Mathematical Programming 44 (1989) 297-335
North-Holland

AN IMPLEMENTATION OF KARMARKAR'S ALGORITHM FOR LINEAR PROGRAMMING

Ilan ADLER, Mauricio G.C. RESENDE* and Geraldo VEIGA
Department of Industrial Engineering and Operations Research, University of California, Berkeley, CA 94720, USA

Narendra KARMARKAR
AT&T Bell Laboratories, Murray Hill, NJ 07974, USA

Received 29 June 1987
Revised manuscript received 14 March 1989

This paper describes the implementation of power series dual affine scaling variants of Karmarkar's algorithm for linear programming. Based on a continuous version of Karmarkar's algorithm, two variants resulting from first and second order approximations of the continuous trajectory are implemented and tested. Linear programs are expressed in an inequality form, which allows for the inexact computation of the algorithm's direction of improvement, resulting in a significant computational advantage. Implementation issues particular to this family of algorithms, such as treatment of dense columns, are discussed. The code is tested on several standard linear programming problems and compares favorably with the simplex code MINOS 4.0.

Key words: Linear programming, Karmarkar's algorithm, interior point methods.

1. Introduction

We describe in this paper a family of interior point power series affine scaling algorithms based on the linear programming algorithm presented by Karmarkar (1984). Two algorithms from this family, corresponding to first and second order power series approximations, were implemented in FORTRAN over the period November 1985 to March 1986. Both are tested on several publicly available linear programming test problems (Gay, 1985, 1986). We also test one of the algorithms on randomly generated multi-commodity network flow problems (Ali and Kennington, 1977) and on timber harvest scheduling problems (Johnson, 1986).

Several authors (see, e.g., Aronson et al., 1985; Lustig, 1985; Tomlin, 1985; Tone, 1986) have compared implementations of interior point algorithms with simplex method codes, but have been unable to obtain competitive solution times. An implementation of a projected Newton's barrier method reported by Gill et al. (1986) presents the first extensive computational evidence indicating that an interior point algorithm can be comparable in speed with the simplex method.

In our computational experiments, solution times for the interior point implementations are, in most cases, less than those required by MINOS 4.0 (Murtagh and Saunders, 1977). Furthermore, we are typically able to achieve 8 digit accuracy

* Current address: AT&T Bell Laboratories, Murray Hill, NJ 07974, USA.

in the optimal objective function value without experiencing the numerical difficulties reported in previous implementations.

MINOS is a FORTRAN code intended primarily for the solution of constrained nonlinear programming problems, but includes an advanced implementation of the simplex method. An updated version, MINOS 5.0 (Murtagh and Saunders, 1983), features a scaling option and an improved set of routines for computing and updating sparse *LU* factors. This latest version of MINOS was not available at the University of California, Berkeley, where the computational tests described in this paper were carried out. We believe that for the purposes of this study, MINOS 4.0 constitutes a reasonable benchmark simplex implementation. Furthermore, as evidenced by the results reported in Gill et al. (1986) we do not expect MINOS 5.0 to perform significantly faster for the test problems considered here.

The plan of the paper is as follows. In Section 2, we describe our Algorithm I, a basic interior point method, commonly referred to as the *affine scaling* algorithm. When Algorithm I takes infinitesimal steps at each iteration, the resulting continuous trajectory is described by a system of differential equations. In Section 3, we discuss the family of algorihms constructed by truncating the Taylor expansion representing the solution of this system of differential equations. A first order approximation to the Taylor expansion results in Algorithm I. We also implement Algorithm II, which is obtained by truncating the Taylor expansion to a second order polynomial. We show in Section 4 how an initial interior solution can be obtained and in Section 5 how the algorithms can be applied to general linear programming problems. In Section 6, we describe the stopping criterion used in the computational experiments. Section 7 discusses some implementation issues, including symbolic and numerical factorizations, using an approximate scaling matrix, reducing fill-in during Gaussian elimination and using the conjugate gradient algorithm with preconditioning to solve problems with dense columns. In Section 8, we report the computational results of running the algorithms on three sets of test problems. The first set is a family of publicly available linear programs from a variety of sources. The others are, respectively, randomly generated multi-commodity network flow problems and timber harvest scheduling problems. We compare these results to those of the simplex code MINOS 4.0 and for the case of the multi-commodity network flow problems to MCNF85, a specialized simplex based code for multi-commodity network flow (Kennington, 1979). Conclusions are presented and future research is outlined in Section 9.

2. Description of the algorithm

Consider the linear programming problem:

$$\text{P:} \qquad \text{maximize} \quad c^{\mathrm{T}}x \tag{2.1}$$

$$\text{subject to} \quad Ax \leq b, \tag{2.2}$$

where c and x are n-vectors, b an m-vector and A is a full rank $m \times n$ matrix, where $m \geqslant n$ and $c \neq 0$. In assumptions relaxed later, we require P to have an interior feasible solution x^0.

Rather than expressing P in the standard equality form, we prefer the inequality formulation (2.1) and (2.2). As discussed later in this section, there are some computational advantages in the selection of search directions under this formulation. In Section 5, we show how to apply this approach to linear programs in standard form.

The algorithm described below is a variation of Karmarkar's original projective algorithm (Karmarkar, 1984), substituting an affine transformation for the projective transformation, and the objective function for the potential function. Recently, it came to our attention that this algorithm was first proposed independently by Dikin (1967). Similar algorithms have been discussed by Barnes (1986) and Vanderbei, Meketon and Freedman (1986). They, however, express the linear programming problem in (standard) equality form. Following the taxonomy referred to in Hooker (1986), the algorithms presented in this paper can be classified as *dual affine scaling* algorithms. However, these algorithms are equivalent to their primal counterparts applied to problems in inequality form.

Starting at x^0, the algorithm generates a sequence of feasible interior points $\{x^1, x^2, \ldots, x^k, \ldots\}$ with monotonically increasing objective values, i.e.,

$$b - Ax^k > 0 \tag{2.3}$$

and

$$c^{\mathrm{T}}x^{k+1} > c^{\mathrm{T}}x^k, \tag{2.4}$$

terminating when a *stopping criterion* to be discussed later is satisfied. Introducing slack variables to the formulation of P, we have:

$$\text{P:} \qquad \text{maximize} \quad c^{\mathrm{T}}x \tag{2.5}$$

$$\text{subject to} \quad Ax + v = b, \tag{2.6}$$

$$v \geqslant 0, \tag{2.7}$$

where v is the m-vector of slack variables.

Affine variants of Karmarkar's algorithm consist of a scaling operation applied to P, followed by a search procedure that determines the next iterate. At each iteration k, with v^k and x^k as the current iterates, a linear transformation is applied to the solution space,

$$\hat{v} = D_v^{-1}v, \tag{2.8}$$

where

$$D_v = \operatorname{diag}(v_1^k, \ldots, v_m^k). \tag{2.9}$$

The slack variables are scaled so that x^k is equidistant to all hyperplanes generating the closed half-spaces whose intersection forms the transformed feasible polyhedral set,

$$\{x \in \mathbb{R}^n \mid D_v^{-1}Ax \leq D_v^{-1}b\}. \tag{2.10}$$

Rewriting equations (2.1) and (2.2) in terms of the scaled slack variables, we have

$\hat{\mathrm{P}}$: maximize $c^{\mathrm{T}}x$ (2.11)

subject to $Ax + D_v\hat{v} = b$, (2.12)

$v \geq 0$. (2.13)

The set of feasible solutions for P is denoted by

$$X = \{x \in \mathbb{R}^n \mid Ax \leq b\}, \tag{2.14}$$

and the set of feasible scaled slacks in $\hat{\mathrm{P}}$ is

$$\hat{V} = \{\hat{v} \in \mathbb{R}^m \mid \exists x \in X, Ax + D_v\hat{v} = b\}. \tag{2.15}$$

As observed in Gonzaga (1988), under the full-rank assumption for A, there is a one to one relationship between X and $\hat{V}$, with

$$\hat{v}(x) = D_v^{-1}(b - ax) \tag{2.16}$$

and

$$x(\hat{v}) = (A^{\mathrm{T}}D_v^{-2}A)^{-1}A^{\mathrm{T}}D_v^{-1}(D_v^{-1}b - \hat{v}). \tag{2.17}$$

There is also a corresponding one to one relationship linking feasible directions h_x in X and $h_{\hat{v}}$ in $\hat{V}$, with

$$h_{\hat{v}} = -D_v^{-1}Ah_x \tag{2.18}$$

and

$$h_x = -(A^{\mathrm{T}}D_v^{-2}A)^{-1}A^{\mathrm{T}}D_v^{-1}h_{\hat{v}}. \tag{2.19}$$

Observe from (2.18) that a feasible direction in $\hat{V}$ lies on the range space of $D_v^{-1}A$.

As in other presentations of affine variants of Karmarkar's algorithm, the search direction selected at each iteration is the projected objective function gradient with respect to the scaled variables. Since only the slack variables are changed by the affine transformation, using (2.11) and (2.17), we can compute the gradient of the objective function with respect to $\hat{v}$,

$$\nabla_{\hat{v}}c(x(\hat{v})) = (\nabla_{\hat{v}}x(\hat{v}))^{\mathrm{T}}\nabla_x c(x) = -D_v^{-1}A(A^{\mathrm{T}}D_v^{-2}A)^{-1}c. \tag{2.20}$$

The gradient with respect to the scaled slacks lies on the range space of $D_v^{-1}A$, making a projection unnecessary. Consequently, the search direction in $\hat{V}$ is

$$h_{\hat{v}} = -D_v^{-1}A(A^{\mathrm{T}}D_v^{-2}A)^{-1}c, \tag{2.21}$$

and from (2.19) the corresponding feasible direction in X can be computed,

$$h_x = (A^{\mathrm{T}}D_v^{-2}A)^{-1}c. \tag{2.22}$$

Applying the inverse affine transformation to $h_{\hat{v}}$, we obtain the corresponding feasible direction for the unscaled slacks,

$$h_v = -A(A^T D_v^{-2} A)^{-1} c. \tag{2.23}$$

Under the assumption that $c \neq 0$, unboundedness is detected if $h_v \geqslant 0$. Otherwise, the next iterate is computed by taking the maximum feasible step in the direction h_v, and retracting back to an interior point according to a *safety factor* γ, $0 < \gamma < 1$, i.e.

$$x^{k+1} = x^k + \alpha h_x, \tag{2.24}$$

where

$$\alpha = \gamma \times \min\{-v_i^k/(h_v)_i \mid (h_v)_i < 0,\ i = 1, \ldots, m\}. \tag{2.25}$$

The above formulation allows for inexact projections without loss of feasibility. Even if h_x is not computed exactly in (2.22), a pair of feasible search direction can still be obtained by computing

$$h_v = -Ah_x. \tag{2.26}$$

Given an interior feasible solution x^0, a stopping criterion and a safety factor γ, Pseudo-code 2.1 describes Algorithm I as outlined in this section. As in subsequent instances in this paper, algorithms are expressed in an Algol-like algorithmic notation described by Tarjan (1983).

Pseudo-code 2.1. Algorithm I:

```
procedure Algorithm I ($A, b, c, x^0$, stopping criterion, $\gamma$)
 1  $k := 0$;
 2  do stopping criterion not satisfied $\rightarrow$
 3     $v^k := b - Ax^k$;
 4     $D_v := \mathrm{diag}(v_1^k, \ldots, v_m^k)$;
 5     $h_x := (A^T D_v^{-2} A)^{-1} c$;
 6     $h_v := -Ah_x$;
 7     if $h_v \geqslant 0 \rightarrow$ return fi;
 8     $\alpha := \gamma \times \min\{-v_i^k/(h_v)_i \mid (h_v)_i < 0,\ i = 1, \ldots, m\}$;
 9     $x^{k+1} := x^k + \alpha h_x$;
10     $k := k + 1$;
11  od
end Algorithm I.
```

3. A family of interior-point algorithms using power series approximations

Consider the continuous trajectory generated by Algorithm I when infinitesimal steps are taken at each iteration (Bayer and Lagarias, 1989). Let us denote the path of interior feasible solutions for P by $(\bar{x}(\tau), \bar{v}(\tau))$, where the real parameter τ is the continuous counterpart to the iteration counter. For any value of τ, the corresponding search directions $h_{\bar{x}(\tau)}$ and $h_{\bar{v}(\tau)}$ can be computed by expressions (2.22) and (2.23). Alternatively, the search directions can be described by an equivalent system of linear equations,

$$-A^{\mathrm{T}} D_{\bar{v}(\tau)}^{-2} h_{\bar{v}(\tau)} = c, \tag{3.1}$$

$$A h_{\bar{x}(\tau)} + h_{\bar{v}(\tau)} = 0. \tag{3.2}$$

By taking infinitesimal steps, the resulting continuous trajectory is such that

$$\frac{\mathrm{d}\bar{x}}{\mathrm{d}\tau}(\tau) = h_{\bar{x}(\tau)} \quad \text{and} \quad \frac{\mathrm{d}\bar{v}}{\mathrm{d}\tau}(\tau) = h_{\bar{v}(\tau)}. \tag{3.3}$$

Given the system of linear equations (3.1) and (3.2), we replace the search directions by the corresponding derivatives with respect to the trajectory parameter, resulting in the system of nonlinear, first order differential equations

$$-A^{\mathrm{T}} D_{\bar{v}(\tau)}^{-2} \frac{\mathrm{d}\bar{v}}{\mathrm{d}\tau}(\tau) = c, \tag{3.4}$$

$$A \frac{\mathrm{d}\bar{x}}{\mathrm{d}\tau}(\tau) + \frac{\mathrm{d}\bar{v}}{\mathrm{d}\tau}(\tau) = 0, \tag{3.5}$$

with the boundary conditions

$$\bar{x}(0) = x^0 \quad \text{and} \quad \bar{v}(0) = v^0, \tag{3.6}$$

where the initial solution (x^0, v^0) is given and satisfies $Ax^0 + v^0 = b$, $v^0 > 0$.

By following the trajectory satisfying (3.4)-(3.6), we can, theoretically, obtain the optimal solution to P (Adler and Monteiro, 1988). In practice, we build an iterative procedure where, after replacing the current iterate for the initial solution in the boundary condition, an approximate solution to (3.4)-(3.6) is computed, using a truncated Taylor power series expansion. The next iterate is determined through a search on the approximate trajectory.

As discussed later in this section, the search procedure requires a suitable reparametrization of the continuous trajectory,

$$\tau = \rho(t), \tag{3.7}$$

where $\rho(t)$ is a monotonically increasing, infinitely differentiable real function such that

$$x(t) = \bar{x}(\rho(t)) \quad \text{and} \quad v(t) = \bar{v}(\rho(t)). \tag{3.8}$$

Reparametrizing (3.4)-(3.6), we have

$$-A^{\mathrm{T}} D_{v(t)}^{-2} \frac{\mathrm{d}v}{\mathrm{d}t}(t) = \frac{\mathrm{d}\rho}{\mathrm{d}t}(t) c, \tag{3.9}$$

$$A \frac{\mathrm{d}x}{\mathrm{d}t}(t) + \frac{\mathrm{d}v}{\mathrm{d}t}(\tau) = 0, \tag{3.10}$$

and the boundary conditions

$$x(0)=x^0 \quad \text{and} \quad v(0)=v^0. \tag{3.11}$$

Since we cannot compute an exact solution to (3.9)-(3.11), we use approximate solutions to the system of differential equations as the backbone of a family of iterative algorithms. At each iteration k, the algorithm restarts the trajectory with an initial solution $(x(0), v(0))=(x^{k-1}, v^{k-1})$. A new iterate is generated by moving on the approximate trajectory without violating nonnegativity of the slack variables. The approximate solutions can be computed by means of a truncated Taylor expansion of order r (Karmarkar et al., 1989), such that for $t>0$,

$$x(t)\approx \tilde{x}(t)=x(0)+\sum_{i=1}^{r}\frac{t^i}{i!}\frac{\mathrm{d}^i x}{\mathrm{d}t^i}(0) \tag{3.12}$$

and

$$v(t)\approx \tilde{v}(t)=v(0)+\sum_{i=1}^{r}\frac{t^i}{i!}\frac{\mathrm{d}^i v}{\mathrm{d}t^i}(0). \tag{3.13}$$

From (3.9) and (3.10), we can compute derivatives of all orders for $x(t)$ and $v(t)$. Consider the functions

$$F(t)=-D_{v(t)}^{-1} \quad \text{and} \quad G(t)=D_{v(t)}, \tag{3.14}$$

which satisfy

$$F(t)G(t)=-I. \tag{3.15}$$

Applying Leibniz's differentiation theorem to the product $F(t)G(t)$, we have

$$\sum_{j=0}^{i}\frac{i!}{(i-j)!\,j!}\frac{\mathrm{d}^{i-j}F}{\mathrm{d}t^{i-j}}(t)\frac{\mathrm{d}^j G}{\mathrm{d}t^j}=0 \quad \text{for } i\geq 1. \tag{3.16}$$

From (3.16), the derivatives of $F(t)$ can be computed recursively by

$$\frac{\mathrm{d}^i F}{\mathrm{d}t^i}(t)=D_{v(t)}^{-2}\frac{\mathrm{d}^i G}{\mathrm{d}t^i}(t)-D_{v(t)}^{-1}\sum_{j=1}^{i-1}\frac{i!}{(i-j)!\,j!}\frac{\mathrm{d}^{i-j}F}{\mathrm{d}t^{i-j}}(t)\frac{\mathrm{d}^j G}{\mathrm{d}t^j}(t) \quad \text{for } i\geq 1. \tag{3.17}$$

Taking derivatives of both sides of (3.9), and rewriting the left-hand side of the equation in terms of $F(t)$, we have

$$-A^{\mathrm{T}}\frac{\mathrm{d}^i F}{\mathrm{d}t^i}(t)e=\frac{\mathrm{d}^i \rho}{\mathrm{d}t^i}(t)c \quad \text{for } i\geq 1. \tag{3.18}$$

By combining (3.17) with (3.18), and annexing the appropriate derivative of (3.10), we have a system of linear equations that recursively computes the derivatives of $x(t)$ and $v(t)$ evaluated at $t=0$,

$$-A^{\mathrm{T}}D_{v(0)}^{-2}\frac{\mathrm{d}^i v}{\mathrm{d}t^i}(0)=\frac{\mathrm{d}^i \rho}{\mathrm{d}t^i}(0)c+A^{\mathrm{T}}D_{v(0)}^{-1}\sum_{j=1}^{i-1}\frac{i!}{(i-j)!\,j!}\frac{\mathrm{d}^{i-j}F}{\mathrm{d}t^{i-j}}(0)\frac{\mathrm{d}^j v}{\mathrm{d}t^j}(0), \tag{3.19}$$

$$A\frac{\mathrm{d}^i x}{\mathrm{d}t^i}(0)+\frac{\mathrm{d}^i v}{\mathrm{d}t^i}(0)=0. \tag{3.20}$$

In an implementable reformulation of (3.19) and (3.20), we eliminate the binomial terms by defining

$$F_i = \frac{1}{i!}\frac{\mathrm{d}^i F}{\mathrm{d}t^i}(0), \tag{3.21}$$

$$\rho_i = \frac{1}{i!}\frac{\mathrm{d}^i \rho}{\mathrm{d}t^i}(0), \tag{3.22}$$

$$z_x^i = \frac{1}{i!}\frac{\mathrm{d}^i x}{\mathrm{d}t^i}(0) \quad \text{and} \quad z_v^i = \frac{1}{i!}\frac{\mathrm{d}^i v}{\mathrm{d}t^i}(0). \tag{3.23}$$

Solving (3.19) and (3.20), we have

$$z_x^i = \rho_i (A^{\mathrm{T}} D_{v(0)}^{-2} A)^{-1} c + (A^{\mathrm{T}} D_{v(0)}^{-2} A)^{-1} A^{\mathrm{T}} D_{v(0)}^{-1} \sum_{j=1}^{i-1} F_{i-j} z_v^j \tag{3.24}$$

and

$$z_v^i = -A z_x^i. \tag{3.25}$$

The approximate trajectory based on a power series of order r is rewritten as

$$\tilde{x}(t) = x(0) + \sum_{i=1}^{r} t^i z_x^i \quad \text{and} \quad \tilde{v}(t) = v(0) + \sum_{i=1}^{r} t^i z_v^i. \tag{3.26}$$

At iteration k, the next iterate is computed as

$$x^{k+1} = \tilde{x}(\alpha) \quad \text{and} \quad v^{k+1} = \tilde{v}(\alpha), \tag{3.27}$$

with

$$\alpha = \gamma \times \sup\{t \mid 0 \leq t \leq 1,\ \tilde{v}_i(t) \geq 0,\ i = 1, \ldots, m\}, \tag{3.28}$$

where $0 < \gamma < 1$.

By selecting a suitable reparametrization $\rho(t)$, the line search on $\tilde{v}(t)$ that computes α in (3.28) can be limited to the interval $0 \leq t \leq 1$. Since $\tilde{x}(t)$ and $\tilde{v}(t)$ depend solely on derivatives of $\rho(t)$ evaluated at $t = 0$, the desired reparametrization is fully characterized by constants $\rho_1, \ldots, \rho_r$. Values for ρ_i are computed by selecting a row index l, and forcing

$$\tilde{v}_l(1) = 0 \tag{3.29}$$

and

$$\frac{\mathrm{d}^i x_l}{\mathrm{d}t^i}(0) = 0 \quad \text{for } i = 2, \ldots, r. \tag{3.30}$$

Consider the search directions

$$h_x^1 = (A^{\mathrm{T}} D_{v(0)}^{-2} A)^{-1} c \tag{3.31}$$

and

$$h_v^1 = -Ah_x^1. \tag{3.32}$$

From (3.24) and (3.25), we have

$$z_x^1 = \rho_1 h_x^1 \quad \text{and} \quad z_v^1 = \rho_1 h_v^1. \tag{3.33}$$

Once again, under the assumptions that A is a full rank and $c \neq 0$, unboundedness is detected if $h_v \geq 0$. Otherwise, we determine the row index l by performing a ratio test between $v(0)$ and search direction h_v^1, i.e.,

$$l = \underset{1 \leq i \leq m}{\operatorname{argmin}}\{-v(0)/(h_v^1)_i \mid (h_v^1)_i < 0\}. \tag{3.34}$$

This operation corresponds to searching along the first order approximation of trajectory $v(t)$. From the conditions imposed on the reparametrization by (3.30), the truncated Taylor expansion in (3.26) is such that

$$\tilde{v}_l(1) = v(0) + \rho_1 h_v^1, \tag{3.35}$$

and from (3.29) and (3.33), we compute

$$\rho_1 = -v_l(0)/(h_v^1)_l. \tag{3.36}$$

Iteratively, for $i = 2, \ldots, r$, we compute ρ_i and z_x^i. Based on (3.24),

$$z_x^i = \rho_i h_x^1 + h_x^i \tag{3.37}$$

where

$$h_x^i = (A^{\mathrm{T}} D_{v(0)}^{-2} A)^{-1} A^{\mathrm{T}} D_{v(0)}^{-1} \sum_{j=1}^{i-1} F_{i-j} z_v^j. \tag{3.38}$$

To satisfy (3.30), we have

$$\rho_i = -(h_x^i)_l/(h_x^1)_l. \tag{3.39}$$

Pseudo-code 3.1 formalizes the algorithm based on a truncated power series of order r. The computational experiments reported in Section 8 include Algorithm II, which is the second order version of this algorithm. In a practical implementaion, the additional computational effort, when compared to Algorithm I, is dominated by the solution of systems of symmetric linear equations to compute h_x^i for $i = 2, \ldots, r$. As in Algorithm I, if $\tilde{x}(t)$ is not obtained exactly, a pair of feasible trajectories can still immediately be available. By computing

$$z_v^i = -Ax_x^i \quad \text{for } i = 1, \ldots, r, \tag{3.40}$$

we guarantee that $(\tilde{x}(t), \tilde{v}(t))$ is a pair of feasible trajectories for $0 \leq t \leq \alpha$, where $0 \leq \alpha \leq 1$ is the maximum feasible step.

Pseudo-code 3.1. Power Series Algorithm:

```
procedure PowerSeries ($A, b, c, x^0$, stopping criterion, $\gamma, r$)
1   $k := 0$;
2   do stopping criterion not satisfied $\rightarrow$
3      $x(0) := x^k$;
4      $v(0) := b - Ax(0)$;
5      $D_{v(0)} := \mathrm{diag}(v_1(0), \ldots, v_m(0))$;
6      $h_x^1 := (A^T D_{v(0)}^{-2} A)^{-1} c$; $h_v^1 := -Ah_x^1$;
7      if $h_v^1 \geq 0 \rightarrow$ return fi;
8      $l := \mathrm{argmin}_{1 \leq i \leq m} \{-v(0)/(h_v^1)_i \mid (h_v^1)_i < 0\}$;
9      $\rho_1 := -v_l(0)/(h_v^1)_l$;
10     $z_x^1 := \rho_1 h_x^1$; $z_v^1 := \rho_1 h_v^1$;
11     $F_1 := D_{v(0)}^{-2} z_v^1$;
12     for $i = 2, \ldots, r \rightarrow$
13        $h_x^i := (A^T D_{v(0)}^{-2} A)^{-1} A^T D_{v(0)}^{-1} \sum_{j=1}^{i-1} F_{i-j} z_v^j$;
14        $\rho_i := -(h_x^i)_l / (h_x^1)_l$;
15        $z_x^i := \rho_i h_x^1 + h_x^i$; $z_v^i := -Az_x^i$;
16        $F_i := D_{v(0)}^{-2} z_v^i - D_{v(0)}^{-1} \sum_{j=1}^{i-1} F_{i-j} z_v^j$;
17     rof;
18     $\tilde{x}(t) := x(0) + \sum_{i=1}^{r} t^i z_x^i$; $\tilde{v}(t) := v(0) + \sum_{i=1}^{r} t^i z_v^i$;
19     $\alpha := \gamma \times \sup\{t \mid 0 \leq t \leq 1, \tilde{v}_i(t) \geq 0, i = 1, \ldots, m\}$;
20     $x^{k+1} := \tilde{x}(\alpha)$;
21     $k := k + 1$;
22  od
end PowerSeries.
```

4. Initial solution

Algorithm I and the truncated power series algorithms require that an initial interior solution x^0 be provided. Since such solution does not necessarily exist for a generic linear program, and a starting solution close to the faces of the feasible polyhedral set can imply in very slow convergence (Megiddo and Shub, 1986), we propose a Phase I/Phase II scheme, where we first solve an artificial problem with a single artificial variable having a large cost coefficient assigned to it.

Firstly, we compute a tentative solution for P,

$$x^0 = (\|b\|_2 / \|Ac\|_2) c. \tag{4.1}$$

For the computational experiments reported in Section 8, we compute the initial value for the artificial variable as

$$x_a^0 = -2 \times \min\{(b - Ax^0)_i \mid i = 1, \ldots, m\}. \tag{4.2}$$

Although it did not occur in any of the test problems reported in this paper, the computation in (4.2) can be such that $x_a^0 \approx 0$. Consequently, for application to generic linear programs, we recommend an alternative computation of the initial value of the artificial variable,

$$x_a^0 = 2 \times \|b - Ax^0\|_2. \tag{4.3}$$

The $n+1$-vector (x^0, x_a^0) is an interior solution of the Phase I linear programming problem

$$\mathrm{P}_a: \qquad \text{maximize} \quad c^{\mathrm{T}}x - Mx_a \tag{4.4}$$

$$\text{subject to} \quad Ax - ex_a \leqslant b, \tag{4.5}$$

where

$$e = (1, 1, \ldots, 1)^{\mathrm{T}}. \tag{4.6}$$

The large artificial cost coefficient is computed as a function of the problem data,

$$M = \mu \times c^{\mathrm{T}}x^0 / x_a^0, \tag{4.7}$$

where μ is a large constant.

Initially, the algorithm is applied to P_a with a modified stopping criterion. In this Phase I stage, the algorithm either identifies an interior feasible solution, or, if no such solution exists, finds a solution that satisfies the stopping criterion for problem P. With ε_{f} defined as the feasibility tolerance, the modified stopping criterion for Phase I is formulated as follows:

(i) If $x_a^k < 0$ at iteration k, then x^k is an interior feasible solution for problem P.

(ii) If the algorithm satisfies the regular stopping criterion and $x_a^k > \varepsilon_{\mathrm{f}}$, P is declared infeasible.

(iii) If the algorithm satisfies the regular stopping criterion and $x_a^k < \varepsilon_{\mathrm{f}}$, either unboundedness is detected or an optimal solution is found. In this case, P has no interior feasible solution.

If Phase I terminates according to condition (i), the algorithm is applied to problem P starting with the last iterate of Phase I and using the regular stopping criterion.

5. Application to the general linear programming problem

It is not common practice to formulate a linear programming problem as in (2.1) and (2.2). Instead, a standard form is usually preferred,

$$\text{LPP:} \qquad \text{minimize} \quad b^{\mathrm{T}}y \tag{5.1}$$

$$\text{subject to} \quad A^{\mathrm{T}}y = c, \tag{5.2}$$

$$y \geqslant 0, \tag{5.3}$$

where A is an $m\times n$ matrix, c an n-vector and b and y m-vectors. The dual linear programming problem of (5.1)-(5.3), however, has the desired form:

$$\text{LPD:}\quad \text{maximize}\quad c^{\mathrm{T}}x \tag{5.4}$$

$$\text{subject to}\quad Ax\leq b, \tag{5.5}$$

where x is an n-vector. Note that LPD is identical to P, as defined in (2.1) and (2.2). At iteration k, with current solutions x^k and v^k, a tentative dual solution is defined as

$$y^k=D_v^{-2}A(A^{\mathrm{T}}D_v^{-2}A)^{-1}c, \tag{5.6}$$

where

$$D_v=\operatorname{diag}(v_1^k,\ldots,v_m^k). \tag{5.7}$$

This computation of the tentative dual solution, similar to the one suggested by Todd and Burrell (1986), can be performed by scaling the first order search direction in each iteration of the algorithm. From (2.23), for iteration k,

$$y^k=D_v^{-2}h_v. \tag{5.8}$$

The tentative dual solution minimizes the deviation from complementary slackness with respect to the current iterate x^k, relaxing the nonnegativity constraints (5.3) (Chandru and Kochar, 1986). Formally, consider the problem

$$\text{minimize}\quad \sum_{j=1}^{m}(v_j^k y_j)^2 \tag{5.9}$$

$$\text{subject to}\quad A^{\mathrm{T}}y=c. \tag{5.10}$$

The tentative dual solution in (5.6) is the solution to the Karush-Kuhn-Tucker stationary conditions of minimization problem given by (5.9) and (5.10). Under nondegeneracy, given a sequence of feasible primal solutions converging to an optimal solution, the corresponding sequence of tentative dual estimates also converges to the optimal dual solution.

6. Stopping criterion

In the computational experiments reported in this study, both Algorithms I and II are terminated whenever the relative improvement to the objective function is small, i.e.,

$$|c^{\mathrm{T}}x^k-c^{\mathrm{T}}x^{k-1}|/\max\{1,|c^{\mathrm{T}}x^{k-1}|\}<\varepsilon, \tag{6.1}$$

where ε is a given small positive tolerance.

The tentative dual solution y^k, computed in (5.6), can be used to build an alternative stopping criterion. If y^k and v^k satisfy

$$A^{\mathrm{T}}y^k=c, \tag{6.2}$$

$$y^k \geqslant 0, \tag{6.3}$$

$$y_j^k v_j^k = 0, \quad j = 1, 2, \ldots, m, \tag{6.4}$$

then, by duality theory of linear programming, y^k is optimal for LPP and x^k is optimal for LPD. Since (6.2) is automatically satisfied, a stopping criterion, replacing (6.1), such that the algorithm terminates whenever

$$y_j^k \geqslant -\varepsilon_1 \|y^k\|_2, \quad j = 1, 2, \ldots, m, \tag{6.5}$$

and

$$|y_j^k v_j^k| \leqslant \varepsilon_2 \|y^k\|_2 \|v^k\|_2, \quad j = 1, 2, \ldots, m, \tag{6.6}$$

for given small positive tolerances ε_1 and ε_2. The relations in (6.5) and (6.6) serve as a verification that x^k and y^k are indeed approximate optimal solutions for LPD and LPP, respectively.

Unboundedness of LPD is, theoretically, detected by the algorithm whenever $h_v \geqslant 0$, i.e., the ratio tests in (2.25) and (3.34), involving the first order search direction fail. In practice, an additional test is required, LPD is declared unbounded whenever the objective function value exceeds a supplied bound.

7. Implementation issues

In this section, we briefly discuss some important characteristics of this implementation. A detailed description of the data structures and programming techniques used in the implementation of Algorithms I and II is the subject of Adler et al. (1989).

7.1. Computing search directions

As in other variants of Karmarkar's algorithm, the main computational requirement of the algorithms described in this paper consists of the solution of a sequence of sparse symmetric positive definite systems of linear equations determining the search directions for each iteration. In Algorithm I, for each iteration k, a linear trajectory is determined by the feasible direction computed in

$$B_k h_x = c, \tag{7.1.1}$$

with

$$B_k = A^{\mathrm{T}} D_v^{-2} A, \tag{7.1.2}$$

where

$$D_v = \operatorname{diag}(v_1^k, \ldots, v_m^k). \tag{7.1.3}$$

Under the full-rank assumption for A, the system of linear equations in (7.1.1) is symmetric and positive definite. Such systems of linear equations are usually solved by means of the LU factorization,

$$B_k = LU, \tag{7.1.4}$$

where L is an $m \times m$ unit lower triangular matrix (a matrix that has exclusively ones in its main diagonal), and U is an $m \times m$ upper-triangular matrix. In the case of positive definite matrices, this LU factorization always exists and is unique. Furthermore, if the system is also symmetric, the factor L can be trivially obtained from U with

$$L = D_U^{-1} U^{\mathrm{T}}, \tag{7.1.5}$$

where D_U is a diagonal matrix with elements drawn from the diagonal of U. Rewriting (7.1.1), we have

$$(LU) h_x = c. \tag{7.1.6}$$

Direction h_x in (7.1.6) can be determined by solving two triangular systems of linear equations, performing a *forward substitution*

$$Lz = c, \tag{7.1.7}$$

followed by a *back substitution*

$$U h_x = z. \tag{7.1.8}$$

In each iteration of the higher order algorithms described in Section 3, the additional directions in (3.37) and (3.38) also involve the solution of linear systems identical to (7.1.1), except for different right-hand sides. Solving these additional systems of linear equations involve only the back and forward substitution operations, using the same LU factors. Furthermore, during the execution of the algorithm only D_v changes in (7.1.2), while A remains unaltered. Therefore, the nonzero structure of $A^{\mathrm{T}} D_v^{-2} A$ is static throughout the entire solution procedure. An efficient implementation of the Gaussian elimination procedure can take advantage of this property by performing, in the beginning of the algorithm, a single *symbolic factorization* step, i.e., operations that depend solely on the nonzero structure of the system matrix. For example, at this stage of the algorithm, we determine the nonzero structure of LU factors and build a list of the numerical operations performed during the Gaussian elimination procedure. At each iteration k of the algorithm, the actual numerical values of B_k are computed and incorporated in the symbolic information. Next, the *numerical factorization* is executed, by traversing the list of operations, yielding the LU factors.

7.2. Using an approximate scaling matrix

Based on the theoretical approach suggested by Karmarkar (1984), a significant reduction in the computational effort can be achieved by using an approximate scaling matrix when computing the numerical values for matrix B_k at each iteration. The matrix for the system of linear equations in (7.1.1) is replaced by

$$\tilde{B}_k = A^{\mathrm{T}} \tilde{D}_k^{-2} A, \tag{7.2.1}$$

where $\tilde{D}_k$ is an approximate scaling matrix, computed by selectively updating the scaling matrix used in the preceding iteration. The approximate scaling matrix at each iteration k is computed as follows:

$$\tilde{D}_k(i,i)=\begin{cases}\tilde{D}_{k-1}(i,i), & \text{if } |D_v(i,i)-\tilde{D}_{k-1}(i,i)|/|\tilde{D}_{k-1}(i,i)|<\delta,\\ D_v(i,i), & \text{if } |D_v(i,i)-\tilde{D}_{k-1}(i,i)|/|\tilde{D}_{k-1}(i,i)|\geq\delta,\end{cases} \tag{7.2.2}$$

for a given $\delta>0$. After computing the current approximate scaling matrix according to (7.2.2), we define

$$\Delta=\tilde{D}_k^{-2}-\tilde{D}_{k-1}^{-2}. \tag{7.2.3}$$

Then, we have the update expression

$$\tilde{B}_k=\tilde{B}_{k-1}+A^{\mathrm{T}}\Delta A. \tag{7.2.4}$$

This enables us to update the linear system matrix by rescaling only a reduced set of columns of A^{T}.

7.3. *Ordering for sparsity*

When a sparse matrix is factored, fill-in usually occurs. The triangular factors contain nonzero elements in positions where B_k has zeros. Fill-in degrades the performance of the sparse Gaussian elimination used to compute the LU factorization, also affecting the back and forward substitution operations. It is possible to reduce fill-in by performing a permutation of columns and rows in B_k.

If P is a permutation matrix, then (7.1.1) and

$$(PB_kP^{\mathrm{T}})Ph_x=Pc \tag{7.3.1}$$

are equivalent systems. Furthermore, there exists a permutation matrix $\bar{P}$ such that the fill-in generated in the triangular factors is minimized. Unfortunately, finding this permutation matrix is an NP-complete problem (Yannakakis, 1981). However, the *minimum degree* and *minimum local fill-in* ordering heuristics (Rose, 1972) have been shown to perform well in practice (Duff, Erisman and Reid, 1986). We use either the minimum degree heuristic as implemented in subroutine MD of the Yale Sparse Matrix Package (Eisenstat et al., 1982), or the minimum local fill-in implementation in de Carvalho (1987).

7.4. *Treating dense columns in the coefficient matrix*

In the presence of a few dense columns in A^{T}, $A^{\mathrm{T}}D_v^{-2}A$ will be impracticably dense, regardless of the permutation matrix. Consequently, we face prohibitively high computational effort and storage requirements during Gaussian elimination. To remedy this situation, we make use of a hybrid scheme in which we first perform an incomplete factorization of $A^{\mathrm{T}}D_v^{-2}A$. Next, we use the incomplete Cholesky factors as preconditioners for a conjugate gradient method to solve the system of linear equations defined in (7.1.1).

Let $(N, \bar{N})$ be a partition of the column indices of A^{T}, such that the columns of A_N^{T} have density smaller than a given parameter λ. At iteration k, the incomplete Cholesky factors $\tilde{L}_k$ and $\tilde{L}_k^{\mathrm{T}}$ are such that

$$A_N^{\mathrm{T}} D_v^{-2} A_N = \tilde{L}_k \tilde{L}_k^{\mathrm{T}}. \tag{7.4.1}$$

Using the conjugate gradient algorithm, we solve the system of linear equations

$$Qu = f, \tag{7.4.2}$$

where

$$Q = \tilde{L}_k^{-1}(A^{\mathrm{T}} D_v^{-2} A)(\tilde{L}_k^{\mathrm{T}})^{-1}, \tag{7.4.3}$$

and

$$f = \tilde{L}_k^{-1} c. \tag{7.4.4}$$

The search direction h_x is computed by performing a back substitution operation, solving

$$u = \tilde{L}_k^{\mathrm{T}} h_x. \tag{7.4.5}$$

Given a termination tolerance $\varepsilon_{cg} > 0$, the conjugate gradient algorithm is outlined in Pseudo-code 7.4.1.

Pseudo-code 7.4.1. Conjugate Gradient Algorithm:

```
procedure ConjugateGradient $(Q, f, \varepsilon_{cg})$
1   $u_0 := f$;
2   $r_0 := Qu_0 - f$;
3   $p_0 := -r_0$;
4   $i := 0$;
5   do $\|r_i\|_2^2 \geq \varepsilon_{cg} \rightarrow$
6       $q_i := Qp_i$;
7       $\alpha_i := \|r\|_2^2 / p_i^{\mathrm{T}} q_i$;
8       $u_{i+1} := u_i + \alpha_i p_i$;
9       $r_{i+1} := Qu_{i+1} - f$;
10      $\beta_i := \|r_{i+1}\|_2^2 / \|r_i\|_2^2$;
11      $p_{i+1} := -r_{i+1} + \beta_i p_i$;
12      $i := i + 1$;
13  od
end ConjugateGradient.
```

8. Test problems

In this section, we report the computational results of running our implementations of Algorithms I and II on a set of linear programming test problems (Gay, 1985,

1986) available through NETLIB (Dongarra and Grosse, 1987). NETLIB is a system designed to provide efficient distribution of public domain software to the scientific community through computer networks, e.g. Arpanet, UNIX UUCP network, CSNET, Telenet and BITNET. We also report the results of running Algorithm I on linear programs generated by two models. The first set is composed of randomly generated multi-commodity network flow problems generated with MNETGN (Ali and Kennington, 1977). The other is a collection of timber harvest scheduling problems generated by FORPLAN (Johnson, 1986), a linear programming based system used for long-range planning by the US Forest Service.

8.1. The NETLIB test problems

This collection of linear programs consists of problems contributed by a variety of sources. It includes linear programming test problems from the Systems Optimization Laboratory at Stanford University, staircase linear programs generated by Ho and Loute (1981), and many real world problems. We considered in our tests all of the available problems that do not have a BOUNDS section in their MPS representation, since the current version of our code cannot handle bounded variables implicitly.

Table 8.1.1 presents the statistics for 31 test problems obtained from NETLIB, ordered by increasing number of nonzero elements in the coefficient matrix A, after adding slack variables. The dimensions of the problems include null or fixed variables, sometimes removed by a preprocessor before applying a linear programming algorithm. The number of rows does not include the objective function. In column 4, the number of nonzero elements of A is displayed. Column 5 gives the number of nonzero elements of the lower triangular portion of the permuted $(PA)^TPA$ matrix, using the minimum degree ordering heuristic. Column 6 gives the nonzero elements in the Cholesky factor. The fill-in is the difference between columns 6 and 5. The entries in columns 5 and 6 for problem *Israel* are those used by the algorithm, i.e., after 6 dense columns of A^T are dropped.

All test runs, with the exception of those of Algorithm II, were carried out on the IBM 3090 at U.C.-Berkeley under VM/SP CMS. Algorithm II test runs were conducted on the IBM 3081-K at U.C.-Berkeley, also under VM/SP CMS. FORTRAN programs were compiled using the FORTVS compiler with compiler options OPT(3), NOSYM and NOSDUMP. The reported CPU times were computed with utility routine DATETM, taking into account preprocessing (e.g. input matrix cleanup, ordering and symbolic factorization) and all operations until termination. However, they exclude the effort required to translate the linear programs from MPS format.

Execution times for Algorithm I and Algorithm II are compared to those of the simplex code MINOS 4.0. The reported CPU times for MINOS are those of subroutine DRIVER, which also excludes the time to read and translate the MPS input file. The results of solving the NETLIB test problems by MINOS 4.0 are displayed in Table 8.1.2. In running MINOS, we use default parameters except for LOG FREQ = 200 and ITERATIONS = 7500. The problem size parameters, ROWS, COLUMNS and ELEMENTS,

Table 8.1.1
NETLIB test problem statistics

Problem	Rows	Columns	nonz(A)	nonz($(PA)^TPA$)[1]	nonz(L)
Afiro	27	51	102	90	107
ADLittle	56	138	424	377	404
Scagr7	129	185	465	606	734
Sc205	205	317	665	654	1182
Share2b	96	162	777	871	1026
Share1b	117	253	1179	967	1425
Scorpion	388	466	1534	1915	2324
Scagr25	471	671	1725	2370	2948
ScTap1	300	660	1872	1686	2667
BrandY	220	303	2202	2190	2850
Scsd1	77	760	2388	1133	1392
Israel	174	316	2443	3545	3707
BandM	305	472	2494	2929	4114
Scfxm1	330	600	2732	3143	4963
E226	223	472	2768	2683	3416
Scrs8	490	1275	3288	1953	5134
Beaconfd	173	295	3408	1720	1727
Scsd6	147	1350	4316	2099	2545
Ship04s	402	1506	4400	2827	3134
Scfxm2	660	1200	5469	6306	9791
Ship04l	402	2166	6380	4147	4384
Ship08s	778	2467	7194	3552	4112
ScTap2	1090	2500	7334	6595	14870
Scfxm3	990	1800	8206	9469	14619
Ship12s	1151	2869	8284	4233	5063
Scsd8	397	2750	8584	4280	5879
ScTap3	1480	3340	9534	8866	19469
CzProb	929	3340	10043	6265	6655
25FV47[2]	821	1876	10705	10919	33498
Ship08l	778	4363	12882	6772	7128
Ship12l	1151	5533	16276	8959	9501

[1] Using the minimum degree ordering heuristic.
[2] *25FV47* is also known as *BP822*.

were set to the exact number of rows, columns and nonzero elements of the coefficient matrix. In this fashion, MINOS selects the value for the all important PARTIAL PRICE parameter according to its built-in default strategy. Using the default parameters, MINOS 4.0 achieved 7 digit accuracy on all problems, if compared to the optimal objective values reported in Gay (1986).

Since the NETLIB test problems are not in the form described by (2.1) and (2.2), Algorithms I and II solve the corresponding dual linear programming problems. A primal optimal solution is obtained from (5.6), involving the last scaling matrix D_v and the last search direction for the dual slacks. By contrast, MINOS solves the test problems in its original form. Tables 8.1.3 and 8.1.4 summarize computational results for Algorithm I on the set of test problems, using, respectively, the minimum degree

Table 8.1.2
MINOS 4.0 test statistics (IBM 3090)—NETLIB test problems

Problem	Phase I Iter.	Total iter.	Time (sec.)	Objective value
Afiro	2	6	0.01	−4.6475319 e +02
ADLittle	28	119	0.23	2.2549495 e +05
Scagr7	38	81	0.29	−2.3313897 e +06
Sc205	30	111	0.45	−5.2201997 e +01
Share2b	83	119	0.26	−4.1573247 e +02
Share1b	238	391	1.27	−7.6589255 e +04
Scorpion	146	215	1.06	1.8781247 e +03
Scagr25	228	614	4.54	−1.4753432 e +07
ScTap1	184	305	1.98	1.4122500 e +03
BrandY	185	313	1.34	1.5185099 e +03
Scsd1	97	261	1.02	8.6666659 e +00
Israel	72	281	1.54	−8.9664483 e +05
BandM	207	472	2.54	−1.5862800 e +02
Scfxm1	238	437	2.04	1.8416758 e +04
E226	69	568	3.68	−1.8638928 e +01
Scrs8	135	523	5.64	9.0429701 e +02
Beaconfd	9	41	0.18	3.3592486 e +04
Scsd6	189	576	2.69	5.0500000 e +01
Ship04s	309	395	3.54	1.7987147 e +06
Scfxm2	432	792	8.18	3.6660260 e +04
Ship04l	441	593	7.18	1.7933246 e +06
Ship08s	470	657	9.16	1.9200982 e +06
ScTap2	667	1121	22.02	1.7248071 e +03
Scfxm3	646	1196	19.31	5.4901252 e +04
Ship12s	561	718	12.78	1.4892361 e +06
Scsd8	585	1590	14.53	9.0499997 e +02
ScTap3	696	1273	32.87	1.4240000 e +03
CzProb	1035	1955	39.08	2.1851968 e +06
25FV47	1262	7157	217.67	5.5018494 e +03
Ship08l	594	958	14.72	1.9090552 e +06
Ship12l	782	1268	26.10	1.4701879 e +06

and the minimum local fill-in ordering heuristics. Table 8.1.5 presents results for Algorithm II, which uses the minimum degree ordering heuristic.

Both Algorithms I and II were tested with the following parameter settings. The safety factor parameter was set to $\gamma = 0.99$ for the first 10 iterations and $\gamma = 0.95$ thereafter. Both algorithms were terminated when the relative improvement in the objective function fell below $\varepsilon = 10^{-8}$. In Phase I, the value of the artificial variable cost defined in (4.7) was determined by the constant $\mu = 10^5$. The diagonal update tolerance was set to $\delta = 0.1$ and the column density parameter for building the incomplete Cholesky factorization in the conjugate gradient algorithm was set to $\lambda = 0.3$.

Table 8.1.6 compares execution times for the three algorithms. CPU times displayed for MINOS and Algorithm I were measured on the IBM 3090, while those for Algorithm

Table 8.1.3
Algorithm I test statistics (IBM 3090)—NETLIB test problems (minimum degree ordering heuristic)

Problem	Phase I iter.	Total iter.	Time (sec.)	Objective value
Afiro	1	20	0.04	−4.6475315 e +02
ADLittle	1	24	0.12	2.2549496 e +05
Scagr7	3	25	0.17	−2.3313898 e +06
Sc205	4	29	0.28	−5.2202061 e +01
Share2b	4	28	0.29	−4.1573224 e +02
Share1b	7	39	0.58	−7.6589319 e +04
Scorpion	5	25	0.51	1.8781248 e +03
Scagr25	3	28	0.69	−1.4753433 e +07
ScTap1	6	34	0.85	1.4122488 e +03
BrandY	38	38	1.52	1.5185099 e +03
Scsd1	2	19	0.41	8.6666666 e +00
Israel[1]	8	38	4.00	−8.9664483 e +05
BandM	7	33	1.54	−1.5862802 e +02
Scfxm1	33	33	1.97	1.8416759 e +04
E226	40	40	1.56	−1.8751929 e +01
Scrs8	39	39	2.30	9.0429695 e +02
Beaconfd	23	23	0.69	3.3592486 e +04
Scsd6	2	22	0.83	5.0500000 e +01
Ship04s	5	31	1.26	1.7987145 e +06
Scfxm2	38	38	4.38	3.6660261 e +04
Ship04l	5	31	2.31	1.7933245 e +06
Ship08s	5	34	1.58	1.9200982 e +06
ScTap2	7	33	6.71	1.7248068 e +03
Scfxm3	37	37	6.66	5.4901254 e +04
Ship12s	4	35	1.91	1.4892361 e +06
Scsd8	2	24	1.68	9.0500000 e +02
ScTap3	7	36	9.44	1.4240000 e +03
CzProb	3	46	9.76	2.1851927 e +06
25FV47	44	55	46.17	5.5018494 e +03
Ship08l	5	34	4.00	1.9090552 e +06
Ship12l	5	34	4.89	1.4701879 e +06

[1] The conjugate gradient algorithm was triggered when running this test problem.

II are from the IBM 3081-K. For the sake of consistency, when computing the CPU time ratio between MINOS and Algorithm II, the execution times for MINOS are those of the IBM 3081-K. Figures 1 and 2 illustrate graphically the performances of MINOS and Algorithm I.

Table 8.1.7 focuses on five subgroups of problems, where each subgroup is made up of problems having the same structure, or generated by the same model. In this table, ratios of MINOS iterations to Algorithm I iterations, MINOS CPU time per iteration to Algorithm I CPU time per iteration and MINOS total CPU time to Algorithm I total CPU time are given.

Table 8.1.8 presents detailed results related to the generation of primal solutions in Algorithm I. Let t be the number of iterates generated by the algorithm. The

Table 8.1.4

Algorithm I test statistics (IBM 3090)—NETLIB test problems (minimum local fill-in ordering heuristic)

Problem	Phase I iter.	Total iter.	Time (sec.)	Objective value
Afiro	1	20	0.05	−4.6475315 e +02
ADLittle	1	24	0.13	2.2549496 e +05
Scagr7	3	24	0.18	−2.3313898 e +06
Sc205	4	28	0.26	−5.2202061 e +01
Share2b	4	29	0.32	−4.1573224 e +02
Share1b	7	38	0.47	−7.6589319 e +04
Scorpion	5	24	0.49	1.8781248 e +03
Scagr25	3	29	0.70	−1.4753433 e +07
ScTap1	6	33	0.80	1.4122488 e +03
BrandY	38	38	1.49	1.5185099 e +03
Scsd1	2	19	0.44	8.6666666 e +00
Israel[1]	8	37	4.01	−8.9664483 e +05
BandM	7	30	1.26	−1.5862802 e +02
Scfxm1	33	33	1.66	1.8416759 e +04
E226	34	34	1.58	−1.8751929 e +01
Scrs8	39	39	2.28	9.0429695 e +02
Beaconfd	23	23	0.62	3.3592486 e +04
Scsd6	2	22	0.84	5.0500000 e +01
Ship04s	5	30	1.01	1.7987145 e +06
Scfxm2	38	39	3.83	3.6660261 e +04
Ship04l	5	28	1.39	1.7933245 e +06
Ship08s	5	32	1.35	1.9200982 e +06
ScTap2	7	34	5.52	1.7248068 e +03
Scfxm3	37	40	5.87	5.4901254 e +04
Ship12s	4	35	1.75	1.4892361 e +06
Scsd8	2	23	1.82	9.0500000 e +02
ScTap3	7	36	7.78	1.4240000 e +03
CzProb	3	52	3.64	2.1851927 e +06
25FV47	44	54	31.70	5.5018494 e +03
Ship08l	5	31	2.44	1.9090552 e +06
Ship12l	5	32	3.43	1.4701879 e +06

[1] The conjugate gradient algorithm was triggered when running this test problem.

normalized duality gap

$$|b^{\mathrm{T}}y^t - c^{\mathrm{T}}x^t|/|b^t y^t| \qquad (8.1.1)$$

is given in column 2. Column 3 presents the maximum normalized primal infeasibility

$$\max\{|A_i^{\mathrm{T}}y^t - c_i|/\|c\|_2 \,|\, i=1,2,\ldots,n\}. \qquad (8.1.2)$$

Column 4 gives the minimum normalized primal entry,

$$\min\{y_i^t/\|y^t\|_2 \,|\, i=1,2,\ldots,m\} \qquad (8.1.3)$$

and column 5 the maximum normalized complementarity violation

$$\max\{|y_i^t v_i^t|/\|y^t\|_2\|v^t\|_2 \,|\, i=1,2,\ldots,m\}. \qquad (8.1.4)$$

Table 8.1.5

Algorithm II test statistics (IBM 3081-K)—NETLIB test problems (minimum degree ordering heuristic)[1]

Problem	Phase I iter.	Total iter.	Time (sec.)	Objective value
Afiro	1	15	0.08	−4.6475315 e +02
ADLittle	1	18	0.27	2.2549496 e +05
Scagr7	3	19	0.43	−2.3313898 e +06
Sc205	3	20	0.66	−5.2202061 e +01
Share2b	4	21	0.69	−4.1573224 e +02
Share1b	5	33	1.47	−7.6589319 e +04
Scorpion	4	19	1.22	1.8781248 e +03
Scagr25	3	21	1.71	−1.4753433 e +07
ScTap1	5	23	2.00	1.4122488 e +03
BrandY	24	24	2.79	1.5185099 e +03
Scsd1	2	16	1.10	8.6666666 e +00
Israel	6	29	8.22	−8.9664483 e +05
BandM	4	24	3.46	−1.5862802 e +02
Scfxm1	30	30	4.94	1.8416759 e +04
E226	30	30	3.54	−1.8751929 e +01
Scrs8	29	29	5.07	9.0429695 e +02
Beaconfd	17	17	1.51	3.3592486 e +04
Scsd6	2	18	2.13	5.0500000 e +01
Ship04s	4	22	2.85	1.7987145 e +06
Scfxm2	29	29	9.28	3.6660261 e +04
Ship04l	4	21	4.71	1.7933245 e +06
Ship08s	4	21	3.30	1.9200982 e +06
ScTap2	5	25	13.55	1.7248068 e +03
Scfxm3	30	30	14.25	5.4901254 e +04
Ship12s	3	23	4.17	1.4892361 e +06
Scsd8	2	18	4.22	9.0500000 e +02
ScTap3	6	27	19.59	1.4240000 e +03
CzProb	3	35	14.33	2.1851927 e +06
Ship08l	4	23	8.12	1.9090552 e +06
Ship12l	4	23	10.07	1.4701879 e +06

[1] Problem *25FV47* was not solved with Algorithm II, as its current implementation does not incorporate a dense window data structure (Adler et al., 1989) necessary to solve this problem under a 4 Mbytes memory limit.

From the results above, we make the following observations:

• Iterations for Algorithm I vary from 19 to 55 (see Tables 8.1.3 and 8.1.4), growing slowly with problem size.

• Algorithm I is, in general, faster than MINOS, with a speed-up of up to 10.7 (see Table 8.1.6 and Figures 1 and 2). The total problem set execution time was 5.14 times faster. MINOS, was faster on 5 small problems, all having less than 225 rows and 3500 nonzero matrix elements.

• If the test problems are categorized into three groups according to number of nonzero elements: "small" (*Afiro–Ship04s*), "medium" (*Scfxm2–Ship12s*) and "large" (*Scsd8–Ship12l*), Algorithm I is, for each corresponding category, on the

Table 8.1.6

Comparison of run times (IBM 3090)—NETLIB test problems (minimum local fill-in ordering heuristic)

Problem	MINOS time (sec.)	Alg. I time (sec.)	Alg. II time[1] (sec.)	MINOS/Alg. I time ratio	MINOS/Alg. II time ratio[1]
Afiro	0.01	0.05	0.08	0.2	0.5
ADLittle	0.23	0.13	0.27	1.8	1.7
Scagr7	0.29	0.18	0.43	1.6	1.2
Sc205	0.45	0.26	0.66	1.7	1.6
Share2b	0.26	0.32	0.69	0.8	0.9
Share1b	1.27	0.47	1.47	2.7	1.8
Scorpion	1.06	0.49	1.22	2.2	2.1
Scagr25	4.54	0.70	0.71	6.5	6.1
ScTap1	1.98	0.80	2.00	2.5	1.7
BrandY	1.34	1.49	2.79	0.9	1.1
Scsd1	1.02	0.44	1.10	2.3	2.2
Israel	1.54	4.01	8.22	0.4	0.4
BandM	2.54	1.26	3.46	2.0	1.9
Scfxm1	2.04	1.66	4.94	1.2	1.0
E226	3.68	1.58	3.54	2.3	1.7
Scrs8	5.64	2.28	5.07	2.5	1.9
Beaconfd	0.18	0.62	1.51	0.3	0.3
Scsd6	2.69	0.84	2.13	3.2	3.9
Ship04s	3.54	1.01	2.85	3.5	2.1
Scfxm2	8.18	3.83	9.28	2.1	1.8
Ship04l	7.18	1.39	4.71	5.2	2.5
Ship08s	9.16	1.35	3.30	6.8	4.6
ScTap2	22.02	5.52	13.52	4.0	2.8
Scfxm3	19.31	5.87	14.25	3.3	2.5
Ship12s	12.78	1.75	4.17	7.3	5.6
Scsd8	14.53	1.82	4.22	8.0	7.7
ScTap3	32.87	7.78	18.95	4.2	3.0
CzProb	39.08	3.64	14.33	10.7	4.3
25FV47	217.67	31.70	NA	6.9	NA
Ship08l	14.72	2.44	8.12	6.0	4.0
Ship12l	26.10	3.43	10.07	7.6	5.5

[1] IBM 3081-K CPU times and minimum degree ordering heuristic in Algorithm II.

average 1.8, 4.0 and 6.8 times faster than MINOS. One may infer a growth in the relative speed of Algorithm I with respect to MINOS, as problem sizes increase.

• In Table 8.1.7, where problems with similar structure are grouped together, one can observe that MINOS's disadvantage in number of iterations grows with problem size, while its advantage in time per iteration seems to level off for the larger problems. This is so, because MINOS refactors the basis after a fixed number of iterations. Refactorization requires work that is in the same order of one iteration of Algorithm I, dominating the work carried out in the intermediate iterations. Also, as the problem size increases, the overhead incurred by Algorithm I in the preprocessing is absorbed and therefore one should expect the ratio of work per iteration

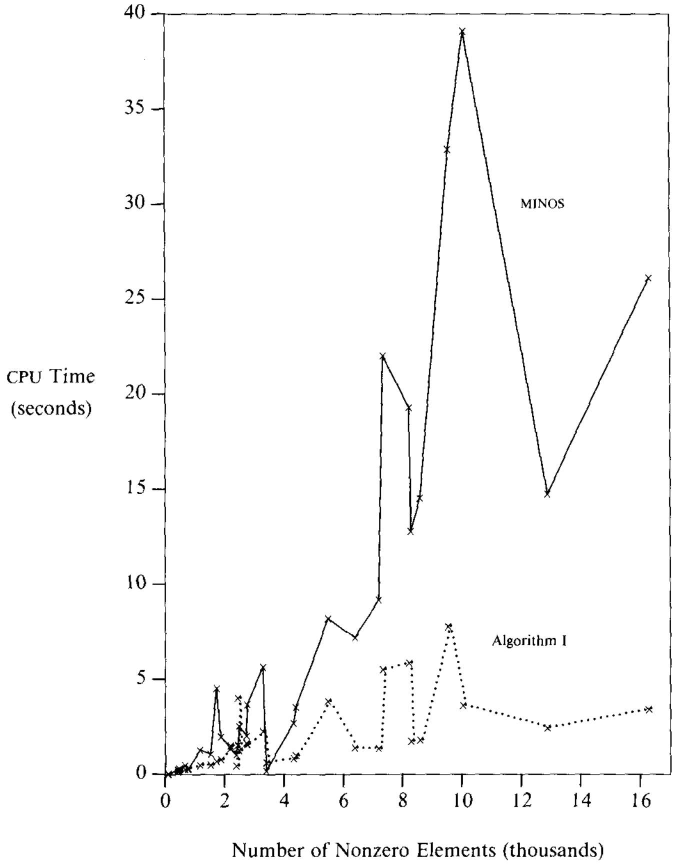

Fig. 1. MINOS 4.0 and Algorithm I (CPU times, IBM 3090)—NETLIB test problems (minimum local fill-in ordering heuristic in Algorithm I). (Test problem *25FV47* is not included in this graph.)

between the two algorithms to be at worst proportional to the inverse of the refactorization frequency. Thus, relative to simplex based codes, Algorithm I becomes increasingly faster as the problems get larger.

• The optimal objective values reported in Table 8.1.3 are accurate within 6 and 8 digits if compared to the values reported in (Gay, 1985, 1986). The primal solution computed at termination was accurate in most cases with the exception of problems *CzProb* and *25FV47* (see Table 8.1.8). The primal-dual relation presented for problem *Israel* are based on an implementation using direct factorization to compute the search directions at each iteration.

• All of the above considerations made for Algorithm I are valid for Algorithm II. Furthermore, Algorithm II requires on average 26% less iterations to converge to 8 digit accuracy than Algorithm I. However, this does not translate into any significant gain in solution time, since the extra work per iteration in the current implementation offsets the savings in the number of iterations. We believe that by

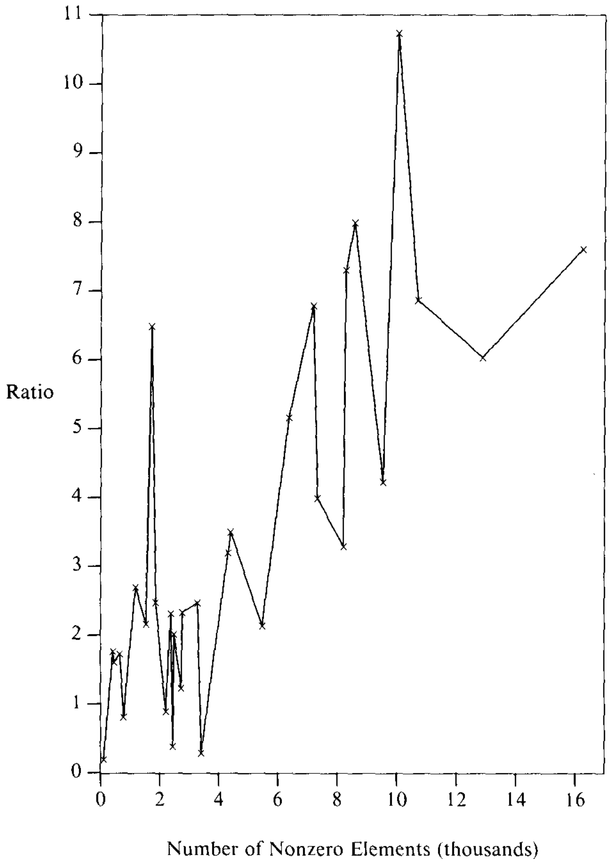

Fig. 2. MINOS 4.0/Algorithm I (CPU time ratio, IBM 3090)—NETLIB test problems (minimum local fill-in ordering heuristic in Algorithm I).

refining the extra computation, further reduction in work per iteration in Algorithm II is possible, leading to a faster implementation.

• The conjugate gradient algorithm, with column density parameter $\lambda = 0.3$, was important for solving test problem *Israel.* Factorization times for that problem were reduced significantly, when compared with an earlier version of Algorithm I, where the conjugate gradient routine was not implemented. Even though dropping a few dense columns did not significantly affect convergence of the dual solution, it did not generate a direction precise enough for computing an accurate primal solution at termination. This, however, can be accomplished by means of a more exact solution to the system defining the last direction. Actually, by using an exact factorization in the last iteration the accuracy of the primal solution for problem *Israel* was similar to those reported for the other problems.

• Comparing Table 8.1.3 and Table 8.1.4 we observe a clear advantage in using the minimum local fill-in ordering heuristic. While the ordering algorithm for this

Table 8.1.7

Iteration, time per iteration and total time ratios for NETLIB subgroups (IBM 3090) (minimum local fill-in ordering heuristic)

Problem	Rows	Columns	MINOS/Alg. I iter. ratio	MINOS/Alg. I time/iter. ratio	MINOS/Alg. I time ratio
Scsd1	77	760	13.7	0.169	2.32
Scsd6	147	1350	26.2	0.122	3.20
Scsd8	397	2750	69.1	0.115	7.98
Scfxm1	330	600	13.2	0.093	1.23
Scfxm2	660	1200	20.3	0.105	2.14
Scfxm3	990	1800	29.9	0.110	3.29
ScTap1	300	660	9.2	0.268	2.47
ScTap2	1090	2500	33.0	0.121	3.99
ScTap3	1480	3340	35.4	0.119	4.22
Ship04s	402	1506	10.3	0.340	3.50
Ship08s	778	2467	20.5	0.330	6.79
Ship12s	1151	2869	20.5	0.356	7.30
Ship04l	402	2166	21.2	0.244	5.17
Ship08l	778	4363	30.9	0.195	6.03
Ship12l	1151	5533	39.6	0.192	7.61

heuristic usually involves longer processing times, the reduction in fill-in results in a faster Gaussian elimination procedure. Since the same ordering is used in every iteration of the linear programming algorithm, the total savings more than offsets the extra processing in the ordering procedure. The sum of the solution times (including reordering) for all test problems was reduced from 119.10 seconds to 89.11 seconds with minimal local fill-in, corresponding to a 25% reduction. Solution times for Algorithm I with minimal local fill-in were faster on 21 of the 31 test problems and on 11 of the 12 problems having more than 5000 nonzero elements (see de Carvalho, 1987).

8.2. Multi-commodity network flow problems

In this section, we report on 11 multi-commodity network flow problems generated with MNETGN (Ali and Kennington, 1977), a random multi-commodity network flow problem generator. MNETGN generates a random network structure based on the number of arcs and nodes supplied by the user. Additional user specifications include the number of commodities and the ranges of arc costs and capacities. Table 8.2.1 displays the basic data for this set of test problems. We generated multi-commodity network flow problem by varying the number of nodes and commodities and keeping the same relative number of arcs. The test problems were solved with Algorithm I, MINOS and MCNF85 (Kennington, 1979), a special purpose simplex code for multi-commodity network flow problems. Algorithm I used the minimum degree ordering heuristic and the same parameter selection described in Section

Table 8.1.8
Primal dual relations-Algorithm I (IBM 3090)—NETLIB test problems

Problem	Relative duality gap	Max. normal. primal infeas.	Min. normal. primal entry	Max. normal. complem. viol.
Afiro	3.56 e −09	1.58 e −15	−5.76 e −24	7.99 e −12
ADLittle	3.69 e −09	1.05 e −11	−9.67 e −22	3.06 e −12
Scagr7	3.16 e −09	2.31 e −10	−4.02 e −20	6.27 e −13
Sc205	2.41 e −09	2.23 e −13	−2.61 e −02	2.73 e −13
Share 2b	6.83 e −11	2.09 e −12	−1.06 e −16	1.84 e −12
Share1b	3.33 e −09	3.26 e −09	−1.73 e −21	8.31 e −15
Scorpion	2.84 e −09	5.27 e −12	−6.07 e −13	1.14 e −12
Scagr25	1.06 e −08	1.32 e −10	−9.73 e −23	4.50 e −13
ScTap1	2.25 e −09	1.41 e −13	−2.96 e −21	9.40 e −14
BrandY	1.34 e −09	4.00 e −06	−6.91 e −04	5.43 e −14
Scsd1	3.31 e −09	4.90 e −13	−1.05 e −14	1.17 e −11
Israel	1.22 e −09	5.46 e −12	−5.38 e −24	1.11 e −15
BandM	1.06 e −08	2.12 e −09	−1.28 e −16	1.76 e −13
Scfxm1	1.20 e −07	7.29 e −07	−8.78 e −19	5.03 e −13
E226	4.10 e −09	3.36 e −07	−5.94 e −05	1.51 e −13
Scrs8	1.70 e −09	7.28 e −06	−1.67 e −09	3.55 e −15
Beaconfd	1.44 e −09	1.42 e −06	−6.48 e −06	9.23 e −12
Scsd6	9.81 e −10	4.30 e −10	−1.61 e −16	7.15 e −13
Ship04s	8.34 e −09	2.41 e −09	−4.95 e −14	1.45 e −11
Scfxm2	1.54 e −08	5.99 e −10	−1.78 e −19	2.19 e −13
Ship04l	2.15 e −09	2.25 e −11	−1.75 e −16	3.01 e −12
Ship08s	1.56 e −09	1.09 e −11	−4.24 e −18	1.40 e −12
ScTap2	1.06 e −08	9.57 e −15	−6.09 e −20	2.00 e −13
Scfxm3	9.29 e −09	4.11 e −10	−5.37 e −20	8.30 e −14
Ship12s	8.16 e −09	1.78 e −09	−1.74 e −15	2.96 e −12
Scsd8	4.26 e −09	1.35 e −14	−4.67 e −21	6.15 e −13
ScTap3	1.73 e −09	1.08 e −14	−1.01 e −20	4.23 e −14
CzProb	1.60 e −09	1.11 e −05	−1.02 e −19	1.19 e −14
25FV47	1.22 e −08	5.43 e −03	−3.17 e −06	7.54 e −15
Ship08l	4.11 e −10	1.02 e −09	−3.05 e −18	4.29 e −13
Ship12l	1.27 e −09	9.38 e −10	−2.19 e −17	2.83 e −13

8.1. All runs were carried out on an IBM 3090. Tables 8.2.2–8.2.4 and Figure 3 present the statistics for these runs.

We list below a few observations on the test results reported on this section.

• Multicommodity networks provide an important class of test problems on which a general purpose simplex method performs poorly (see Table 8.2.3 and Figure 3). The behavior of our implementation of Algorithm I confirms our earlier observation that the relative speed-up in relation to the simplex method grows with size. A similar leveling off of the time per iteration ratios between the two simplex based codes and Algorithm I is observed (see Tables 8.2.5 and 8.2.6). The only exception was for the MCNF85 to Algorithm I ratio on the 5-commodity problems. The trend in the data suggests that leveling off should only occur for larger networks, i.e.

Table 8.2.1

Multi-commodity network flow test problems statistics

Problem	Nodes	Commodities	LP rows	LP columns	nonz(A)
MUL031	300	1	1406	2606	5212
MUL041	400	1	1888	2488	6976
MUL051	500	1	2360	4360	8720
MUL061	600	1	2853	5253	10506
MUL043	400	3	2008	4027	9670
MUL053	500	3	2489	4949	11876
MUL063	600	3	2981	5882	14126
MUL035	300	5	2081	4436	11196
MUL045	400	5	2793	5948	15068
MUL055	500	5	3477	7637	19182
MUL065	600	5	4135	8800	22140

Table 8.2.2

Algorithm I test statistics (IBM 3090)—Multicommodity networks (minimum degree ordering heuristic in Algorithm I)

Problem	Phase I iter.	Total iter.	Time (sec.)
MUL031	3	33	6.51
MUL041	2	28	11.79
MUL051	2	33	23.07
MUL061	2	31	30.96
MUL043	2	27	24.27
MUL053	2	30	47.84
MUL063	2	28	58.57
MUL035	2	30	55.81
MUL045	2	30	104.34
MUL055	2	36	204.71
MUL065	2	35	309.50

networks with more than 600 nodes. As before, the number of iterations grows slowly, never exceeding 36.

• It is interesting to note that despite being a general purpose implementation, Algorithm I performed comparably to MCNF85. A tailored implementation of Algorithm I (e.g. with integer operations and specialized Gaussian elimination) could improve on current results. Furthermore, the trend in the data (Table 8.2.6) suggests that for larger problems Algorithm I could outperform MCNF85.

• The objective values for the solutions obtained with our implementation of Algorithm I achieved 8 digits accuracy when compared to the values obtained by MINOS. Also a feasible primal solution was recovered at the end of the algorithm with the same degree of accuracy.

Table 8.2.3

MINOS 4.0 test statistics (IBM 3090)—Multicommodity networks

Problem	Phase I iter.	Total iter.	Time (sec.)
MUL031	362	940	12.73
MUL041	457	1351	24.11
MUL051	620	1807	40.65
MUL061	817	2358	63.88
MUL043	2498	5926	196.38
MUL053	3366	8115	345.69
MUL063	4855	12286	677.31
MUL035	2930	6691	249.71
MUL045	4611	12193	666.58
MUL055	5919	15238	1045.67
MUL065	7693	21915	1885.34

Table 8.2.4

MCNF85 test statistics (IBM 3090)—Multicommodity networks

Problem	Total iter.	Time (sec.)
MUL031	931	7.42
MUL041	1087	11.39
MUL051	1892	24.87
MUL061	3082	53.45
MUL043	4983	45.23
MUL053	5158	55.60
MUL063	16983	243.35
MUL035	4132	34.64
MUL045	10558	111.30
MUL055	8862	121.66
MUL065	16624	260.44

8.3. Timber harvest scheduling problems

In this section, we report on 11 timber harvest scheduling problems generated with FORPLAN (Johnson, 1986). Based on data collected for a United States national forest, a series of timber harvest scheduling models of increasing sizes were generated. Under the framework of FORPLAN, a forest is divided into management units called *analysis areas*, each comprising a collection of acres from across the forest, sharing similar silvicultural and economic characteristics. Our test problems were created by successively increasing the number of analysis areas, resulting in models that represent subsets of the original forest. FORPLAN is widely used throughout the United States national forest system, yielding very large linear programs that pose

Table 8.2.5

MINOS/Algorithm I performance ratios—Multicommodity networks (minimum local fill-in ordering heuristic)

Problem	Nodes	Commodities	MINOS/Alg. I iter. ratio	MINOS/Alg. I time/iter. ratio	MINOS/Alg. I time ratio
MUL031	300	1	28.5	0.069	1.96
MUL041	400	1	48.3	0.042	2.04
MUL051	500	1	54.8	0.032	1.76
MUL061	600	1	76.1	0.027	2.06
MUL043	400	3	219.5	0.037	8.09
MUL053	500	3	270.5	0.027	7.23
MUL063	600	3	438.8	0.026	11.56
MUL035	300	5	223.0	0.020	4.47
MUL045	400	5	406.4	0.016	6.39
MUL055	500	5	423.3	0.012	5.11
MUL065	600	5	626.1	0.010	6.09

Table 8.2.6

MCNF85/Algorithm I performance ratios—Multicommodity networks (minimum local fill-in ordering heuristic)

Problem	Nodes	Commodities	MCNF85/Alg. I iter. ratio	MCNF85/Alg. I time/iter. ratio	MCNF85/Alg. I time ratio
MUL031	300	1	28.2	0.0404	1.14
MUL041	400	1	38.8	0.0249	0.97
MUL051	500	1	57.3	0.0188	1.08
MUL061	600	1	99.4	0.0174	1.73
MUL043	400	3	184.6	0.0101	1.86
MUL053	500	3	171.9	0.0068	1.16
MUL063	600	3	606.5	0.0069	4.15
MUL035	300	5	137.7	0.0045	0.62
MUL045	400	5	351.9	0.0030	1.07
MUL055	500	5	246.2	0.0024	0.59
MUL065	600	5	475.0	0.0018	0.84

a considerable challenge to linear programming codes. Table 8.3.1 presents the basic statistics of this family of test problems.

From Table 8.3.1, we observe that the test problems have significantly more variables than constraints. This characteristic is accentuated with problem size. In this situation, most advanced simplex implementations provide a choice of pricing strategies. In particular, MINOS allows for *partial pricing*, a scheme in which the columns of the coefficient matrix are partitioned into equal segments. In each *pricing* operation, the search for the incoming basic variable is limited to one segment. This reduces the work required for each operation, but, of course, has no predictable

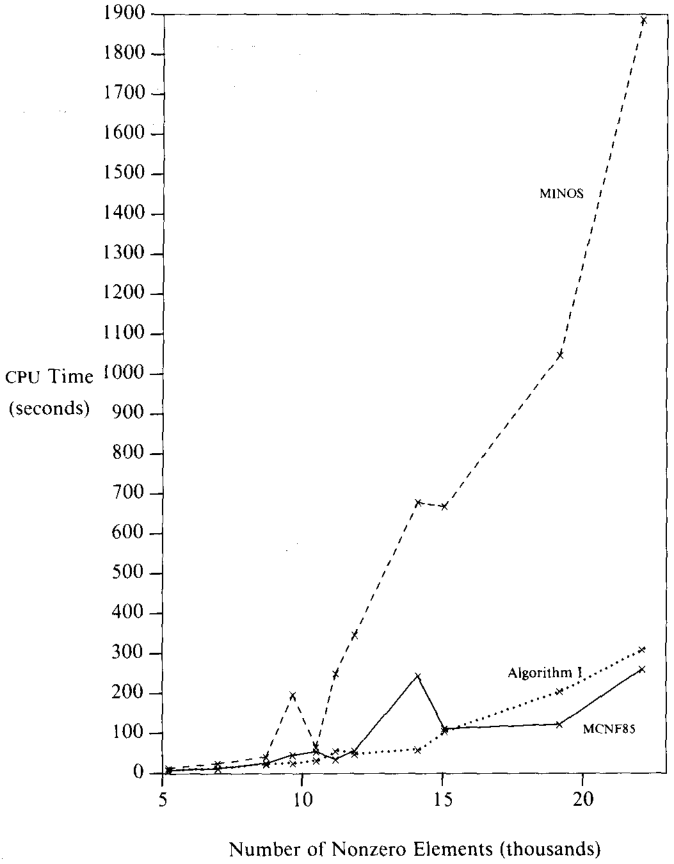

Fig. 3. MINOS 4.0, MCNF85 and Algorithm I (CPU times, IBM 3090)—Multicommodity flows (minimum degree ordering heuristic in Algorithm I).

Table 8.3.1

Timber harvest scheduling test problems statistics

Problem	LP Rows	LP Columns	nonz(A)
FPK010	55	744	6021
FPK040	58	973	8168
FPK050	64	1502	12765
FPK080	76	2486	21617
FPK100	87	3357	30104
FPK150	109	4993	43774
FPK200	130	6667	59057
FPK300	177	9805	86573
FPK400	220	12570	109697
FPK500	266	16958	149553
FPK600	316	19991	176346

effect on the total number of iterations. For this class of test problems, the manipulation of the PARTIAL PRICE parameter, which sets the number of segments in the partition, can impact the performance of MINOS enormously. Consequently, we compare MINOS with Algorithm I using the following pricing strategies:

• *Total pricing strategy*: Each pricing operation examines the total set of variables.

• *Default pricing strategy*: Sets the PARTIAL PRICE parameter to half of the ratio of number of columns to number of rows.

• *Improved pricing strategy*: Sets the PARTIAL PRICE parameter to eight times the ratio of number of columns to number of rows. We arrived at this stragegy after extensive testing with some of the test problems.

Tables 8.3.2–8.3.8 display the results of running the test problems with Algorithm I and MINOS under the three pricing strategies. Figure 4 illustrates graphically the

Table 8.3.2

Algorithm I test statistics—Timber harvest scheduling problems (minimum degree ordering heuristic)

Problem	Phase I iter.	Total iter.	Time (sec.)
FPK010	4	38	0.85
FPK040	4	40	1.10
FPK050	4	41	1.64
FPK080	5	49	2.95
FPK100	5	49	4.27
FPK150	5	49	6.47
FPK200	5	56	9.60
FPK300	6	52	14.99
FPK400	5	43	24.81
FPK500	6	67	34.11
FPK600	6	71	43.80

Table 8.3.3

MINOS 4.0 test statistics (total pricing)—Timber harvest scheduling problems

Problem	Partial pricing	Phase I iter.	Total iter.	Time (sec.)
FPK010	1	449	534	3.52
FPK040	1	579	736	5.63
FPK050	1	688	878	9.87
FPK080	1	1318	1574	28.81
FPK100	1	1521	1827	43.18
FPK150	1	2350	2768	93.77
FPK200	1	2867	3446	154.78
FPK300	1	4570	5407	351.82
FPK400	1	5375	7300	600.75
FPK500	1	7322	10156	1112.68
FPK600	1	3488	12009	1582.80

Table 8.3.4

MINOS (total pricing)/Algorithm I performance ratios—Timber harvest scheduling problems (minimum degree ordering heuristic in Algorithm I)

Problem	Partial pricing	MINOS/Alg. I iter. ratio	MINOS/Alg. I time/iter. ratio	MINOS/Alg. I time ratio
FPK010	1	14.05	0.2947	4.14
FPK040	1	18.40	0.2782	5.12
FPK050	1	21.41	0.2810	6.02
FPK080	1	32.12	0.3040	9.77
FPK100	1	37.29	0.2712	10.11
FPK150	1	56.49	0.2566	14.49
FPK200	1	61.54	0.2620	16.12
FPK300	1	103.98	0.2257	23.47
FPK400	1	169.77	0.1426	24.21
FPK500	1	151.58	0.2152	32.62
FPK600	1	169.14	0.2137	36.14

Table 8.3.5

MINOS 4.0 test statistics (default pricing)—Timber harvest scheduling problems

Problem	Partial pricing	Phase I iter.	Total iter.	Time (sec.)
FPK010	1	449	534	3.52
FPK040	1	579	736	5.63
FPK050	11	724	902	2.74
FPK080	16	973	1259	4.48
FPK100	19	1460	1832	7.39
FPK150	22	1860	2352	11.37
FPK200	25	2707	3369	17.94
FPK300	27	3244	4370	29.74
FPK400	28	4536	6035	49.13
FPK500	31	6934	9732	93.89
FPK600	31	3421	11364	123.62

computational performance of the algorithms. All runs were carried out on an IBM 3090 with same characteristics as in tests reported in Section 8.1. We use minimum degree ordering heuristic in Algorithm I.

We close this section with a few observations on the results of the runs with the timber harvest scheduling problems.

- The theoretical worst case analysis of Karmarkar's algorithm (Karmarkar, 1984) suggests that the number of iterations should grow linearly with the largest dimension of the coefficient matrix. This behavior is not confirmed for the variants of the algorithm described here. In this set of test problems, with number of columns ranging from 744 to 19 991, the number of iterations until convergence grows sublinearly, varying from 38 to 71, consistent with our earlier observation that the number of iterations grows slowly with problem size.

Table 8.3.6

MINOS (default pricing)/Algorithm I performance ratios—Timber harvest scheduling problems (minimum degree ordering heurstic in Algorithm I)

Problem	Partial pricing	MINOS/Alg. I iter. ratio	MINOS/Alg. I time/iter. ratio	MINOS/Alg. I time ratio
FPK010	1	14.05	0.2947	4.14
FPK040	1	18.40	0.2782	5.12
FPK050	11	22.00	0.0759	1.67
FPK080	16	25.69	0.0591	1.52
FPK100	19	37.39	0.0463	1.73
FPK150	22	48.00	0.0366	1.76
FPK200	25	60.16	0.0311	1.87
FPK300	27	84.04	0.0236	1.98
FPK400	28	140.35	0.0141	1.98
FPK500	31	145.25	0.0190	2.75
FPK600	31	160.06	0.0176	2.82

Table 8.3.7

MINOS 4.0 test statistics (improved pricing)—Timber harvest scheduling problems

Problem	Partial pricing	Phase I iter.	Total iter.	Time (sec.)
FPK010	104	48	1738	2.54
FPK040	128	137	812	1.52
FPK050	160	79	478	1.04
FPK080	184	113	333	1.13
FPK100	360	77	379	1.89
FPK150	480	105	542	2.96
FPK200	480	174	558	3.96
FPK300	440	455	924	6.61
FPK400	440	461	1049	8.41
FPK500	480	387	843	9.30
FPK600	480	729	1329	13.26

• This collection of test problems present an unusual behavior when solved with MINOS. Decreasing the number of columns considered in each pricing operation reduced not only the computation effort in each simplex operation, but also the total number of iterations. The number of pivots associated with solution paths followed by different versions of the simplex method may vary greatly. In this sense, interior point methods display a more robust behavior. This recommends caution in carrying out computational experiments involving the simplex method, as varying a single parameter among different runs of MINOS, the execution time for the two largest test problems is reduced by a factor of over 100.

• Our implementation of Algorithm I is faster than the most straightforward version of MINOS by a wide margin (see Table 8.3.3 and Figure 4) and is up to 2.8

Table 8.3.8

MINOS (improved pricing)/Algorithm I performance ratios—Timber harvest scheduling problems (minimum degree ordering heuristic in Algorithm I)

Problem	Partial Pricing	MINOS/Alg. I iter ratio	MINOS/Alg. I time/iter ratio	MINOS/Alg. I time ratio
FPK010	104	45.74	0.0653	2.99
FPK040	128	20.30	0.0681	1.38
FPK050	160	11.66	0.0544	0.63
FPK080	184	6.80	0.0564	0.38
FPK100	360	7.73	0.0572	0.44
FPK150	480	11.06	0.0414	0.46
FPK200	480	9.96	0.0414	0.41
FPK300	440	17.77	0.0248	0.44
FPK400	440	24.40	0.0139	0.34
FPK500	480	12.58	0.0217	0.27
FPK600	480	18.72	0.0162	0.30

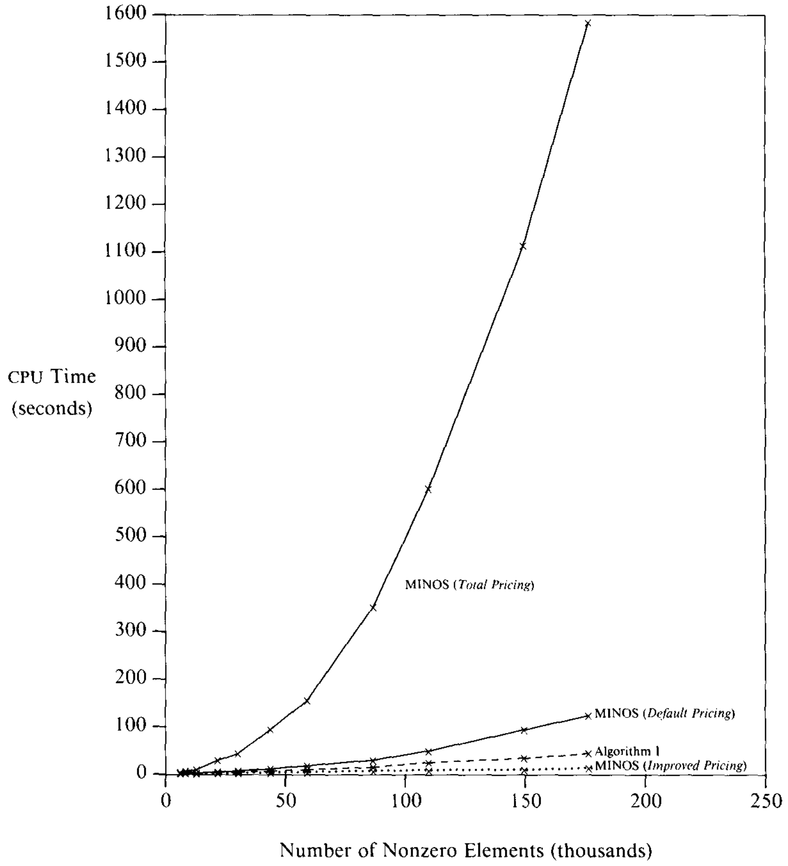

Fig. 4. MINOS 4.0 and Algorithm I (CPU times, IBM 3090)—Timber harvest problems (minimum degree ordering heuristic in Algorithm I).

times faster for runs using the default pricing strategy. However, MINOS runs up to 3 times faster when using the improved pricing strategy in a pricing strategy obtained after extensive experimentation.

• The objective values for the solutions obtained with our implementation of Algorithm I achieved 8 digits accuracy when compared to the values obtained by MINOS. Also a feasible primal solution was recovered at the end of the algorithm with the same degree of accuracy.

8.4. *Interpretation of computational experiments*

We make the following interpretation of the results of the computational experiments.

• Our implementation of Algorithm I attained 7 and 8 digit accuracy in the objective functional for most test problems without numerical difficulties. The data structures and programming techniques that led to these results are described in Adler et al. (1989).

• The selection of an initial interior solution as described in Section 6 plays a significant role in the fast convergence observed in all test problems. In this procedure, we attempt to start the algorithm far from faces of the feasible polyhedral set, avoiding the difficulties described by Megiddo and Shub (1986).

• Recovering a dual solution (primal solution in the format of the test problems) is an intricate proposition. Convergence of the dual solution estimates in (5.6) is guaranteed under nondegeneracy only, and examples can be built where it converges to an infeasible solution. However, our test results display good behavior for the dual solutions.

• The stopping criterion described in Section 6 resulted in the correct solution within the desired accuracy. An alternative criterion that checks dual complementarity properties could have been used without loss in efficiency.

• Using $\delta = 0.1$ in updating the $A^{\mathrm{T}}D_v^{-2}A$ matrix, Algorithm I obtained savings of over 10% in execution times, when compared to full updating, without degradation of the method's numerical stability.

• Algorithm I is sensitive to the density of the $A^{\mathrm{T}}D_v^{-2}A$ matrix. hence, besides being sparse, the A^{T} matrix should have a small number of dense columns. Maintaining this matrix sparse is essential to the algorithm.

9. Conclusion and extensions

The computational results presented in this paper illustrate the potential of interior point methods for linear programming. While the implementation described here was developed in a short period of time, it still outperforms MINOS 4.0 on the majority of problems tested.

For test problems from the NETLIB collection, solution time speed-ups 6 or higher are observed on most of mid-sized problems, and increase with problem size. Seven and eight digit accuracy in the objective function value was achieved on all test problems, except for problems *ScTap1*, *CzProb* and *25FV47*. An optimal complementary primal–dual pair was obtained for all test problems with exception of *CzProb* and *25FV47*. Numerical difficulties with the *LU* towards the end of the algorithm account for these inaccuracies.

Among the test problems is a set of multi-commodity network flow problems. The interior algorithm was clearly superior to MINOS and was also competitive with a specialized simplex algorithm, MCNF85. For the set of timber harvest scheduling problems, linear programs with significantly more variables than constraints, the number of iterations required by Algorithm I grows slowly with the number of variables. However, specialized pricing strategies make the simplex method a better tool for solving this type of linear programming problems.

The implementations of interior point algorithms described in this paper are still preliminary. Since many improvements are still possible this approach seems promising as a general purpose solver for large real-world linear programming problems. Of the several extensions planned to our implementation, some are immediately required:

- Implicit treatment of upper and lower bounds on the variables.
- Feasibility adjustment of the tentative primal solution computed at the end of the algorithm.

Other extensions planned are:

- Preprocessor for increasing sparsity of input matrices.
- Higher order approximations to the solution of system differential equations (3.9)–(3.11).
- Optimal basis identification for early termination.
- Bi-directional search for determining the step direction.
- Develop primal and primal–dual implementations.
- Include steps in potential function gradient direction.
- Implementations for special LP structures, e.g. network flows and GUB.
- Implementation for parallel computer architectures.

Acknowledgment

The authors acknowledge the help volunteered by several individuals in coding the numerous modules that make up the backbone of this implementation, which started as a joint class project for I. Adler's Fall 1985 *Large-Scale Systems* graduate seminar at Berkeley. In particular, W. Bein, D. Chou, C.C. Chyu, S. Cosares, M. de Carvalho, J. Doucet, J. Harrison, T. Kaplan, R.C. Monteiro, D. Popken, P.C. Samaratunga, A. Svoboda and Z. Wei were involved in the programming effort. We also wish to thank M. Khellaf and A. Sanstad for valuable discussions related to the implementation of Algorithm II. Furthermore, the authors are indebted to one of the anonymous referees for many constructive suggestions.

This research was partially funded by the Brazilian Council for the Development of Science and Technology—CNPq and Management Sciences Staff, US Forest Service—USDA.

References

I. Adler, N. 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(2) (1989).

I. Adler and R.C. Monteiro, "Limiting behaviour of the affine-scaling continuous trajectories for linear programming problems," Report ESRC 88-9, Engineering Systems Research Center, University of California (Berkeley, CA, 1988).

A.I. Ali and J.L. Kennington, "MNETGN program documentation," Technical Report IEOR 77003, Department of Industrial Engineering and Operations Research, Southern Methodist University (Dallas, TX, 1977).

J. Aronson, R. Barr, R. Helgason, J. Kennington, A. Loh and H. Zaki, "The projective transformation algorithm by Karmarkar: A computational experiment with assignment problems," Technical Report 85-OR-3, Department of Operations Research, Southern Methodist University (Dallas, TX, August 1985).

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

D.A. Bayer and J.C. Lagarias, "The nonlinear geometry of linear programming: I. Affine and projective rescaling trajectories," to appear in Transactions of the AMS (1989).

M.L. de Carvalho, "On the work needed to factor a symmetric positive definite matrix," Technical Report ORC 87-14, Operations Research Center, University of California (Berkeley, CA, 1987).

V. Chandru and B.S. Kochar, "A class of algorithms for linear programming," Research Memorandum 85-14, School of industrial Engineering, Purdue University (West Lafayette, IN, 1986).

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

J.J. Dongarra and E. Grosse, "Distribution of mathematical software via electronic mail," *Communications of the ACM* 30 (1987) 403-414.

I.S. Duff, A.M. Erisman and J.K. Reid, *Direct Methods for Sparse Matrices* (Claredon Press, Oxford, 1986).

S.C. Eisenstat, M.C. Gurshy, M.H. Schultz and A.H. Sherman, "The Yale sparse matrix package, I. The symmetric codes," *International Journal of Numerical Methods in Engineering* 18 (1982) 1145-1151.

D.M. Gay, "Electronic mail distribution of linear programming test problems," *Mathematical Programming Society Committee on Algorithms Newsletter* 13 (December 1985) 10-12.

D.M. Gay, "Electronic mail distribution of linear programming test problems," Numerical Analysis Manuscript 86-0, AT&T Bell Laboratories (Murray Hill, NJ, 1986).

P.E. Gill, W. Murray, M.A. Saunders, J.A. Tomlin and M.H. Wright, "On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method," *Mathematical Programming* 36 (1986) 183-209.

C. Gonzaga, "Interior point algorithms for linear programming problems with inequality constraints," Report ES-140/88, COPPE-Federal University of Rio de Janeiro (Rio de Janeiro, Brazil, 1988).

J.K. Ho and E. Loute, "A set of staircase linear programming test problems," *Mathematical Programming* 20 (1981) 245-250.

J.H. Hooker, "Karmarkar's linear programming algorithm," *Interfaces* 16 (1986) 75-90.

K.N. Johnson, "FORPLAN version 1: An overview," Technical Report, Land Management Planning-System Section, USDA, Forest Service (Fort Collins, CO, 1986).

N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorica* 4 (1984) 373-395.

N. Karmarkar, J. Lagarias, L. Slutsman and P. Wang, "Power series variants of Karmarkar type algorithms," Technical Report, AT&T Bell Laboratories (Murray Hill, NJ, 1989).

J. Kennington, "A primal partitioning code for solving multicommodity flow problems (version 1)," Technical Report 79008, Department of Industrial Engineering and Operations Research, Southern Methodist University (Dallas, TX, 1979).

I.J. Lustig, "A practical approach to Karmarkar's algorithm," Technical Report SOL 85-5, Systems Optimization Laboratory, Stanford University (Stanford, CA, 1985).

N. Megiddo and M. Shub, "Boundary behavior of interior point algorithms for linear programming," IBM Research Report RJ5319, Almadén Research Center (San Jose, CA, 1986).

B.A. Murtagh and M.A. Saunders, "MINOS user's guide," Technical Report 77-9, Systems Optimization Laboratory, Stanford University (Stanford, CA, 1977).

B.A. Murtagh and M.A. Saunders, "MINOS 5.0 user's guide," Technical Report 83-20, Systems Optimization Laboratory, Stanford University (Stanford, CA, 1983).

D.J. Rose, "A graph-theoretical study of the numerical solution of sparse positive definite systems of linear equations," in: R.C. Read, ed., *Graph Theory and Computing* (Academic Press, New York, 1972) pp. 183-217.

R.E. Tarjan, *Data Structures and Network Algorithms* (Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983).

M.J. Todd and B.P. Burrell, "An extension of Karmarkar's algorithm for linear programming using dual variables," *Algorithmica* 1 (1986) 409-424.

J.A. Tomlin, "An experimental approach to Karmarkar's projective method for linear programming," Manuscript, Ketron, Inc. (Mountain View, CA, 1985).

K. Tone, "An implementation of a revised Karmarkar method," Interim Report, Graduate School for Policy Science, Saitama University (Urawa, Saitama 338, Japan, 1986).

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

M. Yannakakis, "Computing the minimum fill-in is NP-complete," *SIAM Journal on Algebraic and Discrete Methods* 2 (1981) 77-79.