\
\Delta z &= X^{-1}Z(\hat{\nu} - \Delta x) \\
\Delta p &= PQ^{-1}(\gamma_q - \Delta q) \\
\Delta v &= VW^{-1}(\gamma_w - \Delta w).
\end{aligned}$$

System (2.7) is called the *reduced KKT system.* We emphasize that both E and $\bar{D}$ have *strictly positive* diagonal entries. The fact that E has strictly positive diagonal entries is a direct consequence of our allowing w and p to be

strictly positive and having the algorithm force these variables toward zero as optimality is approached (as opposed to setting them to zero from the start).

2.3. Compute Step Lengths. Finally, for Newton's method the desired step length is one, but a ratio test evaluates whether it needs to be shortened to maintain strict positivity of the nonnegative variables.

2.4. Solving the Reduced KKT System. There are many possible orders in which one can solve the reduced KKT system. For example, if we were to use the first equation in (2.7) to solve for Δx and then eliminate it from the second equation, the resulting system for Δy involves the matrix

$$ADA^T \text{ where } D = (\bar{D} + H)^{-1}$$

(this is the *normal-equations* approach used in OB1 [21, 22] and CPLEX-barrier). Similarly, if we solve the second equation for Δy and then eliminate this variable from the first equation, the resulting system for Δx involves the matrix

$$A^T E^{-1} A.$$

(This is the approach advocated by the optimization group at the National Institute of Standards and Technology [6, 33]). Both of these may entail fill-in, which in some cases can be considerable. For example, if the Hessian H is not diagonal or if the matrix A has a dense column, the first form will suffer "catastrophic fill-in". On the other hand, if A has a dense row, the second form will be very bad. If A has both dense columns and dense rows, then both of these forms will suffer unnecessarily large amounts of fill-in and one would prefer, in that case, to work with the larger system to find a pivot order that does not generate so much fill-in. This idea was first suggested by Turner [35] and has been adopted by Saunders et al. [12, 13, 14] and by Fourer and Mehrotra [10]. All three use a Bunch-Parlett factorization of the indefinite system. Solving the larger system is also the approach adopted in LOQO, but LOQO does not employ a Bunch-Parlett factorization. Instead, LOQO modifies a Cholesky factorization to solve *symmetric quasidefinite systems* ([37, 38]). Equation (2.7) is an example of such a system. The matrices defining these systems share with symmetric semidefinite matrices the desirable property that the (diagonal) pivots can be selected based only on a fill-in minimizing heuristic, i.e., without regard for the numerical values, which may only be known later.

There are two types of fill-in minimizing heuristics: myopic heuristics, such as *minimum-degree* [11], which sequentially attempt to minimize the fill-in produced in each subsequent stage of elimination, and global heuristics, such as *nested-dissection* [31], which analyze the overall structure to find good orderings. In the symmetric quasidefinite system (2.7), there is an obvious global

structure that one can exploit. For example, the lower-right block is a diagonal matrix (as is the upper-left block whenever H is diagonal or absent). Hence, initial pivots selected from the lower-right block can only produce fill-in in the upper-left block. It is generally advantageous to exploit this structure. Hence, we employ a priority minimum-degree method in LOQO. Each diagonal element is assigned a small integer representing an elimination priority. At first, pivots are selected only from the elements assigned priority zero. Within this priority class, pivots are selected according to the usual minimum degree heuristic. Only after all priority zero pivots have been eliminated do we proceed to the priority one elements. Again, within this priority class, the elimination order is determined using the minimum-degree heuristic. This process is continued through each priority class until all elements are eliminated.

By default, LOQO uses a heuristic to determine how to set the priority classes. However, for multi-stage nonlinear stochastic programs we have altered the code so that the priority classes are set to reflect the inherent scenario tree structure. We call this ordering tree dissection; it is discussed in detail in the next section.

2.5. Further Issues. There are many further implementations issues that we have not touched on such as

- Initializing $(x, \ldots, q)$.
- Detecting optimality, unboundedness, and infeasibility.
- Setting μ appropriately in each iteration.
- Computing step lengths.
- Using the so-called predictor-corrector technique to find better search directions.
- Using iterative refinement and diagonal perturbation to improve numerical stability.

These issues and more are discussed in depth in [36]. Applications of interior point methods in stochastic programming and robust optimization can be found in [4, 5, 17, 28, 42].

3. Tree Dissection.

As mentioned, the primary bottleneck in interior-point linear programming is the solution of a series of sparse linear systems of equations. These systems are normally solved using a direct method, where the matrix to be factored, for example, $M = ADA^T$ is modified into the form $M = LDL^T$, where L is lower triangular with unit diagonal and D is diagonal, if the Hessian is diagonal. The first step in this factorization is a symmetric permutation of M to reduce *fill* in the factor L. As mentioned, the minimum degree ordering

heuristic has proven to be an effective, general purpose heuristic for reducing fill (e.g., LOQO, CPLEXBarrier). Unfortunately, as we will soon discuss, minimum degree is ineffective for the matrices generated for multi-period stochastic optimization problems. These problems possess an appealing structure that the minimum degree heuristic does not exploit. This section describes the structure that is present in these problems and the approach we use to capture this structure in the ordering.

3.1. The Graph Theoretic Approach to Sparse Factorization. Our ordering approach is perhaps best explained in terms of the graph underlying sparse matrix factorization. In graph terms, a symmetric sparse matrix M is represented as an undirected graph $G = (V, E)$, where there is one vertex $v \in V$ for every row/column in the graph, and one edge $(i, j) \in E$ for every non-zero value $A_{i,j}$ in $\boldsymbol{M}$. The process of factoring a matrix $\boldsymbol{M}$ into $\boldsymbol{M} = \boldsymbol{L}\boldsymbol{L}^T$ is one of eliminating the nodes, $\{1 \ldots n\}$, from $\boldsymbol{M}$. When node i is eliminated, the nodes adjacent to i become adjacent to one another. Any such edge that did not previously exist represents *fill* in the factor matrix. In other words, that edge represents a non-zero in the factor $\boldsymbol{L}$ that did not exist in $\boldsymbol{M}$. For example as shown in Figure 2, the elimination of node 1 causes the matrix to become completely dense at the subsequent iteration. In contrast, Figure 3 shows that eliminating node 2 preserves the sparsity in the reduced matrix.

The permutation of the matrix determines the order in which the nodes are eliminated. The minimum degree heuristic sequences the nodes in G such that, at every point, the eliminated node has the minimum degree. The intuition behind this approach is that the minimum degree node corresponds to the one whose elimination generates the least fill in that step on average.

An alternative approach, called *nested dissection*, performs the ordering by finding *separators* in G, where a separator is a set of nodes whose removal from G creates two disconnected components of roughly equal size. The separator nodes are eliminated *after* the remaining nodes. By postponing the elimination of separator nodes, it is guaranteed that the nodes in one component can not become adjacent to those in the other component before the separator nodes are eliminated. This decomposes the problem into two disjoint sub-problems. The dissection process is then repeated recursively on the pieces created by top-level separator.

3.2. The Structure of Multi-Period Modeling Matrices. Let us now consider the overall structure of a multi-period modeling constraint matrix. Recall that the matrix contains s replications of the original linear programming constraint matrix, where s is the number of scenarios. All involve different variables and different constraints, thus creating a block-diagonal global constraint matrix. In addition, the global constraint matrix includes several

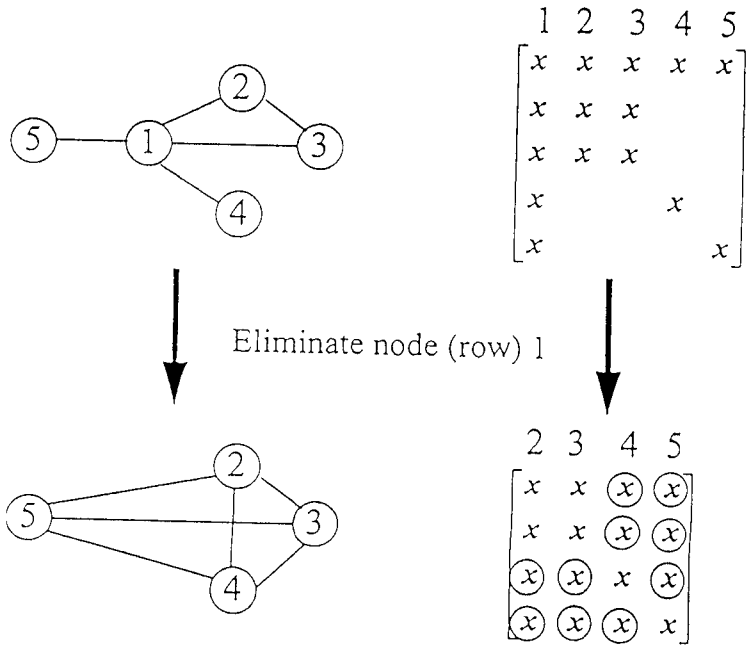

FIGURE 2. Graph representation of Cholesky factorization

non-anticipativity constraints that equate some time-period variables to reflect immutable choices made in the past. The actual set of linking constraints is determined by the scenario tree. In particular, the variables for the first time-period are equal across scenarios 1 through s. Those for the second time period are equal across 1 through $s/2$ and across $s/2+1$ through s. Equality across a set of scenarios is achieved by linking variables in adjacent pairs of scenarios. Figure 1 shows the structure of an 8 scenario model.

To simplify our presentation, we assume that the original constraint matrix is dense. In practice, of course, this is a pessimistic assumption. In the context of multistage stochastic program, simple analysis reveals that:

- Scenario constraint nodes are adjacent to all other scenario constraint nodes from the same scenario.
- Scenario constraint nodes are adjacent to all linking constraint nodes related to that scenario.

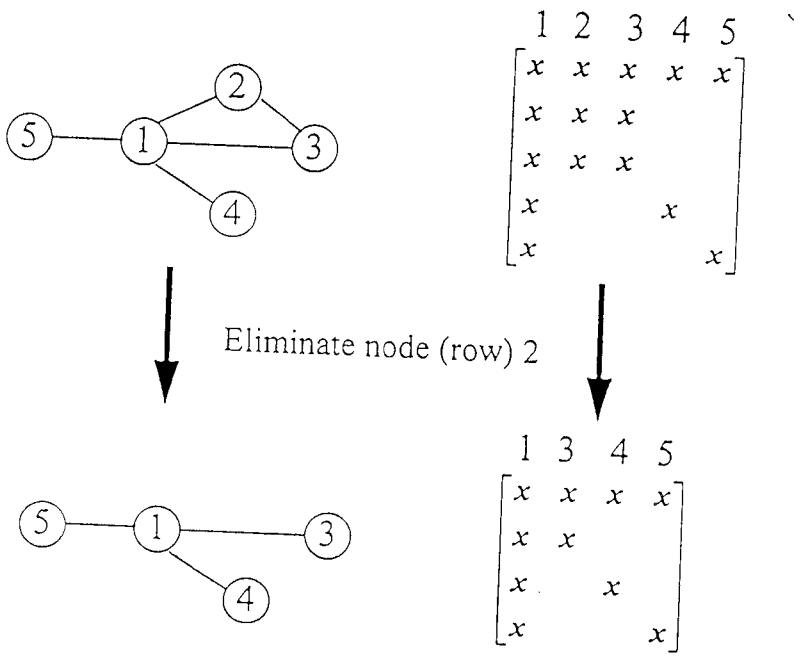

FIGURE 3. Graph representation of Cholesky factorization – alternative elimination

- A linking constraint node is adjacent to the single linking constraint node that links that variable to the corresponding variable in a neighboring scenario.
- A linking constraint node for variable x is *not* adjacent to any linking constraint nodes associated with variables other than x.

3.3. Tree Dissection. The structure captured by the scenario tree provides a natural method for ordering the matrix in the reduced KKT system (2.7), or the alternative normal-equations. Specifically, we can take advantage of the graph separator property of nested-dissection to trivially decompose the problem. Consider three sub-graphs of G, G_1, G_2, and S, where G_1 contains all scenario constraint nodes for scenarios 1 through $s/2$ plus all linking constraint nodes that link pairs of variables from these scenarios. Sub-graph G_2 contains the corresponding nodes for scenarios $s/2+1$ through s. Sub-graph S contains

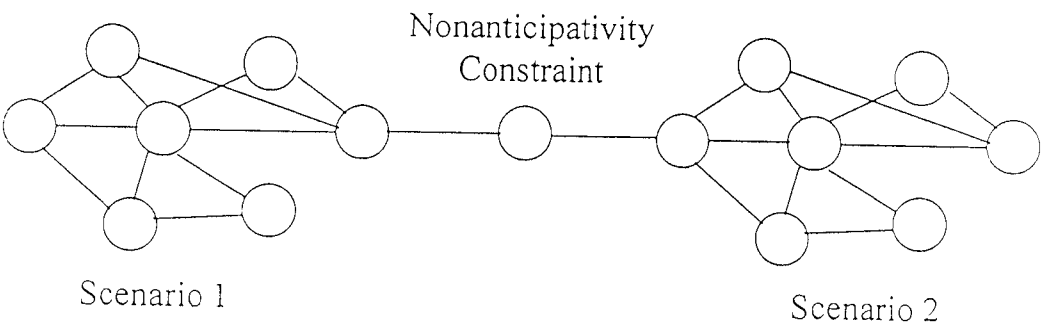

FIGURE 4. Conceptual graph for split variable stochastic program

the linking constraint nodes associated with the constraints that link period-one variables from scenarios $s/2$ and $s/2+1$. Note that there are no edges between G_1 and G_2. Thus, S forms a small graph separator for G (see Figure 4 for a graph representation).

Now consider graph G_1. The set of linking constraint nodes that link variables in scenarios $s/4$ and $s/4+1$ form a separator in this graph. This separator is twice the size of the separator for G, but it is still quite small. The process of choosing a small set of linking constraint nodes as separators can continue recursively until the problem has been decomposed into a set of original one-scenario constraint matrices. At no point in this process do two scenario constraint nodes become adjacent. We call this process of applying a nested dissection type ordering approach to the scenario tree *tree dissection*.

3.4. Asymptotic Growth Rates for Tree Dissection. Recall that in tree dissection, the factorization happens in stages, where at each stage some set

of separators divides the problem into sub-problems. At stage 1, a single separator of size n_s, the number of variables in scenario s, divides the problem into two parts. In stage 2, two separators, each of size $2n_s$, divide the problem into four parts. In stage 3, four separators of size $3n_s$ divide the problem into eight parts, and so on. Finally, in stage $\log s$, the problem has been divided into individual constraint matrices. The factorization itself proceeds in the reverse order: it first eliminates the constraint nodes, then the lowest level of separators nodes, then the next level, and so on. Three observations allow the complexity of the factorization to be bounded:

- The factorization problem that results from the elimination of all scenario nodes and the lowest level of separator nodes in an s-scenario problem is identical to the factorization problem the results from the elimination of all scenario nodes in an $s/2$-scenario problem.
- The cost of eliminating a linking node is $O(\log^2 s)$.
- The cost of eliminating a scenario node is $O(\log^2 s)$.

These observations form the basis of a simple recurrence. The solution to this recurrence gives an asymptotic growth rate of $O(s \log^3 s)$. In practice, we have found that the runtime is linear in the number of scenarios, as long as adequate memory is available.

3.5. Minimum Degree Ordering. Now consider the ordering that would be produced by the minimum degree heuristic for this matrix. The initial degree of a scenario constraint node is equal $m + 2n$, where m and n are the number of constraints and variables in a single scenario, respectively. The node is adjacent to all other nodes in the same scenario (n), and all linking constraint nodes associated with variables in that scenario ($2n$ since there are two linking constraints for each variable). In contrast, the linking constraint nodes are adjacent to $1 + 2n$ nodes, where n is the number of variables in a scenario. A linking constraint node is adjacent to the other linking constraint node associated with the same variable (1 adjacency) and to all scenario constraint nodes within the same scenario ($2n$). If the degrees are compared, one can see that the MMD heuristic eliminates linking constraint nodes before eliminating scenario constraint nodes. It is also apparent that the linking constraint nodes that form the top-level separator in our tree-dissection approach have no distinguishing properties in MMD. They could easily be eliminated first. In other words, MMD is unable to capture the structure inherent in the multistage stochastic program.

4. Computational Results.

The test problem comes from the area of financial optimization. An individual needs to decide on an investment strategy given her current asset position

and outstanding debts. This example portrays a doctor who has $80,000 in assets, $65,000 in medical school loans (two variable rate and one fixed rate), and an income of $75,000. We model 7 different asset categories she can choose from, and she has the possibility of repaying some or all of the loans before their termination date. The planning horizon is divided into discrete time stages (annual reviews) at which loan payments are made, assets can be rebalanced, and income received. The decision variables represent the amount of money allocated to each asset category and the outstanding balance in each liability category at every time period. The constraints represent flow of funds among categories as well as interest payments and transaction costs. There have been a number of papers on how to formulate this problem in the stochastic programming framework, with linear constraints and linear, quadratic, or convex objective functions. See [2], [8], or [25]. In these tests we look at the linear objective of maximizing expected wealth at the end of the planning horizon.

The randomness in the problem occurs both on the asset and liability side of the customer's future balance sheet. Asset returns, on stocks, bonds, cash, and other asset categories are random variables, and adjustable rate loans will depend on uncertain future interest rates. We forecast scenarios which are consistent streams of economic factors and asset returns. The scenarios possess the standard tree structure described in section 2 (see [2] or [26] for more details).

The "base" (single-scenario) model possesses 96 constraints, 156 variables, and 6 time stages, corresponding to annual portfolio rebalancings. Each of the first 5 time stages takes 16 constraints and 25 decisions variables. When randomness is introduced, problem size will depend on the number of scenarios and the scenario tree structure. Table 1 shows model size. In the table, "Branchings" denotes the number of splits at each level of the tree, "Compact Form" refers to the formulation in system 1.3, and "Split Variable Form" refers to the formulation for the solution procedure described in this paper (system 1.1).

All problems were run on a Silicon Graphics Power Challenge workstation with a R8000/R8010 chip running at 100 MHz and 4GB main memory. The tree dissection method was incorporated into LOQO version 2.07 [36]. Comparisons were rendered with this version of LOQO and CPLEX version 3.0.

Table 2 shows the arithmetic operations required to factor the reduced KKT system (2.7) of the split variable formulation. "Liu Min Degree" refers to the original and most widely used implementation of multiple minimum degree ordering [20]. "LOQO" refers to default settings for version 2.07 and "LOQO Dual First" prioritizes eliminating Δy from system (2.7) before eliminating Δx . There are two important points to note. First, the tree dissection method

		Compact Form		Split Variable Form	
Scenarios	Branchings	Constraints	Variables	Constraints	Variables
1,024	16x4x4x2x2	65,000	109,000	146,000	160,000
2,048	16x8x4x2x2	131,000	218,000	299,000	319,000
4.096	32x8x4x2x2	263,000	435,000	597,000	639,000
8,192	32x16x4x2x2	525,000	869,000	1,195,000	1,278,000
16,384	64x16x4x2x2	1,050,000	1,738,000	2,390,000	2,556,000

TABLE 1. Problem size for tree dissection tests, rounded to nearest thousand.

greatly outperforms the standard minimum degree ordering techniques found in LOQO. On the 4,096 scenario problem, there is a 160-fold reduction in the number of operations, and both LOQO and Liu multiple minimum degree exceeded available memory trying to factor the larger eight and sixteen thousand scenario problems. Second, the number of operations for tree dissection grows linearly in the number of scenarios. This bodes well for algorithm scalability in terms of solution time given adequate memory.

Runtime comparisons can be found in Table 3. Problems were solved to a primal-dual relative optimality gap of less than 10^{-5}. Tree dissection greatly improves primal-dual algorithms, such as LOQO, for multistage stochastic programs. Of course, the full reduction in operation counts doesn't translate exactly to run times since the factorization is only a part (albeit a large part) of total computation. On the 4,096 scenario problem, there is still an improvement by a factor of 10 versus LOQO Dual First and the problem could not be solved with default LOQO. These initial tests indicate that we can solve smaller problems faster and larger problems that were unsolvable with standard methods. As other researchers have discovered, the number of interior-point iterations appears to be relatively independent of problem size. In addition, the algorithm appears to have almost linear scalability in the number of scenarios, especially as the problem size increases.

LOQO compares favorably with commercial codes for interior point methods. Table 4 shows a comparison of LOQO with CPLEX on the primal formulation and CPLEX on the dual formulation for this financial problem. Although the site license at Princeton University is restricted to a 32 MB workstation, we still see LOQO and CPLEX perform comparably on these small problems. Further evidence of the viability of LOQO *vis á vis* commercial solvers can be found in [36].

Scenarios	Liu Min Degree	LOQO default	LOQO "DUAL First"	Tree Dissection
512	4,300	15,300	655	102
1,024	10,500	57,200	1,660	207
2,048	82,800	384,000	8,100	453
4,096	182,000	1,010,000	25,800	909
8,192	1,480,000	NS	129,000	1,910
16,384	NS	NS	309,000	3,830

TABLE 2. Arithmetic operations (in millions) to factor reduced KKT system, three significant digits. NS denotes problem not solved due to excessive memory requirements.

Scenarios	LOQO default	LOQO "DUAL First"	Tree Dissection	Iter. for Tree Diss.
512	423	18	7.5	31
1,024	NS	45	24	49
2,048	NS	431	86	73
4,096	NS	1776	167	70
8,192	NS	NS	330	57
16,384	NS	NS	600	51

TABLE 3. Runtimes (in minutes) until solution. Final relative optimality gap $< 10^{-5}$. NS denotes problem not solved due to excessive memory requirements.

Scenarios	LOQO	CPLEX	CPLEX(on dual)
32	32.6	12.7	13.1
64	100	55.8	57.6
128	340	391	388

TABLE 4. Run Times (Seconds R4000 SGI Computer) for Stochastic Programming without Tree Dissection

5. FUTURE DIRECTIONS

The tree dissection method improves the efficiency of nonlinear interior-point algorithms by a considerable amount for large-scale multistage stochastic programs – over 150 times for the large examples. Thereby, we can solve multistage stochastic programs that recently required decomposition approaches, such as nested Benders [7], or diagonal quadratic approximation [3]. This expands the scope of stochastic programs by opening up new application domains.

What are some possibilities? The first is near "real-time" scheduling in

which decisions are made on a regular basis (such as truck dispatching or airline scheduling) throughout the day. The fleet assignments are continually monitored and controlled via the optimization module. Of course, the optimization model does take some time to solve, even with massively parallel computers, hence, there is a lag between the decision intervals. A similar application is possible in the area of finance in which trading strategies can be designed to capture market imperfections.

Another application lies in extending the split variable approach for more general linear programs. If tree dissection is effective for split variables in stochastic programming, we can induce the tree structure on any LP by adding new split variables. This approach needs to be tested with large examples to determine its tractability.

In the near future, we plan to parallelize the tree dissection method using the ideas of Ed Rothberg, and to integrate the approach with the diagonal quadratic approximation algorithm [3].

REFERENCES

[1] D. Bai, T. Carpenter, and J. Mulvey. Stochastic programming to promote network survivability. Technical Report Report SOR-94-14, Department of Civil Engineering and Operations Research, Princeton University, 1994.

[2] A.J. Berger. *Large scale stochastic optimization with applications to finance.* PhD dissertation, Department of Civil Engineering and Operations Research, Princeton University, Princeton, NJ 08544, 1995.

[3] A.J. Berger, J.M. Mulvey, and A. Ruszczyński. An extension of the DQA algorithm to convex stochastic programs. *SIAM Journal on Optimization*, 4:735–753, 1994.

[4] J.R. Birge and D.F. Holmes. Efficient solution of two-stage stochastic linear programs using interior point methods. *Computational Optimization and Applications*, pages 245–276, 1992.

[5] J.R. Birge and L. Qi. Computing block-angular Karmarkar projections with applications to stochastic programming. *Management Science*, (12):1472–1479, 1988.

[6] P.T. Boggs, P.D. Domich, J.E. Rogers, and C. Witzgall. An interior-point method for linear and quadratic programming problems. Technical Report NISTIR-4556, National Institute of Standards and Technology, 1991.

[7] G.B. Dantzig and G. Infanger. Large-scale stochastic linear programs: importance sampling and Benders decomposition. Technical Report Report SOL-91-4, Department of Operations Research, Stanford University, 1991.

[8] G.B. Dantzig and G. Infanger. Multi-stage stochastic linear programs for portfolio optimization. *Annals of Operations Research*, (45):59–76, 1993.

[9] J. Dupačová. Multistage stochastic programs: the state-of-the-art and selected bibliography. *Kybernetika*, 31:151–174, 1995.

[10] R. Fourer and S. Mehrotra. Solving symmetric indefinite systems in an interior point method for linear programming. *Math. Prog.*, 62:15–40, 1991.

[11] A. George and J.W.H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31:1–19, 1989.

[12] P.E. Gill, W. Murray, D.B. Ponceleón, and M.A. Saunders. Primal-dual methods in linear programming. Technical Report SOL 91-3, Systems Optimization Laboratory, Stanford Univ., Stanford, CA, April 1991.

[13] P.E. Gill, W. Murray, D.B. Ponceleón, and M.A. Saunders. Solving reduced KKT systems in barrier methods for linear and quadratic programming. Technical Report SOL 91-7, Systems Optimization Laboratory, Stanford Univ., Stanford, CA, July 1991.

[14] P.E. Gill, W. Murray, D.B. Ponceleón, and M.A. Saunders. Preconditioners for indefinite systems arising in optimization. *SIAM J. Matrix Anal. Appl.*, 13(1):292–311, 1992.

[15] B. Golub, M.R. Holmer, R. McKendall, L. Pohlman, and S.A. Zenios. A stochastic programming model for money management. Technical report, Decision Science Department, The Wharton School, University of Pennsylvania, Philadelphia, PA 19104, 1993.

[16] G. Infanger. *Planning under uncertainty: solving large-scale stochastic linear programs.* Boyd and Fraser Publishing Company, 1994.

[17] E.R. Jessup, D. Yang, and S.A. Zenios. Parallel factorization of structure matrices arising in stochstic programming. *SIAM Journal on Optimization*, (4):833–846, 1994.

[18] P. Kall and S.W. Wallace. *Stochastic Programming.* John Wiley & Sons, New York, 1994.

[19] A.J. King and T. Warden. Stochastic programming for strategic portfolio management. 15th International Programming Symposium, Ann Arbor, August, 1994.

[20] J. Liu. Modification of the minimum degree algorithm by multiple elimination. *ACM Trans. Math. Software*, 11:141–153, 1985.

[21] I.J. Lustig, R.E. Marsten, and D.F. Shanno. Computational experience with a primal-dual interior point method for linear programming. *Lin. Alg. and Appl.*, 152:191–222, 1991.

[22] I.J. Lustig, R.E. Marsten, and D.F. Shanno. On implementing Mehrotra's predictor-corrector interior point method for linear programming. *SIAM J. on Optimization*, 2:435–449, 1992.

[23] I.J. Lustig, J.M. Mulvey, and T.J. Carpenter. Formulating stochastic programs for interior point methods. *Operations Research*, 39:757–770, 1991.

[24] J.M. Mulvey. Financial planning via multi-stage stochastic optimization. To appear *ORSA Journal of Computing*, 1995.

[25] J.M. Mulvey. A surplus optimization perspective. *Investment Management Review*, 3:31–39, 1989.

[26] J.M. Mulvey. Generating scenarios for the Towers Perrin investment systems. Technical report, Princeton University, Princeton, NJ, 1994. To appear *Interfaces*, 1995.

[27] J.M. Mulvey and A. Ruszczyński. A new scenario decomposition method for large-scale stochastic optimization. *Operations Research*, 1995.

[28] J.M. Mulvey, R. J. Vanderbei, and S.A. Zenios. Robust optimization of large-scale systems. *Operations Research*, (2):264–281, 1995.

[29] S.S. Nielsen and S.A. Zenios. A stochastic programming model for funding single premium defered annuities. Technical report, Decision Science Department, The Wharton School, University of Pennsylvania, Philadelphia, PA 19104, 1992.

[30] M.V. Pereira and L.M.V.G. Pinto. Multistage stochastic optimization applied to energy planning. *Mathematical Programming*, pages 359–375, 1991.

[31] S. Pissanetzky. *Sparse Matrix Technology.* Academic Press, 1984.

[32] R.T. Rockafellar and R.J.-B. Wets. Nonanticipativity and $\mathcal{L}^1$-martingales in stochastic optimization problems. *Math. Programming Study*, 6:170–187, 1976.
[33] J.E. Rogers, P.T. Boggs, and P.D. Domich. A predictor-corrector-O3D formulation for quadratic programming. Technical report, National Institute of Standards and Technology, 1994.
[34] S. Sen, R. D. Doverspike, and S. Cosares. Network planning with random demand. Technical report, Univ. of Arizona, Tucson, AZ, 85721, 1992.
[35] K. Turner. Computing projections for the Karmarkar algorithm. *Linear Algebra and Its Applications*, 152:141–154, 1991.
[36] R.J. Vanderbei. LOQO: An interior point code for quadratic programming. Technical Report SOR-94-15, Department of Civil Engineering and Operations Research, Princeton University, Princeton, NJ 08544, 1994.
[37] R.J. Vanderbei. Symmetric quasi-definite matrices. *SIAM J. Optimization*, 1994. To appear.
[38] R.J. Vanderbei and T.J. Carpenter. Symmetric indefinite systems for interior point methods. *Mathematical Programming*, (58):1–32, 1993.
[39] R.J. Vanderbei, A. Duarte, and B. Yang. An algorithmic and numerical comparison of several interior-point methods. Technical Report SOR 94-05, Princeton University, 1994.
[40] R.J.-B. Wets. On the relation between stochastic and deterministic optimization. In: Control Theory, Numerical Methods and Computer Systems Modelling (A. Bensoussan and J.L. Lions,eds.), Lecture Notes in Econom. and Math. Systems 107, Springer, Berlin 1975:350–361.
[41] R.J.-B. Wets. Stochastic programming. In: Handbook on OR and MS (G.L. Nemhauser et al., eds.), 1, North-Holland, Amsterdam 1989:573–629.
[42] D. Yang and S.A. Zenios. A scalable parallel interior point algorithm for stochastic linear programming and robust optimization. Report 95-07, Department of Public and Business Administration, University of Cyprus, Nicosia, Cyprus, 1995.

STATISTICS AND OPERATIONS RESEARCH, PRINCETON UNIVERSITY, PRINCETON, NJ 08544
E-mail address: ajberger@emerald.princeton.edu

STATISTICS AND OPERATIONS RESEARCH, PRINCETON UNIVERSITY, PRINCETON, NJ 08544
E-mail address: mulvey@macbeth.princeton.edu

SILICON GRAPHICS INC., 2011 N. SHORELINE BOULEVARD, MOUNTAIN VIEW, CA 94043
E-mail address: rothberg@multnomah.engr.sgi.com

STATISTICS AND OPERATIONS RESEARCH, PRINCETON UNIVERSITY, PRINCETON, NJ 08544
E-mail address: rvdb@princeton.edu