EFFICIENT SOLUTION OF TWO STAGE STOCHASTIC LINEAR PROGRAMS USING INTERIOR POINT METHODS

John R. Birge
and
Derek Holmes

Department of Industrial & Operations Engineering
University of Michigan
Ann Arbor, MI 48109-2117

Efficient Solution of Two Stage Stochastic Linear Programs using Interior Point Methods

J. Birge

Department of Industrial and Operations Engineering

University of Michigan

Ann Arbor, MI 48109

(313) 764-9422

email: John.R.Birge@um.cc.umich.edu

D. Holmes

Department of Industrial and Operations Engineering

University of Michigan

Ann Arbor, MI 48109

(313) 764-9422

30 December 1991.

email: Derek.Holmes@um.cc.umich.edu

Keywords: Interior-point algorithms, stochastic programming.

Abstract

Solving deterministic equivalent formulations of two stage stochastic linear programs using interior point methods may be computationally difficult, due to the need to factorize quite dense search direction matrices (e.g. AA^T). Several methods for improving the algorithmic efficiency of interior point algorithms by reducing the density of these matrices have been proposed in the literature. Reformulating the program decreases the effort required to find a search direction, but at the expense of increased problem size. Using transpose product formulations (e.g. $A^T A$) work well, but are highly problem dependent. Schur complements require solutions to potentially near singular matrices, and so suffer from numeric instabilities. Explicit factorizations of the search direction matrices eliminate these problems while only requiring the solution to several small, independent linear systems. These systems may be distributed across multiple processors. Computational experience with these methods suggest that substantial performance improvements are possible with each method, but the least computational effort is required by the use of explicit factorizations.

1 Introduction

Many practical problems with uncertain parameters can be modeled as stochastic programs. Some examples include cash and portfolio management models (Mulvey [34]), electric power generation capacity planning (Louveaux [28]), and forestry management (Gassman, [19]). A survey of documented stochastic programming formulations can be found in King [27]. Most basic among stochastic programs are Stochastic Linear Programs (SLPs) which are the stochastic extensions of standard linear programs.

Even small linear programs may lead, however, to large SLPs and extensive computational requirements since the size of these problems typically grows exponentially with the number of stochastic parameters in the formulation. The recent advent of interior point methods for the solution of large linear programs (Marsten, et.al. [30], Carolan, et. al. [10], and Monma and Morton [33]), however, holds great promise for the efficient solution of these problems. The basic requirement is the efficient solution of a sequence of large, symmetric, positive definite systems of linear equations. Generally, the solutions to these systems are obtained by factoring the coefficient matrix into some equivalent triangular matrix and back solving with a given right hand side. The ease with which the factorizations are obtained decreases significantly as the density of the coefficient matrix increases. Unfortunately, the structure of SLPs can lead to quite dense systems, limiting the use of interior point methods for their solution.

The purpose of this paper is to review various methods for improving the efficiency of solving the linear systems associated with (two stage) Stochastic Linear Programs, and report both serial and parallel computational experience with one particularly promising method described in Birge and Qi [6]. This method has been shown to have a worst case computational complexity at least an order of the number of variables better than that of the standard Karmarkar algorithm. In Section 2, we review the structure of stochastic linear programs with fixed recourse and the computational requirements of interior point programs. In Section 3, we look at methods for improving the running time of the interior point codes, and focus on the Birge and Qi method. Computational results appear in Section 4. A brief summary is contained in Section 5.

2 Preliminaries

2.1 Stochastic Linear Programs

Two-stage stochastic linear programs with fixed recourse that are defined over a discrete probability space have the general form

$$\text{minimize} \quad c^T x_0 + \mathcal{Q}(x_0, \xi)$$
$$\text{subject to} \quad A_0 x_0 = b_0, \tag{1}$$
$$x_0 \geq 0,$$

where $\mathcal{Q}(x_0, \xi)$ is a recourse functional describing the expected costs of undertaking a specific action x_0 before the uncertainty characterized by the random variable ξ is resolved. The expectation of the recourse cost is obtained by

$$\mathcal{Q}(x_0, \xi) = \sum_{l=1}^{N} p_l Q(x_0, \xi_l),$$

1

where p_l is the probability that the lth scenario occurs (i.e. $p_l = P[(\xi) = \xi_l]$ where ξ_l is a realization of the random variable ξ, defined on $(\Xi, \mathcal{A}, P)$), and $Q(x_0, \xi_l)$ is the recourse cost obtained by solving the following recourse problem:

$$Q(x_0, \xi_l) = \inf\{d_l y_l \mid W_l y_l = b_l - T_l x_0, y_l \geq 0, y_l \in \Re^{n_1}\}, \quad \xi_l = (d_l, b_l, T_l, W_l), \tag{2}$$

for each scenario $l = 1, \ldots N$. Here, a decision x_0 is made before ξ_l is known, and a "corrective" optimal action y_l is taken after ξ_l is known. The cost of the second action is $Q(x_0, \xi_l)$, and the expected cost with respect to the random variable ξ is $\mathcal{Q}(x_0, \xi)$. Note here that the solution to $Q(x_0, \xi_l)$ assumes that x_0 has been fixed. This *nonanticipativity* restriction requires that all first stage decisions are invariant with respect to future outcomes.

Substituting the recourse program into (1) and simultaneously minimizing over $(x_0, y_1, \ldots y_N)$, we obtain

$$
\begin{array}{llll}
\text{minimize} & c_0^T x_0 & + \sum_{l=1}^N c_l^T y_l \\
\text{subject to} & A_0 x_0 & & = b^0 \\
& T_l x_0 & + W_l y_l & = b_l & l = 1, \ldots N \\
& x_0, & y_l & \geq 0,
\end{array}
\tag{3}
$$

where $c_0, x_0 \in \Re^{n_0}, T_l \in \Re^{m_l \times n_0}, A_0 \in \Re^{m_0 \times n_0}, W_l \in \Re^{m_l \times n_l}, c_l^T = p_l d_l^T \in \Re^{n_l}$ for $l = 1, \ldots N$. Note that this problem has $n = n_0 + \sum_{l=1}^N n_l$ columns and $m = m_0 + \sum_{l=1}^N m_l$ constraints. For purposes of discussion, we assume that A_0, W_l have full row rank, $m_l \leq n_l, l = 0, \ldots N$, and $n_0 \leq \sum_{l=1}^N n_l$.

This problem, classified as a dual block angular linear program, was first studied by Beale [4] and Dantzig [14]. Several special purpose algorithms for solving linear programs with this special structure have been developed, including the L-shaped method of Van Slyke and Wets [39] and the decomposition method pioneered by Dantzig and Madansky [15]. Interior point algorithms such as those proposed by Karmarkar [26] and Marsten, et. al. [30] applied to these problems have been discussed by Birge and Qi [6], and Carpenter, et. al. [11].

2.2 Interior Point Methods

In the last decade, several breakthroughs in general purpose linear programming algorithms have been made using path following methods [21]. Karmarkar [26] pioneered these breakthroughs with the first algorithm that could be proven to converge to an optimal solution in polynomial, or $O(n^2 m^2 L)$ time, where n is the size of the problem and L is a measure of the problem's data. By contrast, the worst case complexity of the simplex method cannot be bounded by a polynomial.

For the purposes of discussion, we focus on what is generally called the dual affine scaling method [1] as applied to the dual-block angular program described above. Consider a linear program in the following standard equality form:

$$
\begin{array}{lll}
(\mathbf{P}) & \text{minimize} & c^T x \\
& \text{subject to} & Ax = b, \\
& & x \geq 0,
\end{array}
$$

where $A \in \Re^{m \times n}$ and has full row rank, $b \in \Re^m$ is a resource vector, and $c \in \Re^n$ is the objective function vector. The dual affine variant finds an optimal solution to the dual (D) to the problem

(P):

$$(\mathbf{D}) \quad \text{maximize} \quad b^T y$$
$$\text{subject to} \quad A^T y \le c,$$

where $y \in \Re^m$ is a dual vector to the equality constraints of (P). In general, we could also re-write (P) in the polyhedral inequality form of (D) and apply the same algorithm. In this case, the method performs the same steps as the procedure called primal affine scaling ([40] [3] [16]). The only difference is in the form of the matrix factorization ([8]). We can, therefore, use simply *affine scaling* to refer to these methods.

Applying affine scaling to problem (D) requires an initial interior point y^0 that satisfies dual feasibility ($A^T y^0 < c$), and successively iterates on a current point to find another interior point with a better objective function value. The algorithm proceeds as follows:

---

1. $k = 0$. Select a stopping criterion (e.g., Stop if $b^T y^{k+1} - b^T y^k < \epsilon$, where $\epsilon > 0$ is a small positive number.

2. Stop if optimality criterion is satisfied.

3. Calculate dual slack variables: $\nu^k = c - A^T y^k$.

4. Calculate the search direction, which is a function of a Newton step taken toward the "center" of the feasible region and a steepest descent step in some transformed space:

    (a) Let $D^k = \text{diag}\{(1/\nu_1^k), \ldots, (1/\nu_m^k)\}$.

    (b) Let $dy = (A(D^k)^2 A^T)^{-1} b$

    (c) Let $dv = -A^T dy$

5. Calculate a step size:

    (a) Let $\alpha = \gamma \times \min\{\nu_i^k/ - (dv)_i : (dv)_i < 0, i = 1, \ldots, m\}$
    Here, $0 < \gamma < 1$ is a step size parameter to insure that the next iterate will remain interior to the feasible solution set. For most practical purposes, $0.95 \le \gamma < 1$ is sufficient.

6. Update dual variables, primal variables, and counters:

    (a) Let $y^{k+1} = y^k + \alpha dy$

    (b) Let $x^{k+1} = (D^k)^2 dv$

    (c) Let $k = k + 1$

    (d) Goto 2

Dual Affine Algorithm

3

The vast majority of the computational effort required in the above procedure is to calculate a solution to the system $(AD^2A^T)dy = b$ (the iteration counter will be dropped whenever the context is clear), or to calculate some factorization of the matrix to enable quick solution of the system. These computations are common to every interior point algorithm developed thus far (Shanno and Bagchi, [37]). The matrix $M \equiv AD^2A^T$ is a large ($M \in \Re^{m \times m}$) symmetric, positive definite matrix for which several solution methods have been developed (see, e.g. Golub and Van Loan [23]). Generally speaking, a direct inversion of M is an inefficient solution method, and is seldom used in large scale implementations.

There are two main strategies for solving the system $(AD^2A^T)dy = b$. They are *iterative methods* and *direct methods*.

1. *Iterative methods* generate a sequence of approximations to dy. Since these methods only involve vector-matrix multiplication, they are computationally more attractive and require less storage than alternative solution procedures. Convergence to solutions of these linear systems can be unacceptably slow, unless special tricks (e.g., matrix preconditioners) are employed. Examples of iterative methods include the Jacobi, Gauss-Seidel, Chebychev, Lanczos, and conjugate gradient methods. Meijerink and van der Vorst [32] discuss the use of these methods in interior point algorithms.

2. *Direct methods* calculate the exact solution to the set of equations $(AD^2A^T)dy = b$ by factoring the matrix (AD^2A^T), and using backwards/forwards substitution to find dy. The most common schemes in use are (LU) factorization and Cholesky (LL^T) factorization. The effectiveness of these methods depends on the use of special data structures and pivoting rules, and on the characteristics of the coefficient matrix itself. Examples of software implementations include YSMP (Eisenstadt [18]) and SPARSPAK (Chu, et. al. [13]). Direct and iterative methods can also be combined by using ideas from the direct solution procedures to generate an effective preconditioner that improves the convergence of iterative methods. This paper will focus on implementations of interior point algorithms that use direct methods only.

The ease with which direct methods may be used depends heavily on the amount of *fill-in*, or density of the factorized matrix. Matrices which can be rearranged to minimize fill-in can be stored using less memory, and hence require fewer operations to update the factorization or obtain a solution. However, matrices that are ill-structured will generate an extremely dense matrix M, and hence can be quite inefficient to solve.

The density of the matrix (AD^2A^T) largely depends on the number of dense columns that are contained in the original matrix A. Unfortunately, the dual block angular program (3) described in §2.1 (potentially) contains many dense columns. To see this, let $D_0^2 \in \Re^{m_0 \times m_0}$ and $D_l^2 \in \Re^{m_l \times m_l}$ be defined by $D_l^2 = \text{diag} \{(\nu_1^l)^{-2}, \ldots, (\nu_{m_l}^l)^{-2}\}, l = 0, \ldots, N$. Suppose further that $T^l = T, W^l = W, l = 1, \ldots, N$. Then the required system to solve is:

$$AD^2A^T = \begin{bmatrix} A_0 & & & \\ T & W & & \\ \vdots & & \ddots & \\ T & & & W \end{bmatrix} \times \begin{bmatrix} D_0^2 & & & \\ & D_1^2 & & \\ & & \ddots & \\ & & & D_N^2 \end{bmatrix} \times \begin{bmatrix} A_0^T & T^T & \cdots & T^T \\ & W^T & & \\ & & \ddots & \\ & & & W^T \end{bmatrix}$$

4

$$= \begin{bmatrix} A_0 D_0^2 A_0^T & A_0 D_0^2 T^T & A_0 D_0^2 T^T & \cdots & A_0 D_0^2 T^T \\ T D_0^2 A_0^T & T D_0^2 T^T + W D_1^2 W^T & T D_0^2 T^T & \cdots & T D_0 T^T \\ T D_0^2 A_0^T & T D_0^2 T^T & T D_0^2 T^T + W D_2^2 W^T & \cdots & T D_0 T^T \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ T D_0^2 A_0^T & T D_0^2 T^T & T D_0^2 T^T & \cdots & T D_0 T^T + W D_N^2 W^T \end{bmatrix}$$

Clearly, the presence of the columns associated with the T matrices creates an extremely dense M matrix to factorize. For this reason, Arantes and Birge [2] found that dual block angular programs in the primal form are expensive (if possible) to solve, even with basic preprocessing or row reordering to reduce fill-in. The extent to which the fill-in affects computational performance will be explored empirically in Section 4. As we will see in the next section, there are many ways to approach the problem of dense columns in a coefficient matrix to reduce fill-in and improve solution times.

3 Methods for Reducing Computational Requirements

Several modifications to both the formulation of the block angular program and the implementation of the interior point algorithm required for its solution have been proposed. Their intent is either to reduce the number of dense columns that are in the coefficient matrix of the linear program or to separate them explicitly from the other (non-dense) columns. Four alternatives will be explored in this section: reformulation of the program to split up dense columns (Carpenter et. al. [11]), solution of the dual to the program (or a factorization formed from the dual, Arantes and Birge [2] and Birge, Freund and Vanderbei [8]), the use of the Schur Complement to remove dense columns (Lustig, et. al. [29]), and direct solution by a special factorization of the matrix AD^2A^T (Birge and Qi [6]).

3.1 Reformulation of the Program

Carpenter, Lustig, and Mulvey [11] consider a two-stage generalized stochastic network derived from a portfolio management model (developed in Mulvey and Vladimirou [35]). The model can be written in the standard form given in (3), where A_0 constrains the flow of capital between assets in the first stage of the problem, and W contains the stochastic arc multipliers that describe the yields of each asset modeled.

To remove the dense columns associated with the first stage decisions, they *split* the first n_0 columns into scenario dependent variables. Whereas nonanticipativity in the original formulation (3) is enforced by the fact that the first n_0 columns are invariant with respect to each scenario, they enforce nonanticipativity by including explicit constraints that guarantee the invariance. Specifically, if $x_0(l)$ is the first stage decision given that scenario l occurs, nonanticipativity requires that

$$x_0(l+1) = x_0(l) \text{ for all } l = 1, \ldots, N-1.$$

The resulting formulation (called the full-splitting formulation) removes nonzeros from the first stage columns for scenario l from all constraints except those associated with first stage columns

for scenarios $l-1$ and $l+1$, and is

$$
\begin{array}{llllllll}
\min & c_0^T x_0 & + & c_1^T y_1 & + & \cdots & + & c_N^T y_N & \\
\text{st} & A_0 x_0 & & & & & & = & b^0 \\
& I x_0 & -I x_1 & & & & & = & 0 \\
& & T x_1 & +W y_1 & & & & = & b_1 \\
& & I x_1 & & -I x_2 & & & = & 0 \\
& & & & \ddots & & & = & \vdots \\
& & & & & T x_N & +W y_N & = & b_N
\end{array} \tag{4}
$$

By concentrating the non-zero elements of the constraint matrix of (4) around the diagonal, the density of the matrix AD^2A^T is reduced. However, the full-splitting formulation also increases the overall size of the problem. In a set of ten portfolio test problems run in [11], the average increase in the number of rows was 47.2% and the average increase in the number of columns was 12.8%.

Further improvements to the full splitting formulation can be made by only splitting those first-stage variables which have non-zero elements in the rows of T_l, and leaving the remaining first-stage decision variables in the A matrix only. This *partial splitting* representation can effectively limit the increase in problem size that occurs in the full splitting formulation. For the stochastic network test problems in the paper, the average row and column growths for the partial splitting model were 12.8% and 5.2%, respectively.

Although reformulating the dual block angular problem increases its size, the resulting improvement in AD^2A^T fill-in can be substantial. On the same test problems mentioned above, the original formulation (3) had (on average) 2.47 times more nonzeros than the partial splitting representation. As a result, the Cholesky factorizations of the AD^2A^T matrices are also less dense (on average 2.8 times less), and can be solved more efficiently. As might be expected, the reduction in matrix density also has a substantial impact on solution times. When solved using a commercial implementation of a primal-dual interior point algorithm (OB1 [29]), the full splitting formulation was (on average) 4.65 times faster than the original formulation. The partial splitting formulation was (on average) 10.8 times faster than the original formulation. The average speedup of the partial splitting model over the original formulation run on a well-known simplex based LP solver (MINOS 5.3, developed by Murtagh and Saunders, [36]) was 5.58.

Unfortunately, the effectiveness of reformulating the linear program is largely dependent on the relative sizes and forms of the first stage coefficient matrices. In general, two-stage stochastic linear programs may have large first stage coefficient matrices or many first stage decision variables which are in linking constraints to recourse decisions. Since Carpenter, et. al. only considered one type of stochastic decision problem, other methods which are not as problem specific might be more robust.

3.2 Using the Transpose Product Factorization

The density of the matrix AD^2A^T was shown in Section 2.2 to be quite high when a problem which exhibits a dual block angular structure is solved using dual affine scaling. Arantes and Birge [2] suggested that reformulating the primal problem in the polyhedral inequality form would allow more efficient computation since the relevant matrix for computation is $A^T D^2 A$ which is much sparser than AD^2A^T. Reformulation is in fact not necessary (see [8] where it is shown that

computations with the AD^2A^T matrix structure can always be replaced by computations with the A^TD^2A structure).

It is, however, illustrative to think of this approach as solving the dual to the original problem using again the dual affine scaling method. The dual of (3) is

$$
\begin{aligned}
\max \quad & b_0^T y_0 + \sum_{l=1}^N b_l^T y_l \\
\text{st} \quad & A_0^T y_0 + \sum_{l=1}^N T_l^T y_l \leq c_0 \\
& W_l^T y_l \leq c_l
\end{aligned}
\tag{5}
$$

Since the resulting coefficient matrix B of (6) has $n = n_0 + \sum_{l=1}^N n_l$ rows and $m = m_0 + \sum_{l=1}^N m_l$ columns, BD^2B^T is of order $n \times n$, and so may be considerably larger than the coefficient matrix of (3). However, the matrix BD^2B^T exhibits the A^TD^2A matrix structure and enables an efficient Cholesky factorization. Specifically, the matrix BD^2B^T is

$$
BD^2B^T = \begin{bmatrix}
A_0^T D_0^2 A_0 + \sum_{l=1}^N T^T D_l T & T^T D_1^2 W & \cdots & T^T D_N^2 W \\
W^T D_1^2 T & W D_1 W^T & 0 & 0 \\
\vdots & 0 & \ddots & 0 \\
W^T D_N^2 T & 0 & 0 & W D_N^2 W^T
\end{bmatrix}
\tag{6}
$$

As an example of the different fill-in characteristics of the primal and dual problems, pictures of the AD^2A^T and BD^2B^T matrices are shown in Figure 1 for a test problem. [1]

Since the density of the Cholesky factorization of a matrix depends on the symmetry of that matrix, the factorization of the matrix (6) should be sparse. Solution times reported by Arantes and Birge for a common test set (Ho and Loute, [24], Birge, [9]) were much faster using the BD^2B^T form over the AD^2A^T form. The largest problem they tried (32 scenarios, 7023 nonzeroes in M) was a full order of magnitude faster with BD^2B^T. However, the off-diagonal blocks of the matrix BD^2B^T contain matrix products with the recourse matrix W, whereas the matrix AD^2A^T does not. Hence, if W is unusually large or dense, solving with BD^2B^T may not be as efficient. More extensive comparisons may be found in Section 4. In general, solving stochastic linear programs with the form based on the A^TD^2A structure (as in BD^2B^T) appears preferable to solving using the AD^2A^T structure.

3.3 Schur Complement

Many implementations of interior point algorithms avoid dense columns in coefficient matrices by explicitly removing them and accounting for them separately. The mechanism for solving the system $(AD^2A^T)dy = b$ is the *Schur complement*, which involves solving a small, dense matrix derived from a larger sparse matrix.

Consider a coefficient matrix A which can be partitioned into $[A_d \vdots A_s]$, where $A_s \in \Re^{m \times n_s}$ is a submatrix containing only sparse columns, and $A_d \in \Re^{m \times n_d}$ contains only dense columns. In the case of the two-stage dual block angular program, A_d contains the columns corresponding to the first stage decisions and A_s contains the columns corresponding to the second stage decisions.

Let D_s and D_d be diagonal matrices corresponding to A_s and A_d. Then $AD^2A^T = A_s D_s^2 A_s^T + A_d D_d^2 A_d^T$. Let the Cholesky factorization of $A_s D_s^2 A_s^T$ be LL^T, $V = A_d D_d$, and $\delta = -V^T dy$. Then

---

[1]SC205 with 16 scenarios. For problem characteristics, please refer to Section 6.

$VV^T = A_d D_d^2 A_d^T$ and solving the system

$$\begin{bmatrix} LL^T & -V \\ V^T & I \end{bmatrix} \begin{bmatrix} dy \\ \delta \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix} \tag{7}$$

is equivalent to solving $(AD^2A^T)dy = b$. From the first set of equations in (7), we get

$$LL^T dy = b + V\delta \quad \text{or}$$
$$dy = (LL^T)^{-1}(b + V\delta). \tag{8}$$

From the second set of equations in (7), we find that

$$V^T dy + I\delta = 0 \tag{9}$$

Substituting (8) into (9),

$$[I + V^T(LL^T)^{-1}V]\delta = -V^T(LL^T)^{-1}b \tag{10}$$

The matrix $I + V^T(LL^T)^{-1}V$ is a Schur complement and a dense matrix of order $n_d \times n_d$. Methods specific to the solution of dense matrices (e.g., a dense Cholesky factorization) can be used to solve (10) while sparse solution methods can be used to calculate (8) and the right hand side of (10). Once δ has been calculated, then the search direction is the solution to

$$LL^T dy = b + V\delta.$$

The entire method requires $n_d + 1$ sparse Cholesky backsolves (n_d solutions to $(LL^T)z = (V)._i$ and one solution to $(LL^T)z = b$) as well as a dense backsolve for δ.

As shown in Lustig et. al. ([29]), use of the Schur complement can significantly reduce the overall effort needed to solve the search direction system, since removal of dense columns results in quite sparse Cholesky factorizations. However, as reported in Choi, et. al. ([12]), there are two disadvantages to using the Schur complement. As the number of dense columns grows large, the effort needed to solve the dense matrix $I + V^T(LL^T)^{-1}V$ grows markedly. As an example, Choi, et. al [12] solved the ISRAEL problem in the NETLIB test set (Gay, [20]) with different sizes of the dense submatrix. With the number of dense columns set at 40, the time required to solve the problem was over twice that when 6 columns were included in the dense partition.

The second disadvantage is possible numerical instability. For dual block angular programs, moving a first stage column into A_d leaves a column with no nonzeroes in $A_s D_s^2 A_s^T$. For example, consider a one scenario problem, with coefficient matrix, $A = \begin{bmatrix} A_0 & 0 \\ T & W \end{bmatrix}$. In this case, $A_d = \begin{bmatrix} A_0 \vdots T \end{bmatrix}^T$ and $AD^2A^T = A_d D_d^2 A_d^T + A_s D_s^2 A_s^T =$

$$\begin{bmatrix} A_0 \\ T \end{bmatrix} \begin{bmatrix} D_1^2 \end{bmatrix} \begin{bmatrix} A_0^T & T^T \end{bmatrix} + \begin{bmatrix} 0 \\ W \end{bmatrix} \begin{bmatrix} D_2^2 \end{bmatrix} \begin{bmatrix} 0 & W^T \end{bmatrix} \tag{11}$$

$$= \begin{bmatrix} A_0 D_1^2 A_0^T & A_0 D_1^2 T^T \\ T D_1^2 A_0^T & T D_1^2 T^T \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & W D_2^2 W^T \end{bmatrix}. \tag{12}$$

8

Since $A_s D_s^2 A_s^T$ is singular, the above procedure fails. Even if A_s is forced to have full rank, the procedure is likely to suffer from numerical instability. To address this problem, various methods have been proposed to improve the accuracy of the algorithm. For example, Lustig, et. al. [29] uses iterative refinement to improve the accuracy of solutions to equations involving LL^T. For some problems in the NETLIB test set [20], however, they are only able to guarantee the solution to one decimal place. This reflects the inherent difficulty with problems with many dense columns.

While loss of numerical accuracy may be a problem for dual block angular programs with many dense columns, the use of the Schur complement may improve solution times substantially for problems with a few dense columns. For a class of stochastic network models, Carpenter et. al. [11] halved solution times using the Schur complement over a splitting reformulation (Section 3.1). However, the inherent loss of accuracy associated with the Schur complement suggests that it may not be desirable for solving dual block angular programs.

In many general linear programs, an identity (or a substantial part of an identity) matrix exists in the original constraint matrix, A, due to slack or surplus variables. In this case, AA^T may be written as $A_d A_d^T + I_s$. Now, the Schur complement approach can be used to write $(AA^T)^{-1} = (I_s + A_d A_d^T)^{-1} = I_s - A_d(I_d + A_d^T A_d)^{-1} A_d^T$, where we use I_d to denote an identity of rank n. Other diagonal coefficients corresponding to slack variable values can be used in place of I_s and I_d with the same basic result. In this way, solution using the AA^T structure is replaced by solution with the $A^T A$ structure, which may be sparser. This is the approach in [8] that allows the dual factorization form to be used in solving the primal problem.

Even when an identity does not exist, one can be added to the $A_s D_s^2 A_s^T$ part of $AD^2 A^T$ and subtracted from $A_d D_d^2 A_d^T$. This allows $I_s + A_s D_s^2 A_s^T$ to remain well conditioned. The next section describes this method to maintain accuracy in a form similar to the Schur complement.

3.4 Explicit Factorization of Dual Block Angular Programs

The solution to the set of equations that determine search directions in affine scaling algorithms may also be accomplished by decompositions specific to the dual block angular structure. Birge and Qi [6] proposed the better conditioned form of the Schur complement using a generalized version of the Sherman-Morrison-Woodbury formula (see, e.g. Golub and Van Loan [23]). While the full calculation of the step size in affine scaling may be performed generally in $O(m^2 n)$ operations, the decomposition they propose reduces the computational complexity to $O(n^2)$ operations (assuming $n_l \sim n$ for all $l = 0, \ldots, N$). In this subsection, we review the theoretical result obtained and discuss the procedure for its implementation. Serial and parallel computational results for this particular method are reported in Section 4.

The main result obtained by Birge and Qi notes that the matrix $AD^2 A^T$ may be written as the sum of a block diagonal matrix D (the $I_s + A_s D_s^2 A_s^T$ segment) and the product of two similar matrices U and V (defined below). Given this representation, the Sherman-Morrison-Woodbury formula, which is reproduced here for convenience, may be used to find the inverse of the matrix. The notation used in the following follows that of Section 2.2.

Lemma 1 : *For any matrices A, U, V such that A and $(I + V^T A^{-1} U)$ are invertible,*

$$(A + UV^T)^{-1} = A^{-1} - A^{-1} U(I + V^T A^{-1} U)^{-1} V^T A^{-1}. \tag{13}$$

Proof:

$$(A + UV^T)^{-1}(A + UV^T) = I + A^{-1} U V^T - A^{-1} U(I + V^T A^{-1} U)^{-1} V^T -$$

9

$$A^{-1}U(I + V^T A^{-1}U)^{-1}V^T A^{-1}UV^T$$
$$= I + A^{-1}UV^T - A^{-1}U(I + V^T A^{-1}U)^{-1}(I + V^T A^{-1}U)V^T$$
$$= I + A^{-1}UV^T - A^{-1}UV^T = I.$$

The lemma is the main tool in deriving the BQ factorization. We slightly change the form of the M matrix in [6] to obtain the following result. The proof follows the same development as [6].

Theorem 1 : *Consider the feasible region of the dual block angular program given in (3), and some dual solution $(y^0, \ldots, y^N)$. Let $M = AD^2 A^T, S = \text{diag}\{S_0, S_1, \ldots, S_N\}$, where $S_l = W_l D_l^2 W_l^T, l = 1, \ldots, N$, $S_0 = I_2 \in \Re^{m_0 \times m_0}$, and $D_l^2 = \text{diag}\{(\nu_1^l)^{-2}, \ldots, (\nu_{m_l}^l)^{-2}\}$. Furthermore, let I_1 and I_2 be identity matrices of dimension n_0 and m_0, respectively. Also, let*

$$G_1 = (D_0)^2 + \sum_{l=0}^{N} T_l^T S_l^{-1} T_l, \qquad G_2 = -A_0 G_1^{-1} A_0^T$$

$$U = \begin{pmatrix} A_0 & I_2 \\ T_1 & 0 \\ \vdots & \vdots \\ T_N & 0 \end{pmatrix}, \qquad V = \begin{pmatrix} A_0 & -I_2 \\ T_1 & 0 \\ \vdots & \vdots \\ T_N & 0 \end{pmatrix}$$

If A_0 and W_l has full row rank, for $l = 1, \ldots, N$, G_2 and M are invertible and

$$M^{-1} = S^{-1} - S^{-1}U \begin{bmatrix} I_1 & G_1^{-1}A_0^T \\ 0 & -I_2 \end{bmatrix} \begin{bmatrix} I_1 & 0 \\ 0 & G_2^{-1} \end{bmatrix} \begin{bmatrix} I_1 & 0 \\ A_0 & I_2 \end{bmatrix} \begin{bmatrix} G_1^{-1} & 0 \\ 0 & I_2 \end{bmatrix}. \tag{14}$$

Proof: In the affine scaling algorithm, ν_k is positive, so D is invertible. By assumption, W_l has full row rank, so S_l is invertible for $l = 1, \ldots, N$ and S is invertible. Let $\bar{D} = \text{diag}\{D_0, I_2\}$ and

$$\tilde{U} = \begin{pmatrix} A_0 D_0 & I_2 \\ T_1 D_0 & 0 \\ \vdots & \vdots \\ T_N D_0 & 0 \end{pmatrix}, \qquad \tilde{V} = \begin{pmatrix} A_0 D_0 & -I_2 \\ T_1 D_0 & 0 \\ \vdots & \vdots \\ T_N D_0 & 0 \end{pmatrix}.$$

Note that $\tilde{U} = U\bar{D}$ and $\tilde{V} = V\bar{D}$. By construction, $M = S + \tilde{U}\tilde{V}^T$. Applying Lemma 1 (the Sherman-Morrison-Woodbury formula), M is invertible and

$$M^{-1} = (S + \tilde{U}\tilde{V}^T)^{-1}$$
$$= S^{-1} - S^{-1}\tilde{U}(I + \tilde{V}^T S^{-1}\tilde{U})^{-1}\tilde{V}^T S^{-1} \ (\text{ iff } S \text{ and } I + \tilde{V}^T S^{-1}\tilde{U} \text{ are invertible })$$
$$= S^{-1} - S^{-1}U\bar{D}^2(I + \bar{D}^T V^T S^{-1}U\bar{D})^{-1}V^T S^{-1}$$
$$= S^{-1} - S^{-1}U\bar{D}^2\bar{D}^{-2}(\bar{D}^{-2} + V^T S^{-1}U)^{-1}V^T S^{-1}$$
$$= S^{-1} - S^{-1}U(\bar{D}^{-2} + V^T S^{-1}U)^{-1}V^T S^{-1}.$$

Let

$$G = \begin{bmatrix} G_1 & A_0^T \\ -A_0 & 0 \end{bmatrix} \text{ where } G_1 = (D_0)^2 + \sum_{l=0}^{N} T_l^T S_l^{-1} T_l.$$

$$G_2 = -A_0 G_1^{-1} A_0^T.$$

10

Then

$$V^T S^{-1} U = \begin{bmatrix} A_0 A_0^T + \sum_{l=1}^{N} T_l S_l^{-1} T_l & A_0^T \\ -A_0 & -I_2 \end{bmatrix}$$

and $\bar{D}^2 + V^T S^{-1} U = G$. So,

$$M^{-1} = S^{-1} - S^{-1} U G^{-1} V^T S^{-1} \tag{15}$$

if and only if $(I + \check{V}^T S^{-1} \check{U})$, or G, is invertible.

G_1 can be expanded as

$$G_1 = (D_0)^2 + A_0 A_0^T + \sum_{l=1}^{N} T_l S_l^{-1} T_l^T.$$

By construction, $(D_0)^2$ and $A_0 A_0^T$ are positive definite and symmetric. Since S_l is positive definite for all $l = 1, \ldots, N$, $T_l S_l^T T_l^T$ is positive definite and symmetric. The sum of positive definite matrices is again positive definite, so G_1 is positive definite and symmetric. So, G_1^{-1} exists, is symmetric, and can be written as $G_1 = G_1^{1/2} G_1^{1/2}$, where $G_1^{1/2}$ is also symmetric. By assumption, A_0 has full row rank, so $A_0 G_1^{1/2}$ has full row rank. Consequently, $G_2 = -A_0 G_1^{-1} A_0^T$ is invertible, and

$$G \begin{bmatrix} I_1 & G_1^{-1} A_0^T \\ 0 & -I_2 \end{bmatrix} \begin{bmatrix} I_1 & 0 \\ 0 & G_2^{-1} \end{bmatrix} \begin{bmatrix} I_1 & 0 \\ A_0 & I_2 \end{bmatrix} \begin{bmatrix} G_1^{-1} & 0 \\ 0 & I_2 \end{bmatrix}$$

$$= \begin{bmatrix} G_1 & A_0^T \\ -A_0 & 0 \end{bmatrix} \begin{bmatrix} G_1^{-1} + G_1^{-1} A_0^T G_2^{-1} A_0 G_1^{-1} & G_1^{-1} A_0^T G_2^{-1} \\ -G_2^{-1} A_0 G_1^{-1} & -G_2^{-1} \end{bmatrix} = \begin{bmatrix} I_1 & 0 \\ 0 & I_2 \end{bmatrix}.$$

Since G is invertible, (15) holds. Consequently, M is invertible and (14) holds.

Using Theorem 1 to explicitly compute the inverse of the matrix M is not the most efficient way to determine the search direction dy. However, the system of equations $M dy = b$ may be efficiently solved using Theorem 1 by solving (in order)

$$Sp = b \qquad Gq = V^T p \qquad Sr = Uq \tag{16}$$

and setting $dy = p - r$. To verify that $dy = M^{-1} b$, note that

$$\begin{aligned} dy &= p - r = S^{-1} b - S^{-1} U q \\ &= [S^{-1} - S^{-1} U (G^{-1} V^T S^{-1})] b \\ &= M^{-1} b \end{aligned}$$

Further simplification of the second equation of (16) may be made by symbolically expanding G into its components G_1 and A_0. Let $q^T = (q_1^T, q_2^T)$, where $q_1 \in \Re^{n_0}$ and $q_2 \in \Re^{m_0}$. Then solving

$$\begin{bmatrix} G_1 & A_0^T \\ -A_0 & 0 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} \hat{p}_1 \\ \hat{p}_2 \end{bmatrix} = \begin{bmatrix} A_0^T & T_1^T & \cdots & T_N^T \\ -I_2 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} p_0 \\ \vdots \\ p_N \end{bmatrix} \tag{17}$$

implies that

$$\begin{aligned} q_2 &= (A_0 G_1^{-1} A_0^T)^{-1} (\hat{p}_2 - A_0 G_1^{-1} \hat{p}_1) \\ &= -G_2^{-1} (\hat{p}_2 - A_0 G_1^{-1} \hat{p}_1) \\ q_1 &= (G_1)^{-1} (\hat{p}_1 - A_0^T q_2). \end{aligned}$$

11

Solving (16) requires Cholesky factorizations of S_l, G_1, and G_2. At each iteration of the affine scaling algorithm, an update of S_l to reflect the current dual solution must precede the solution of $Mdy = b$. Given the updated matrices $(D_l)^2$ and subsequent updates of the Cholesky factorizations of S_l, dy may be found using the algorithm described below.

Procedure finddy (S, $A_0 \ldots A_N$, b, dy)
begin

1. (Solve $Sp = b$). Solve $S_l p_l = b_l$ for $p_l, l = 0, \ldots, N$.

2. (Solve $Gq = V^T p$).

   (a) Form G_1 by solving $S_l(u_l)_i = (A_l)_{\cdot i}$ for $(u_l)_i, l = 0, \ldots, N, i = 1, \ldots, n_l$) and setting $(G_1)_{\cdot i} = (D_0)_{ii} + T_l^T(u_l)_i$.

   (b) Form $\hat{p}_1$ and $\hat{p}_2$ using (17).

   (c) Solve $G_1 u = \hat{p}_1$ for u and set $v = \hat{p}_2 - A_0 u$.

   (d) Form G_2 by solving $(G_1)w_i = (A_0^T)_{\cdot i}$ for w_i and setting $G_2 = -A_0[w_1 \cdots w_m]$.

   (e) Solve $G_2 q_2 = v$ for q_2, and solve $G_1 q_1 = \hat{p}_1 - A_0^T q_2$ for q_1.

3. (Solve $Sr = Uq$). Set $r_0 = A_0 q_1 + q_2$ and solve $S_l r_l = A_l q_1$ for $r_l \in \Re^{m_l}, l = 1, \ldots, N$.

4. (Form dy). Set $(dy)_l = p_l - r_l$ for $l = 0, \ldots, N$. Return dy.

end.

Forming dy using Birge and Qi's Decomposition

There are several advantages to using this approach to find the search direction dy. Unlike the Schur complement, the method does not suffer from ill-conditioned working matrices and requires no iterative improvement to maintain solution accuracy. To illustrate this point, a one scenario problem with constraint matrix, $A = \begin{bmatrix} A_0 & 0 \\ T & W \end{bmatrix}$. In this case, $AD^2A^T = S + \tilde{U}\tilde{V}^T =$

$$\begin{bmatrix} I & 0 \\ 0 & WD_1^2W^T \end{bmatrix} + \begin{bmatrix} A_0 D_0^2 A_0^T - I & A_0 D_0^2 T^T \\ TD_0^2 A_0^T & TD_0^2 T^T \end{bmatrix}. \tag{18}$$

Comparing (18) to (12) shows the effect of adding the identity matrix to eliminate the singularity in the sparse system. We note also that this approach can in general be used to obtain better conditioned factorizations for problems with dense columns. Although we have only considered dual block angular problems so far, this is not necessary. In the case of $A = \begin{bmatrix} A_0 & Q \\ T & W \end{bmatrix}$, we can write $AD^2A^T = S + \tilde{U}\tilde{V}^T =$

$$\begin{bmatrix} I + QD_1^2Q^T & 0 \\ 0 & WD_1^2W^T \end{bmatrix} + \begin{bmatrix} A_0 D_0^2 A_0^T - I & A_0 D_0^2 T^T + QD_1^2 W^T \\ TD_0^2 A_0^T + WD_1^2 Q^T & TD_0^2 T^T \end{bmatrix}. \tag{19}$$

An implementation based on (19) can use $S = \begin{bmatrix} I + QD_1^2Q^T & 0 \\ 0 & WD_1^2W^T \end{bmatrix}$ just as in Theorem 1, where $Q = 0$. In this way, instability in $QD_1^2Q^T$ can be countered.

Note that this approach does not increase problem size as in splitting or dual factorizations. Birge and Qi [6] also show that the decomposition given in Theorem 1, when applied to the dual problem (6), allows efficient updating from a smaller stochastic linear program with few scenarios to a larger problem with more scenarios.

A substantial portion of the computational effort required for the procedure lies in the factorization of S_l and the solution of systems of equations involving S_l. Since these calculations may be performed independently, coarse-grained parallelization or distributed processing may be used to provide a substantial gain in computational efficiency. With one processor for each scenario, the simulataneous running time to solve for the search direction may be reduced from $O(N(n_1^3 + n_1^2 n_0 + n_0 n_1^2 + n_0^2 n_1))$ as given by Birge and Qi to $O(n_1^3 + n_1^2 n_0 + n_0 n_1^2 + N n_0^2)$, a reduction of roughly $\min\{N, n_1\}$. Finer grain systems with many processors could achieve further reductions by peforming matrix additions and multiplications in parallel, but we concentrate on distributed systems due to their wide availability. In the next section, we provide computational experience on both serial and distributed implementations.

4 Computational Experience

In [6], only the theoretical efficiency of the BQ decomposition was demonstrated. We wish to demonstrate that practical computational advantages also exist by comparing this factorization with the other methods mentioned for stochastic linear programs. We chose to compare the method against the interior point solver in IBM's Optimization Subroutine Library (OSL) [25] since that package is widely available and, as such, represents a benchmark for general purpose interior point implementations. Our goal was to determine whether the specialized BQ factorization improves upon general purpose performance mainly in terms of the effort required in each iteration of the respective methods. We also wished to see whether a distributed computing implementation of the BQ decomposition could provide further efficiencies.

The OSL interior point approach is a primal barrier method described in Gill, et. al. ([21]). In the current study, we compare the BQ factorization to the OSL implementation for three formulations: (1) the primal problem (3), (2) the dual of (3) (so that the transpose factorization was used), and (3) the split variable formulation (4) of the primal problem.

4.1 Problem Characteristics

The test problems used in the current study are stochastic versions of a set of staircase test problems (Ho and Loute, [24]) as in Birge ([9]). The problems in the test include:

SC205   A dynamic multisector development planning model.
SCRS8   A technological assessment model for the study of the transition from
                 fossil fuel use to renewable energy sources.
SCAGR7   A multiperiod dairy farm expansion model.
SCSD8   A model to find the minimal design of a multistage truss.
SCFXM1   A production scheduling problem.

Deterministic equivalent problems (3) and split variable formulations (4) were created using a test problem generator developed by Birge, et. al. [7]. Dual problems were generated from the

13

original deterministic formulations prior to any input into an LP solver. Linear programs were converted by all LP solution implementations into the standard equality form given in Section 2.2 prior to solution.

For each problem in the test set, deterministic equivalents with 4, 8, 16, 32, and 64 scenarios (distinct realizations of ζ) were created. The exceptions are scfxm1 and scsd8, which were only solved to 32 scenarios. The characteristics of each problem are given in Table 1. The number of nonzeros in the AD^2A^T matrices for the various formulations is also given in Table 1. Figure 2 graphically shows the number of Cholesky nonzeroes for the sc205 and scsd8 problems.

Problem	Scen- arios	Rows	Columns (1)	A Matrix Nonzeros	Density (Percent)	AD^2A^T Nonzeros Dual	Primal	Split
sc205.1	4	101	102	270	2.62%	367	478	623
sc205.2	8	189	190	510	1.42%	695	1200	1441
sc205.3	16	365	366	990	0.74%	1343	3454	3323
sc205.4	32	717	718	1950	0.38%	2639	11236	8316
sc205.5	64	1421	1422	3870	0.19%	5231	39882	18928
scagr7.1	4	167	180	546	1.82%	1164	1191	1129
scagr7.2	8	319	340	1050	0.97%	2248	3443	2546
scagr7.3	16	623	660	2058	0.50%	4416	11123	5417
scagr7.4	32	1231	1300	4074	0.25%	8752	39155	11593
scagr7.5	64	2447	2580	8106	0.13%	17424	145907	25743
scfxm1.1	4	684	1014	3999	0.58%	28564	7669	9585
scfxm1.2	8	1276	1914	7319	0.30%	56111	14997	18449
scfxm1.3	16	2460	3714	13959	0.15%	125689	31819	38952
scfxm1.4	32	4828	7314	27239	0.08%	249402	75641	90220
scrs8.1	4	140	189	457	1.73%	1877	1235	1367
scrs8.2	8	252	341	849	0.99%	5195	4771	3550
scrs8.3	16	476	645	1633	0.53%	4955	23137	10251
scrs8.4	32	924	1253	3201	0.28%	9563	88732	29337
scrs8.5	64	1820	2469	6337	0.14%	18779	347247	89687
scsd8.1	4	90	630	1890	3.33%	31677	1433	2673
scsd8.2	8	170	1190	3650	1.80%	59595	3916	7066
scsd8.3	16	330	2310	7170	0.94%	117113	12112	19892
scsd8.4	32	650	4550	14210	0.48%	232287	41588	65143
	(1) Slack Columns not included							

Table 1. Problem Characteristics

As can be seen from Table 1, the scfxm1 and scsd8 problems contained many more nonzeroes in the dual Cholesky factorization than in the primal Cholesky factorization. This behavior runs counter to the supposition given in Section 3.2 that solving the dual problem reduces the fill-in in the projection matrix factorization.

Resolving this discrepancy requires the consideration of the sizes of the submatrices A_0, T, and W of the deterministic equivalent problem. The major block components off the diagonal of the matrix AD^2A^T are TA_0^T and TT^T. In contrast, the block components of the system A^TD^2A (shown in 6) off the diagonal are matrices with the same nonzero structure as W^TT. If this matrix is comparatively large or dense, then the dual projection matrix may be more dense than the primal projection matrix. Table 2 shows the block component densities and sizes for each of the test problems considered. The two exceptional problems, SCFXM1 and SCSD8, have many columns and have relatively dense TW^T blocks. These results imply that consideration of the block

14

characteristics of the deterministic equivalent problem is important before deciding which form of the problem to solve. (We should note that Carpenter and Vanderbei ([41]) recently proposed another alternative for use in a primal-dual algorithm which allows some compromise between the AD^2A^T or A^TD^2A factorization forms.)

	T Matrix		W matrix		Primal		Dual	Primal and Dual	
Problem	Rows	Columns	Rows	Columns	TA_0^T	TT^T	TW^T	WW^T	$A_0A_0^T$
sc205	13	14	22	22	26	22	25	102	54
scfxm1	92	114	149	225	2729	12996	168	5937	3450
scagr7	19	20	39	40	73	80	67	370	124
scrs8	28	37	28	38	98	185	100	248	221
scsd8	10	70	21	140	1732	9900	1732	4880	1420

Table 2. Block Characteristics

Another statistic which may help distinguish which form of the problem to solve is the relative row and column densities of the deterministic equivalent linear program. Table 3 shows these densities for the two largest instances of each problem. The row (column) density is defined as the number of nonzeroes in a row (column) divided by the number of rows (columns) in the entire constraint matrix. Since the problem scfxm1 has an unusually low maximum column density, solving the dual problem may not necessarily remove many nonzeros from its Cholesky factorization. Likewise, the problem scsd8 has an unusually high maximum row density, which is consistent with the small number of rows in its primal problem.

	Row Densities (1)		Column Densities (2)	
Problem	Minimum	Maximum	Minimum	Maximum
sc205.4	0.000	0.696	0.279	4.881
sc205.5	0.000	0.352	0.141	4.715
scagr7.4	0.077	0.769	0.081	18.278
scagr7.5	0.039	0.388	0.041	18.349
scfxm1.3	0.027	3.096	0.041	2.114
scfxm1.4	0.014	1.572	0.021	2.071
scrs8.4	0.080	1.117	0.108	14.286
scrs8.5	0.041	0.567	0.055	14.286
scsd8.3	0.433	3.723	0.303	15.152
scsd8.4	0.220	1.890	0.154	15.077
Notes:	(1) Defined as the number of nonzeroes in a row divided by the number of rows (2) Defined as the number of nonzeroes in a column divided by the number of columns			

Table 3. Row and Column Densities

4.2 Computational test parameters and algorithms

The deterministic equivalent test problems, as described in the previous section , were solved using OSL [2] and an implementation of the BQ decomposition on an IBM RS/6000 320H workstation [3] for

---

[2]Subsequent improvements to OSL show lower solution times than those quoted here.

[3]System performance measures are given as: 41.2 SPECs, 37 IMIPs, and 11.7 MFLOPs

the serial comparisons and a network of six DEC 5000/320 workstations for the distributed implementation of BQ. Data structures and recommended parameter values used for the BQ implementation followed those in ([1]). The implementations were written in the FORTRAN programming language, and complied using the XLF RS/6000 compiler with all applicable optimizations.

The solution of the sparse systems of equations required to find the search direction dy in the standard dual affine method were obtained using Cholesky factorizations of each S_l in the BQ decomposition. These factorizations were computed using SPARSPAK, developed by Chu, et. al. [13]. The SPARSPAK routines use a minimum degree ordering heuristic to permute the columns and rows to obtain relatively sparse factorizations. Since the nonzero structure of the AD^2A^T matrix remains the same on each iteration, only the entries of the matrix are updated at each iteration. As suggested in Karmarkar [26] and Adler, et. al. [1], an approximate scaling matrix D is used to determine a search direction at each iteration. Instead of using $D_k = diag\{1/\nu_1^k, \ldots, 1/\nu_n^k\}$ in AD^2A^T, each implementation uses $\tilde{D}_k$, where $(\tilde{D}_k)_{ii}$ is $(\tilde{D}_{k-1})_{ii}$ except in those elements that differ (relatively) by at least a fixed amount ϵ_u from the corresponding element in the exact scaling matrix. So, for $i = 1, \ldots, n$ and $\epsilon_u = 0.1 > 0$,

$$
(\tilde{D}_k)_{ii} = \begin{cases} (\tilde{D}_{k-1})_{ii} & \text{if } \frac{|(D_k)_{ii}-(\tilde{D}_{k-1})_{ii}|}{(\tilde{D}_{k-1})_{ii}} < \epsilon_u \\ (D_k)_{ii} & \text{if } \frac{|(D_k)_{ii}-(\tilde{D}_{k-1})_{ii}|}{(\tilde{D}_{k-1})_{ii}} \geq \epsilon_u \end{cases}
$$

Feasibility in the BQ implementation was obtained using a Big M scheme (proposed by Adler et. al. [1]) and a Phase I/Phase II stopping criterion. Specifically, a feasible solution may be obtained by solving

$$
\begin{aligned}
\max \quad & b^T - My_a \\
\text{subject to} \quad & A^Ty - e^Ty_a \leq c.
\end{aligned}
$$

If the algorithm stops and $y_a^k < 0$, then the algorithm proceeds without the artificial variable and y^k as an interior point to the original problem. If $y_a^k < \epsilon_f$ [4] then the problem is unbounded or an optimal solution was found. Otherwise, the problem is infeasible. The initial interior point used for each phase one procedure was $y^0 = (\|c\|/\|A^Tb\|)b$.

Obtaining the search direction dy using the BQ decomposition requires many solutions to linear systems of the form $S_l x = W_l D_l^2 W_l^T x = y$ for all $l = 0, \ldots, N$. Since these systems are independent of each other, and their symbolic Cholesky factorizations need to be performed only once, they are ideal "processes" to compute in a distributed computing environment. As part of this study, a parallel implementation of the BQ decomposition which assigns groups of scenario blocks S_l to different processors was developed. The implementation consists of several node programs and a host program. Each node program uses SPARSPAK to compute and solve Cholesky factorizations for each block in a set of blocks for which it is responsible. [5] At each iteration, the host program sends a set of right hand side vectors $y_{i_1}, \ldots, y_{i_N}$ to each node program and collects the solutions $S_{i_1}^{-1}y_{i_1}, \ldots, S_{i_N}^{-1}y_{i_N}$. The host program also performs all other tasks, such as setting up the problem, finding the search direction dy from the solutions given by the node programs, and printing the final results.

Communications between the host program and each node program were managed by the Parallel Virtual Machine (PVM) package, developed by Beguelin, et. al. ([5]). PVM allows rapid

---

[4] $\epsilon_f = 10^{-6}$, in the implementations used here.

[5] Blocks were spread as evenly as possible across processors.

prototyping of distributed applications across a number of UNIX based environments. Testing was performed on a network of six DEC 5000/320 workstations connected by a shared ethernet.

The primal barrier point method implemented in the OSL package is described in Gill, et. al. ([21]). Given a feasible interior point, the algorithm projects the steepest descent direction of a barrier problem onto the nullspace of a set of equality constraints. This algorithm is fundamentally different from the dual affine scaling method used in the BQ implementation, since primal feasibility is maintained at all times. Like the dual affine scaling algorithm, however, the majority of the computational effort is spent solving linear systems of the form $AD^2 A^T dy = b$. OSL also uses a minimum degree row-ordering heuristic to find an efficient Cholesky factorization of the projection matrix. The OSL implementation used for this study was programmed to consider all columns as sparse columns in a full-sized Cholesky factorization. All default parameters were used to solve the problems, except that the stopping criterion was designed to solve problems to six significant digits. No preprocessing was performed on the problems prior to solution, and the method to find a feasible interior point used was the same as that employed in the BQ decomposition.

4.3 Computational Results

All problems were solved using only interior point iterations. As mentioned above, no preprocessing was used and all system parameters were set to their system defaults. Solutions were obtained to within six significant digits of (known) optimality except for the 32 scenario instances of the scfxm1 and scsd8 problems. These problems were not solved to optimality due to computational difficulties with the BQ implementation. The difficulty in these cases was most likely in round-off accumulation in the construction of the G_1 matrix. Although we believe that more precise solutions with each $Sp = b$ system might have eliminated this problem, it does suggest that some numerical difficulties may still exist in the BQ factorization. The tradeoff is, however, for increased speed and distributed capabilities as shown below.

Table 4 shows the interior point solution times (see Figures 3 and 4) and the total solution times for the dual, primal and split formulations, as well as the BQ decomposition. As might be expected, the dual requires less time to solve than the primal problem, except when the number of columns in the primal problem is unusually large or the product TT^T is unusually dense. Considering the split formulation may considerably reduce computational requirements, but is not as advantageous as the dual problem when the number of scenarios grows large. Finally, finding the solution using the BQ decomposition *embedded in the dual affine algorithm* is the fastest option.

Problem	Solution Time (seconds) (1)				Total Time (seconds) (2)			
	Dual	Primal	Split	BQ	Dual	Primal	Split	BQ
sc205.1	0.59	0.70	0.76	0.243	0.87	1.00	1.05	0.264
sc205.2	1.07	1.32	1.46	0.481	1.56	1.78	1.98	0.52
sc205.3	2.03	3.07	2.82	1.014	2.87	3.94	3.75	1.102
sc205.4	4.08	9.33	6.64	1.987	5.55	10.86	8.45	2.206
sc205.5	9.42	44.74	15.44	4.219	12.56	47.87	18.97	4.828
scagr7.1	2.66	1.31	1.32	0.391	4.22	1.95	1.90	0.444
scagr7.2	4.80	2.92	2.66	0.861	5.79	3.96	3.72	0.971
scagr7.3	9.56	8.03	5.33	1.831	11.45	9.89	7.33	2.089
scagr7.4	14.55	41.84	11.48	4.011	18.02	45.57	15.34	4.673
scagr7.5	27.29	263.17	33.23	9.124	34.19	270.27	40.82	11.064
scfxm1.1	20.67	10.03	10.29	4.76	23.30	12.89	13.27	5.239
scfxm1.2	43.57	21.30	19.37	6.91	48.14	26.82	24.86	8.701
scfxm1.3	104.00	50.78	46.73	10.46	112.74	60.99	57.07	12.56
scfxm1.4	NA	135.02	144.03	NA	NA	154.72	164.27	NA
scrs8.1	1.92	1.07	1.23	0.651	2.40	1.53	1.80	0.693
scrs8.2	4.57	2.53	2.55	1.46	5.44	3.37	3.58	1.544
scrs8.3	5.55	12.39	6.18	1.836	6.99	13.88	8.14	2.022
scrs8.4	11.52	63.10	34.08	4.691	14.38	66.03	37.91	5.155
scrs8.5	25.30	459.20	60.06	11.997	30.86	464.90	67.55	13.292
scsd8.1	11.25	2.66	2.75	0.909	12.75	4.28	4.42	1.065
scsd8.2	21.40	5.44	5.76	1.693	24.29	8.61	8.85	2.01
scsd8.3	44.93	13.32	13.47	3.139	50.24	19.24	19.60	3.832
scsd8.4	93.82	42.25	39.58	NA	104.50	54.30	51.58	NA
Notes:	(1) Includes factorization time and interior point solution time							
	(2) Includes solution time, reading and printing time							

Table 5. Solution Times

Since two different methods and implementations were used in this study, direct solution time comparisons between problems solved with the BQ decomposition and the primal, dual and split variable formulation do not accurately represent the relative merits of each solution method. Since this study is ultimately concerned with the efficiency of solving linear systems of the form $AD^2 A^T dy = b$, more meaningful comparisons may be made by comparing the CPU time required per interior point iteration.

Table 6 (and Figures 5 and 6) shows the (average) CPU time in seconds required per interior point iteration performed on each problem. With the exception of some smaller problems (e.g. sc205.1, scfxm1.1, and scfxm1.2), using the BQ decomposition is considerably faster than any non-decomposition based solution technique. As the number of scenarios in the deterministic equivalent increases, the BQ decomposition becomes particularly attractive.

Problem	Iterations				Time per Iteration (secs)			
	Primal	Dual	Split	BQ	Primal	Dual	Split	BQ
sc205.1	26	52	33	19	0.027	0.011	0.032	0.014
sc205.2	25	20	34	22	0.053	0.054	0.058	0.024
sc205.3	30	21	34	23	0.102	0.097	0.110	0.048
sc205.4	32	22	36	23	0.292	0.185	0.235	0.096
sc205.5	21	24	38	24	2.130	0.393	0.499	0.201
scagr7.1	23	121	33	24	0.057	0.022	0.058	0.019
scagr7.2	24	74	35	27	0.122	0.065	0.106	0.036
scagr7.3	31	73	34	28	0.259	0.131	0.216	0.075
scagr7.4	33	51	34	32	1.268	0.285	0.451	0.146
scagr7.5	37	39	53	34	7.113	0.700	0.770	0.325
scfxm1.1	42	35	46	15	0.239	0.591	0.288	0.349
scfxm1.2	48	37	42	15	0.444	1.178	0.592	0.460
scfxm1.3	49	38	50	14	1.036	2.737	1.141	0.747
scfxm1.4	18	NA	61	14	7.501	NA	2.693	0.933
scrs8.1	20	39	24	24	0.054	0.049	0.075	0.029
scrs8.2	26	38	24	17	0.097	0.120	0.149	0.091
scrs8.3	22	33	28	25	0.563	0.168	0.291	0.081
scrs8.4	28	32	82	33	2.254	0.360	0.462	0.156
scrs8.5	28	30	27	32	16.400	0.843	2.502	0.415
scsd8.1	18	17	20	18	0.148	0.662	0.221	0.059
scsd8.2	20	17	20	18	0.272	1.259	0.443	0.112
scsd8.3	20	17	20	24	0.666	2.643	0.980	0.160
scsd8.4	20	17	20	22	2.113	5.519	2.579	0.271

Table 6. Solution Time per Iteration

The merits of each solution method are highlighted by their relative speedup factors. Table 7 and Figures 7 and 8 give these statistics for the stochastic test set. The absolute speedup factors shown in Table 6 are the ratios between the solution times for each of the solution methods being compared. Likewise, the speedup factors based on time per iteration are the the ratios between the the average CPU time per iteration for each solution method.

In terms of absolute solution time, Table 7 shows that solving the dual is not necessarily faster than solving the primal for small problems. However, as the number of scenarios increases, solving a problem that does not have an exceptionally dense or large projection matrix is much easier when the dual formulation is used. In the extreme, solving the dual to problem scrs8.5 (64 scenarios) was over 18 times faster than solving the primal. As the scfxm1 and scsd8 problems show, the ratios of solution times for solving the dual compared to the primal are limited to at most approximately 2.5. Solving the split variable formulation offers smaller performance gains (over solving the primal) than solving the dual problem, but works for all problems. The problem scfxm1.4 suggests that these performance gains may be limited for exceptionally large problems and that there may be a breakeven point for each problem where the performance loss resulting from matrix size increase associated with solving the split formulation offsets the gain obtained through reduced cholesky fill-in.

Comparisons of speedup factors based on solution time per iteration show similar results. With the exception of the scsd8 problem, the BQ decomposition is on average 1.95 (std. dev 0.59) times faster than solving with the dual factorization. These results confirm the claims made in [6]. The problems scsd8 and scfxm1 offer higher speedups, and show trends toward increasing speedup with

problem size. The same conclusions appear to hold, though with less certain speedup factors, when the BQ decomposition is compared with the split variable formulation.

Problem	Absolute Speedups (1)				Speedups based on time per iteration			
	Primal Dual	Primal Split	Dual BQ	Split BQ	Primal Dual	Primal Split	Dual BQ	Split BQ
sc205.1	1.19	0.92	2.43	3.13	2.37	0.85	0.82	2.29
sc205.2	1.23	0.90	2.22	3.04	0.99	0.91	2.26	2.46
sc205.3	1.51	1.09	2.00	2.78	1.06	0.93	2.02	2.30
sc205.4	2.29	1.41	2.05	3.34	1.57	1.24	1.93	2.45
sc205.5	4.75	2.90	2.23	3.66	5.43	4.27	1.95	2.48
scagr7.1	0.49	0.99	6.80	3.38	2.59	0.99	1.19	3.11
scagr7.2	0.61	1.10	5.57	3.09	1.88	1.14	1.80	2.96
scagr7.3	0.84	1.51	5.22	2.91	1.98	1.20	1.76	2.89
scagr7.4	2.88	3.64	3.63	2.86	4.44	2.81	1.95	3.09
scagr7.5	9.64	7.92	2.99	3.64	10.16	9.24	2.15	2.37
scfxm1.1	0.49	0.97	4.34	2.16	0.40	0.83	1.69	0.83
scfxm1.2	0.49	1.10	6.31	2.80	0.38	0.75	2.56	1.29
scfxm1.3	0.49	1.09	9.94	4.47	0.38	0.91	3.66	1.53
scfxm1.4	NA	0.94	NA	NA	NA	2.79	NA	2.89
scrs8.1	0.56	0.87	2.95	1.89	1.09	0.71	1.70	2.60
scrs8.2	0.55	0.99	3.13	1.75	0.81	0.65	1.32	1.64
scrs8.3	2.23	2.00	3.02	3.37	3.35	1.94	2.08	3.59
scrs8.4	5.48	1.85	2.46	7.26	6.26	4.87	2.30	2.96
scrs8.5	18.15	7.65	2.11	5.01	19.45	6.56	2.03	6.02
scsd8.1	0.24	0.97	12.38	3.03	0.22	0.67	11.18	3.74
scsd8.2	0.25	0.94	12.64	3.40	0.22	0.61	11.27	3.96
scsd8.3	0.30	0.99	14.31	4.29	0.25	0.68	16.55	6.14
scsd8.4	0.45	1.07	NA	NA	0.38	0.82	20.38	9.52
Notes:	(1) Ratio of Solution Times							

Table 7. Speedup Comparisons

As mentioned previously, the structure of the BQ decomposition lends itself to distributed processing. Table 8 (and Figure 9 in the appendix) give the best run times for the sc205 problem [6] obtained using a parallel version of the BQ implementation with those obtained using the serial version of the decomposition. The solution times given in Table 7 are "wall clock" times necessary to calculate the iterations required by the dual affine scaling algorithm [7]. The parallel implementation clearly gives a substantial performance improvement over a serial implementation of the BQ decomposition for large problems. For example, solving the 64 scenario sc205 problem is almost three times faster using four processors than the serial version. Speedups are not linear with the number of processors since the communication requirements quickly overtake the benefits provided by the distribution of computational work. The 8 and 16 scenario problems suggest that three or four scenarios per processor offer the best performance.

---

[6]Run times were obtained using a network of six DEC 5000/320 workstations.

[7]Unfortunately, since the network connecting the processors was a shared resource, obtaining accurate solution times for larger number of processors was quite difficult. Fully utilizing the parallel implementation requires communications bandwidths unobtainable over a shared ethernet.

		Solution Time (seconds) (1)						
		Serial	Number of Processors					
Problem	Scenarios	Algorithm	1	2	3	4	5	6
sc205.1	4	3	5	5	5	5	-	-
sc205.2	8	6.7	9	7	8	7	9	11
sc205.3	16	13.2	17	13	12	12	14	15
sc205.4	32	28	38	22	22	19	23	23
sc205.5	64	55.5		34	31	20	93	
Notes:	(1) Solution times are best encountered in at least 10 runs							
	(2) Blank times unavailable due to system limitations							

Table 7. Parallel Solution Times

5 Conclusions

This paper reviewed the need for and characteristics of solving (discrete) stochastic linear programs with fixed recourse (or more generally, dual block angular linear programs) using interior point methods. Direct application of interior point methods to larger problems of this nature is computationally difficult if not infeasible. The reason for this lies in the many dense columns which are characteristic of these problems. To resolve this problem, four methods that directly address the problem of dense columns were reviewed. Reformulation of the program to break up these dense columns can improve the performance of interior point methods, but may require a substantial increase in problem size. Solving the problem's dual (or using the transpose factorization) is also possible and generally works well in practice. However, solving the dual for problems with dense or large recourse matrices does not work well. Partitioning the constraint matrix into two matrices with dense and non-dense columns is another alternative but suffers from inherent numerical instability.

Decomposing the constraint matrix into the sum of two matrices and applying the Sherman-Morrison-Woodbury formula is the BQ decomposition alternative studied here in detail. Computational experience with this method suggests that it substantially reduces the effort needed to solve for the search direction at each iteration. Solving the dual or primal requires a Cholesky factorization of a sparse, but still large matrix. By contrast, the BQ decomposition technique requires several smaller, but independent, factorizations. In practice, this effort is small compared to that required by non-decomposition based techniques. Computational experience indicates that speedups may increase with the number of scenarios. Taking advantage of the ease with which the decomposition may be parallelized further reduces the computational requirements necessary to solve dual block angular programs, and in practice offers substantial performance improvements.

References

[1] I. Adler, N. Karmarkar, M.G.C. Resende, and G. Veiga, 1989."An Implementation of Karmarkar's Algorithm for Linear Programming," *Mathematical Programming 44*, 297-335. Errata in *Mathematical Programming 50*, 415, 1991.

[2] J. Arantes and J. Birge, 1989. "Matrix Structure and Interior Point Methods in Stochastic Programming," Presentation, *Fifth International Stochastic Programming Conference*, Ann Arbor, Michigan.

[3] E. R. Barnes, 1986. " A variation on Karmarkar's algorithm for solving linear programming problems," *Mathematical Programming 36*,174-182.

[4] E.M.L. Beale, 1955. "On Minimizing a Convex Function subject to Linear Inequalities," *Journal of the Royal Statistical Society B 17* 173-184.

[5] A. Beguelin, J. Dongarra, A. Geist, R. Mancheck, and V. Sunderam, 1991. "A User's Guide to PVM: Parallel Virtual Machine," Report ORNL/TM-11826, Oak Ridge National Laboratory, Deparment of Energy, Oak Ridge, Tennessee.

[6] J. R. Birge and L. Qi, 1988. "Computing Block-angular Karmarkar Projections with Applications to Stochastic Programming," *Management Science 34:12*, 1472-1479.

[7] J. R. Birge, H. I. Gassman, and D. Holmes, 1991. STOPGEN stochastic linear program deterministic equivalent problem generator.

[8] J. R. Birge, R. M. Freund, and R. J. Vanderbei, 1990. " Prior reduced fill-in in the solution of equations in interior point algorithms," Technical Report, Sloan School of Management, MIT (Cambridge, MA, 1990), to appear in *Operations Research Letters*.

[9] J. R. Birge, 1985. "Decomposition and Partitioning Methodsfor Multistage Stochastic Linear Programs," *Operations Research 33*, 989-1007.

[10] W.J. Carolan, J.E. Hill, J.L. Kennington, S. Niemi, and S.J. Wichmann, 1990. "An empirical evaluation of the KORBX algorithms for military airlift applications," *Operations Research 9:2*, 169-184.

[11] J. Carpenter, I. Lustig, and J. Mulvey, 1991. "Formulating Stochastic Programs for Interior Point Methods," *Operations Research 39:5*, 757-770.

[12] I.C. Choi, C.L. Monma, and D.F. Shanno, 1988. "Further Development of a Primal-Dual Interior Point Method," Report 60-88, RutgersCenter for Operations Research, Rutgers University, New Brunswick, New Jersey.

[13] E. Chu, A. George, J. Liu, and E. Ng, 1984. "SPARSPAK: Waterloo Sparse Matrix Package User's Guide for SPARSPAK-A," Research Report CS-84-36, Department of Computer Science, University of Waterloo, Waterloo, Ontario.

[14] G. Dantzig, 1955. "Linear Programming under Uncertainty," *Management Science 1* 197-206.

[15] G. Dantzig, and A. Mandansky, 1961. "On the Solution of Two Stage Linear Programs under Uncertainty," *Proceedings of the Fourth Berkely Symposium on Mathematics and Probability Vol. 1*, Universityof California Press, Berkely, California, 165-176.

[16] I.I. Dikin, 1967. " Iterative solution of problems of linear and quadratic programming," *Soviet Mathematics Doklady 8*, 674-675.

[17] J. Edwards, 1988. "A Proposed Standard Input Format for Computer Codes which solve Stochastic Programs with Recourse," **Numerical Techniques for Stochastic Optimization**, Yu. Ermoliev and R.J.-B. Wets (eds), 215-228.

[18] S.C. Eisenstadt, M.C. Gurshy, M.H. Shultz, and A.H. Sherman, 1981. "The Yale Sparse Matrix Package, I. The Symmetric Codes," *ACM Transactions on Mathematical Software*.

[19] H.I. Gassman, 1989. "Optimal harvest of a forest in the presence of uncertainty," *Canadian Journal of Forest Research 19*, 1267-1274.

[20] D.M. Gay, 1985. "Electronic Mail Distribution Linear Programming Test Problems," Mathematical Programming Society COAL Newsletter, December.

[21] P.E. Gill, W. Murray, M.A. Saunders, J.A. Tomlin, and M.H. Wright, 1986. "On Projected Newton Methods for Linear Programming and Equivalence to Karmarkar's Projection Method," *Mathematical Programming 36*, 183-201.

[22] P.E. Gill, W. Murray, and M.A. Saunders, 1988. "A single phase dual barrier method for Linear Programming," Report SOL-88-10, Systems Optimization Laboratory, Stanford University, Stanford, California.

[23] G.H. Golub and C.F. van Loan, 1983. **Matrix Computations**, The Johns Hopkins University Press, Baltimore, Maryland.

[24] J.K. Ho and E. Loute, 1981. "A Set of Linear Programming Test Problems," *Mathematical Programming 20* 245-250.

[25] International Business Machines Corp., 1991. "Optimization Subroutine Libary Guide and Reference, " document SC23-0519-01, International Business Machines Corp.

[26] N. Karmarkar, 1984. "A New Polynomial Time Algorithm for Linear Programming," *Combinatorica 4*, 373-395.

[27] A.J. King, 1988. "Stochastic Programming Problems: Examples from the Literature," pp. 543-567 **Numerical Techniques for Stochastic Optimization**, Yu. Ermoliev and R.J.-B. Wets (eds).

[28] F.V. Louveaux, 1980. " A solution method for multistage stochastic programs with recourse with application to an energy investment problem,", *Operations Research 28*, 889-902.

[29] I. Lustig, R. Marsten, and D. Shanno, 1989. "ComputationalExperience with a Primal-Dual Interior Point Method for Linear Programming," Technical Report SOR 89-17, Department of Civil Engineering and Operations Research, Princeton University, Princeton, New Jersey.

[30] R. Marsten, R. Subramanian, M. Saltzman, I. Lustig, and D. Shanno, 1990. "Interior Point Methods for Linear Programming: Just call Newton, Lagrange, and Fiacco and McCormick!," *Interfaces 20:4*, 105-116.

[31] N. Megiddo, 1986. "Pathways to the Optimal Set in Linear Programming," Technical Report RJ 5295, IBM Almaden Research Center, San Jose, California.

[32] J.A. Meijerink and H.A. van der Vorst, 1977. "An IterativeSolution Method for Linear Equation Systems of which the Coefficient matrixis a Symmetric M-matrix," *Mathematical Computations, 31* 148-162.

[33] C.L. Monma and A.J. Morton, 1987. "Computational Experience with a Dual Affine Variant of Karmarkar's Method for Linear Programming," *Operations Research Letters 6*, 261-267.

[34] J.M. Mulvey, 1984. "A Network Portfolio Approach for Cash Management," *Journal of Cash Management 4*, 46-48.

[35] J.M. Mulvey and H. Vladimirou, 1989. "Stochastic Network Optimization Models for Investment Planning," to appear, B. Shelby (ed.) *Annals of Operations Research*, special issue on Networks and Applications.

[36] B.A. Murtaugh and M.A. Saunders, 1983. "MINOS 5.1 User's Guide," Technical Report SOL-83-20R, Systems Optimization Laboratory, Stanford University, Stanford, California.

[37] D.F. Shanno, and A. Bagchi, 1990. "A Unified View of Interior Point Methods for Linear Programming," *Annals of Operations Research 22*, 55-70.

[38] D. Shanno, 1988. "Computing Karmarkar Projections Quickly," *Mathematical Programming 41*, 61-71.

[39] R. Van Slyke and R. Wets, 1969. "L-Shaped Linear Programs with Applications to Optimal Control and Stochastic Programming," *SIAM Journal of Applied Mathematics 17*, 638-663.

[40] R. J. Vanderbei, M. S. Meketon, and B. A. Freedman, 1986. " A modification of Karmarkar's linear programming algorithm," *Algorithmica 1*, 395-407.

[41] R. J. Vanderbei and T.J. Carpenter, 1991. "Symmetric Indefinite Systems for interior point methods." Technical Report SOR 91-7,Dept. of Civil Engineering and Operations Research, Princeton University.

Appendix
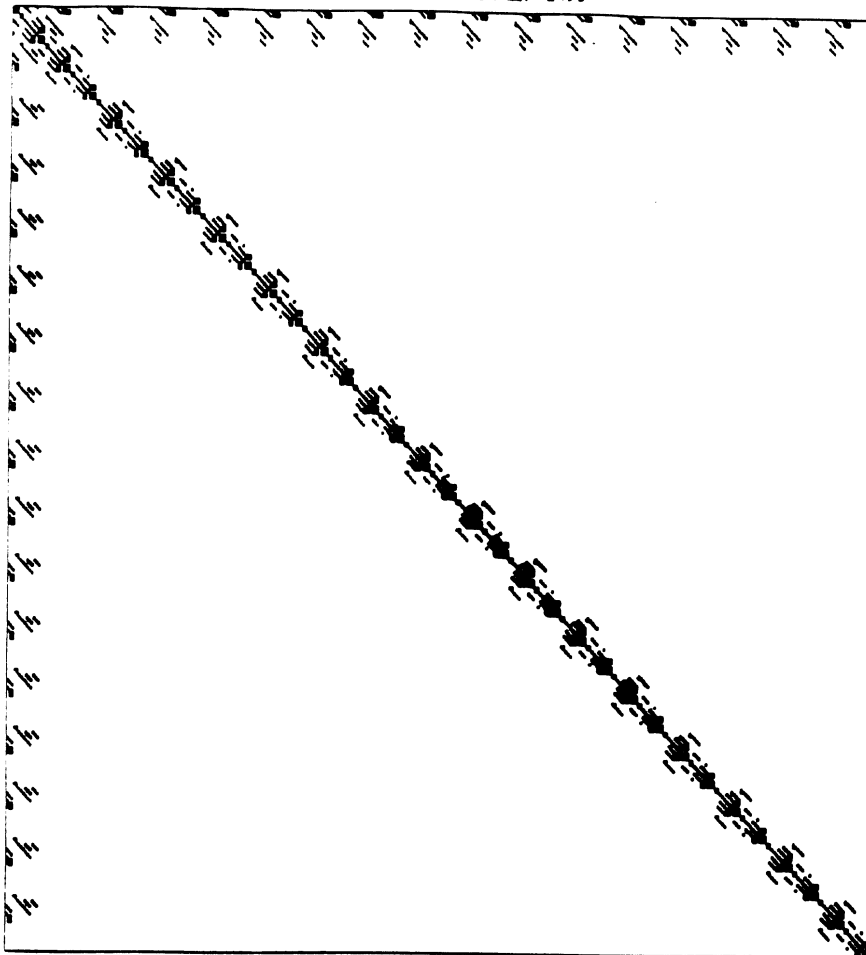
Rows: 366 Cois: 366 Nzs: 2493

Figure 1a: SC205, 16 scenarios, Dual
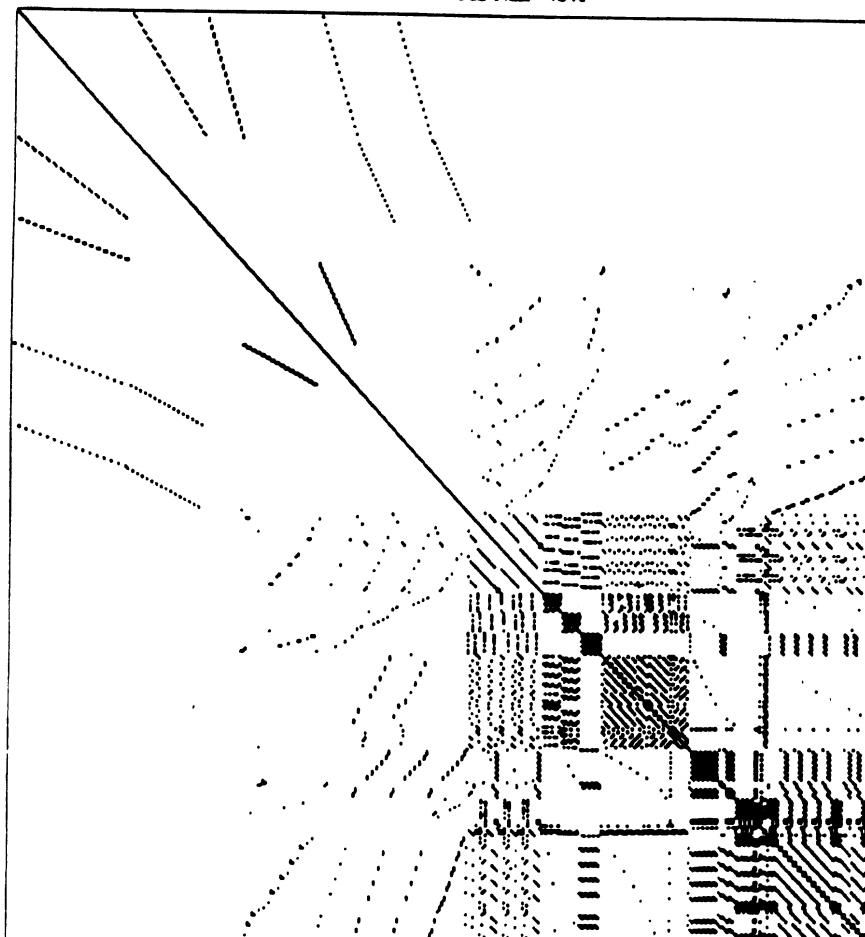

Rows: 365 Cols: 365 Nzs: 4841

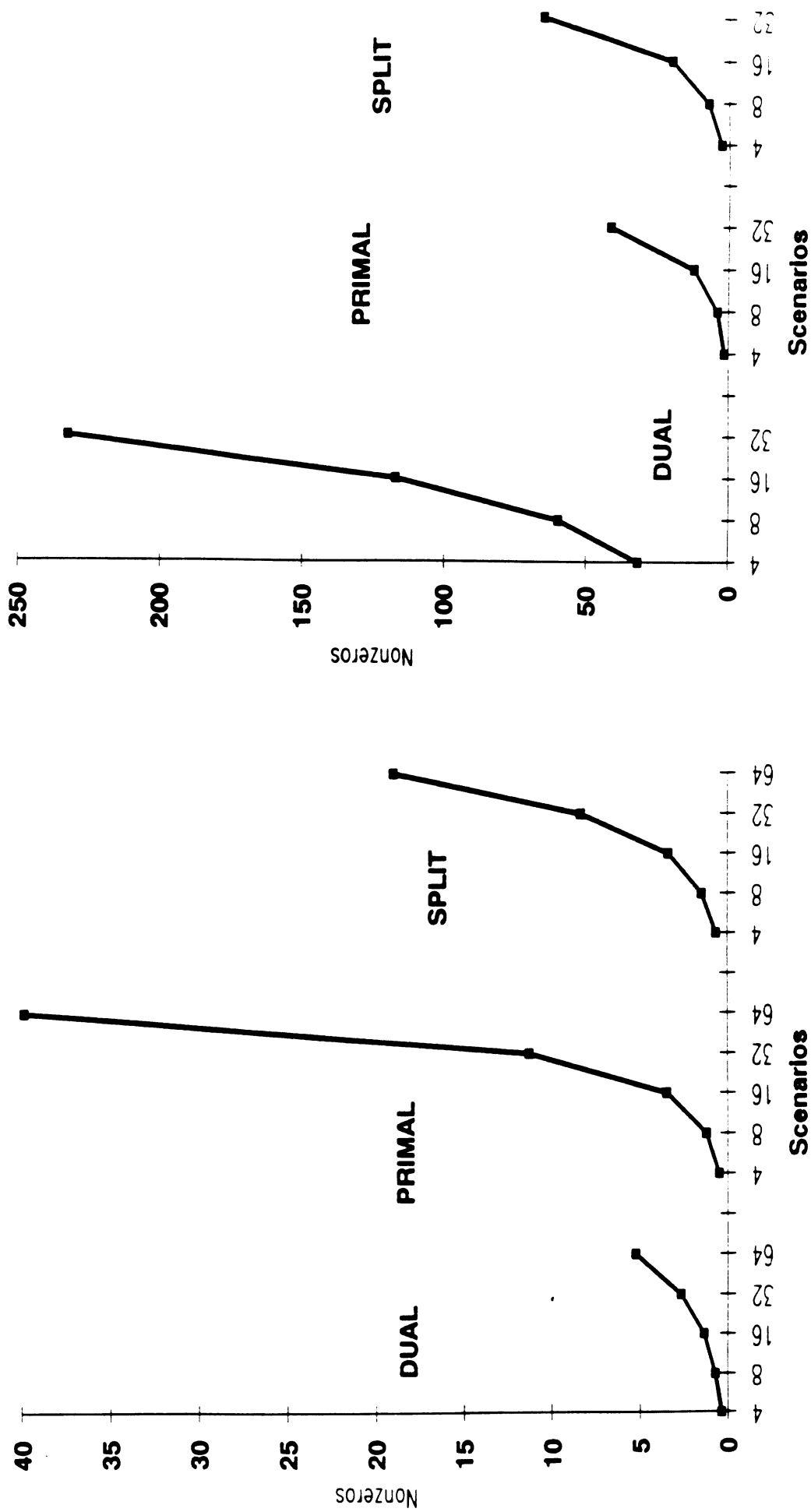Figure 1b: SC205, 16 scenarios, Primal

Figure 2. Cholesky nonzeroes for SC205 and SCSD8

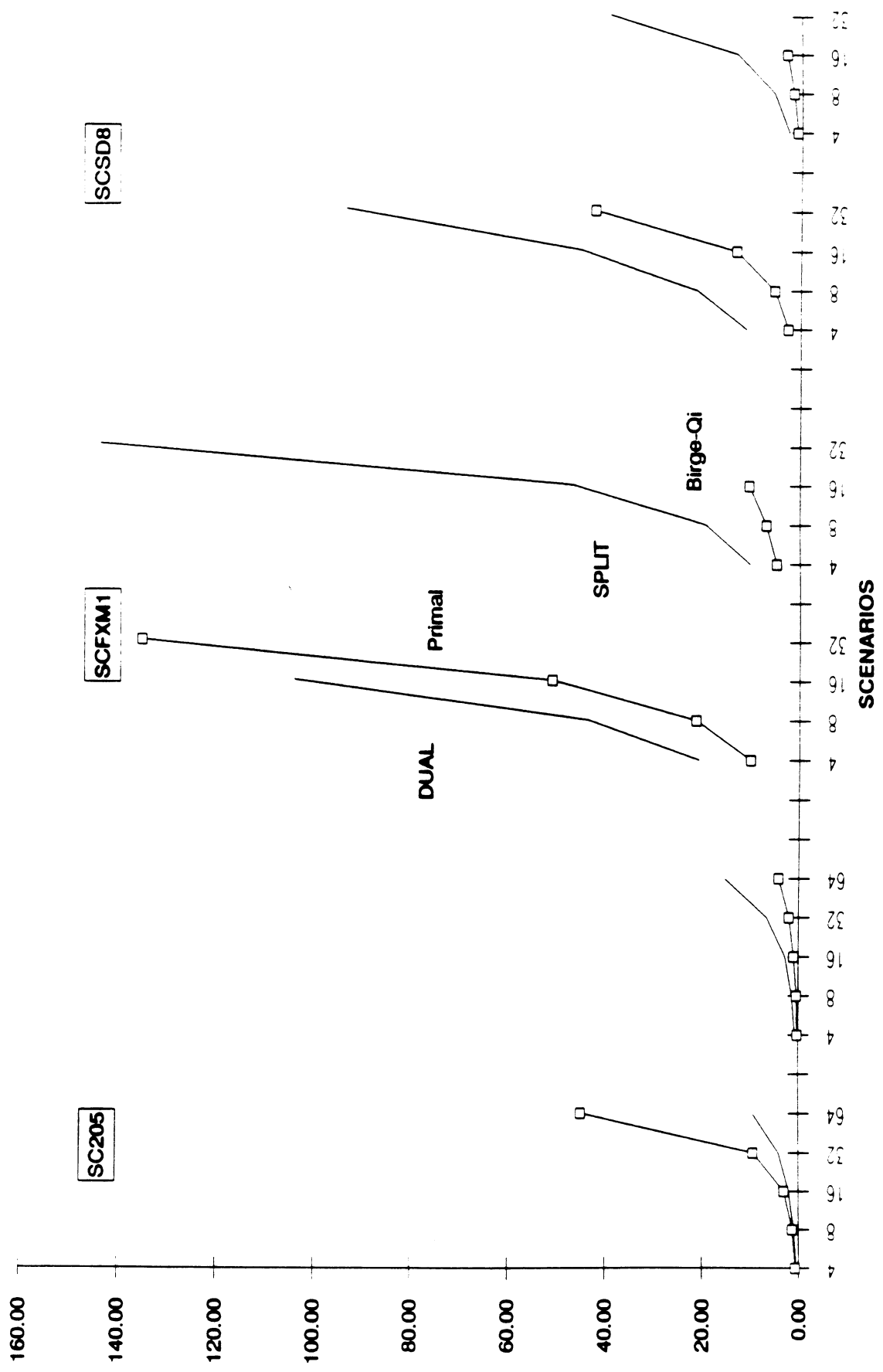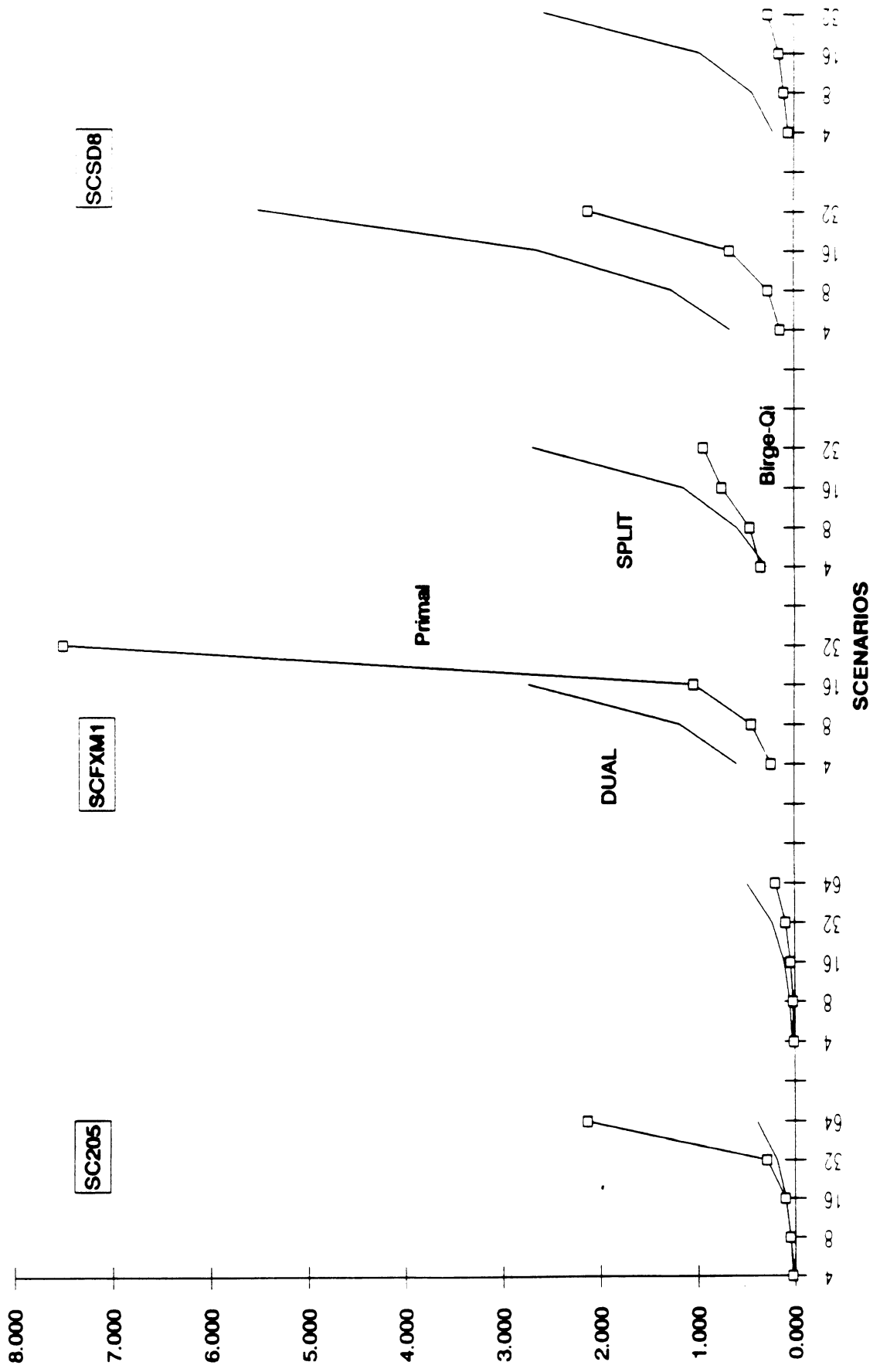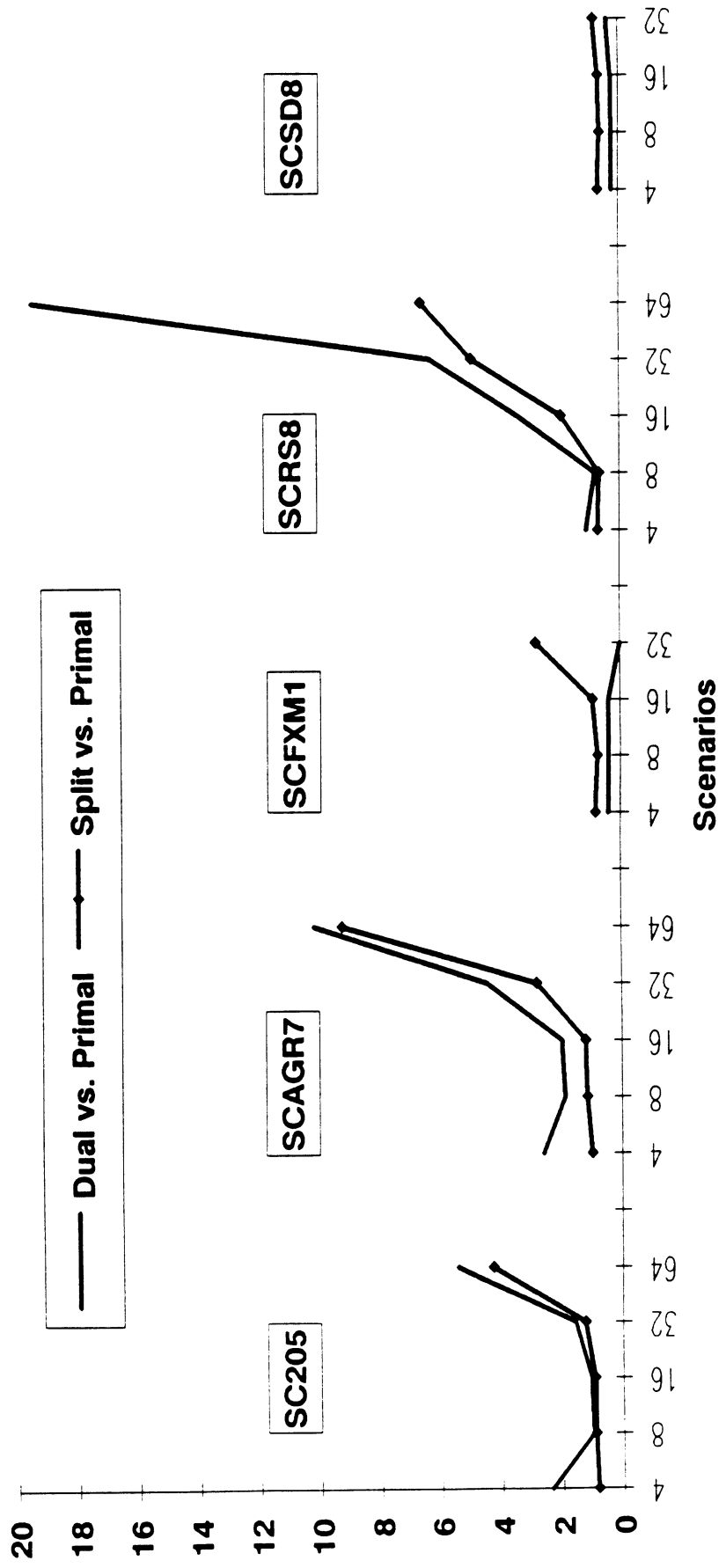Figure 3. Solution Times (seconds) for SC205, SCFXM1, and SCSD8

Figure 4. Solution Times (seconds) for SCRS8 and SCAGR7

Figure 5. Solution Times per iteration (seconds) for
SC205, SCFXM1, and SCSD8

Figure 6. Solution Times per iteration (seconds) for SCRS8 and SCAGR7

Figure 7. Speedups associated with solving the Primal
(based on solution time per iteration)

Figure 8. Speedups associated with using the BQ decomposition (based on solution time per iteration)
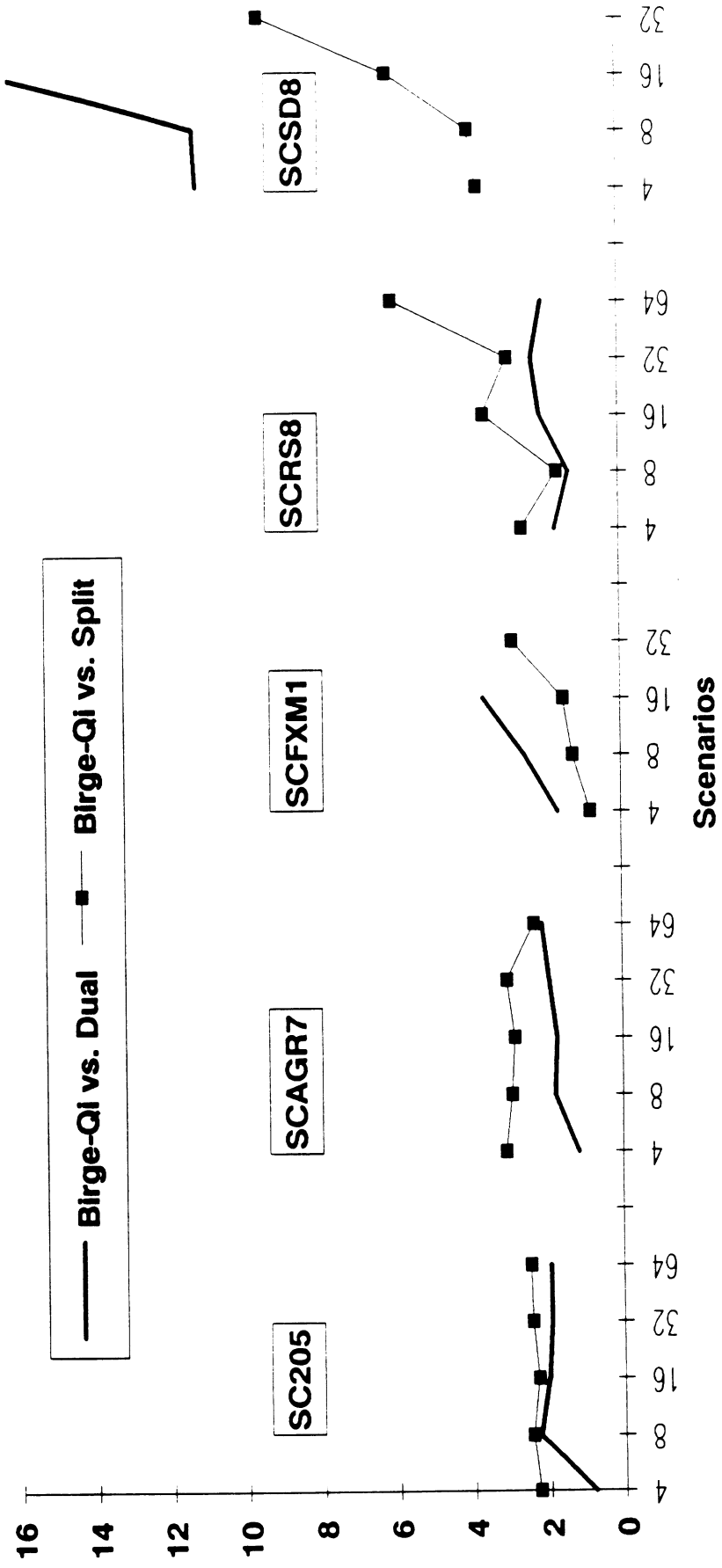
Figure 9. Distributed Solution Times
(SC205)