

Efficient Solution of Two-Stage Stochastic Linear Programs Using Interior Point Methods

J.R. BIRGE AND D.F. HOLMES

Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI 48109.

Received January 15, 1992, Revised August 3, 1992.

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., $A^T A$) works well but is highly problem dependent. Schur complements may require solutions with potentially near singular matrices. 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 suggests that substantial performance improvements are possible with each method and that, generally, explicit factorizations require the least computational effort.

Keywords: Interior point algorithms, stochastic programming.

1. Introduction

Many practical problems with uncertain parameters can be modeled as stochastic programs. Some examples include cash and portfolio management models ([31]), electric power generation capacity planning models ([26]), and forestry management problems ([19]). A survey of documented stochastic programming formulations can be found in ([25]). Most basic among stochastic programs are discretely distributed 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 ([29], [11], and [30]), however, holds great promise for the efficient solution of these problems. The basic computational requirement for these algorithms 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 backsolving 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) SLPs, and report both serial and parallel computational experience with one particularly promising method described by Birge and Qi ([10]). 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 projective scaling 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 an affine version of 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

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

where $Q(x_0, \xi)$ 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

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

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

$$Q(x_0, \xi_l) = \inf \{d_l y_l | W_l y_l = b_l - T_l x_0, y_l \geq 0, y_l \in \mathbb{R}^m\}, \xi_l = (d_l, b_l, T_l, W_l), \tag{2}$$

for each scenario $l = 1, \dots, N$. Here, a decision x_0 is made before ξ_l is known, and an optimal "corrective" 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 $Q(x_0, \xi)$. 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, \dots, y_N) , we obtain

$$\begin{aligned}
 &\text{minimize} && c_0^T x_0 + \sum_{l=1}^N c_l^T y_l \\
 &\text{subject to} && A_0 x_0 = b_0 \\
 & && T_l x_0 + W_l y_l = b_l \quad l = 1, \dots, N \\
 & && x_0, y_l \geq 0,
 \end{aligned} \tag{3}$$

where $c_0, x_0 \in \mathbb{R}^{n_0}$; $T_l \in \mathbb{R}^{m_l \times n_0}$; $A_0 \in \mathbb{R}^{m_0 \times n_0}$; $W_l \in \mathbb{R}^{m_l \times n_l}$; and $c_l = p_l d_l \in \mathbb{R}^{n_l}$ for $l = 1, \dots, N$. 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 and W_l have full row rank, $m_l \leq n_l$, $l = 0, \dots, N$, 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 ([5]) 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 ([35]) and the decomposition method of Dantzig and Madansky ([15]). Interior point algorithms such as those proposed by Karmarkar ([24]) and Marsten et al. ([29]) applied to these problems have been discussed by Birge and Qi ([10]) and Lustig et al. ([28]).

2.2. Interior point methods

In the last decade, several breakthroughs in general-purpose linear programming algorithms have been made using interior point methods ([29]). Karmarkar ([24]) pioneered these breakthroughs with the first practical interior point method that could be proven to converge to an optimal solution in polynomial, or $O(n^3 L)$ 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:

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

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

$$\begin{aligned} \text{(D)} \quad & \text{maximize} \quad b^T y \\ & \text{subject to} \quad A^T y \leq c, \end{aligned}$$

where $y \in \mathbb{R}^m$ is a dual vector to the equality constraints of (P). In general, we could also rewrite (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 ([4], [16], and [37]). The only difference is in the form of the matrix factorization ([8]). We can, therefore, use simply *affine scaling* to refer to these methods.

Given a dual feasible interior point, the dual affine algorithm described below may be used to find an optimal solution.

Procedure DualAffine(A, b, c , *stopping criterion*).

1. $k = 0$
2. Stop if optimality criterion is satisfied.
3. Let $v^k = c - A^T y^k$.
4. Calculate the search direction.
 - (a) Let $D^k = \text{diag}\{(1/v_1^k), \dots, (1/v_m^k)\}$.
 - (b) Let $dy = (A(D^k)^2 A^T)^{-1} b$.
 - (c) Let $dv = -A^T dy$.
5. Calculate a step size.
 - (a) Let $\alpha = \gamma \times \min\{v_i^k / -(dv)_i : (dv)_i < 0, i = 1, \dots, m\}$, where $0 < \gamma < 1$.
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.

The vast majority of the computational effort required in the above procedure is to calculate a solution to the symmetric positive definite system $(AD^2 A^T)dy = b$ (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 ([34]). There are two main strategies for solving the system $(AD^2 A^T)dy = b$.

They are *iterative methods*, which generate sequences of approximations to dy using simple matrix-matrix multiplications, and *direct methods*.

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 ([18]) and SPARSPAK ([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. Rearranging matrices to minimize fill-in reduces memory usage and the number of operations to update the factorization or obtain a solution. However, matrices that are ill-structured may not yield sparse Cholesky factorizations. Problems with these types of matrices may be quite difficult 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 Section 2.1 (potentially) contains many dense columns. To see this, let $D_l^2 \in \mathbb{R}^{m_l \times m_l}$ be defined by $D_l^2 = \text{diag}\{(\nu_1^l)^{-2}, \dots, (\nu_{m_l}^l)^{-2}\}$, $l = 0, \dots, N$. Suppose further that $T^l = T$, $W^l = W$, $l = 1, \dots, N$. Then solving the system requires a factorization of

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

$$= \begin{bmatrix} A_0D_0^2A_0^T & A_0D_0^2T^T & A_0D_0^2T^T & \dots & A_0D_0^2T^T \\ TD_0^2A_0^T & TD_0^2T^T + WD_1^2W^T & TD_0^2T^T & \dots & TD_0^2T^T \\ TD_0^2A_0^T & TD_0^2T^T & TD_0^2T^T + WD_2^2W^T & \dots & TD_0^2T^T \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ TD_0^2A_0^T & TD_0^2T^T & TD_0^2T^T & \dots & TD_0^2T^T + WD_N^2W^T \end{bmatrix}$$

Clearly, the presence of the columns associated with the T matrices creates an extremely dense matrix to factorize. For this reason, Arantes and Birge ([3]) 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 (see also [28]). 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 have been proposed to both the formulation of the block angular program and the implementation of the interior point algorithm required for its solution. 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 (nondense) columns. Four alternatives will be explored in this section: reformulation of the program to split up dense columns ([28]), solution of the dual to the program (or a factorization formed from the dual, [3] and [8]), the use of the Schur complement to remove dense columns ([27] and [28]), and direct solution by a special factorization of the matrix AD^2A^T ([10]).

3.1. Reformulation of the program

Lustig et al. ([28]) consider a two-stage generalized stochastic network derived from a portfolio management model (developed in [32]). 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. Instead of enforcing nonanticipativity in the original formulation (3) by keeping the first n_0 columns 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

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

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

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

By concentrating the nonzero 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 10 portfolio test problems run in ([28]), 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 that have nonzero elements in the rows of T_i , 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 Cholesky factorizations of the AD^2A^T matrices with splitting were on average 2.8 times less dense than those obtained from the original formulation without splitting. The split formulations themselves were solved using a commercial implementation of a primal-dual interior point algorithm, OB1 ([27]), 10.8 times faster than the original formulation. The splitting formulation was (on average) 10.8 times faster than the original formulation. On average, solving the split formulations using OB1 was 5.58 times faster than the simplex method MINOS 5.3, developed by Murtagh and Saunders ([33]).

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 that are in linking constraints to recourse decisions. An example from the test set used here ([7] and [22]) is SCFXM1, which has 52 of 114 first-stage variables linked to second-stage variables. For this problem (see Section 4.3), splitting variables and dual formulations offer no computational advantage.

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 that exhibits a dual block angular structure is solved using dual affine scaling. Arantes and Birge ([3]) suggested that reformulating the primal problem in the polyhedral inequality form would allow more efficient computation since the relevant matrix for computation is A^TD^2A , 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

$$\begin{aligned} \max \quad & b_0^T y_0 + \sum_{i=1}^N b_i^T y_i \\ \text{st} \quad & A_0^T y_0 + \sum_{i=1}^N T_i^T y_i \leq c_0 \\ & W_i^T y_i \leq c_i. \end{aligned} \quad (5)$$

Since the resulting coefficient matrix B of the dual formulation has $n = n_0 + \sum_{i=1}^N n_i$ rows and $m = m_0 + \sum_{i=1}^N m_i$ 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. Here,

$$BD^2B^T = \begin{bmatrix} A_0^T D_0^2 A_0 + \sum_{i=1}^N T_i^T D_i T_i & T^T D_1^2 W & \cdots & T^T D_N^2 W \\ W^T D_1^2 T & W D_1 W^T & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ W^T D_N^2 T & 0 & 0 & W D_N W^T \end{bmatrix} \quad (6)$$

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. (SC205 with 16 scenarios. For problem characteristics, please refer to Section 6.)

The Cholesky factorization of the matrix (6) is generally also sparse. Solution times reported by Arantes and Birge for a common test set ([7] and [22]) were much faster using the BD^2B^T form over the AD^2A^T form. The largest problem they tried (32 scenarios, 7,023 nonzeros in AD^2A^T) 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 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.

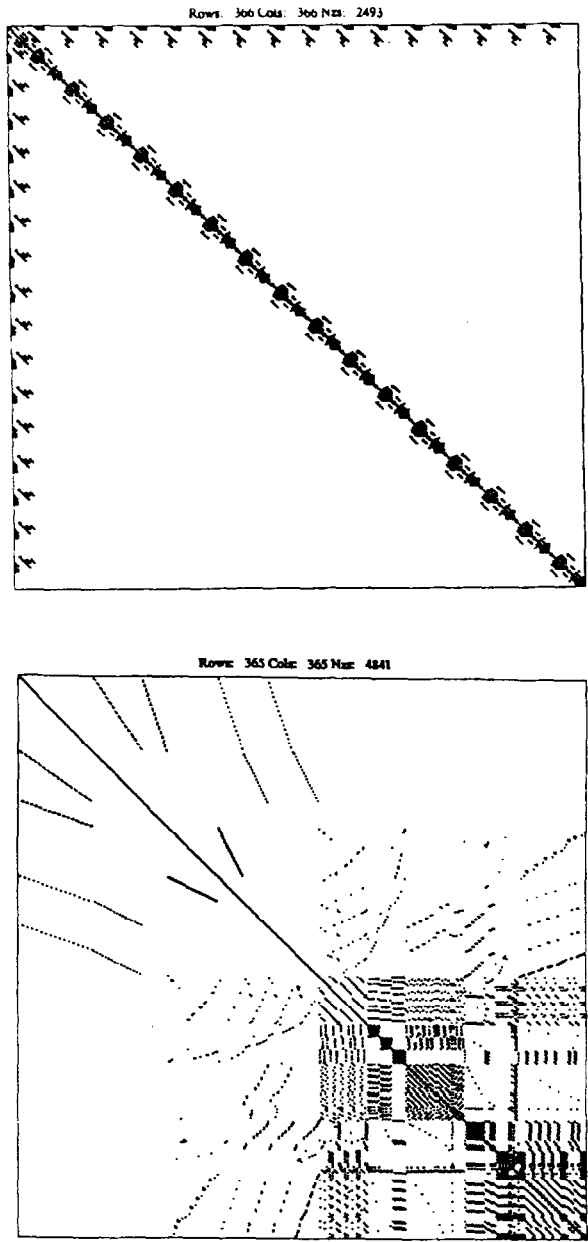


Figure 1. Nonzero structure of sc205, 16 scenarios: (Top) Dual; (Bottom) primal.

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 that can be partitioned into $[A_d; A_s]$, where $A_s \in \mathbb{R}^{m \times n_s}$ is a submatrix containing only sparse columns, and $A_d \in \mathbb{R}^{m \times n_d}$ contains only dense columns. In the case of the two-stage dual block angular program, A_d contains (at most) 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 $AD^2A^T = A_sD_s^2A_s^T + A_dD_d^2A_d^T$. Let the Cholesky factorization of $A_sD_s^2A_s^T$ be LL^T , $V = A_dD_d$, and $\delta = -V^Tdy$. Then $VV^T = A_dD_d^2A_d^T$ and solving the system

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

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

$$\begin{aligned} LL^Tdy &= b + V\delta \quad \text{or} \\ dy &= (LL^T)^{-1}(b + V\delta). \end{aligned} \quad (8)$$

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

$$V^Tdy + I\delta = 0 \quad (9)$$

Substituting (8) into (9),

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

The matrix $I + V^T(LL^T)^{-1}V$ is a Schur complement, 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

$$LL^Tdy = b + V\delta.$$

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

As shown in ([27]), 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. The overall effort required to obtain the sparse factorization is still $O(n_s^3)$, so efficiency gains

cannot be guaranteed. However, as reported by 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, they solved the ISRAEL problem in the NETLIB test set ([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 six 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 may leave a column with no nonzeros in $A_s D_s^2 A_s^T$. For example, consider a one-scenario problem, with coefficient matrix $A = \begin{bmatrix} A_0 & 0 \\ T & W \end{bmatrix}$. If all first-stage columns are dense,

$$A_d^T = \begin{bmatrix} A_0^T & T^T \end{bmatrix} \text{ and } AD^2A^T = A_d D_d^2 A_d^T + A_s D_s^2 A_s^T =$$

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

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

As long as there are rows with nonzeros in dense columns but not in sparse columns, $A_s D_s^2 A_s^T$ is singular, and the Schur complement procedure fails. Even if A_s is forced to have full rank, the procedure is likely to suffer from numerical instability (this is investigated empirically in the next section). To address this problem, various methods have been proposed to improve the accuracy of the algorithm. For example, Lustig et al. ([27]) 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 were 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, Lustig et al. ([28]) halved solution times using the Schur complement over a splitting reformulation (Section 3.1). Since these problems may be split so that each row contains a nonzero in a dense column and a nonzero in a sparse column (which maintains the full row rank of the sparse Cholesky factorization), they encountered no significant loss of accuracy. However, the possible loss of accuracy associated with the Schur complement suggests that it may not be desirable for solving general 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 decomposition procedures specific to the dual block angular structure. Birge and Qi ([10]) proposed a better-conditioned method using a generalized version of the Sherman-Morrison-Woodbury formula. 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 $l = 0, \dots, N$). 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 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,*

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

Proof.

$$\begin{aligned} (A + UV^T)^{-1}(A + UV^T) &= I + A^{-1}UV^T - A^{-1}U(I + V^T A^{-1}U)^{-1}V^T - \\ &\quad A^{-1}U(I + V^T A^{-1}U)^{-1}V^T A^{-1}UV^T \\ &= I + A^{-1}UV^T - A^{-1}U(I + V^T A^{-1}U)^{-1} \\ &\quad (I + V^T A^{-1}U)V^T \\ &= I + A^{-1}UV^T - A^{-1}UV^T = I. \end{aligned}$$

Lemma 1 is the main tool in deriving the factorization in Birge and Qi's paper ([10]) (which we refer to as the BQ factorization). We slightly change the form of the M matrix they use to obtain the following result. The proof follows the same development as in ([10]).

THEOREM 1. *Consider the feasible region of the dual block angular program given in (3), written as $Ax = b, x \geq 0$. Let $M = AD^2A^T$, $S = \text{diag}\{S_0, S_1, \dots, S_N\}$, where $S_l = W_l D_l^2 W_l^T$, $l = 1, \dots, N$, $S_0 = I_2 \in \mathbb{R}^{m_0 \times m_0}$, and $D_l^2 = \text{diag}\{(\nu_1^l)^{-2}, \dots, (\nu_{m_l}^l)^{-2}\}$. Furthermore, let I_1 and I_2 be identity matrices of dimension n_0 and m_0 , respectively. Also, let*

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

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

If A_0 and $W_l, l = 1, \dots, N$ have full row rank, then M and $G_2 \equiv -A_0 G_1^{-1} A_0^T$ are invertible and

$$M^{-1} = S^{-1} - S^{-1} U G^{-1} V^T S^{-1}. \quad (14)$$

Proof. In the affine scaling algorithm, $\nu_k > 0$ for all k , so D is always invertible. By assumption, W_l has full row rank, so S_l is invertible for $l = 1, \dots, N$. Since $S = \text{diag}\{S_1, \dots, S_N\}$ augmented with an identity matrix, S is also invertible. Let $\bar{D} = \text{diag}\{D_0, I_2\}$, $\bar{U} = U\bar{D}$ and $\bar{V} = V\bar{D}$. By construction, $M = S + \bar{U}\bar{V}^T$.

To apply Lemma 1, $I + \bar{V}^T S^{-1} \bar{U}$ must be invertible. Now, $I + \bar{V}^T S^{-1} \bar{U} = I + \bar{D} V^T S^{-1} U \bar{D} = \bar{D}(\bar{D}^{-2} + V^T S^{-1} U) \bar{D}$ will be invertible if and only if \bar{D} is invertible and $\bar{D}^{-2} + V^T S^{-1} U$ is invertible. Since

$$G = \begin{bmatrix} D_0^{-2} + A_0^T A_0 + \sum_{l=1}^N T_l^T S_l^{-1} T_l & A_0^T \\ -A_0 & I_2 - I_2 \end{bmatrix}$$

$$= \begin{bmatrix} D_0 & 0 \\ 0 & I_2 \end{bmatrix}^{-2} + \begin{bmatrix} A_0^T A_0 + \sum_{l=1}^N T_l^T S_l^{-1} T_l & A_0^T \\ -A_0 & -I_2 \end{bmatrix}$$

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

and \bar{D} is invertible, $I + \bar{V}^T S^{-1} \bar{U}$ will be invertible if and only if G is invertible.

By construction, $(D_0)^{-2}$ and $A_0^T A_0$ are positive definite and symmetric. Since S_l is positive definite for all $l = 1, \dots, N$, $T_l^T S_l^{-1} T_l$ is positive definite and symmetric. The sum of positive definite matrices is again positive definite, so G_1 is positive definite and symmetric, and hence invertible. The symmetric matrix G_1^{-1} has

rank n_0 , 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. Also, the rank of G is $m_0 + n_0$, so G is invertible.

Applying Lemma 1, M is invertible and

$$\begin{aligned}
 M^{-1} &= (S + \bar{U} \bar{V}^T)^{-1} \\
 &= S^{-1} - S^{-1} U \bar{D} (I + \bar{D}^T V^T S^{-1} U \bar{D})^{-1} \bar{D} V^T S^{-1} \\
 &= S^{-1} - S^{-1} U \bar{D} \bar{D}^{-1} (\bar{D}^{-2} + V^T S^{-1} U)^{-1} \bar{D}^{-1} \bar{D} V^T S^{-1} \\
 &= S^{-1} - S^{-1} U (\bar{D}^{-2} + V^T S^{-1} U)^{-1} V^T S^{-1} \\
 &= S^{-1} - S^{-1} U G^{-1} V^T S^{-1}.
 \end{aligned}$$

□

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)

$$Sp = b, \quad Gq = V^T p, \quad Sr = Uq \quad (15)$$

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

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

Further simplification of the second equation of (15) 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 $q_1 \in \mathbb{R}^{m_0}$ and $q_2 \in \mathbb{R}^{n_0}$. Then solving

$$\begin{bmatrix} G_1 & A_0^T \\ -A_0 & 0 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} \hat{p}_1 \\ \hat{p}_2 \end{bmatrix} = \begin{bmatrix} A_0^T & T_1^T & \cdots & T_N^T \\ -I_2 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} p_0 \\ \vdots \\ p_N \end{bmatrix} \quad (16)$$

implies that

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

Solving (15) requires Cholesky factorizations of S_i, G_1 , and G_2 . At each iteration of the affine scaling algorithm, an update of S_i to reflect the current dual solution must precede the solution of $M dy = b$. Given the updated matrices

$(D_l)^2$ and subsequent updates of the Cholesky factorizations of S_l, dy may be found using the algorithm (using Birge and Qi's decomposition) described below.

Procedure finddy ($S, A_0, T_1 \dots T_N, b, dy$).
begin.

1. (Solve $S_p = b$). Solve $S_l p_l = b_l$ for $p_l, l = 0, \dots, N$.
2. (Solve $G_q = V^T p$).
 - (a) Form G_1 by solving $S_l(u_l)_i = (T_e)_i$ for $(u_l)_i, l = 0, \dots, N, i = 1, \dots, n_l$ and setting $(G_1)_{ii} = (D_0)_{ii} + \sum_{e=1}^N T_l^T(u_l)_i$.
 - (b) Form \hat{p}_1 and \hat{p}_2 using (16).
 - (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 $(G_1)w_i = (A_0^T)_i$ for w_i and setting $G_2 = -A_0[w_1 \dots w_m]$.
 - (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 $S_l r_l = T_l q_1$ for $r_l \in \mathbb{R}^{m_l}, l = 1, \dots, N$.
4. (Form dy). Set $(dy)_l = p_l - r_l$ for $l = 0, \dots, N$. Return $dy_x = (dy_0, \dots, dy_N)$.

end.

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

$$\begin{bmatrix} I & 0 \\ 0 & WD_1^2W^T \end{bmatrix} + \begin{bmatrix} A_0D_0^2A_0^T - I & A_0D_0^2T^T \\ TD_0^2A_0^T & TD_0^2T^T \end{bmatrix}. \quad (17)$$

Comparing (17) 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 $AD^2A^T = S + \bar{U}\bar{V}^T =$

$$\begin{bmatrix} I + QD_1^2Q^T & 0 \\ 0 & WD_1^2W^T \end{bmatrix} + \begin{bmatrix} A_0D_0^2A_0^T - I & A_0D_0^2T^T + QD_1^2W^T \\ TD_0^2A_0^T + WD_1^2Q^T & TD_0^2T^T \end{bmatrix}. \quad (18)$$

An implementation based on (18) 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 ([10]) 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.

The third advantage of this approach is the natural theoretical limit on the number of operations required to solve the search direction system. The computational complexity of the procedure is given in Birge and Qi ([10]) as $O(N(n_1^3 + n_1^2n_0 + n_0n_1^2 + n_1n_0^2))$. Although more effort is necessary to calculate the working matrices G_1 and G_2 than is required by the Schur complement, the overall complexity is linear in the number of scenarios. The Schur complement's arithmetic complexity, in contrast, is $O(N^3n_1^3)$.

Finally, since the factorizations of each S_i 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 simultaneous running time to solve for the search direction may be reduced from $O(N(n_1^3 + n_1^2n_0 + n_0n_1^2 + n_0^2n_1))$ as given by Birge and Qi to $O(n_1^3 + n_1^2n_0 + n_0n_1^2 + Nn_0^2)$, a reduction of roughly $\min\{N, n_1\}$. Finer-grain systems with many processors could achieve further reductions by performing 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 [10], 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, version 1) ([23]) 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 [21]. In the current study, we compare the BQ decomposition 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. Stability comparisons between the BQ

decomposition and the Schur complement are also given.

4.1. Problem characteristics

The test problems used in the current study are stochastic versions of a set of staircase test problems ([22]) as in [7]. 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. ([9]). Dual problems were generated from the 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 ξ) 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 Cholesky factorizations of the AD^2A^T matrices for the various formulations is also given in Table 1. Figure 2 graphically shows the number of Cholesky nonzeros for the SC205 and SCSD8 problems.

As can be seen from Table 1, the SCFXM1 and SCSD8 problems contained many more nonzeros in the dual Cholesky factorization than in the primal Cholesky factorization. This behaviour 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 characteristics of the deterministic equivalent problem is important before deciding which form

Table 1. Problem characteristics.

Problem	Scenarios	Rows	Columns ¹	A Matrix Nonzeros	Density (Percent)	AD^2A^T Cholesky Nonzeroes		
						Dual	Primal	Split
SC205.1	4	101	102	270	2.62%	367	496	623
SC205.2	8	189	190	510	1.42%	695	1,200	1,441
SC205.3	16	365	366	990	0.74%	1,343	3,454	3,323
SC205.4	32	717	718	1,950	0.38%	2,639	11,236	8,316
SC205.5	64	1,421	1,422	3,870	0.19%	5,231	39,882	18,928
SCAGR7.1	4	167	180	546	1.82%	1,164	1,191	1,129
SCAGR7.2	8	319	340	1,050	0.97%	2,248	3,443	2,546
SCAGR7.3	16	623	660	2,058	0.50%	4,416	11,123	5,417
SCAGR7.4	32	1,231	1,300	4,074	0.25%	8,752	39,155	11,593
SCAGR7.5	64	2,447	2,580	8,106	0.13%	17,424	145,907	25,743
SCFXM1.1	4	684	1,014	3,999	0.58%	28,564	7,669	9,585
SCFXM1.2	8	1,276	1,914	7,319	0.30%	56,111	14,997	18,449
SCFXM1.3	16	2,460	3,714	13,959	0.15%	125,689	31,819	38,952
SCFXM1.4	32	4,828	7,314	27,239	0.08%	249,402	75,641	90,220
SCRS8.1	4	140	189	457	1.73%	1,877	1,235	1,367
SCRS8.2	8	252	341	849	0.99%	5,195	4,771	3,550
SCRS8.3	16	476	645	1,633	0.53%	4,955	23,137	10,251
SCRS8.4	32	924	1,253	3,201	0.28%	9,563	88,732	29,337
SCRS8.5	64	1,820	2,469	6,337	0.14%	18,779	347,247	89,687
SCSD8.1	4	90	630	1,890	3.33%	31,677	1,433	2,673
SCSD8.2	8	170	1,190	3,650	1.80%	59,595	3,916	7,066
SCSD8.3	16	330	2,310	7,170	0.94%	117,113	12,112	19,892
SCSD8.4	32	650	4,550	14,210	0.48%	232,287	41,588	65,143
¹ Slack columns not attached.								

of the problem to solve. (We should note that Vanderbei and Carpenter ([36]) 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.)

Another statistic that 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 nonzeros 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

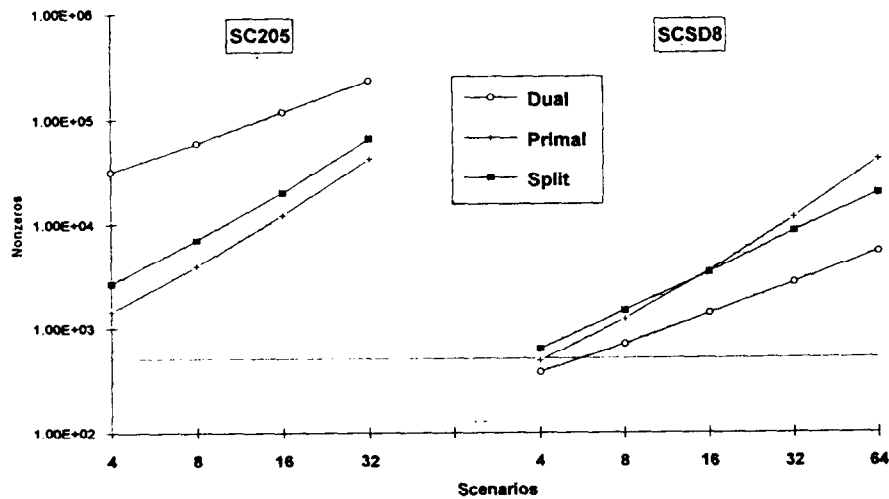


Figure 2. Cholesky nonzeros for sc205 scsd8.

Table 2. Block characteristics.

Problem	T Matrix		W matrix		Primal		Dual	Primal and Dual	
	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
SCFX1	92	114	149	225	2,729	12,996	168	5,937	3,450
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	1,732	9,900	1,732	4,880	1,420

high maximum row density, which is consistent with the small number of rows in its primal problem.

4.2. Computational test parameters and algorithms

The deterministic equivalent test problems, described in the previous section, were solved using OSL (subsequent improvements to OSL show substantially lower solution times than those quoted here) and an implementation of the BQ decomposition on an IBM RS/6000 320H workstation (system performance measures are given as: 41.2 SPECs, 37 IMIPs, and 11.7 MFLOPs) for the

Table 3. Row and column densities.

Problem	Row Densities ¹		Column Densities ²	
	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 nonzeros in a row divided by the number of rows. ² Defined as the number of nonzeros in a column divided by the number of columns.			

serial comparisons and a network of six DEC 5000/320 workstations for the distributed implementation of BQ. An implementation of the dual affine scaling algorithm using Schur complements was also developed for the RS/6000 for stability comparisons. The data structures used were extensions of those found in ([2]), while the recommended parameter values followed those in ([1]). The implementations were written in the FORTRAN programming language and compiled using the RS/6000 XLF 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_i 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. (The computational results obtained were insensitive to the heuristic type.) 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. Dense working matrices were solved using LINPACK ([17]).

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

$$\begin{array}{ll} \max & b^T y - M y_a \\ \text{subject to} & A^T y - e^T y_a \leq c. \end{array}$$

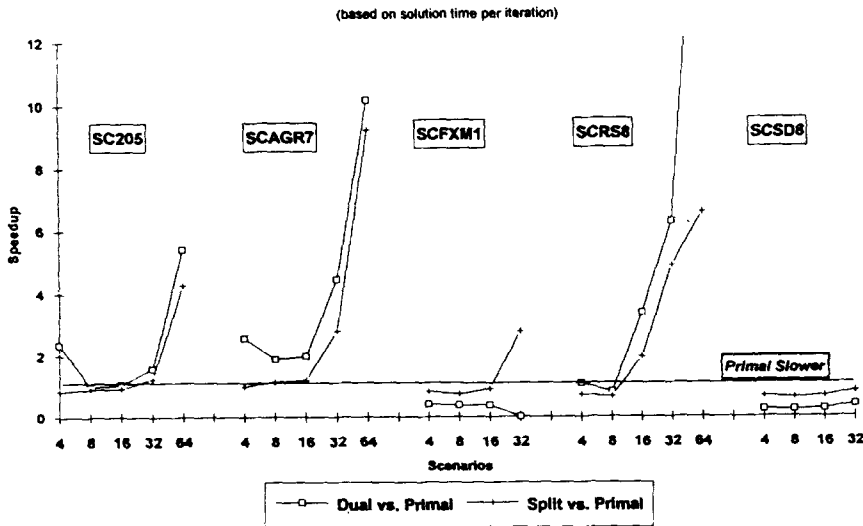


Figure 3. Speedups associated with solving the primal (based on solution time per iteration).

The artificial variable column was not stored explicitly, so its search direction component was computed symbolically. 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 $y_a^k < \epsilon_f$ ($\epsilon_f = 10^{-6}$, in the implementations used here) 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 $y^0 = (\|c\|/\|A^T b\|)b$.

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 $l = 0, \dots, N$. 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 that 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 (blocks were spread as evenly as possible across processors). At each iteration, the host program sends a set of right-hand side vectors y_{i_1}, \dots, y_{i_N} to each node program and collects the solutions $S_{i_1}^{-1} y_{i_1}, \dots, S_{i_N}^{-1} y_{i_N}$. 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.

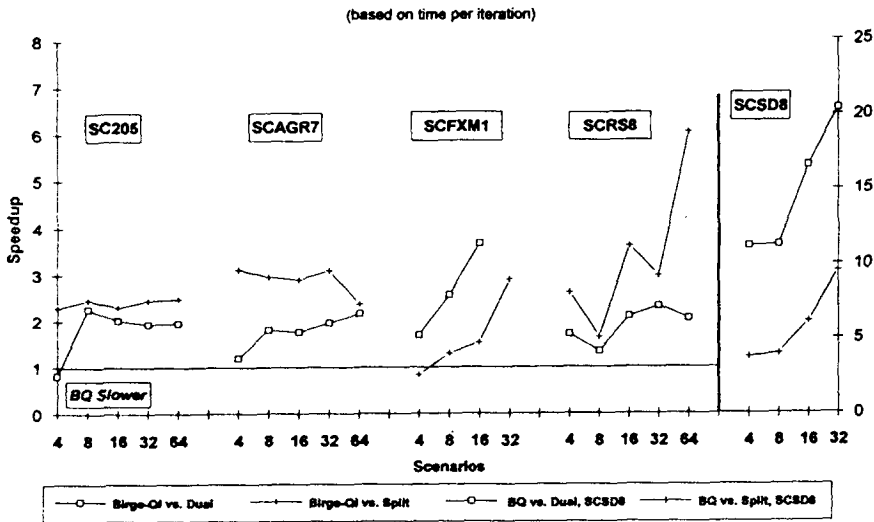


Fig. 4. Speedups associated with using the BQ decomposition (based on solution time per iteration).

Communications between the host program and each node program were managed by the Parallel Virtual Machine (PVM) package, developed by Beguelin et. al ([6]). PVM allows rapid 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 ([21]). Given a feasible interior point, the algorithm projects the steepest descent direction of a barrier problem onto the null space 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 $AD^2A^T dy = b$. 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. 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. These difficulties will be discussed in Section 4.4.

Table 4 shows the interior point solution times and the total solution times for the dual, primal and split formulations (as solved by OSL's primal barrier

Table 4. Solution times.

[illegible]

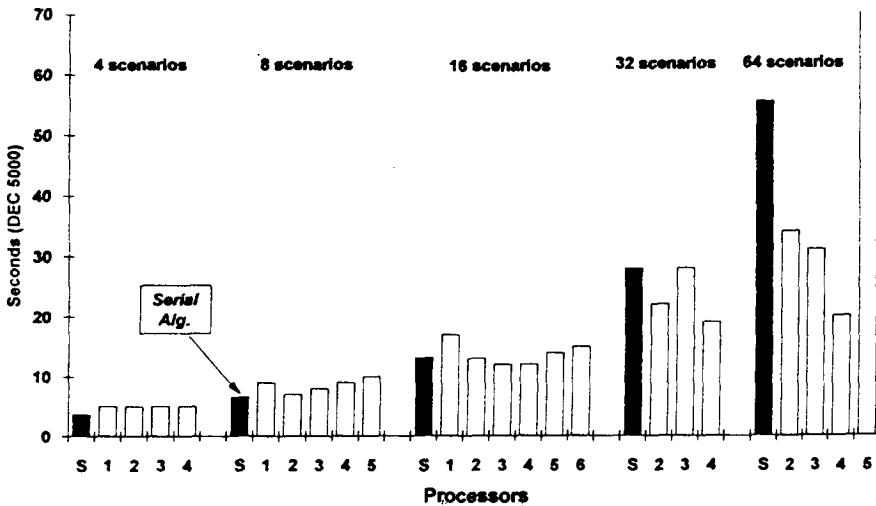


Figure 5. Distributed solution times.

method), 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.

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 $AD^2A^T dy = b$, more meaningful comparisons may be made by comparing the CPU time required per interior point iteration.

Table 5 (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 nondecomposition-based solution technique. As the number of scenarios in the deterministic equivalent increases, the BQ decomposition becomes particularly attractive.

The merits of each solution method are highlighted by their relative speedup factors. Table 6 gives these statistics for the stochastic test set. The absolute speedup factors shown in Table 6 (Figures 3 and 4) are the ratios between the solution times for each of the solution methods being compared. Likewise, the

Table 5. Solution time per iteration.

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

speedup factors based on time per iteration are the ratios between the average CPU time per iteration for each solution method.

In terms of absolute solution time, Table 6 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

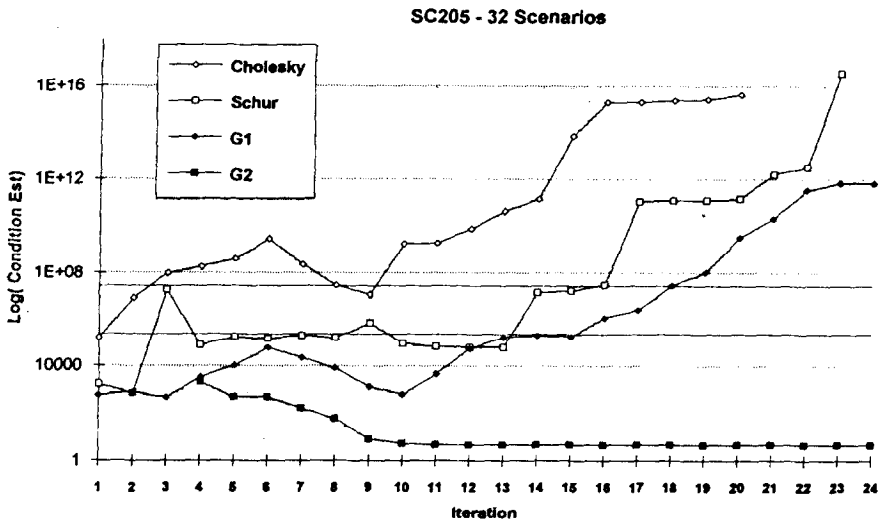


Figure 6. Condition estimates—Working matrices.

suggests that these performance gains may be limited for exceptionally large problems and that there may be a break-even 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 (standard deviation 0.59) times faster than solving with the dual factorization. These results confirm the claims made in ([10]). The problems SCSD8 and SCFXM1 offer higher speed-ups, 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.

As mentioned previously, the structure of the BQ decomposition lends itself to distributed processing. Table 7 (and Figure 5 in the appendix) give the best run times for the SC205 problem (run times were obtained using a network of six DEC 5000/320 workstations) 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. (Unfortunately, since the network connecting the processors was a shared resource, obtaining accurate solution times for larger numbers of processors was quite difficult. Fully utilizing the parallel implementation requires communications bandwidths unobtainable over a shared ethernet.) 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.

Table 6. Speedup comparisons.

Problem	Speedups based on time per iteration ¹			
	<u>Primal</u>	<u>Primal</u>	<u>Dual</u>	<u>Split</u>
	Dual	Split	BQ	BQ
SC205.1	2.37	0.85	0.82	2.29
SC205.2	0.99	0.91	2.26	2.46
SC205.3	1.06	0.93	2.02	2.30
SC205.4	1.57	1.24	1.93	2.45
SC205.5	5.43	4.27	1.95	2.48
SCAGR7.1	2.59	0.99	1.19	3.11
SCAGR7.2	1.88	1.14	1.80	2.96
SCAGR7.3	1.98	1.20	1.76	2.89
SCAGR7.4	4.44	2.81	1.95	3.09
SCAGR7.5	10.16	9.24	2.15	2.37
SCFXM1.1	0.40	0.83	1.69	0.83
SCFXM1.2	0.38	0.75	2.56	1.29
SCFXM1.3	0.38	0.91	3.66	1.53
SCFXM1.4	2.79	NA	NA	2.89
SCRS8.1	1.09	0.71	1.70	2.60
SCRS8.2	0.81	0.65	1.32	1.64
SCRS8.3	3.35	1.94	2.08	3.59
SCRS8.4	6.26	4.87	2.30	2.96
SCRS8.5	19.45	6.56	2.03	6.02
SCSD8.1	0.22	0.67	11.18	3.74
SCSD8.2	0.22	0.61	11.27	3.96
SCSD8.3	0.25	0.68	16.55	6.14
SCSD8.4	0.38	0.82	20.38	9.52
Notes:	¹ Ratio of solution times.			

Table 7. Speedup comparisons.

Problem	Scenarios	Solution Time (seconds) ¹						
		Algorithm	Number of Processors					
			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	— ²	34	31	20	93	— ²
Notes:	¹ Solution times are best encountered in at least 10 runs.							
	² Blank times unavailable due to system limitations.							

4.4. Stability issues

As mentioned previously, a major advantage of the BQ method over the Schur complement is its natural numeric stability. On the problems currently under study, the BQ decomposition proved to be much more stable than our implementation of the dual affine scaling algorithm using Schur complements.

The Schur complement suffered numeric instability from two sources. As mentioned previously, the sparse matrix $A_s D_s^2 A_s^T$ may be rank deficient and, hence, ill-conditioned. Rows with dense nonzeros that have no sparse nonzeros become "null" rows that reduce the rank of the system. Table 8 shows this behavior on two relatively easy (SC205 and SCAGR7) problems and one quite difficult problem (SCFXM1). The easy problems had a small number of basic variables in the optimal solution. Two approaches were used to solve the SCFXM1 problem. SCFXM1 has the interesting property that only eight of 92 rows of T contain nonzeros. The first approach placed all columns with nonzeros in both A_0 and T in A_d , while the second approach placed the densest 15% (maximum) of the A_0 columns in A_d . Neither approach worked well. In many cases, we were unable to get the Schur complement implementation to converge to the optimal solution.

The second source of instability arises when the Schur complement $I + V^T(LL^T)^{-1}V$ becomes unstable due to dual slack variables corresponding to basic dense columns approaching zero. In contrast, the main working matrix of the BQ decomposition (G_1) was more stable, but tended to become more unstable as the algorithm progressed. Condition estimates for each iteration of a typical problem are shown in Figure 6. Since G_1 is defined as $G_1 = D_0^2 + A_0^T A_0 + \sum_{i=1}^L T_i^T S_i^{-1} T_i$, it should tend to reduce the effect of instabilities in any given component, until the diagonal entries of D_0^2 dominate the matrix. Table 8 shows this effect for

the SC205 problem. If there are many columns in $(A_0^T, T^T)^T$ basic in the optimal solution, G_1 may become unacceptably unstable. Table 9 summarizes these characteristics for the problems in Table 8.

Table 9 suggests that the number of dense basic columns is the most important factor that affects the stability of the dual affine scaling algorithm, regardless of the factorization used to solve the search direction system $AD^2A^T dy = b$. Table 9 also suggests that an excessive number of null rows for large problems adversely affects the stability of the Schur complement procedure.

Table 8. Condition estimate comparisons.

Problem	Log of $A_0 D_0^2 A_0^T$ Cholesky Condition Estimate After				Log of G_1 Condition Estimate After				Average Log of S_i Condition Estimate After			
	5 its	10 its	15 its	20 its	5 its	10 its	15 its	20 its	5 its	10 its	15 its	20 its
SC205.1	7.96	10.65	15.07	- ²	2.53	4.81	9.15	13.98	12.43	10.23	5.76	- ¹
SC205.2	4.31	6.09	13.41	- ²	3.00	3.20	5.75	11.68	12.90	12.42	8.84	4.99
SC205.3	7.65	8.22	14.46	- ²	3.49	2.88	5.60	9.97	12.92	12.50	8.28	4.41
SC205.4	8.59	9.28	13.84	15.64	3.99	2.80	5.24	9.54	12.95	12.60	9.38	6.10
SC205.5	9.61	8.04	15.72	- ²	4.26	3.06	5.53	8.08	12.81	12.73	10.33	6.45
SCAGR7.1	6.65	6.27	9.69	13.86	3.52	6.02	8.31	12.51	9.28	10.06	10.24	9.66
SCAGR7.2	7.13	7.01	9.43	12.86	3.16	6.04	6.73	10.85	8.61	9.22	10.43	10.34
SCAGR7.3	7.67	8.69	- ¹	- ¹	3.75	6.29	6.65	9.98	8.57	8.26	9.50	10.71
SCFXM1 run with columns in A_0 and T removed.												
SCFXM1.1	14.35	14.94	- ²	-	5.05	6.79	- ²	-	4.70	4.70	-	-
SCFXM1.2	15.80	- ²	- ²	-	5.37	6.93	- ²	-	4.65	4.65	-	-
SCFXM1.3	Could not run				5.72	6.93	- ²	-	4.65	4.65	-	-
SCFXM1.4	Could not run				6.03	6.78	- ²	-	4.65	2.24	-	-
SCFXM1 run with all dense columns removed.												
SCFXM1.1	11.08	15.15	- ²	-	5.05	6.79	- ²	-	4.70	4.70	-	-
SCFXM1.2	11.39	14.79	- ²	-	5.37	6.93	- ²	-	4.65	4.65	-	-
	¹ Matrix declared effectively singular by SPARSPAK/LINPACK.											
	² LINPACK/SPARSPAK unable to estimate condition number.											

Overall, the BQ method appears generally more stable for these dual block angular programs than the Schur complement decomposition. Other factorizations of this type that take even finer structural details into account might improve this further.

5. Conclusions

This paper reviewed the need for and characteristics of solving (discrete) stochastic

Table 9. Stability measures.

Problem	Dense Basic		First-Stage
	Null	Columns	Basic Columns
	Rows	(Dense Columns)	(First-Stage Columns)
sc205.1	1	6(8)	11(14)
sc205.2	1	6(8)	11(14)
sc205.3	1	5(8)	9(14)
sc205.4	1	5(8)	9(14)
sc205.5	1	5(8)	9(14)
SCAGR7.1	25	3(12)	9(20)
SCAGR7.2	52	5(12)	14(20)
SCAGR7.3	20	5(12)	14(20)
SCFXM1.1	0	30(57)	70(114)
SCFXM1.2	0	30(57)	70(114)
SCFXM1.3	0	30(57)	70(114)
SCFXM1.4	0	30(57)	70(114)

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 that 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 nondense 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, and is also numerically stable. 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 nondecomposition-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.

Acknowledgments

This material is based upon work supported by the National Science Foundation under award No. EECS-8815101.

References

- [1] I. Adler, N. Karmarkar, M.G.C. Resende, and G. Veiga, "An implementation of Karmarkar's algorithm for linear programming," *Math. Programming*, 44, 297–335, 1989. Errata in *Math. Programming*, vol. 50, p. 415, 1991.
- [2] I. Adler, N. Karmarkar, M.G.C. Resende, and G. Veiga, "Data structures and programming techniques for the implementation of Karmarkar's algorithm," *ORSA J. on Comput.*, vol. 1, pp. 84–106, 1989.
- [3] J. Arantes and J. Birge, "Matrix structure and interior point methods in stochastic programming," Presentation at *Fifth Int. Stochastic Programming Conf.*, Ann Arbor, MI, 1989.
- [4] E. R. Barnes, "A variation on Karmarkar's algorithm for solving linear programming problems," *Math. Programming*, vol. 36, pp. 174–184, 1986.
- [5] E.M.L. Beale, "On minimizing a convex function subject to linear inequalities," *J. Royal Stat. Soc.* vol. B, 17, pp. 173–184, 1955.
- [6] A. Beguelin, J. Dongarra, A. Geist, R. Manchek, and V. Sunderam, "A user's guide to PVM: Parallel virtual machine," Report ORNL/TM-11826, Oak Ridge Nat. Lab., Dept. of Energy, Oak Ridge, TN, 1991.
- [7] J. R. Birge, "Decomposition and partitioning methods for multistage stochastic linear programs," *Oper. Res.*, vol. 33, pp. 989–1007, 1985.
- [8] J. R. Birge, R. M. Freund, and R. J. Vanderbei, "Prior reduced fill-in in the solution of equations in interior point algorithms," Tech. Report, Sloan School of Management, M.I.T. Cambridge, MA, 1990.
- [9] J. R. Birge, H. I. Gassman, and D. Holmes, STOPGEN stochastic linear program deterministic equivalent problem generator, 1991.
- [10] J. R. Birge and L. Qi, "Computing block-angular Karmarkar projections with applications to stochastic programming," *Mgt. Sci.*, 34:12, 1472–1479, 1988.
- [11] W.J. Carolan, J.E. Hill, J.L. Kennington, S. Niemi, and S.J. Wichmann, "An empirical evaluation of the KORBX algorithms for military airlift applications," *Oper. Res.* vol. 9:2, pp. 169–184, 1990.
- [12] I.C. Choi, C.L. Monma, and D.F. Shanno, "Further development of a primal-dual interior point method," Report 60-88, Rutgers Ctr. for Oper. Res., Rutgers Univ., New Brunswick, NJ, 1988.
- [13] E. Chu, A. George, J. Liu, and E. Ng, "SPARSPAK: Waterloo sparse matrix package user's guide for SPARSPAK-A," Res. Report CS-84-36, Dept. of Comput. Sci., Univ. of Waterloo, Waterloo, Ontario, 1984.
- [14] G. Dantzig, "Linear programming under uncertainty," *Mgt. Sci.*, vol. 1, pp. 197–206, 1955.
- [15] G. Dantzig and A. Madansky, "On the solution of two stage linear programs under uncertainty," *Proc. of the Fourth Berkely Symp. on Math. and Probability Vol. 1*, Univ. of California Press, Berkely, CA, pp. 165–176, 1961.

- [16] I.I. Dikin, "Iterative solution of problems of linear and quadratic programming," *Soviet Math. Doklady*, vol. 8, pp. 674-675, 1967.
- [17] J. Dongarra, J.R. Bunch, C.B. Moler, and G. W. Stewart, *LINPACK Users Guide*, SIAM Publications, Philadelphia, PA, 1978.
- [18] S.C. Eisenstadt, M.C. Gurshy, M.H. Shultz, and A.H. Sherman, "The Yale sparse matrix package, I. The symmetric codes," *ACM Trans. on Math. Software*, 1981.
- [19] H.I. Gassman, "Optimal harvest of a forest in the presence of uncertainty," *Can. J. forest Res.*, vol. 19, pp. 1267-1274, 1989.
- [20] D.M. Gay, "Electronic mail distribution linear programming test problems," *Math. Programming Soc. COAL Newsletter*, December, 1985.
- [21] P.E. Gill, W. Murray, M.A. Saunders, J.A. Tomlin, and M.H. Wright, "On projected Newton methods for linear programming and equivalence to Karmarkar's projection method," *Math. Programming*, vol. 36, pp. 183-201, 1986.
- [22] J.K. Ho and E. Loute, "A set of linear programming test problems," *Math. Programming*, vol. 20, pp. 245-250, 1981.
- [23] Int. Business Machines Corp. (IBM) "Optimization subroutine library guide and reference," document SC23-0519-01, 1991.
- [24] N. Karmarkar, "A new polynomial time algorithm for linear programming," *Combinatorica*, vol. 4, pp. 373-395, 1984.
- [25] A.J. King, "Stochastic programming problems: Examples from the literature," **Numerical Techniques for Stochastic Optimization**, Yu. Ermoliev and R.J.-B. Wets, (eds), 543-567, 1988.
- [26] F.V. Louveau, "A solution method for multistage stochastic programs with recourse with application to an energy investment problem," *Oper. Res.* vol. 28, pp. 889-902, 1980.
- [27] I. Lustig, R. Marsten, and D. Shanno, "Computational Experience with a primal-dual interior point method for linear programming," *Linear Algebra and its Applications*, vol. 152, pp. 191-222, 1991. (also Technical Report SOR 89-17, Dept. of Civ. Engg. and Oper. Res., Princeton Univ., Princeton, NJ).
- [28] I. Lustig, J. Mulvey, and T. Carpenter, "Formulating stochastic programs for interior point methods," *Oper. Res.* vol. 39:5, pp. 757-770, 1991.
- [29] R. Marsten, R. Subramanian, M. Saltzman, I. Lustig, and D. Shanno, "Interior point methods for linear programming: Just call Newton, Lagrange, and Fiocco and McCormick!" *Interfaces*, vol. 20:4, pp. 105-116, 1990.
- [30] C.L. Monma and A.J. Morton, "Computational experience with a dual affine variant of Karmarkar's method for linear programming," *Oper. Res. Letters*, vol. 6, pp. 261-267, 1987.
- [31] J.M. Mulvey, "A network portfolio approach for cash management," *J. Cash Mgt.* vol. 4, pp. 46-48, 1984.
- [32] J.M. Mulvey and H. Vladimirov, "Stochastic network optimization models for investment planning," B. Shelby, ed., to appear, *Ann. Oper. Res.*, special issue on Networks and Applications, 1989.
- [33] B.A. Murtaugh and M.A. Saunders, "MINOS 5.1 user's guide," Tech. Report SOL 83-20R, Systems Optimization Lab., Stanford Univ., Stanford, CA, 1983.
- [34] D.F. Shanno, and A. Bagchi, "A unified view of interior point methods for linear programming," *Ann. of Oper. Res.*, vol. 22, pp. 55-70, 1990.
- [35] R. Van Slyke and R. Wets, "L-shaped linear programs with applications to optimal control and stochastic programming," *SIAM J. Appl. Math.*, vol. 17, pp. 638-663, 1969.
- [36] R.J. Vanderbei and T.J. Carpenter, "Symmetric indefinite systems for interior point methods." Tech. Report SOR 91-7, Dept. of Civ. Engg. and Oper. Res., Princeton Univ., Princeton, NJ, 1991.
- [37] R.J. Vanderbei, M.S. Meketon, and B.A. Freedman, "A modification of Karmarkar's linear programming algorithm," *Algorithmica*, vol. 1, pp. 395-407, 1986.