ALPO: Another Linear Program Optimizer

Robert J. Vanderbei

Program in Statistics & Operations Research

Princeton University

Princeton, NJ 08544

BITNET: rvdb@princeton.edu

October 9, 1992

Subject Classification: Linear programming algorithms

Abstract

This paper describes an implementation of the one-phase primal-dual path-following algorithm for solving linear programming problems. The design is intended to be simple, portable and robust. These design goals are achieved without sacrificing state-of-the-art performance. We give a rather complete description of the algorithm and the implementation. Computational results obtained for the NETLIB suite of problems run on a Silicon Graphics workstation are also presented.

This paper documents a new linear program solver called ALPO. ALPO is written entirely in C and can easily be ported to a variety of computing environments. ALPO is intended to be simple, portable and robust. It also turns out to be efficient (see Section 3).

ALPO is easy to use. Other than specifying a problem and the level of precision desired in the answer, there are no user adjustable parameters. ALPO can be run as a stand-alone program (called `alpo`), or it can be accessed from either C or FORTRAN as a small library of functions. If it is executed from the command line, it takes its input from industry-standard MPS format files.

A function called `optlp` is the heart of the system. It uses the one-phase primal-dual path-following algorithm to solve problems presented in standard form (all MPS format problems are first converted to standard form). This algorithm is one of the recently discovered interior point methods for linear programming that has appeared as a result of the intense research that has ensued from the publication of N. Karmarkar's seminal paper [12]. As its basic engine, `optlp` uses Cholesky factorization routines (with a minimum-local-fill ordering heuristic) for solving symmetric nonnegative definite systems of equations.

Whenever there was a choice between using a simple algorithm or a sophisticated one, we have opted for the simple algorithm. It is our belief that the performance penalty for such a compromise is generally modest (see Section 3). As a result the entire code, consisting of less than 1400 non-comment lines (about 2000 lines in total), is quite short and understandable. By way of comparison, just the data input routine in KORBX (`kmps`) has about 5000 lines.

Since we are currently seeing so many rapid advances in our understanding of efficient interior point methods for mathematical programming, it would be very useful to have a platform from which different methods and new advances could be studied easily. ALPO is well suited for this purpose. As an example, ALPO was recently modified to solve separable quadratic programming problems. This modification took less than a day to design, implement and debug.

The one-phase primal-dual path-following algorithm is described in the next section.

Section 2 gives a brief account of some of the implementation details. Finally, Section 3 presents the performance of ALPO on the standard NETLIB suite.

1 The Algorithm

The algorithm implemented in ALPO is the one-phase primal-dual path-following algorithm more or less as described in [6], [17] or [15]. This algorithm appeared as a result of Karmarkar's discovery of an efficient polynomial time algorithm for linear programming (see [12], [19], [4] and [13]). For completeness, we include in this section a description of this algorithm. The algorithm operates on linear programming problems presented in standard form:

$$\text{minimize } c^T x$$

$$Ax = b$$

$$x \geq 0.$$

This linear programming problem is called the *primal problem*. The set of points $\{x : Ax = b,\ x \geq 0\}$ is called the *primal polytope*. If this set is nonempty, the primal problem is said to be *feasible*. Points x in the primal polytope are called *primal feasible* points. The $m \times n$-matrix A is called the *constraint matrix* and the function $c^T x$ is called the *objective function*.

Associated with this primal problem is its dual:

$$\text{maximize } b^T y$$

$$A^T y \leq c.$$

It is often convenient to write the dual in equality form which we do by introducing slack variables r (also called *reduced costs*):

$$\text{maximize } b^T y$$

$$A^T y + r = c$$

$$r \geq 0.$$

The fact that the dual is related to the primal manifests itself in the *Weak Duality Theorem* (see, e.g. [7]):

Theorem 1 . *If x is feasible for the primal and y is feasible for the dual, then $b^T y \leq c^T x$. If equality holds then x is optimal for the primal and y is optimal for the dual.*

The difference $c^T x - b^T y$ is called the *duality gap*.

1.1 The central path

Fix $\mu > 0$. Let (x_μ, y_μ, r_μ) be a solution to the following system of $2n + m$ (nonlinear) equations in $2n + m$ unknowns:

$$\begin{aligned} Ax &= b && (1.1)\\ A^T y + r &= c && (1.2)\\ XRe &= \mu e, && (1.3) \end{aligned}$$

where e denotes the n-vector of all ones, and the capital letters X and R denote the diagonal matrices with x and r, respectively, on the diagonal. The first m equations (1.1) are part of the primal feasibility requirement and the next n equations (1.2) are part of the dual feasibility requirement. The last n equations are related to the duality gap. Indeed, assuming that (x, y, z) satisfy (1.1), (1.2) and (1.3), it is easy to see that

$$\begin{aligned} \mu n &= r^T x && (1.4)\\ &= c^T x - b^T y. \end{aligned}$$

Proposition 2 . *If either the primal or the dual polytope is bounded and has nonempty interior, then there exists a unique solution to (1.1), (1.2), (1.3) in the domain given by $x \geq 0$, $r \geq 0$.*

This result is well-known (see e.g. [5]) but we include a short proof here since it shows the connection between the central trajectory and an associated barrier problem.

Proof. Suppose that the primal polytope is bounded and has nonempty interior (the proof for the other case is similar and omitted). Consider the following logarithmic-barrier problem associated with the original primal problem:

$$\text{minimize } c^T x - \mu \sum_j \ln x_j$$

$$Ax = b$$

$$x \geq 0.$$

For any $\mu > 0$, this objective function is strictly convex and is infinite at the boundary of the polytope. Since the primal polytope is bounded, the objective function must attain its minimum. The strict convexity implies that the solution is unique. This unique solution is characterized as the unique feasible point for which the gradient of the objective function lies in the row space of A:

$$c - \mu X^{-1} e = A^T y$$

where y represents the coefficients of the linear combination of the rows of A used to represent the gradient. Introducing the notation $r = \mu X^{-1} e$, it is easy to see that these conditions are equivalent to (1.1), (1.2), (1.3). □

The map $\mu \mapsto (x_\mu, y_\mu, r_\mu)$ is called the *central path*. Let (x^*, y^*, r^*) denote a limit point of the central path as μ tends to zero. It follows immediately from (1.1), (1.2) and (1.4) that x^* is primal feasible, (y^*, r^*) is dual feasible and that the duality gap is zero. Hence, x^* is optimal for the primal problem and (y^*, r^*) is optimal for the dual.

Assume for a moment that the primal polytope is bounded and has nonempty interior. The *analytic center* of the primal polytope is defined to be the unique solution to the following problem:

$$\text{minimize } -\sum_j \ln x_j$$

$$Ax = b$$

$$x \geq 0.$$

It is easy to see that as μ tends to infinity, x_μ tends toward the analytic center of the primal polytope and from (1.4) that (y_μ, r_μ) tends to infinity. An analogous statement holds in the case where the dual polytope is bounded and has nonempty interior. (From (1.4), we see that it is impossible for both polytopes to be bounded and have nonempty interiors.)

The primal-dual path-following algorithm can now be described quite simply. We start with any triple (x, y, r) satisfying $x > 0$ and $r > 0$ and with any $\mu > 0$. We next use one step of Newton's method to try to find a point closer to (x_μ, y_μ, r_μ). We then let this new point be our current (x, y, r), we reduce μ appropriately and we start over. This iterative process is continued until primal and dual feasibility is attained and the duality gap is smaller than some predetermined tolerance.

All that remains is to investigate the exact form of the equations for Newton's method and to describe how we reduce μ appropriately. The choice of μ is quite simple. From (1.4) we see that

$$\mu = \frac{r^T x}{n}$$

gives us a measure of the "μ" value for the current point. It is a good idea to aim for a point on the central path whose μ value is substantially less than this. Hence, we choose

$$\mu = 0.1 \, \frac{r^T x}{n}$$

(unless the primal objective value is less than the dual objective value in which case we boost μ up to $\mu = 2 \, r^T x / n$).

1.2 Newton's method

The iterative method we use to home in on (x_μ, y_μ, r_μ) is Newton's method applied to (1.1), (1.2), (1.3). Newton's method is just the freshman-calculus technique for finding a root of a nonlinear equation by approximating it with a succession of linear equations. The only difference is that now the problem is not one-dimensional. Indeed, suppose we want to find z^* which satisfies $F(z^*) = 0$ where $F(z)$ is a map from $\mathcal{R}^d$ to $\mathcal{R}^d$. Using the multidimensional

Taylor's series expansion, we see that

$$F(z + \Delta z) \approx F(z) + \partial F(z)\Delta z, \tag{1.5}$$

where $\partial F(z)$ denotes the matrix whose $(i,j)^{\text{th}}$ entry is

$$[\partial F(z)]_{i,j} = \frac{\partial F_i}{\partial z_j}(z).$$

Hence, to find an approximate solution to $F(z) = 0$, simply set the right-hand side of (1.5) to zero. This tells us that Δz should be the solution to the following system of linear equations

$$\partial F(z)\Delta z = -F(z)$$

and that z should be updated to $z + \Delta z$. This is *Newton's method*. Note that if $F(z)$ is linear, Newton's method finds the root in one step. However, if $F(z)$ is nonlinear, then one must iterate the above procedure and hope (or prove) that it homes in on a solution.

Applying Newton's method to (1.1), (1.2), (1.3), we get

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ R & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta r \end{bmatrix} = - \begin{bmatrix} Ax - b \\ A^T y + r - c \\ XRe - \mu e \end{bmatrix}.$$

Writing this out, we obtain

$$A\Delta x = \rho \tag{1.6}$$

$$A^T \Delta y + \Delta r = \sigma \tag{1.7}$$

$$R\Delta x + X\Delta r = \phi, \tag{1.8}$$

where

$$\rho = b - Ax,$$

$$\sigma = c - A^T y - r,$$

and

$$\phi = \mu e - XRe.$$

If we multiply (1.7) by AXR^{-1} and then use (1.8) followed by (1.6), we see that

$$\Delta y = (AD^2A^T)^{-1}(AD^2(\sigma - \gamma) + \rho), \tag{1.9}$$

where D is the positive definite diagonal matrix satisfying

$$D^2 = XR^{-1},$$

and

$$\gamma = X^{-1}\phi = \mu X^{-1}e - Re.$$

Once Δy is known, it is easy to solve for Δr and Δx:

$$\Delta r = \sigma - A^T\Delta y \tag{1.10}$$

and

$$\Delta x = D^2(\gamma - \Delta r). \tag{1.11}$$

The desired update is then

$$x \leftarrow x + \Delta x \tag{1.12}$$

$$y \leftarrow y + \Delta y \tag{1.13}$$

$$r \leftarrow r + \Delta r. \tag{1.14}$$

However, there is no guarantee that this update will preserve the nonnegativity of x and r, so a shorter step is taken as necessary to keep $x > 0$ and $r > 0$.

1.3 Step Direction Decomposition

Substituting (1.10) and then (1.9) into (1.11), we get the following formula for Δx:

$$\Delta x = \mu DPDX^{-1}e - DPDc + D^2A^T(AD^2A^T)^{-1}\rho \tag{1.15}$$

where

$$P = I - DA^T(AD^2A^T)^{-1}AD$$

is the matrix that projects vectors orthogonally onto the null space of AD. The three terms in (1.15) are called the *centering step direction*, the *objective step direction*, and the *feasibility step direction*, respectively. The centering step direction Δx^{ctr} and the objective step direction Δx^{obj} lie in the null space of A (i.e., $A\Delta x^{ctr} = 0$ and $A\Delta x^{obj} = 0$) and hence do not change the degree of infeasibility of the current point. Therefore, it is only the feasibility step direction that works toward finding a feasible primal solution. The centering step direction repels the solution away from walls of the primal polytope. This helps prevent the algorithm from "jamming". The objective step direction points in the direction of steepest descent of the objective function (after a scaling change of coordinates).

Initially, μ and ρ are large and the algorithm concentrates on finding a feasible point near the center of the primal polytope. When a feasible point is found, ρ becomes essentially zero and the feasibility step becomes insignificant (except to correct for small infeasibilities that creep in due to finite precision arithmetic). As the algorithm iterates through a sequence of feasible points, μ is progressively decreased and so eventually the centering step direction is dwarfed by the objective step direction. Once this happens, it only takes a few iterations to home in on an optimal solution.

A similar analysis can be carried out for the dual step directions Δy and Δr. Indeed, it is not hard to see that Δr can be written as the sum of three terms:

$$\Delta r = \mu A^T (AD^2A^T)^{-1} AR^{-1}e - A^T(AD^2A^T)^{-1}b + D^{-1}PD\sigma.$$

Again, the first term is a centering step direction, the second term is a objective step direction and the third term is a feasibility step direction.

1.4 Step Lengths

Instead of the update shown in (1.12), (1.13), (1.14), the general update is

$$x \leftarrow x + \frac{1}{\alpha_p}\Delta x \tag{1.16}$$

$$y \leftarrow y + \frac{1}{\alpha_d}\Delta y \tag{1.17}$$

$$r \leftarrow r + \frac{1}{\alpha_d}\Delta r, \tag{1.18}$$

where α_p is given by

$$\alpha_p = \max\{1, -\frac{\Delta x_0}{0.95x_0}, \ldots, -\frac{\Delta x_{n-1}}{0.95x_{n-1}}\}$$

and α_d is

$$\alpha_d = \max\{1, -\frac{\Delta r_0}{0.95r_0}, \ldots, -\frac{\Delta r_{n-1}}{0.95r_{n-1}}\}.$$

(Note that we index n-vectors from 0 to $n-1$ to be consistent with typical C usage instead of indexing from 1 to n.) This choice of α_p and α_d guarantees that x and r remain strictly positive.

1.5 Upper Bounds

A standard form linear programming problem having upper bounds is an LP presented in the following form:

$$\text{minimize } c^T x$$

$$Ax = b$$

$$0 \leq x \leq u,$$

where u is a vector of *upper bounds*. It is easy to convert an LP in this form to the usual standard form by introducing slack variables for the difference between the upper bounds vector and the current point. However, it is computationally more efficient to handle upper bounds implicitly. Indeed, there are now $4n + m$ equations defining the central path:

$$\begin{aligned} Ax &= b \\ x + t &= u \\ A^T y - s + r &= c \\ XRe &= \mu e \\ TSe &= \mu e. \end{aligned}$$

Applying Newton's method to this system, we get

$$\begin{aligned} A\Delta x &= \rho \\ \Delta x + \Delta t &= \tau \\ A^T\Delta y - \Delta s + \Delta r &= \sigma \\ R\Delta x + X\Delta r &= \phi \\ S\Delta t + T\Delta s &= \psi, \end{aligned}$$

where

$$\rho = b - Ax,$$

$$\tau = u - x - t,$$

$$\sigma = c - A^T y + s - r,$$

$$\phi = \mu e - XRe,$$

and

$$\psi = \mu e - TSe.$$

Solving this system, we get

$$\Delta y = (AD^2A^T)^{-1}[AD^2(\sigma - \gamma + \delta - ST^{-1}\tau) + \rho],$$

$$\Delta x = D^2(-\sigma + \gamma - \delta + ST^{-1}\tau + A^T\Delta y),$$

$$\Delta t = \tau - \Delta x,$$

$$\Delta r = \gamma - RX^{-1}\Delta x,$$

$$\Delta s = \delta - ST^{-1}\Delta t,$$

where

$$D^2 = (RX^{-1} + ST^{-1})^{-1} = (RT + SX)^{-1}XT,$$

$$\gamma = X^{-1}\phi = \mu X^{-1}e - Re,$$

$$\delta = T^{-1}\psi = \mu T^{-1}e - Se.$$

Although there are now more variables to keep track of and hence more computational overhead, when a problem has lots of upper bounds it is more efficient to use these implicit formulas rather than handling the upper bounds explicitly (see Section 2). If only a subset of the variables have finite upper bounds, then one needs only add variables t and s corresponding to those variables that have finite upper bounds. In this way, problems which have no finite upper bounds incur hardly any extra overhead with this technique.

Formulas for implicit handling of upper bounds can be found in several papers (see e.g. [11], [20] and [15]). The formulas given in [15] are special cases of those given here obtained by setting ρ, σ, and τ to zero (since their formulas assumed the current solution is primal/dual feasible).

2 Implementation

In the last few years, a number of papers have been appeared containing suggestions (some quite sophisticated) regarding techniques for developing efficient implementations (see, e.g., [2], [1], [14], [15], [17], [6], [10], [16], [18], [3]). It is interesting to note that some of these sophisticated techniques buy very little in terms of performance (see, e.g., [9]). Other than exploiting the standard sparse matrix data structures, ALPO uses no fancy tricks. Some of the features not found in ALPO include:

- Scaling the data.
- Matrix reduction techniques such as eliminating null and fixed variables.
- Separate handling of dense columns.
- Heuristics to set variables to their bounds.
- Acceleration techniques for the Cholesky factorization subroutines such as storing address lists.
- Preconditioned conjugate gradient technique for solving systems of equations.

```
while ( iter <= MAXIT ) {

        /*--------------------------------------------+
        | calculate residuals rho   = b - Ax          |
        |        and          sigma = c - At y - r    */

        smx(m,n,a,ka,ia,x,rho);
        for ( i=0; i<m; i++ ) rho[i] = b[i] - rho[i];
        smx(n,m,at,kat,iat,y,sigma);
        for ( j=0; j<n; j++ ) sigma[j] = c[j] - sigma[j] - r[j];

        /*--------------------------------------------+
        | calculate mu, gamma = mu X^-1 e - R e, and  |
        | diagonal matrix D^2 = X R^-1                */

        mu = 0.1*dotprod(r,x,n)/n; if (cxpf < ybpf) mu = 20.0*mu;
        for ( j=0; j<n; j++ ) gamma[j] = mu/x[j] - r[j];
        for ( j=0; j<n; j++ )    d2[j] = x[j]/r[j];

        /*--------------------------------------------+
        |               2 t -1                       |
        | calculate  (AD A )                         |
        |                                            */

        for (i=0; i<m; i++) mark[i] = TRUE;
        inv_num(m, n, ka, ia, a, d2, kat, iat, at, rot,
                  mark, iwork, v);
```

Figure 1: Code Fragment Showing First Half of the Primal-Dual Path-Following Algorithm

Assuming that all the data structures have been properly set up, the code that performs the primal-dual path-following algorithm is quite simple (see Figures 1 and 2). The code shown in Figures 1 and 2 is essentially self-explanatory. It depends on three important subroutines. The subroutine `smx` multiplies a sparse matrix (either A or A^T) times a vector. The routine `inv_num` performs a Cholesky factorization of AD^2A^T so that subsequent calls to `solve` can be used to solve systems of equations involving AD^2A^T. The code fragment shown above does not have a stopping rule and will run the full `MAXIT` iterations. Other than incorporating a stopping rule, checking for unbounded or infeasible problems, printing

```
        /*-------------------------------------------+
        | step toward x_mu, y_mu, r_mu.              */

        for (j=0; j<n; j++) dwork[j] = d2[j]*(sigma[j] - gamma[j]);
        smx(m,n,a,ka,ia,dwork,dwork2);
        for (i=0; i<m; i++) dwork2[i] = dwork2[i] + rho[i];
        solve(m,dwork2,dy,mark);
        smx(n,m,at,kat,iat,dy,dwork);
        alpha = 0.0e0;
        for (j=0; j<n; j++) {
                dr[j] = sigma[j] - dwork[j];
                if ( alpha < -dr[j]/r[j] ) alpha = -dr[j]/r[j];
        }
        alpha = alpha/0.95e0;
        if (alpha < 1.0e0) alpha = 1.0e0;
        for (j=0; j<n; j++) r[j] = r[j] + dr[j]/alpha;
        for (i=0; i<m; i++) y[i] = y[i] + dy[i]/alpha;
        alpha = 0.0e0;
        for (j=0; j<n; j++) {
                dx[j] = d2[j]*(gamma[j] - dr[j]);
                if ( alpha < -dx[j]/x[j] ) alpha = -dx[j]/x[j];
        }
        alpha = alpha/0.95e0;
        if (alpha < 1.0e0) alpha = 1.0e0;
        for (j=0; j<n; j++) x[j] = x[j] + dx[j]/alpha;

        /*-------------------------------------------+
        | wind up this iteration                     */

        iter = iter + 1;
}
```

Figure 2: Code Fragment Showing Second Half of the Primal-Dual Path-Following Algorithm

out an iteration log, and dealing with free variables and upper bounds, the code fragment shown in Figures 1 and 2 constitute the entire code for performing the algorithm.

2.1 Free Variables

A variable is called a free variable if its lower bound is minus infinity and its upper bound is plus infinity. A standard technique is to split each free variable into the difference between two nonnegative variables:

$$x_j = x_j^+ - x_j^-$$

where $x_j^+ \geq 0$ and $x_j^- \geq 0$. Lustig, Shanno and Marsten ([15]) report that this technique seems to work well on the NETLIB test problems. We also experimented with this technique in early implementations of ALPO. While it caused no problems on any of the NETLIB problems, it did create difficulties when the linear programming problem had a large number of free variables (which often happens, for example, when solving the dual as if it were the primal).

To see what is happening, let r_j^+ and r_j^- denote the reduced costs associated with x_j^+ and x_j^-, respectively. Since x_j is free, the j^{th} constraint in the dual problem is actually an equality constraint, not an inequality. This means that r_j^+ and r_j^- must both be zero for the dual variables to be feasible. Hence, the algorithm generates a sequence of r's for which these two components are tending to zero. As this happens, either x_j^+/r_j^+ or x_j^-/r_j^- will tend to infinity. Since these ratios appear in the diagonal matrix D^2, we see that this matrix becomes very ill-conditioned. When there are only a few free variables, experience shows that the optimality conditions are met before this ill-conditioning ruins things, but when there are many free variables, this ill-conditioning becomes quite apparent. (It is interesting to note that, of the 10 NETLIB problems that have free variables, only 3 of these problems have free variables which are negative in the optimal solution. CYCLE has 1 negative free variable, STAIR has 6, TUFF has 1 and none of the others have any. CYCLE and TUFF were not part of the test suite when Lustig, Shanno and Marsten did their experiments.)

Because of this difficulty, ALPO handles free variables differently. First, free variables are given a lower bound of -1. Then in the calculation of the primal step length, α_p, the free variables are ignored. Hence, when stepping to a new point, some of the free variables might become negative. If this happens, `optstd` negates those variables which became negative. It then also negates the corresponding objective function coefficient and the nonzeros in the corresponding column of the constraint matrix. Now all variables are again nonnegative and the algorithm proceeds to the next iteration. This method for handling free variables turns out to work much better than splitting them. For example, consider solving the dual of AFIRO (which is one of the smallest of the NETLIB problems) as if it were the primal. This dual problem has 59 variables of which 8 are free. Numerical instability prevented ALPO from solving this problem when it tried to split the free variables, but with the sign switching technique it had no trouble finding an optimal solution. (In the actual implementation, free variables are reflected about the value 1.0 instead of 0.0 so that all the lower bounds can still be zero.)

2.2 Solving the Dual as the Primal

Generally the most time consuming task in the primal-dual path-following algorithm is to solve systems of equations of the form

$$(AD^2A^T)y = \beta, \tag{2.1}$$

where β is some known vector and y is the vector for which we would like to solve. It was observed very early that comparatively dense columns in A translate into comparatively dense blocks in AD^2A^T and this has an adverse effect on the time to solve system (2.1). Many authors have suggested splitting the A matrix into two parts, a sparse part and a dense part, and then treatying the dense part separately using either low rank updates or conjugate gradient techniques. Most researchers who have experimented with these techniques point out that even though they get dramatic improvements, the code looses a great deal of robustness. This robustness problem derives from the fact that sometimes the system of

equations with certain columns deleted is singular when the original system was nonsingular. Handling this situation in a robust manner is numerically challenging.

For this reason ALPO does not employ any column dropping techniques. It does however offer the user another method which sometimes can overcome the difficulties presented by dense columns. The user can request that ALPO solve the dual problem as if it were the primal. This request causes dense columns to become dense rows (which do not present a problem). Unless the original problem has both dense rows and dense columns, this technique can speed up the solution dramatically. We have seen instances where solving the dual as the primal resulted in a 10,000 times speed-up!

To aid the user in deciding whether or not to request this feature, ALPO prints an estimate of the number of nonzeros in AD^2A^T for the usual case of solving the primal as the primal (called the *primal-fillin-estimate*) and for the other choice of solving the dual as the primal (called the *dual-fillin-estimate*). If the dual estimate is noticably smaller than the primal estimate, then it might be very wise to try solving the dual as the primal.

If the user doesn't explicitly specify whether to solve the primal or the dual, ALPO makes the selection based on the smaller between the primal-fillin-estimate with two times the dual-fillin-estimate (there is a further stipulation that solving the dual will not be selected if the original problem has finite positive ranges since the code to dualize such problems doesn't accomodate this possibility — this stipulation precludes ALPO from solving the dual on problem SEBA even though it would be highly advantageous).

The formula for the primal-fillin-estimate is quick and simple:

$$\text{primal-fillin-estimate} = m^2 \left(1 - \prod_{j=0}^{n-1} (1 - \text{nonz}(A_j)^2/m^2) \right) /2,$$

where A_j denotes the j^{th} column of A. The formula for the dual-fillin-estimate is similar, using the rows of A instead of the columns.

3 Performance

We ran `alpo` on the complete set of NETLIB problems [8]. Statistics regarding these problems are shown in Tables 1 and 2. The column labeled *Nonzeros in L* gives the number of nonzeros in the Cholesky factor L of AD^2A^T (excluding the diagonal). The matrix A is reordered once at the beginning of the code to approximately minimize the amount of fill-in created by the Cholesky factorization. The reordering heuristic used is the minimum-local-fill heuristic as described in [21]. The column labeled *Arithmetic Operations* gives the number of arithmetic operations in the calculation of the Cholesky factor L.

The computational results presented in Tables 3 and 4 were obtained on a Silicon Graphics 4D80/GT workstation with 8 megabytes of memory, one processor and a 16.7MHz clock. The code was compiled with the MIPS `cc` compiler using options `-O2 -Olimit 800`. All times were measured on the UNIX command line with the `time` command.

The column labeled *Primal Infeasibility* reports the following relative measure of primal infeasibility:

$$\frac{\|Ax - b\|}{\|b\| + 1}.$$

Similarly, the *Dual Infeasibility* column reports

$$\frac{\|c - A^T y - r\|}{\|c\| + 1}.$$

Currently, `alpo` does not understand generalized-upper-bound type constraints and so no results are reported for STANDGUB.

Our performance results are essentially the same as those presented by other authors who use Cholesky factorization as the basic engine for solving systems of equations. In fact, after normalizing for differences in hardware and accounting for differences in iteration counts, there is little difference in the performance from one code to the next. (The characteristic which really distinguishes one code from the next is the design, robustness and ease of use.)

ALPO choose to solve the dual as the primal on only one problem, AGG. As mentioned earlier, SEBA would have benefited substantially had its dual been solved but ALPO is

currently unable to dualize problems that have finite positive ranges. Since performing the original computational tests reported here, some additional problems were added to the NETLIB suite. Of these new NETLIB problems, two of them (FIT1P and FIT2P) benefit substantially by solving the dual as the primal. Comparisons between the primal and the dual for AGG, FIT1P and FIT2P are given in Table 5. It should be noted that two other new NETLIB problems, FIT1D and FIT2D, are hand-formulated duals of FIT1P and FIT2P, respectively. Table 5 includes performance data for these two problems as well. FIT1D and FIT2D have a large number of variables, but only a handful of constraints. Hence, the arithmetic operations to factor AD^2A^T is small compared to the number of arithmetic operations needed simply to form this matrix. These operation counts are 205,081 for FIT1D and 1,885,858 for FIT2D. Finally, note that AD^2A^T is completely dense in FIT1P and FIT2P.

One of the challenges to all developers of interior point methods is to find a heuristic for initializing the primal and dual solution vectors in such a way as to obtain low iteration counts over a wide spectrum of problems. ALPO got stuck well short of optimality on GREENBEA and just barely missed the optimality criteria on three other problems (CYCLE, GROW15 and GROW22). ALPO comes with several versions of the initialization subroutine `initxyr` stored in files having names that start with `init` followed by a number and then the usual ".c". The initialization subroutine which achieved the results in Tables 3 and 4 is contained in `init0.c`. All of the problems in the NETLIB suite have been solved with at least one of the initialization subroutines included with ALPO.

We now describe the specific initialization heuristic implemented in `init0.c`. First we compute the vector $\tilde{x}$:

$$\tilde{x}_j = \frac{1}{\|A_j\|_2 + 1}.$$

Then compute the scalar:

$$\beta = \frac{\|b\|_2 + 1}{\|A\tilde{x}\|_2 + 1}$$

and set

$$x = 10\beta\tilde{x}.$$

If any component of x is larger than half of that component's upper bound, then it is reset to half of the upper bound. This completes the description of the initialization of x. The dual vector y is simply set to zero. The reduced cost vector r is initialized as follows. If the j^{th} primal variable has a finite upper bound, then r_j is set to one if $c_j < 0$ and to $c_j + 1$ otherwise. On the other hand, if the upper bound is infinite then r_j is set to one if $c_j < 1$ and to c_j otherwise. The initialization of the surplus vector s is similar. Namely, if the j^{th} primal variable has a finite upper bound, then s_j is set to $-c_j + 1$ if $c_j < 0$ and to one otherwise. On the other hand, if the upper bound is infinite then s_j is set to zero regardless of the size of c_j. The slacks to the upper bounds t are first initialized so that

$$t = u - x$$

but then modified so that any t_j that is less than one is reset to one.

It is interesting to see what fraction of time is spent on various tasks. Tables 6 and 7 contain this information for the six subroutines which generally account for the lions share of the total time. The subroutine `lltnum` numerically calculates the Cholesky factor for AD^2A^T. The subroutine `adat` numerically calculates AD^2A^T itself. The subroutine `solve` solves a system of equations using the Cholesky factor computed by `lltnum`. The subroutine `smx` multiplies a sparse matrix (either A or A^T) times a vector. The subroutine `inv_sym` reorders the rows of A so that the Cholesky factor will not have many nonzeros that didn't already appear in AD^2A^T. It uses an explicit implementation of the minimum-local-fill algorithm as described in [22]. The profiling was performed on the Silicon Graphics workstation and so the column labeled *Total Time* represents times obtained on that machine.

4 Acknowledgement

ALPO was developed while the author was a Member of Technical Staff at AT&T Bell Labs. The author would like to thank J. Reeds for pointing out glitches in various preliminary versions of the code. He would also like to thank D. Gay for suggesting the technique used in ALPO for handling free variables. Thirdly, the author would like to thank M.G.C.

Resende for pointing out that the minimum-local-fill algorithm is superior to the minimum-degree algorithm (at least in the linear programming context). Finally, the author would like to thank D. Mitra and M.R. Garey for their support and encouragement.

References

[1] I. Adler, N.K. Karmarkar, M.G.C. Resende, and G. Veiga. Data structures and programming techniques for the implementation of Karmarkar's algorithm. *ORSA Journal on Computing*, 1:84–106, 1989.

[2] I. Adler, N.K. Karmarkar, M.G.C. Resende, and G. Veiga. An implementation of Karmarkar's algorithm for linear programming. *Mathematical Programming*, 44:297–335, 1989.

[3] A. Armacost and S. Mehrotra. A computational comparison of the network simplex method with the dual affine scaling method. *J. Ops. Res. Soc. India (Opsearch)*, 28(1):18–35, 1991.

[4] D. Bayer and J.C. Lagarias. Method and apparatus for optimizing system operational parameters, May 1988. U.S. Patent Number 4,744,027.

[5] D.A. Bayer and J.C. Lagarias. The nonlinear geometry of linear programming. ii Legendre transform coordinates and central trajectories. *Trans. AMS*, 314:527–581, 1989.

[6] I.C. Choi, C.L. Monma, and D.F. Shanno. Further development of a primal-dual interior point method. *ORSA J. on Computing*, 2:304–311, 1990.

[7] V. Chvátal. *Linear Programming*. Freeman, 1983.

[8] D.M. Gay. Electronic mail distribution of linear programming test problems. *Mathematical Programming Society COAL Newslettter*, 1985.

[9] D.M. Gay. Massive memory buys little speed for complete, in-core sparse Cholesky factorizations on some scalar computers. *Lin. Alg. Appl.*, 152, 1991.

[10] P.E. Gill, W. Murray, and M.A. Saunders. A single-phase dual barrier method for linear programming. Technical Report SOL 88-10, Systems Optimization Laboratory, Stanford Univ., Stanford, CA, 1988.

[11] P.E. Gill, W. Murray, M.A. Saunders, J.A. Tomlin, and M.H. Wright. On projected newton methods for linear programming and an equivalence to Karmarkar's projective method. *Mathematical Programming*, 36:183–209, 1986.

[12] N.K. Karmarkar. A new polynomial time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[13] N.K. Karmarkar. Methods and apparatus for efficient resourse allocation, May 1988. U.S. Patent Number 4,744,028.

[14] N.K. Karmarkar and K.G. Ramakrishnan. Implementation and computational results of the Karmarkar algorithm for linear programming, using an iterative method for computing projections. Technical report, AT&T Bell Labs, Murray Hill, NJ, 1989.

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

[16] R.E. Marsten, M.J. Saltzman, D.F. Shanno, G.S. Pierce, and J.F. Ballintijn. Implementation of a dual interior point algorithm for linear programming. *ORSA Journal on Computing*, 1:287–297, 1989.

[17] K.A. McShane, C.L. Monma, and D.F. Shanno. An implementation of a primal-dual interior point method for linear programming. *ORSA Journal on Computing*, 1:70–83, 1989.

[18] S. Mehrotra. Implementations of affine scaling methods: Approximate solutions of systems of linear equations using preconditioned conjugate gradient methods. Technical Report 89-04, Dept. of Ind. Eng. and Mgmt. Sci., Northwestern Univ., Evanston, IL, 1989.

[19] R.J. Vanderbei. Methods and apparatus for efficient resourse allocation, May 1988. U.S. Patent Number 4,744,026.

[20] R.J. Vanderbei. Affine scaling and linear programs with free variables. *Mathematical Programming*, 39:31–44, 1989.

[21] R.J. Vanderbei. A comparison between the minimum-local-fill and minimum-degree algorithms. Technical report, AT&T Bell Laboratories, 1990.

[22] J. Vlach and K. Singhal. *Computer Methods for Circuit Analysis and Design.* Van Nostrand Reinhold, 1983.

Problem Name	Constraints	Variables	Nonzeros in A	Nonzeros in AD^2A^T	Nonzeros in L	Arithmetic Operations
25FV47	821	1876	10705	11074	27821	1518904
80BAU3B	2262	12061	23264	10074	36771	1755486
ADLITTLE	56	138	424	328	355	2394
AFIRO	27	51	102	63	80	188
AGG	163	651	2573	1957	4410	165160
AGG2	516	758	4740	12883	20139	949264
AGG3	516	758	4756	12893	20139	949264
BANDM	305	472	2494	3419	4315	75364
BEACONFD	173	295	3408	2669	2727	61338
BLEND	74	114	522	743	931	13284
BNL1	643	1586	5532	4395	10990	376488
BNL2	2324	4486	14996	13457	75171	10015918
BOEING1	351	726	3827	5077	6744	150374
BOEING2	166	305	1358	1876	2576	54190
BORE3D	233	334	1448	2192	2769	55792
BRANDY	220	303	2202	2541	3204	87880
CAPRI	271	482	1896	2841	5226	151744
CYCLE	1903	3371	21234	27714	52385	2282474
CZPROB	929	3562	10708	7072	7465	82920
D2Q06C	2171	5831	33081	26991	89676	8283074
DEGEN2	444	757	4201	6868	15237	813558
DEGEN3	1503	2604	25432	50178	114444	13677310
E226	223	472	2768	2600	3412	71874
ETAMACRO	400	816	2537	2759	13509	718194
FFFFF800	524	1028	6401	10091	17501	823704
FINNIS	497	1064	2760	3175	6263	116844
FORPLAN	161	492	4634	2856	3542	118268
GANGES	1309	1706	6937	7656	17051	376624
GFRD-PNC	616	1160	2445	835	1520	2920
GREENBEA	2392	5598	31070	33841	69083	3012394
GREENBEB	2392	5598	31070	33841	69083	3012394
GROW15	300	645	5620	3130	5790	108680
GROW22	440	946	8252	4600	8590	161880
GROW7	140	301	2612	1450	2590	47880
ISRAEL	174	316	2443	11053	11210	968306
KB2	43	68	313	402	460	5370
LOTFI	153	366	1136	1043	1724	26310
NESM	662	3105	13470	4116	22287	1210062
PEROLD	625	1506	6148	6433	24161	1634514
PILOT.JA	940	2267	14977	14378	54533	6172060
PILOT.WE	722	2928	9265	4974	15334	612430
PILOT	1441	4860	44375	61538	162213	30815974
PILOT4	410	1123	5264	6411	11461	498030
PILOTNOV	975	2446	13331	12110	51862	5259268

Table 1: Statistics for the NETLIB problems (A-P).

Problem Name	Constraints	Variables	Nonzeros in A	Nonzeros in AD^2A^T	Nonzeros in L	Arithmetic Operations
RECIPE	91	204	687	498	587	7216
SC105	105	163	340	226	459	1864
SC205	205	317	665	451	969	4332
SC50A	50	78	160	101	182	578
SC50B	50	78	148	93	179	592
SCAGR25	471	671	1725	1922	2484	13810
SCAGR7	129	185	465	500	639	3388
SCFXM1	330	600	2732	2903	3986	65098
SCFXM2	660	1200	5469	5826	8111	133922
SCFXM3	990	1800	8206	8749	12318	205418
SCORPION	388	466	1534	1713	2052	13428
SCRS8	490	1275	3288	1708	5299	104590
SCSD1	77	760	2388	1056	1315	24756
SCSD6	147	1350	4316	1952	2398	40114
SCSD8	397	2750	8584	3883	5518	74888
SCTAP1	300	660	1872	1386	2271	22618
SCTAP2	1090	2500	7334	5505	12056	325224
SCTAP3	1480	3340	9734	7386	16318	432272
SEBA	515	1036	4360	51400	53599	10379222
SHARE1B	117	253	1179	884	1123	11646
SHARE2B	96	162	777	775	878	8274
SHELL	536	1777	3558	1705	3641	43756
SHIP04L	402	2166	6380	4228	4388	53632
SHIP04S	402	1506	4400	2912	3076	33728
SHIP08L	778	4363	12882	8512	8904	110848
SHIP08S	778	2467	7194	4728	5128	53760
SHIP12L	1151	5533	16276	10673	11137	135656
SHIP12S	1151	2869	8284	5345	5809	54296
SIERRA	1227	2735	8001	4936	11435	231484
STAIR	356	614	4003	6215	12265	558970
STANDATA	359	1274	3230	1465	2905	40684
STANDMPS	467	1274	3878	2653	4536	72854
STOCFOR1	117	165	501	504	787	5574
STOCFOR2	2157	3045	9357	12738	22191	278134
TUFF	333	628	4561	5077	7863	288006
VTP.BASE	198	346	1051	1815	2665	42332
WOOD1P	244	2595	70216	18046	18082	1535528
WOODW	1098	8418	37487	20421	44522	2567284

Table 2: Statistics for the NETLIB problems (R-Z).

Problem Name	Iteration Count	Primal Objective	Signif. Figures	Primal Infeasibility	Dual Infeasibility	Time (H : M : S)
25FV47	40	$5.501845893e+03$	8	$3.48e-13$	$3.51e-13$	1 : 32.5
80BAU3B	95	$9.872241956e+05$	8	$5.33e-11$	$2.36e-14$	5 : 38.2
ADLITTLE	26	$2.254949634e+05$	8	$9.42e-15$	$3.51e-13$	0.6
AFIRO	13	$-4.647531414e+02$	8	$6.18e-15$	$6.65e-17$	0.1
AGG	60	$-3.599176732e+07$	8	$7.88e-12$	$1.72e-12$	17.1
AGG2	37	$-2.023925232e+07$	8	$5.73e-13$	$3.42e-15$	48.1
AGG3	40	$1.031211595e+07$	8	$1.74e-13$	$8.57e-16$	51.9
BANDM	29	$-1.586280181e+02$	8	$8.37e-13$	$4.49e-14$	6.6
BEACONFD	20	$3.359248595e+04$	8	$1.13e-10$	$2.79e-12$	5.1
BLEND	19	$-3.081214983e+01$	8	$2.04e-13$	$3.69e-16$	0.8
BNL1	83	$1.977629564e+03$	8	$1.63e-12$	$1.38e-14$	55.4
BNL2	57	$1.811236547e+03$	8	$2.71e-12$	$2.09e-14$	11 : 09.0
BOEING1	37	$-3.352135657e+02$	8	$3.69e-11$	$1.04e-15$	15.1
BOEING2	34	$-3.150187276e+02$	8	$3.36e-13$	$3.52e-15$	4.4
BORE3D	38	$1.373080398e+03$	8	$9.77e-10$	$7.18e-16$	5.4
BRANDY	30	$1.518509899e+03$	8	$1.08e-09$	$9.63e-12$	6.2
CAPRI	41	$2.690012920e+03$	8	$1.77e-12$	$4.46e-12$	11.7
CYCLE	57	$-5.226385191e+00$	5	$3.70e-09$	$6.85e-08$	3 : 21.2
CZPROB	78	$2.185196708e+06$	8	$8.91e-11$	$4.89e-14$	51.3
D2Q06C	57	$1.227842114e+05$	8	$2.20e-10$	$5.57e-13$	10 : 20.1
DEGEN2	25	$-1.435177999e+03$	8	$3.91e-10$	$3.98e-15$	31.6
DEGEN3	39	$-9.872939982e+02$	8	$5.94e-09$	$4.54e-15$	12 : 46.7
E226	32	$-1.875192900e+01$	8	$7.21e-13$	$4.67e-14$	6.8
ETAMACRO	41	$-7.557152321e+02$	8	$2.12e-14$	$4.75e-13$	38.8
FFFFF800	53	$5.556795652e+05$	8	$7.35e-13$	$1.17e-08$	1 : 07.1
FINNIS	39	$1.727910661e+05$	8	$1.59e-11$	$4.33e-13$	12.5
FORPLAN	44	$-6.642189597e+02$	8	$1.62e-09$	$4.22e-10$	13.1
GANGES	30	$-1.095857359e+05$	8	$6.82e-11$	$3.59e-14$	27.9
GFRD-PNC	27	$6.902236032e+06$	8	$2.54e-12$	$9.41e-17$	5.2
GREENBEA	145	$-7.253016958e+07$	3	$2.66e-03$	$2.89e-05$	11 : 03.7
GREENBEB	177	$-4.302260231e+06$	8	$1.52e-08$	$2.18e-12$	13 : 26.9
GROW15	24	$-1.068709405e+08$	7	$1.25e-06$	$1.69e-15$	10.6
GROW22	28	$-1.608343365e+08$	9	$5.00e-06$	$1.40e-15$	17.8
GROW7	22	$-4.778781180e+07$	8	$4.53e-08$	$1.30e-15$	4.3
ISRAEL	40	$-8.966448209e+05$	8	$2.60e-14$	$1.28e-16$	47.9
KB2	28	$-1.749900128e+03$	8	$8.72e-09$	$6.78e-16$	0.5
LOTFI	31	$-2.526470604e+01$	8	$7.49e-09$	$2.41e-15$	2.7
NESM	95	$1.407603650e+07$	8	$5.39e-12$	$2.22e-12$	3 : 3.0
PEROLD	74	$-9.380755247e+03$	8	$2.11e-11$	$1.64e-09$	2 : 20.2
PILOT.JA	66	$-6.113136457e+03$	8	$1.66e-11$	$1.96e-10$	7 : 48.0
PILOT.WE	145	$-2.720107527e+06$	8	$1.66e-11$	$1.32e-12$	2 : 48.2
PILOT	59	$-5.574897259e+02$	8	$9.72e-13$	$3.54e-11$	39 : 35.6
PILOT4	63	$-2.581139253e+03$	8	$3.16e-11$	$7.48e-11$	48.6
PILOTNOV	45	$-4.497276188e+03$	8	$9.16e-13$	$6.13e-11$	4 : 56.2

Table 3: Computational Results on SGI workstation (A-P).

Problem Name	Iteration Count	Primal Objective	Signif. Figures	Primal Infeasibility	Dual Infeasibility	Time (H : M : S)
RECIPE	17	$-2.666159997e+02$	8	$2.26e-10$	$4.55e-12$	0.8
SC105	21	$-5.220206114e+01$	8	$3.35e-14$	$9.85e-17$	0.5
SC205	23	$-5.220206117e+01$	8	$9.33e-14$	$2.57e-12$	1.2
SC50A	18	$-6.457507680e+01$	8	$7.55e-15$	$6.25e-17$	0.2
SC50B	16	$-6.999999967e+01$	8	$8.55e-16$	$6.73e-17$	0.1
SCAGR25	33	$-1.475343303e+07$	8	$3.61e-13$	$6.25e-16$	4.6
SCAGR7	29	$-2.331389823e+06$	8	$3.01e-14$	$4.96e-16$	1.0
SCFXM1	35	$1.841675914e+04$	8	$7.43e-10$	$3.17e-13$	7.5
SCFXM2	39	$3.666026174e+04$	8	$6.16e-09$	$1.20e-13$	17.2
SCFXM3	40	$5.490125469e+04$	8	$2.53e-08$	$1.69e-14$	27.1
SCORPION	27	$1.878124825e+03$	8	$4.82e-11$	$4.69e-15$	3.2
SCRS8	35	$9.042969565e+02$	8	$1.07e-10$	$3.82e-16$	12.3
SCSD1	17	$8.666666685e+00$	8	$3.67e-14$	$2.05e-16$	2.7
SCSD6	19	$5.050000013e+01$	8	$9.23e-14$	$3.35e-16$	5.2
SCSD8	18	$9.050000021e+02$	8	$1.20e-11$	$6.92e-16$	10.9
SCTAP1	38	$1.412250001e+03$	8	$2.00e-10$	$4.92e-16$	5.0
SCTAP2	24	$1.724807148e+03$	8	$4.33e-10$	$2.25e-16$	21.3
SCTAP3	25	$1.424000006e+03$	8	$2.14e-10$	$2.20e-16$	29.4
SEBA	45	$1.571160003e+04$	8	$8.57e-10$	$2.43e-15$	10 : 09.2
SHARE1B	45	$-7.658931842e+04$	8	$2.28e-10$	$2.85e-16$	2.6
SHARE2B	21	$-4.157322402e+02$	8	$1.03e-11$	$8.61e-15$	1.0
SHELL	35	$1.208825351e+09$	8	$5.56e-12$	$1.96e-10$	10.9
SHIP04L	25	$1.793324545e+06$	8	$3.85e-10$	$9.28e-16$	10.7
SHIP04S	23	$1.798714705e+06$	8	$2.33e-10$	$8.66e-16$	6.8
SHIP08L	34	$1.909055216e+06$	8	$2.45e-10$	$1.17e-15$	27.7
SHIP08S	30	$1.920098218e+06$	8	$4.25e-10$	$9.29e-16$	14.2
SHIP12L	29	$1.470187925e+06$	8	$4.38e-10$	$3.20e-15$	30.9
SHIP12S	28	$1.489236146e+06$	8	$6.03e-10$	$4.80e-15$	15.9
SIERRA	28	$1.539436220e+07$	8	$1.68e-13$	$1.44e-12$	24.8
STAIR	29	$-2.512669505e+02$	8	$6.39e-11$	$9.58e-11$	22.6
STANDATA	30	$1.257699502e+03$	8	$3.85e-12$	$1.10e-14$	7.2
STANDMPS	36	$1.406017510e+03$	8	$8.22e-11$	$4.37e-15$	11.1
STOCFOR1	19	$-4.113197617e+04$	8	$2.12e-09$	$4.83e-16$	0.7
STOCFOR2	45	$-3.902440851e+04$	8	$1.33e-08$	$1.75e-15$	46.1
TUFF	45	$2.921477709e-01$	8	$2.50e-13$	$2.93e-09$	23.4
VTP.BASE	54	$1.298314627e+05$	8	$3.57e-13$	$2.67e-09$	5.7
WOOD1P	30	$1.442902431e+00$	8	$1.65e-09$	$3.21e-12$	2 : 58.1
WOODW	41	$1.304476342e+00$	8	$3.48e-09$	$1.77e-13$	3 : 15.6

Table 4: Computational Results on SGI workstation (R-Z).

Problem Name	Method	Nonzeros in AD^2A^T	Nonzeros in L	Arithmetic Operations	Time (H : M : S)
AGG	Primal	11,183	15,380	614,990	51.5
AGG	Dual	1,957	4,527	193,589	17.1
FIT1P	Primal	196,251	196,251	82,556,881	45 : 22.9
FIT1P	Dual	23,311	23,316	402,859	80.3
FIT1D	Primal	267	272	4996	29.0
FIT2P	Primal	4,498,500	4,498,500	9,008,999,000	—
FIT2P	Dual	190,098	190,101	3,336,261	48 : 50.2
FIT2D	Primal	296	299	5775	5 : 30.2

Table 5: Comparison Between Primal and Dual Methods.

Problem Name	Percent of Total Time Spent in lltnum	adat	solve	smx	inv_sym	optstd	Total Time (H : M : S)
25FV47	75.0	5.8	4.1	4.2	4.5	2.4	1 : 59.5
80BAU3B	63.7	4.5	3.8	7.2	1.8	13.1	7 : 23.6
ADLITTLE	16.4	11.9	6.0	20.9	4.5	11.9	0.6
AFIRO	0.0	7.7	0.0	7.7	0.0	15.4	0.1
AGG	56.2	10.3	6.0	9.3	3.1	7.2	17.9
AGG2	73.8	7.9	4.9	3.1	6.0	1.6	1 : 05.2
AGG3	74.2	8.0	5.0	3.2	5.4	1.6	1 : 10.0
BANDM	42.2	14.4	7.1	10.6	7.1	6.4	7.7
BEACONFD	27.4	28.6	4.0	12.7	9.4	2.8	5.6
BLEND	36.2	9.6	0.0	9.6	8.5	10.6	0.9
BNL1	64.3	7.6	6.1	8.3	2.2	6.8	1 : 08.9
BNL2	90.3	1.5	2.1	1.2	2.4	1.1	15 : 33.6
BOEING1	46.0	13.1	6.2	9.4	10.4	7.3	18.0
BOEING2	45.7	12.4	6.2	9.9	7.0	8.5	5.0
BORE3D	44.1	14.2	7.2	10.9	6.2	6.7	5.9
BRANDY	43.8	18.1	5.5	9.8	8.0	4.5	6.8
CAPRI	61.3	9.0	6.3	6.5	4.6	5.7	15.4
CYCLE	71.4	7.8	4.9	5.1	4.0	2.9	4 : 40.5
CZPROB	21.8	12.6	5.6	20.0	6.4	21.2	53.4
D2Q06C	85.6	4.1	2.4	2.3	2.5	1.4	16 : 05.5
DEGEN2	69.4	5.0	4.3	3.2	10.5	2.0	32.9
DEGEN3	84.0	2.9	1.7	1.0	8.7	0.4	14 : 59.8
E226	38.8	18.7	5.9	12.8	6.0	6.3	7.6
ETAMACRO	77.8	3.3	4.5	2.6	4.0	3.0	51.6
FFFFF800	72.1	9.0	4.6	4.6	4.5	2.3	1 : 26.8
FINNIS	46.1	9.2	7.8	9.6	5.4	11.1	14.5
FORPLAN	36.1	25.6	4.3	15.6	4.4	4.7	13.5
GANGES	48.9	9.3	7.4	7.9	6.5	7.5	33.8
GFRD-PNC	6.9	7.7	5.3	17.3	3.8	26.5	5.4
GREENBEA	76.0	6.6	4.6	5.3	1.8	3.5	16 : 46.0
GREENBEB	76.5	6.6	4.6	5.3	1.4	3.5	20 : 25.9
GROW15	34.2	20.2	4.9	12.7	4.9	9.7	13.7
GROW22	34.0	19.5	5.2	12.5	5.2	9.5	20.5
GROW7	32.5	19.8	4.6	11.3	5.8	10.1	4.9
ISRAEL	78.1	7.3	3.0	1.7	7.7	0.7	1 : 00.9
KB2	34.5	13.8	1.7	10.3	5.2	8.6	0.5
LOTFI	35.3	11.7	7.3	12.3	4.7	11.7	2.9
NESM	69.2	7.4	3.9	6.4	1.6	7.9	3 : 57.2
PEROLD	82.8	3.6	3.8	2.7	2.2	2.6	3 : 26.2
PILOT.JA	87.7	3.8	2.2	1.6	2.4	1.1	11 : 45.0
PILOT.WE	64.1	7.3	5.4	8.8	1.1	9.6	3 : 09.5
PILOT	90.6	2.8	1.2	0.9	3.4	0.4	57 : 10.5
PILOT4	67.1	9.6	4.9	6.0	3.3	5.1	1 : 03.3
PILOTNOV	86.4	3.0	2.5	1.7	3.3	1.3	6 : 54.9

Table 6: Percent of Time Spent in Certain Subroutines (A-P).

Problem Name	Percent of Total Time Spent in lltnum	adat	solve	smx	inv_sym	optstd	Total Time (H : M : S)
RECIPE	15.4	12.1	4.4	14.3	5.5	16.5	0.9
SC105	13.8	5.2	6.9	13.8	5.2	19.0	0.5
SC205	17.0	9.6	8.1	15.6	3.0	17.0	1.3
SC50A	11.5	3.8	11.5	3.8	3.8	11.5	0.2
SC50B	13.0	8.7	4.3	4.3	0.0	17.4	0.2
SCAGR25	20.7	11.8	8.3	16.4	4.8	15.1	4.7
SCAGR7	20.4	10.7	7.8	13.6	3.9	14.6	1.0
SCFXM1	39.1	16.1	7.1	12.3	4.4	8.4	8.5
SCFXM2	39.3	15.8	7.6	13.0	4.5	9.1	19.8
SCFXM3	40.2	15.4	7.5	12.5	4.5	8.9	31.8
SCORPION	19.0	13.1	8.4	16.8	5.3	12.8	3.1
SCRS8	41.5	8.1	6.5	11.5	3.9	13.0	12.8
SCSD1	16.1	10.1	3.5	17.1	4.2	14.3	2.8
SCSD6	16.3	10.6	3.2	18.2	4.1	16.3	5.6
SCSD8	16.3	9.4	3.7	16.9	3.4	15.7	11.6
SCTAP1	27.2	11.9	6.9	16.5	3.5	16.3	5.3
SCTAP2	46.8	7.4	5.6	9.0	6.1	9.4	25.7
SCTAP3	47.8	7.1	5.5	8.8	6.7	8.9	37.8
SEBA	84.9	4.9	1.5	0.3	7.3	0.4	11 : 16.2
SHARE1B	24.9	16.0	5.5	21.2	2.7	11.6	2.8
SHARE2B	20.2	13.5	5.8	22.1	4.8	12.5	1.0
SHELL	22.8	7.6	5.4	14.7	3.4	24.3	11.8
SHIP04L	17.3	10.7	4.3	17.9	6.0	17.7	11.2
SHIP04S	16.1	10.3	4.7	17.5	5.5	17.5	7.0
SHIP08L	18.7	11.3	5.0	18.8	4.9	19.4	29.1
SHIP08S	16.7	11.0	5.3	18.7	4.7	19.4	14.8
SHIP12L	17.3	10.5	4.8	18.1	5.4	18.7	33.2
SHIP12S	14.5	10.4	5.7	18.1	5.6	18.7	16.9
SIERRA	31.7	7.6	5.8	10.4	5.2	19.0	28.8
STAIR	68.8	8.4	4.8	4.2	5.6	2.6	30.5
STANDATA	24.7	9.1	5.6	16.5	4.3	19.6	7.6
STANDMPS	34.0	10.0	6.3	14.5	4.7	15.0	12.2
STOCFOR1	24.4	9.8	6.1	14.6	3.7	12.2	0.8
STOCFOR2	39.1	10.1	9.1	10.1	5.7	9.8	57.3
TUFF	57.3	12.4	5.7	8.2	5.9	4.0	26.7
VTP.BASE	47.6	10.0	7.4	9.9	3.8	11.8	6.6
WOOD1P	31.8	39.3	1.1	11.4	7.0	1.6	2 : 59.2
WOODW	64.6	7.0	3.3	7.6	5.9	5.4	4 : 09.1

Table 7: Percent of Time Spent in Certain Subroutines (R-Z).