Exploiting hyper-sparsity in the revised simplex method

J.A.J. Hall K.I.M. McKinnon

December 1999

MS 99-014

Presented at 18th Biennial Conference on Numerical Analysis
Dundee, 1st July 1999

Department of Mathematics and Statistics
University of Edinburgh, The King's Buildings, Edinburgh EH9 3JZ
Tel. (33) 131 650 5075 E-Mail : jajhall@maths.ed.ac.uk, ken@maths.ed.ac.uk

Exploiting hyper-sparsity in the revised simplex method

J. A. J. Hall K. I. M. McKinnon

29th December 1999

Abstract

The revised simplex method is often the method of choice when solving large scale sparse linear programming problems, particularly when a family of closely-related problems is to be solved. In each iteration of the method, three matrix-vector products are formed and, even for many sparse problems, the result is typically dense. However it is demonstrated in this paper that for a significant number of practical problems, the result of one or more of these three operations is usually sparse. In this paper, this property of a problem is referred to as hyper-sparsity. Analysis of the commonly-used computational techniques for each of the components of the revised simplex method shows them to be inefficient when applied to problems which exhibit hyper-sparsity and techniques which exploit hyper-sparsity are developed. For such problems, the performance of the revised simplex method when using these techniques is demonstrated to be many times faster than a commercial implementation of both the revised simplex method and barrier method. When applied to network problems, the revised simplex method using these techniques is demonstrated to be comparable in speed to an efficient implementation of the network simplex method.

1 Introduction

Linear programming (LP) is a widely applicable technique both in its own right and as a sub-problem in the solution of other optimization problems. The revised simplex method and the barrier method are the two efficient methods for solving general large sparse LP problems. In a context where families of related LP problems have to be solved, such as in integer programming and decomposition methods, the revised simplex method is usually the more efficient method.

The computational components of the revised simplex method are identified in the remainder of this section and the property of hyper-sparsity is introduced. LP problems which exhibit hyper-sparsity when solved by the revised simplex method are identified by analysing an appropriate collection of test problems drawn from the standard Netlib set [6] and larger problems from the Kennington test set [2] and the authors' personal collection.

Analysis in Section 2 of the commonly-used computational techniques for each of the components of the revised simplex method shows them to be

inefficient when applied to problems which exhibit hyper-sparsity and techniques which exploit hyper-sparsity are developed. For such problems, when these techniques are implemented in the authors' revised simplex solver, EMSOL, the resulting performance of is demonstrated to be many times faster than both the revised simplex and barrier solvers in IBM's Optimization Subroutine Library, OSL [8] .

A significant class of LP problems which are well known to maintain sparsity are those with a near or complete network structure. Results given in Section 3 compare the performance of EMSOL and OSL with that of NETFLO (Kennington's efficient implementation of the network simplex method [9]) when applied to the NETGEN set of network LP problems [10]. The performance of EMSOL is seen to approach that of NETFLO whilst being up to an order of magnitude better than that of OSL. Conclusions and areas for future work are discussed in Section 4.

1.1 The revised simplex method

The simplex method and its computational requirements are most conveniently discussed in the context of LP problems in standard form

$$\begin{array}{rl} \text{minimize} & f = \boldsymbol{c}^T\boldsymbol{x} \\ \text{subject to} & A\boldsymbol{x} = \boldsymbol{b} \quad \boldsymbol{x} \geq \boldsymbol{0} \\ \text{where} & \boldsymbol{x} \in I\!R^n \quad \text{and} \quad \boldsymbol{b} \in I\!R^m. \end{array} \tag{1}$$

However, an efficient implementation of the revised simplex method should be able to solve general bounded LP problems of the form

$$\begin{array}{rl} \text{minimize} & f = \boldsymbol{c}^T\boldsymbol{x} \\ \text{subject to} & A\boldsymbol{x} - \boldsymbol{y} = \boldsymbol{0} \quad \boldsymbol{l}_x \leq \boldsymbol{x} \leq \boldsymbol{u}_x \quad \boldsymbol{l}_y \leq \boldsymbol{y} \leq \boldsymbol{u}_y, \end{array} \tag{2}$$

where some or all of the entries of $\boldsymbol{l}_x$, $\boldsymbol{u}_x$, $\boldsymbol{l}_y$ and $\boldsymbol{u}_y$ may be finite and nonzero, without explicitly adding artificial variables and constraints to convert the problem to standard form. For this reason, implementations of the revised simplex method generally assume that the whole of the identity matrix appears (implicitly) as part of the constraint matrix in (1). However, when operations corresponding to the matrix entries of these 'logical'—as opposed to 'structural'—variables are encountered computationally, an efficient solver should exploit their structure properly. Although the modifications to the simplex method which are required in order to treat general bounded variables do not change the general nature of the method, their consequences are identified below.

In the simplex method, the variables are partitioned into index sets $\mathcal{B}$ of m basic variables and $\mathcal{N}$ of $n - m$ nonbasic variables such that the basis matrix B formed from the columns of A corresponding to the basic variables, is nonsingular. The set $\mathcal{B}$ itself is conventionally referred to as the basis. The columns of A corresponding to the nonbasic variables form the matrix N and the components of $\boldsymbol{c}$ corresponding to the basic and nonbasic variables are referred to as, respectively, the basic costs $\boldsymbol{c}_B$ and non-basic costs $\boldsymbol{c}_N$. When the nonbasic variables are set to zero the values $\hat{\boldsymbol{b}} = B^{-1}\boldsymbol{b}$ of the basic variables, if non-negative, correspond to a vertex of the feasible region. It is readily shown that an optimal solution occurs at a vertex and the simplex method proceeds by

stepping from one vertex to another with lower objective value until an optimal vertex is determined.

When the equations are used to express $\boldsymbol{x}_B = \hat{\boldsymbol{b}} - B^{-1}N\boldsymbol{x}_N$, the basic variables may be eliminated from the objective to give reduced costs $\hat{\boldsymbol{c}}_N^T = \boldsymbol{c}_N^T - \boldsymbol{c}_B^T B^{-1}N$ for the nonbasic variables. Since the reduced costs are the Lagrange multipliers for the nonbasic variables, a sufficient condition for the vertex to be optimal is that the reduced costs are non-negative. If the vertex is not optimal, choosing to move along the edge of the feasible region corresponding to the most negative reduced cost will result in the most rapid reduction in the objective for a unit increase in the nonbasic variable. Since the change in the values of the basic variables corresponding to an increase in nonbasic variable x_q is given by $\boldsymbol{x}_B = \hat{\boldsymbol{b}} - x_q\hat{\boldsymbol{a}}_q$, the value of x_q corresponding to a step to the adjacent vertex of the feasible region is given by the ratio test

$$\alpha = \frac{\hat{b}_p}{\hat{a}_{pq}} \quad \text{where} \quad p = \underset{\hat{a}_{iq}>0}{\operatorname{argmax}} \frac{\hat{b}_i}{\hat{a}_{iq}}.$$

At the new vertex, the variable x_q enters the basis and the variable x_p becomes nonbasic.

However, if $1 + \|\hat{\boldsymbol{a}}_q\|^2$ is large relative to the corresponding measure for another nonbasic variable with similar reduced cost, it is likely that the corresponding step α will be smaller, resulting in a smaller reduction in the objective. For this reason, it is generally preferable to weight the squared reduced costs according to (an approximation to) $1 + \|\hat{\boldsymbol{a}}_q\|^2$ for some (semi-)norm. A popular technique used in commercial implementations of the the simplex method is the *Devex* strategy described by Harris in [7]. This technique maintains an approximation s_j to $1 + \|\hat{\boldsymbol{a}}_j\|_P^2$ for all $j \in \mathcal{N}$, where $\|.\|_P$ is a semi-norm corresponding to the 2-norm taken over an index set P. The weights s_j are updated following each simplex iteration using the vector $\hat{\boldsymbol{a}}_p^T = \boldsymbol{e}_p^T B^{-1}N$. It is readily shown that this vector may also be used to update the reduced costs according to the relation $\hat{\boldsymbol{c}}_N^T := \hat{\boldsymbol{c}}_N^T - \hat{c}_q\hat{\boldsymbol{a}}_p^T$. Note that the vectors $\hat{\boldsymbol{a}}_q$ and $\hat{\boldsymbol{a}}_p^T$ required in each iteration of the simplex method are the pivotal column and pivotal row of the standard simplex tableau, and that references in this paper to the pivotal column and pivotal row refer to these vectors.

Assuming the existence of a set of reduced costs, vector $\hat{\boldsymbol{b}}$ of current values of the basic variables and invertible representation of the basis matrix, the computational components of a typical iteration of the revised simplex method, as illustrated in Figure 1, are well-defined.

There are two particular situations when this typical form of a simplex iteration is modified. One occurs in 'phase I' when, as is generally done, a (piecewise linear) objective which penalises bound violations is used in order to determine a feasible point. If the step results in one or more non-pivotal basic cost changes, the corresponding linear combination of tableau rows must be computed in order to update the reduced costs. This is achieved by a BTRAN $\hat{\boldsymbol{\delta}}^T = \boldsymbol{\delta}^T B^{-1}$ and PRICE operation $\hat{\boldsymbol{a}}_\delta^T = \hat{\boldsymbol{\delta}}^T N$, where the nonzeros in $\boldsymbol{\delta}$ are the changes in the basic costs. Note that, when using Devex, the original 'unit' BTRAN and PRICE are still required so the 'multiple' BTRAN and PRICE constitute additional computation.

The other modification to the typical iteration occurs when x_q has distinct, finite, lower and upper bounds, and the ratio test identifies that the step should

`CHUZC:` Scan $\hat{\boldsymbol{c}}_N$ for a good candidate q to enter the basis.
`FTRAN:` Form $\hat{\boldsymbol{a}}_q = B^{-1}\boldsymbol{a}_q$, where $\boldsymbol{a}_q$ is column q of A.
`CHUZR:` Scan the ratios $\hat{b}_i/\hat{a}_{iq}$ for the row p of a good candidate to leave the basis, where $\hat{\boldsymbol{b}} = B^{-1}\boldsymbol{b}$ (ratio test). Let $\alpha = \hat{b}_p/\hat{a}_{pq}$.
Update $\hat{\boldsymbol{b}} := \hat{\boldsymbol{b}} - \alpha\hat{\boldsymbol{a}}_q$.
`BTRAN:` Form $\boldsymbol{\pi}^T = \boldsymbol{e}_p^T B^{-1}$.
`PRICE:` Form pivotal row $\hat{\boldsymbol{a}}_p^T = \boldsymbol{\pi}^T N$.
Update reduced costs $\hat{\boldsymbol{c}}_N^T := \hat{\boldsymbol{c}}_N^T - \hat{c}_q\hat{\boldsymbol{a}}_p^T$ and Devex weights.
`If` {growth in factors} `then`
`INVERT:` Form factored inverse of B.
`else`
`UPDATE:` Update the inverse of B corresponding to the basis change.
`end if`

Figure 1: Operations in an iteration of the revised simplex method with Devex pricing

correspond to a 'bound swap' for that variable. The basis does not change and so, unless there are basic cost changes as identified above in phase I, the reduced costs do not change and the iteration terminates following CHUZR. However, such iterations are readily exploited and it is convenient to assume in the discussion below that each simplex iteration results in a change of basis.

1.2 The representation of B^{-1}

In each iteration of the simplex method it is necessary to solve two systems, one involving the current basis matrix B and the other its transpose. This is achieved by passing forwards and backwards through the data structure corresponding to a representation of B^{-1}. When using product form or Schur complement UPDATE, B^{-1} is represented as

$$B^{-1} = E_U^{-1}B_0^{-1}, \tag{3}$$

where B_0^{-1} is obtained by INVERT, and E_U corresponds to the subsequent basis changes with its representation determined by UPDATE. Thus FTRAN is performed as

$$\tilde{\boldsymbol{a}}_q = B_0^{-1}\boldsymbol{a}_q \tag{4}$$

followed by

$$\hat{\boldsymbol{a}}_q = E_U^{-1}\tilde{\boldsymbol{a}}_q. \tag{5}$$

Conversely, BTRAN is performed as

$$\tilde{\boldsymbol{\pi}}^T = \boldsymbol{e}_p^T E_U^{-1} \tag{6}$$

followed by

$$\boldsymbol{\pi}^T = \tilde{\boldsymbol{\pi}}^T B_0^{-1}. \tag{7}$$

In a typical implementation of the revised simplex method for large sparse LP problems, B_0^{-1} is not formed explicitly but represented as a product of

elementary matrices derived from the LU decomposition of (a row and column permutation of) B_0. This invertible representation allows B_0^{-1} to be expressed algebraically as $B_0 = \prod_{k=1}^{r} E_k$, where

$$E_k = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & \boldsymbol{\eta}_k & & & \\ & & 1 & & & & \\ & & & \eta_k & & & \\ & & & & 1 & & \\ & & & \boldsymbol{\eta}_k & & \ddots & \\ & & & & & & 1 \end{bmatrix} \leftarrow \text{row } p_k \tag{8}$$

has its non-unit entries in column p_k. The value η_k is referred to as the **pivot** and operations with E_k^{-1} or its transpose may be performed using just η_k, the **eta vector** $\boldsymbol{\eta}_k$ and knowledge of the index p_k. Such an operation is conveniently referred to as 'applying the eta'. Within an implementation, the nonzeros in the vector $\boldsymbol{\eta}_k$ are stored as value-index pairs and the data structure $\{p_k, \eta_k, \boldsymbol{\eta}_k\}_{k=1}^{r}$ is known as the **eta file**.

The simplest form of eta file corresponds directly to the LU decomposition: there are $2m$ eta vectors and, for $k = 1, \ldots, m$ ($k = m+1, \ldots, 2m$), $\boldsymbol{\eta}_k$ consists of the entries below (above) the diagonal of column k ($2m - k + 1$) of L (U), η_k is the corresponding diagonal entry and p_k is k ($2m - k + 1$). However, in more general invertible representations, and when the product form update procedure is used to represent B^{-1} (see below), so-called Gauss-Jordan eta vectors may have nonzero entries both above and below the diagonal, and the number of eta vectors is denoted by r. In a particular LU decomposition, there will usually be trivial columns of L: those for which there are only zero entries below the diagonal. Since these trivial columns, in particular, those columns of L corresponding to logical variables, would correspond to $E_k = I$, they are omitted from the eta file. Less obviously, columns of U where the corresponding row has only zero entries to the right of the diagonal, may be amalgamated with the corresponding L-eta: any nonzero entries above the diagonal and the pivot are combined with the entries below the diagonal in the column of L to form a Gauss-Jordan eta. Eliminating trivial L-etas and amalgamating U-etas with the corresponding L-eta, when possible, can yield an eta file for B_0^{-1} with significantly fewer than $2m$ etas.

The eta file is determined by the INVERT operation as a result of performing Gaussian elimination, the operations of which usually result in nonzero entries being created in the active submatrix, an occurrence known as 'fill-in'. Consequently, INVERT procedures select pivots with the aim of limiting fill-in whilst achieving an acceptable degree of numerical stability. It is typical for the number of nonzero values in the eta file to be no greater than twice the number of nonzeros in B_0 despite the fact that its explicit inverse may be almost full.

An important property of the matrix B_0 when exploiting sparsity in INVERT, and when discussing hyper-sparsity, is the nature of the optimal block triangular ordering of the matrix which may be obtained by row and column permutations. This reordered matrix is optimal in that each diagonal block is irreducible. Such an ordering may be obtained by using the algorithm of Duff [4] to determine a row permutation P such that PB_0 has a transversal, and then using

Tarjan's algorithm [13], to identify a permutation Q such that $Q^T P B_0 Q$ is in optimal block triangular form. Each diagonal block corresponds to a strong component in the representation of PB_0 as a graph. This block triangular form, or an approximation to it, is determined either explicitly or implicitly by practical INVERT procedures. Elimination operations, and hence fill-in, are then restricted to the factors of any non-unit diagonal blocks. For certain LP problems, model characteristics mean that there is typically a strong component of dimension comparable to that of B_0. Other problems have diagonal blocks of very low dimension. In particular, the basis matrices for network LP problems can be re-ordered into strictly triangular form and so a representation for B_0^{-1} may be obtained structurally. If there are only a few small (non-unit) diagonal blocks, the number of etas in the corresponding eta file will be approximately equal to the number of structural variables in the basis, and have approximately the same number of nonzero entries as B_0.

Note that if the optimal block triangular form has a large strong component and if, as is likely, the initial RHS for FTRAN or BTRAN has a nonzero in a row corresponding to that component then its dimension is a lower bound on the number of nonzeros in the solution (if no zeros are created as a result of cancellation). A consequence of this observation is that if the result of FTRAN or BTRAN is typically sparse, it follows that the diagonal blocks in the optimal block triangular ordering of the matrix are small.

1.3 The product form and other update procedures

The product form update of Dantzig and Orchard-Hays [3] is the original and simplest form of update procedure. It represents E_U^{-1} as a product of elementary matrices of the form (8). The representation of each UPDATE operation is obtained directly from the pivotal column and is given by $\boldsymbol{\eta}_k = \hat{\boldsymbol{a}}_q - \hat{a}_{pq}\boldsymbol{e}_p$ and $\eta_k = -1/\hat{a}_{pq}$. In many solvers based on the product form, the representation of the UPDATE operations are appended to the eta file following INVERT, resulting in a single homogeneous data structure. The techniques developed in this paper are implemented within the authors' solver EMSOL which uses the product form update.

The product form update is commonly criticised for its lack of numerical stability and inefficiency with regard to sparsity. Other procedures include the Schur complement update [1] which retains the eta file for B_0^{-1} and represents UPDATE operations as a rectangular matrix of vectors resulting from (4) and a matrix decomposition of a square Schur complement. The column dimension of both of these matrices is equal to the number of columns in B_0 which are not in B. Clearly this figure is bounded by the number of updates which have been performed since INVERT, and it may be significantly less. The entries in the rectangular matrix are analogous to the eta vectors in the product form update but come from $\tilde{\boldsymbol{a}}_q$ (4) rather than $\hat{\boldsymbol{a}}_q$ (5). For these reasons, the Schur complement update procedure is more efficient with regard to sparsity and it can also be shown to have better numerical stability.

Other update procedures such as that due to Forrest and Tomlin [5] modify the representation of B_0^{-1} with respect to subsequent UPDATEs in order to gain still further efficiency with regard to sparsity, whilst having similar numerical stability properties to the Schur complement update.

1.4 Standard FTRAN and BTRAN

When the product form update is used, the FTRAN operation forms the pivotal column $\hat{\boldsymbol{a}}_q = B^{-1}\boldsymbol{a}_q$ by first scattering the vector $\boldsymbol{a}_q$ from its packed form in the constraint matrix into a zeroed workspace vector $\boldsymbol{b}$, and then transforming this into $\hat{\boldsymbol{a}}_q$ according to the algorithm represented as pseudo-code in Figure 2(a). Note that $\boldsymbol{b}$ is referred to as the RHS throughout this transformation process,

```
do k = 1, r
   if ( b_{p_k} ≠ 0 ) then
      b_{p_k} := b_{p_k} / η_k
      b := b − b_{p_k} η_k
   end if
end do
```

(a) FTRAN

```
do k = r, 1, −1
   b_{p_k} := (b_{p_k} + b^T η_k) / η_k
end do
```

(b) BTRAN

Figure 2: Standard FTRAN and BTRAN

with the cases $\boldsymbol{b} = \boldsymbol{a}_q$ and $\boldsymbol{b} = \hat{\boldsymbol{a}}_q$ distinguished by being referred to as the initial RHS and solution respectively.

In general when solving LP problems, the initial RHS is sparse and so b_{p_1} may be expected to be zero. In this case the multiple b_{p_k}/η_k of $\boldsymbol{\eta}_k$ which is added in to $\boldsymbol{b}$ is zero. Eventually, some b_{p_k} is usually nonzero, resulting in 'fill-in' in the $\boldsymbol{b}$. However, since testing b_{p_k} for zero is cheap relative to the floating point operations which would be otherwise performed, this is usually incorporated into an implementation of the revised simplex method.

When the BTRAN operation is used to form the 'unit' $\boldsymbol{\pi}$ vector required to form the pivotal row, the eta file is used to form $\boldsymbol{\pi}^T = \boldsymbol{e}_p^T B^{-1}$. This is achieved by setting to unity the appropriate component of a zeroed workspace vector $\boldsymbol{b}$ and then transforming it into $\boldsymbol{\pi}$ according to the algorithm represented as pseudo-code in Figure 2(b). Unlike FTRAN, there is no simple and immediate way to exploit sparsity in the RHS.

1.5 What is hyper-sparsity?

As identified above, the results of three matrix-vector products must be computed in each iteration of the revised simplex. These are the pivotal column $\hat{\boldsymbol{a}}_q = B^{-1}\boldsymbol{a}_q$ formed by FTRAN, the 'update $\boldsymbol{\pi}$' $\boldsymbol{\pi}^T = \boldsymbol{e}_p^T B^{-1}$ formed by BTRAN and the pivotal row $\hat{\boldsymbol{a}}_p^T = \boldsymbol{\pi}^T N$ formed by PRICE. In this paper, an LP problem is said to exhibit hyper-sparsity if, for at least one of these three operations, a clear majority of the results is sparse. A vector is considered to be sparse if no more than 10% of its entries are nonzero and a clear majority is take to be at least 60%.

The extent to which hyper-sparsity exists in LP problems was investigated for a subset of the standard Netlib test set [6] and larger problems from the Kennington test set [2] and the authors' personal collection. The problems excluded from the Netlib set were those whose solution requires less than one second of CPU, together with FIT1D and FIT2D for which, since they have only 25 and 26 rows respectively, the techniques developed in this paper are

inappropriate. Although the small Netlib problem SHELL is solved by EMSOL in less than a second, it is included for academic interest since it is well known to have a near network structure. Note that a standard simple scaling algorithm is applied to each of the problems for reasons of numerical stability.

When started from a basis of logical variables, the initial basis matrix is the identity so all FTRAN, BTRAN and PRICE operations for early iterations yield sparse results. To include the results of such iterations would flatter (and hence obscure) the gains which can be achieved by exploiting hyper-sparsity. In relation to their computational cost, such iterations also contribute disproportionally to the total number of iterations. Thus each problem is solved from a 'crash' basis obtained using a stabilisation of the algorithm described by Maros [11] which, starting from a logical basis, aims to remove logical variables which are fixed or have finite lower and upper bounds and introduce structural variables which are free or have only one finite bound. This is done whilst ensuring that the initial basis matrix is triangular and has no diagonal entries which are small relative to the other entries in the corresponding column.

The density of each pivotal column $\hat{\boldsymbol{a}}_q$ following FTRAN, update $\boldsymbol{\pi}$ following BTRAN and pivotal row $\hat{\boldsymbol{a}}_p^T$ following PRICE was determined, and the total number of each which was found to be sparse throughout the solution procedure was obtained. Table 1 lists the problems where for none of these three operations a clear majority of the results is sparse. The remaining problems exhibit hyper-sparsity to some extent and are listed in Table 2. Note that the sizes of the problems are given in Tables 6 and 5 respectively. For each of the three operations, the column headed '%' gives the percentage of the results which were sparse and the columns headed '% NZ' give the average density of the result in the cases when it is, respectively, sparse and dense. Note that these tables also include results for 'multiple' BTRAN and PRICE required to update the reduced costs in phase I. Omitting them would exaggerate the extent of hyper-sparsity in problems for which the number such operations is significant. The final column of Table 2 gives the initial letter of the operation for which a clear majority of the results is sparse.

The main observation from the results in Table 2 is that, since the result of FTRAN and PRICE is sparse for most problems, if one of the pivotal row and column is typically sparse then both are. This is unsurprising since the result of PRICE is a row of the same standard simplex tableau of which the result of FTRAN is a column. Indeed, it is interesting to consider the model-specific reasons why the results of all three operations are not typically sparse.

Since the update $\boldsymbol{\pi}$ is just a single row of B_0^{-1} and the pivotal column is usually a linear combination of several columns of B_0^{-1}, it might be expected that the $\boldsymbol{\pi}$ vector is less dense than $\hat{\boldsymbol{a}}_q$. It is, therefore, surprising that for only one problem, CYCLE, $\boldsymbol{\pi}$ is typically sparse and $\hat{\boldsymbol{a}}_q$ is not.

For problems DCP1 and DCP2, their pivotal columns are typically sparse, but not their pivotal rows. These problems are decentralised planning problems for which a typical standard simplex tableau is very sparse with a few dense rows. Thus the pivotal columns are usually sparse. However, the pivot is usually chosen from one of the dense rows.

Amongst the problems which are complementary to the DCPs in that their pivotal rows are typically sparse but not their pivotal columns, the most remarkable are FIT1P and FIT2P: almost all pivotal columns are essentially full

	FTRAN: $\hat{\boldsymbol{a}}_q$			BTRAN: $\boldsymbol{\pi}$			PRICE: $\hat{\boldsymbol{a}}_p^T$		
	Sparse		Dense	Sparse		Dense	Sparse		Dense
Problem	%	% NZ	% NZ	%	% NZ	% NZ	%	% NZ	% NZ
25FV47	3	2.0	58.9	13	0.5	67.9	16	1.7	90.2
BNL1	20	4.5	26.1	44	1.1	58.4	46	1.6	71.4
BNL2	36	2.8	28.9	36	0.9	37.0	40	1.2	57.8
D2Q06C	15	3.2	51.2	27	0.9	67.8	26	1.4	87.0
D6CUBE	2	6.7	65.2	11	0.7	94.7	13	1.5	75.7
DEGEN2	12	5.1	32.4	43	2.5	59.2	40	2.9	66.0
DEGEN3	18	5.5	24.5	49	1.6	59.0	52	1.8	67.7
DFL001	1	2.2	49.3	35	0.2	84.6	36	0.4	90.5
GREENBEA	13	4.3	26.3	57	0.4	75.9	59	0.8	75.6
GREENBEB	14	4.1	27.2	57	0.3	78.3	59	0.7	80.2
GROW15	1	5.5	82.7	7	1.8	87.4	5	6.4	88.1
GROW22	1	5.0	82.1	8	1.4	93.5	6	4.5	93.7
MAROS-R7	14	2.4	78.3	15	2.1	23.5	12	2.1	53.9
MODSZK1	15	4.3	62.2	26	1.1	66.0	31	1.6	69.8
NESM	34	1.8	39.8	25	4.9	36.6	28	3.7	47.5
PEROLD	6	4.0	64.7	24	1.3	78.3	27	1.7	89.0
PILOT	6	1.6	74.8	12	1.4	71.3	10	1.3	93.9
PILOT.JA	7	4.7	67.4	23	1.0	66.1	24	1.2	89.1
PILOT.WE	14	4.9	75.2	36	1.8	84.4	35	1.6	85.4
PILOT4	3	4.6	63.8	24	2.1	70.0	22	1.8	77.2
PILOT87	5	0.8	75.8	17	1.2	74.4	14	0.9	93.6
PILOTNOV	11	4.8	57.7	29	0.8	51.1	33	1.2	77.9
QAP8	0	2.5	81.7	12	0.6	80.4	14	2.2	98.6
SCSD8	8	3.1	41.9	34	1.4	63.7	45	2.8	44.8
STAIR	2	6.6	79.5	20	0.6	63.5	20	1.8	64.4
TRUSS	5	3.1	48.1	40	0.7	91.5	53	1.6	67.6
WOOD1P	1	0.8	50.4	53	1.3	94.9	46	0.8	85.8

Table 1: Sparsity following FTRAN, BTRAN and PRICE for problems not exhibiting hyper-sparsity

and almost all pivotal rows are sparse. For these two problems, most columns of the constraint matrix have only one nonzero entry, with the remainder being very dense. Thus B^{-1} is largely diagonal with a small number of essentially full columns. Most variables chosen to enter the basis have a single nonzero entry in a row corresponding to these dense columns, and so the pivotal column is (a multiple of) one of these dense columns of B^{-1}. Each update $\boldsymbol{\pi}$ is a row of B^{-1} and its resulting sparsity is inherited by the pivotal row since most columns of N in the PRICE operation have only one nonzero entry.

2 Exploiting hyper-sparsity

Each computational component of an iteration of the revised simplex method either forms, or operates with, the result of a matrix-vector product. Each of these components is considered below and it is shown that typical computational

	FTRAN: $\hat{\boldsymbol{a}}_q$			BTRAN: $\boldsymbol{\pi}$			PRICE: $\hat{\boldsymbol{a}}_p^T$			
	Sparse		Dense	Sparse		Dense	Sparse		Dense	>60%
Problem	%	% NZ	% NZ	%	% NZ	% NZ	%	% NZ	% NZ	Sparse
80BAU3B	97	2.5	10.7	71	0.4	47.7	73	0.6	49.5	FBP
CYCLE	50	6.6	14.4	85	3.6	20.4	57	3.1	17.6	B
CZPROB	100	1.3	—	66	0.5	83.3	67	0.8	71.5	FBP
FIT1P	0	—	99.2	99	4.9	10.3	100	3.8	—	BP
FIT2P	14	0.0	98.9	100	1.1	—	100	0.8	—	BP
GANGES	70	3.6	14.8	92	1.8	13.1	78	1.2	27.5	FBP
MAROS	30	3.6	27.5	61	1.0	39.7	64	2.3	53.0	BP
SCFXM3	67	3.5	18.1	71	2.1	22.8	70	2.8	23.9	FBP
SCTAP3	100	1.3	—	100	0.5	—	93	2.2	15.5	FBP
SHELL	100	2.3	—	87	1.1	30.4	91	1.8	15.5	FBP
SHIP08L	100	1.0	—	84	0.3	72.9	85	0.8	23.1	FBP
SHIP12L	100	0.5	—	91	0.2	59.4	91	0.5	20.2	FBP
SHIP12S	100	0.6	—	89	0.4	31.6	89	0.8	26.0	FBP
SIERRA	100	1.9	—	98	0.8	16.2	98	1.3	14.0	FBP
STOCFOR2	24	1.8	54.7	69	2.9	16.0	66	3.2	16.4	BP
STOCFOR3	47	0.3	68.0	100	1.2	—	100	0.9	—	BP
WOODW	41	4.8	18.2	77	0.6	65.6	76	1.2	63.4	BP
DCP1	92	5.1	10.7	55	0.9	15.2	49	1.3	79.2	F
DCP2	100	1.3	—	57	0.7	14.6	52	0.3	81.8	F
DETEQ8	100	0.9	10.3	100	0.1	—	100	0.2	—	FBP
DETEQ27	95	1.2	12.6	100	0.1	—	100	0.1	—	FBP
PDS-02	100	0.7	—	100	0.7	12.9	100	1.0	12.2	FBP
PDS-06	100	1.0	—	97	0.4	17.5	98	0.5	15.1	FBP
PDS-10	100	1.1	—	97	0.2	23.5	97	0.3	22.0	FBP
PDS-20	100	2.1	—	93	0.2	47.5	94	0.3	35.6	FBP

Table 2: Sparsity following FTRAN, BTRAN and PRICE for problems exhibiting hyper-sparsity

techniques are inefficient in the presence of hyper-sparsity. In each case, equivalent computational techniques are developed which exploit hyper-sparsity. Although each individual technique can result in a significant improvement in performance for the corresponding computational component, the effect on the solution time for the problem depends on that of components for which the techniques cannot be applied. Although the overall improvement in performance will be seen to be modest for problems which do not exhibit hyper-sparsity in both FTRAN and BTRAN, for those which do the performance improvement will be seen to be tremendous.

2.1 Relative cost of computational components

Before considering the consequences of hyper-sparsity and techniques by which it can be exploited for each of the computational components of the revised simplex method, it is interesting to consider the extent to which these components contribute to the solution time. Table 3 gives the percentage of CPU which can

be attributed to each of the major computational components in the revised simplex method. This is given for those problems which exhibit hypersparsity for which the CPU time for each of the computational components is sufficiently great to be determined reliably. Note that for both FTRAN and BTRAN the percentage CPU time for the operations (4) and (7) with the INVERT etas, and (5) and (6) with the UPDATE etas, are presented separately in the columns headed I-FTRAN, I-BTRAN, U-FTRAN and U-BTRAN. For some of the problems the frequency with which INVERT is performed is far from being optimal, in that the percentage of CPU attributable to applying UPDATE etas and the percentage of CPU attributable to INVERT are far from being equal. This is done for reasons of direct comparison with the results obtained when the techniques for exploiting hyper-sparsity are used.

2.2 Hyper-sparse FTRAN

For problems with sparse pivotal columns, since there is unlikely to be more than a few hundred UPDATE etas, the major computational cost of FTRAN is the operation (4) which forms $\tilde{\boldsymbol{a}}_q = B_0^{-1}\boldsymbol{a}_q$. This is seen in Table 3 and is particularly marked for the later, larger problems.

When the pivotal column $\hat{\boldsymbol{a}}_q$ computed by FTRAN is sparse, unless there is an extraordinary amount of cancellation, it is expected that only a very small proportion of the INVERT (and UPDATE) eta vectors, needs to be applied. Further, since the number of floating point operations required to perform these few operations can be expected to be of the same order as the number of nonzeros in $\hat{\boldsymbol{a}}_q$, the cost of FTRAN when forming $\tilde{\boldsymbol{a}}_q = B_0^{-1}\boldsymbol{a}_q$ will be dominated by the test for zero when using the standard algorithm illustrated in Figure 2(a). Note that if trivial L-etas are not eliminated and U-etas not amalgamated when possible, resulting in an INVERT eta file with $2m$ etas, the number of tests for zero, and hence the cost of FTRAN, is double what is necessary. The aim of this subsection is to develop a computational technique which identifies the etas which (may) have to be applied without passing through the whole INVERT eta file and testing each value of b_{p_k} for zero.

In developing the algorithm which is illustrated as pseudo-code in Figure 3, observe first that, corresponding to the indices of the nonzeros in the initial RHS, there is a set $\mathcal{K}$ of indices k of etas for which b_{p_k} is nonzero. Note that since there is at most one L-eta and at most one U-eta in a given row, $|\mathcal{K}|$ is at most twice the number of nonzeros in the initial RHS.

If $\mathcal{K}$ is empty then no etas need to be applied so (4) is complete. Otherwise, the least index $k_0 \in \mathcal{K}$ identifies the skip through the eta file to the first eta which needs to be applied. Once this has been done, there is a set $\mathcal{K}'$ corresponding to the updated RHS. The set $\mathcal{K}'$ differs from the initial set $\mathcal{K}$ in that index k_0 is removed and, as a result of any fill-in, new indices ($k > k_0$) may have been introduced. These observations are formalised and generalised as the pseudo-code illustrated in Figure 3, where $\mathcal{E}_k$ is used to denote the set of indices of the nonzeros in $\boldsymbol{\eta}_k$. The lists $P^{(1)}$ and $P^{(2)}$ record, for each row, the index of the first and second eta which have a pivot in the particular row, with a zero index being recorded if there are fewer than two such etas.

Since the set $\mathcal{K}$ must be searched to determine the next eta to be applied, there is some scope for variation in the way that this is achieved and, if the number of entries in $\mathcal{K}$ becomes large, there comes a point at which the cost

Problem	CHUZC	I-FTRAN	U-FTRAN	CHUZR	I-BTRAN	U-BTRAN
80bau3b	20.51	4.73	1.12	1.07	7.57	1.46
cycle	4.52	8.54	5.40	4.40	20.85	6.16
fit1p	5.09	2.31	14.35	6.48	14.81	21.30
fit2p	8.41	3.31	6.30	9.11	10.42	22.41
maros	3.93	7.02	4.78	1.12	15.73	10.67
scfxm3	6.55	7.42	2.18	6.11	23.14	4.80
ship12l	16.02	5.11	0.70	0.53	10.39	1.23
stocfor2	3.22	6.93	5.94	8.66	12.13	22.28
stocfor3	3.90	6.11	3.47	6.46	16.17	27.58
woodw	11.90	2.71	1.45	1.57	7.75	2.08
dcp1	3.45	4.28	3.55	1.65	13.51	3.64
dcp2	3.90	3.06	1.69	0.28	13.95	1.40
deteq8	11.62	6.98	0.37	0.76	16.64	2.69
deteq27	11.17	6.65	0.31	0.75	16.71	4.24
pds-02	15.48	6.35	0.17	1.48	14.78	4.35
pds-06	14.48	7.37	0.48	0.76	15.10	3.69
pds-10	13.68	6.59	0.33	0.62	14.65	3.31
pds-20	13.34	5.15	0.38	0.74	11.67	4.03

				INVERT frequency		Solution
Problem	PRICE	INVERT	UPDATE	Used	Optimal	CPU (s)
80BAU3B	54.92	1.69	0.91	50	40	52.01
CYCLE	30.40	9.92	1.63	50	45	6.96
FIT1P	13.43	3.70	2.31	50	16	1.89
FIT2P	30.27	2.39	2.71	50	14	199.84
MAROS	30.34	5.62	1.97	50	30	3.10
SCFXM3	25.76	8.73	2.62	50	55	1.92
SHIP12L	60.04	1.41	1.94	50	42	5.31
STOCFOR2	15.35	3.22	1.98	50	17	3.63
STOCFOR3	16.57	11.29	2.02	50	30	315.99
WOODW	65.74	1.32	0.38	50	31	14.94
DCP1	55.12	8.64	0.85	50	54	65.60
DCP2	52.49	20.45	0.14	100	257	3484.95
DETEQ8	55.48	1.23	0.15	100	63	495.65
DETEQ27	55.19	1.28	0.10	100	53	6470.06
PDS-02	51.91	0.52	0.61	200	68	10.65
PDS-06	53.02	0.71	0.25	200	82	311.26
PDS-10	56.99	0.71	0.17	200	88	1341.13
PDS-20	60.80	1.64	0.10	200	121	19092.51

Table 3: Relative cost of computational components of the revised simplex method for problems which exhibit hyper-sparsity

$\mathcal{R} = \{i : b_i \neq 0\}$
$\mathcal{K} = \{k : b_{p_k} \neq 0\}$
`repeat`
  $k_0 = \min_{k \in \mathcal{K}}$
  $b_{p_{k_0}} := b_{p_{k_0}} / \eta_{k_0}$
  `for` $i \in \mathcal{E}_{k_0}$ `do`
    `if` $(b_i \neq 0)$ `then`
      $b_i := b_i + b_{p_{k_0}} [\boldsymbol{\eta}_{k_0}]_i$
    `else`
      $b_i := b_{p_{k_0}} [\boldsymbol{\eta}_{k_0}]_i$
      `if` $(P_i^{(1)} > k_0)$ $\mathcal{K} := \mathcal{K} \cup P_i^{(1)}$
      `if` $(P_i^{(2)} > k_0)$ $\mathcal{K} := \mathcal{K} \cup P_i^{(2)}$
      $\mathcal{R} := \mathcal{R} \cup i$
    `end if`
  `end do`
`until` $\mathcal{K} = \emptyset$

Figure 3: Hyper-sparse FTRAN for INVERT etas

of the search exceeds the cost of the tests for zero which it seeks to avoid. In EMSOL, $\mathcal{K}$ is maintained as an unordered list and the average skip through the eta file which has been achieved during the current FTRAN is compared with a multiple of the number of entries in $\mathcal{K}$ to determine the point at which it is preferable to complete FTRAN using the standard algorithm. The following alternative strategies are expected to form the subject of future research. By maintaining $\mathcal{K}$ as a set of buckets, each corresponding to some portion of the eta file, it may be sufficient to search the local bucket to determine the next eta to be applied. The set $\mathcal{K}$ could also be maintained as a heap.

Although the algorithm in Figure 3 does not require the list $\mathcal{R}$ of indices of entries in the RHS which may be nonzero, the operations required to maintain it are included. It is shown below that knowledge of this set can be advantageous during CHUZR.

2.3 Hyper-sparse CHUZR

For problems when the pivotal column is typically sparse, the cost of performing a test for zero for each of the m entries will dominate the small total number of floating point operations which are performed for the few nonzeros in the pivotal column. If a list of indices of entries in the pivotal column which may be nonzero is known, then this overhead is avoided. The nonzero entries in the pivotal column are also required both to update the values of the basic variables following CHUZRand, as described below, to update the product form UPDATE eta file. If the nonzero entries in the workspace vector used to compute the pivotal column are zeroed after being packed onto the end of the UPDATE eta file, this yields a contiguous list of real values to update the values of the basic variables and makes the UPDATE operation near-trivial. A further consequence is that, so long as pivotal columns remain sparse, the only complete pass through the workspace vector used to compute the pivotal column is that required to zero it before the first simplex iteration.

2.4 Hyper-sparse BTRAN

When performing BTRAN using the algorithm illustrated in Figure 2(b), most of the work when applying an eta is the evaluation of the inner product $\boldsymbol{b}^T\boldsymbol{\eta}_k$, the result of which will frequently be (structurally) zero when the RHS is sparse. Then, only if b_{p_k} is nonzero, is there any non-trivial floating point operation. Unfortunately, there is no simple way of determining whether there is a non-empty intersection of the sparsity pattern of $\boldsymbol{b}$ and $\boldsymbol{\eta}_k$ without a computational overhead which is comparable to evaluating the inner product itself. However, worthwhile computational savings follow from the observation that, when applying $\boldsymbol{\eta}_k$ during BTRAN, fill-in can only occur in component p_k in the RHS.

2.4.1 Maintaining a list of the indices of nonzeros in the RHS

During BTRAN, it is simple and cheap to maintain a list of the indices of the nonzeros in the RHS: if b_{p_k} is zero and $\boldsymbol{b}^T\boldsymbol{\eta}_k$ is nonzero then the index p_k is added to the end of a list. If b_{p_k} is nonzero and $b_{p_k}+\boldsymbol{b}^T\boldsymbol{\eta}_k$ is zero then it is said that 'cancellation' occurs, in which case it is desirable, although not essential, for the index p_k to be removed from the list. For problems when $\boldsymbol{\pi}$ is frequently sparse, knowing the indices of all values in the RHS which are (or may be) nonzero permits a valuable saving during the PRICE operation, as identified below.

2.4.2 Reducing trivial inner products and operations with zero

When using the product form update, it is valuable to consider the operations with the UPDATE etas (6) separately from those with INVERT etas (7). For the former, it is possible to eliminate the structurally trivial inner products and significantly reduce the number of operations with zero. For the latter, it is possible to eliminate a significant number of trivial inner products.

UPDATE etas

Let K denote the number of UPDATE operations which have been performed and let $\mathcal{P}$ denote the set of indices of those rows which have been pivotal since INVERT. Note that the inequality $|\mathcal{P}| \leq K$ is strict if a particular row has been pivotal more than once. Since fill-in during BTRAN can only occur in row p_k, it follows that the nonzeros in $\tilde{\boldsymbol{\pi}}^T = \boldsymbol{e}_p^T E_U^{-1}$ are restricted to the components with indices in the set $\mathcal{P}$. Thus, when applying $\boldsymbol{\eta}_k$, only the nonzeros with indices in the set $\mathcal{P}$ contribute to $\boldsymbol{b}^T\boldsymbol{\eta}_k$. Since $|\mathcal{P}|$ is very much smaller than the dimension of B for large problems, it follows that unless this observation is exploited, most of the floating point operations using the UPDATE etas are still trivial. A significant degree of efficiency is thus achieved by maintaining a rectangular array $\mathtt{E_P}$ of dimension $|\mathcal{P}| \times K$ which holds the values of the entries corresponding to set $\mathcal{P}$ in the UPDATE etas, allowing $\tilde{\boldsymbol{\pi}}$ to be formed as a sequence of K short, dense, inner products.

When using this technique, even if $\mathtt{E_P}$ is full, half the floating point operations are trivial since the initial RHS has only one nonzero, and at most one nonzero is created as a result of applying an eta. If the update etas are sparse then $\mathtt{E_P}$ will be largely zero and so will most of the inner products $\boldsymbol{b}^T\boldsymbol{\eta}_k$. This may be

exploited by searching, for each nonzero in the RHS, for the eta which is the first (from the end of the file) with a nonzero in that row. The earliest such eta is then identified as the first which may result in a nonzero value of $\boldsymbol{b}^T\boldsymbol{\eta}_k$. The overhead of maintaining the data structure and the extra work of performing this search is usually much less than the computation which is avoided. Indeed, the initial search frequently identifies that none of the update etas need be applied. Note that this technique is of benefit whether or not the update etas are sparse, although it represents a relatively small saving compared to the overall cost of performing a simplex iteration.

INVERT etas

By being able to identify the nonzeros in UPDATE etas for a given row, is is shown above that there is a significant reduction in the cost of forming $\tilde{\boldsymbol{\pi}}^T = \boldsymbol{e}_p^T E_U^{-1}$ (6). Indeed, if there are no nonzeros in row p of the UPDATE etas, it follows immediately that $\tilde{\boldsymbol{\pi}} = \boldsymbol{e}_p$. This technique may also be applied to the INVERT etas if the list $Q^{(1)}$ of the indices of the last INVERT eta with a nonzero in each row is known, with an index of zero used to indicate that there is no such eta. The greatest index in $Q^{(1)}$ corresponding to the nonzeros in $\tilde{\boldsymbol{\pi}}$ then indicates the first INVERT eta which must be applied. As with the UPDATE etas, the technique may indicate that a significant number of the INVERT etas need not be applied and, if the index is zero, it follows immediately that $\boldsymbol{\pi} = \tilde{\boldsymbol{\pi}}$. More generally, if the list $Q^{(k)}$ of the index of the k^{th} last INVERT eta with a nonzero in each row is recorded for k from 1 to some small limit then several significant steps backwards through the INVERT eta file may be made. However, to implement this technique requires an integer array of dimension equal to that of B_0 for each list so, in EMSOL, only $Q^{(1)}$ and $Q^{(2)}$ are recorded.

2.4.3 Row-wise INVERT eta file

The limitations and/or high storage requirements associated with exploiting hyper-sparsity during BTRAN with the conventional column-wise (INVERT) eta file motivate the formation, after INVERT of an equivalent representation stored row-wise. This may be formed by passing twice through the complete column-wise INVERT eta file and permits the BTRAN operation (7) to be performed using the algorithm given in Figure 3 for FTRAN. For problems in which $\boldsymbol{\pi}$ is typically sparse, although the computational overhead in forming the row-wise eta file is significant, it is far outweighed by the savings when applying it.

2.5 Hyper-sparse PRICE

The matrix-vector product $\boldsymbol{\pi}^T N$ is easily formed as a sequence of inner products between $\boldsymbol{\pi}$ and the appropriate columns of the constraint matrix. In the case when $\boldsymbol{\pi}$ is full, there will be no trivial floating-point operations so this simple technique is optimal. However this is far from being true if $\boldsymbol{\pi}$ is sparse, in which case, by forming $\boldsymbol{\pi}^T N$ as a linear combination of those rows of N which correspond to nonzero entries in $\boldsymbol{\pi}$, all trivial floating point operations are avoided. Although the cost of maintaining a row-wise representation of the columns of N is non-trivial, this is far outweighed by the efficiency with which $\boldsymbol{\pi}^T N$ may be formed.

For problems when $\boldsymbol{\pi}$ is particularly sparse, the cost of testing each entry for zero dominates the small total number of floating point operations which are performed for the few nonzeros in $\boldsymbol{\pi}$. Thus, as with the pivotal column in the case of CHUZR, when the indices of the entries in $\boldsymbol{\pi}$ which may be nonzero are known (as a result of this list being maintained during BTRAN) the cost of searching for the nonzeros in $\boldsymbol{\pi}$ is avoided. As nonzero entries are encountered, the corresponding workspace entry is zeroed, leaving the workspace zeroed in preparation for the next BTRAN. Thus, as with the workspace vector used to compute the pivotal column, so long as $\boldsymbol{\pi}$ vectors remain sparse, the only complete pass through this workspace array when solving an LP problem is that required to zero it before the first simplex iteration.

Since the the row-wise PRICE driven by the indices of nonzeros in $\boldsymbol{\pi}$ described above avoids any trivial operations, this technique is optimal. However, it is advantageous if the list of indices of nonzeros in the pivotal row is maintained during PRICE, so long as the vector remains sparse. Knowledge of this list eliminates the overhead of searching for the nonzeros in the pivotal row, which would otherwise be the dominant cost when updating the reduced costs and Devex weights. As with the workspace vectors used to compute $\boldsymbol{\pi}$ and the pivotal column, so long as pivotal rows remain sparse, the list of indices of nonzeros enables a zeroed workspace vector to be maintained with just a single full pass required to zero it before the first simplex iteration.

2.6 Hyper-sparse CHUZC

Before discussing methods for CHUZC which exploit hyper-sparsity, it should be observed that, since the vector $\boldsymbol{c}_B$ of basic costs may be full, the vector of reduced costs given by

$$\hat{\boldsymbol{c}}_N^T = \boldsymbol{c}_N^T - \boldsymbol{c}_B^T B^{-1} N,$$

may also be full. Further, for most of the solution time, a significant proportion of the reduced costs are negative. Thus, even for LP problems exhibiting hyper-sparsity, the attractive nonbasic variables do not form a small set whose size could then be exploited. However, if the pivotal row is sparse, the number of reduced costs and edge weights which change each iteration is small and it is this which may be exploited to improve the efficiency of CHUZC.

The aim of the hyper-sparse CHUZC algorithm described below is to maintain a list of the most attractive candidates to enter the basis. This is achieved by first performing an initial complete CHUZC to determine a list C_0 of the best s candidates, the best of which is chosen to enter the basis. For the subsequent pivotal row, a list D_0 is formed of the best s candidates not in C_0 whose reduced cost has changed. The best s candidates from both of these lists is used to determine a new list of candidates C_1, the best of which is chosen to enter the basis in the next iteration. Unless this scheme is reset periodically by performing a complete CHUZC, the simplex method could terminate prematurely. For example, candidates which are initially attractive, but not sufficiently so to be included in C_0, and whose reduced cost does not change subsequently, could never be chosen to enter the basis.

Even with this reset mechanism, it is possible that all the candidates in some list C_k may be inferior to some which were not sufficiently attractive to be included in lists C_j or D_j for $j < k$. Thus the variable to enter the basis chosen

from those in C_k is not that which would be chosen by a complete CHUZC. This deviation from equivalence with the revised simplex method when using complete CHUZC is theoretically inelegant and, in practice, leads to a significant increase in the number of iterations required to solve some problems.

The following modification to this algorithm yields a hyper-sparse CHUZC which determines as good a candidate as a complete CHUZC. This modification is based on maintaining a lower bound on the reduced cost (weighted by Devex) of the best candidate not in C_k. Whenever a list C_k or D_k is formed, the weighted reduced cost of its least attractive candidate provides a lower bound on the corresponding value for the most attractive candidate rejected in forming that list. The least such lower bound over all lists C_j and D_j $(j < k)$ is a lower bound on the weighted reduced cost of the best candidate not in C_k. If this value exceeds the weighted reduced cost of the *best* candidate in C_k, then it is possible that a complete CHUZC would determine a better candidate. If this event is used to trigger a reset of C_0 then the hyper-sparse CHUZC guarantees to find as good a candidate as a complete CHUZC.

2.7 Hyper-sparse (preordered) INVERT

The default INVERT in EMSOL is based on the procedure used by Tomlin in LPM1 and a description is given by Pfefferkorn and Tomlin [12]. This procedure identifies, and uses as pivots for as long as possible, rows and columns in the active submatrix which have only a single nonzero. Following this triangularisation phase, any residual active submatrix is then factorised using Gaussian elimination with the order of the columns fixed according to a merit count. Since the pivot in each stage of Gaussian elimination is selected from a predetermined pivotal column, only this column of the active submatrix is required. Thus, rather than apply elimination operations to maintain the up-to-date active submatrix, the up-to-date pivotal column is formed each iteration. When compared with a Markowitz-based procedure which maintains and selects the pivot from the whole up-to-date active submatrix, the Tomlin procedure has a greatly simplified data structure management and pivot search strategy. Thus, for problems where the diagonal blocks in the optimal block triangular form are small, the Tomlin INVERT is significantly faster and yields an INVERT eta file which is no larger than that obtained by a Markowitz INVERT.

The up-to-date pivotal column which is computed in each stage of Gaussian elimination is formed by passing forwards through the file of L-etas computed up to that stage. Even for problems where the pivotal column of the standard simplex tableau is rarely sparse, the pivotal column of the active submatrix during Gaussian elimination is very likely to be sparse. Thus this partial FTRAN operation is particularly amenable to the exploitation of hyper-sparsity. Indeed, it was in the context of INVERT that the authors first observed the dominant overhead of the test for zero in FTRAN-like operations. Note that the data structures require to exploit hyper-sparsity during FTRAN itself, is generated at almost no cost during the course of INVERT.

It is interesting to observe that since the improvement in performance of INVERT is dependent on the residual active submatrix after the triangularisation phase to be large enough for its factorisation cost to be dominant, significant improvement in INVERT is unlikely to coincide with significant improvements in the performance of BTRAN, FTRAN and PRICE as a result of exploiting

hyper-sparsity.

2.8 Hyper-sparse (product-form) UPDATE

The product-form UPDATE requires the nonzeros in the pivotal column to be stored in packed form with the pivot stored as its reciprocal (so that the divisions in FTRAN and BTRAN are effected by multiplication. As identified above, the former is readily achieved during CHUZR and the latter is a scalar operation performed afterwards.

2.9 Controlling the use of techniques to exploit hyper-sparsity

The techniques described above are inefficient in the absence of hyper-sparsity and so should not be applied universally. For problems which do not exhibit hyper-sparsity at all, or for problems where a particular computational component does not exhibit hyper-sparsity, this is easily recognised by monitoring a running average of the density of the result over a number of iterations and switching off the technique for all subsequent iterations if hyper-sparsity is seen to be absent. For a computational component which typically exhibits hyper-sparsity, it is important to identify the situation where the result for a particular iteration is not going to be sparse, and switch to the standard algorithm which will then be more efficient. This is usually achieved by monitoring the density of the result during the operation and switching on some tolerance. Practical experience has shown that the optimal value of such tolerances is very insensitive.

2.10 Results

The efficacy of the techniques described in this section may be judged from the results presented in Tables 4, 6 and 5 which were obtained using EMSOL on a Sun UltraSPARC 140 with 128Mb of memory. The design of EMSOL allows the user to set the value of control variables to indicate that the use of particular computational techniques is either prohibited or forced. However, the default is for the solver to determine (at run-time) when use of a particular technique is appropriate. Although tuning of such control parameters is justified when many problems of a particular nature are to be solved, the results in this paper were obtained with EMSOL running in its default state, unless stated otherwise.

Table 4 gives, for the problems in Table 3, a detailed breakdown of the percentage decrease in the CPU time which is attributable to applying the INVERT etas in FTRAN, applying the INVERT and UPDATE etas in BTRAN, and performing the PRICE operation. For FIT1P and FIT2P, the time taken applying the INVERT etas during FTRAN actually increases. This may be due to the control mechanism not switching off the hyper-sparse technique soon enough and warrants further investigation. However, for the other problems and other operations, the savings correlate pretty well with the proportion of operations in Table 2 which yield sparse results. This indicates that the cost of these operations when the result is sparse has been vastly reduced leaving the total cost of these operations to be dominated, in general, by the cost of those for which the result is not sparse.

Problem	I-FTRAN	I-BTRAN	U-BTRAN	PRICE
80BAU3B	76.73	69.32	77.65	81.90
CYCLE	76.47	88.55	100.00	89.26
FIT1P	-100.00	81.25	91.30	79.31
FIT2P	-3.22	89.75	99.14	95.93
MAROS	16.00	76.79	86.84	72.22
SCFXM3	76.47	49.06	54.55	71.19
SHIP12L	82.76	86.44	100.00	92.96
STOCFOR2	28.57	53.06	95.56	66.13
STOCFOR3	40.66	93.49	99.59	95.66
WOODW	41.86	72.36	78.79	86.59
DCP1	25.80	73.49	83.82	84.10
DCP2	55.18	79.82	89.44	82.47
DETEQ8	89.61	96.35	94.69	98.57
DETEQ27	92.82	97.94	98.49	99.08
PDS-02	87.67	92.35	90.00	98.16
PDS-06	90.71	95.57	94.54	97.30
PDS-10	90.42	96.11	97.55	97.14
PDS-20	89.94	93.77	99.10	95.37

Table 4: Percentage improvement in the performance of computational components of the revised simplex method when exploiting hyper-sparsity

Table 5 shows quite clearly how the techniques described in this section yield a significant improvement in the performance of EMSOL for almost all problems which exhibit hyper-sparsity.

Table 6 indicates the improvement in solution time for the problems which do not exhibit hyper-sparsity when EMSOL is permitted to use the techniques for exploiting hyper-sparsity. With only a few exceptions, these techniques yield little if any improvement in performance. This is not surprising since the techniques for exploiting hyper-sparsity during FTRAN, BTRAN and PRICE are 'switched off', with the exception of the row-wise representation of the entries in UPDATE etas for rows which have been pivotal since INVERT. However, this technique has little overall effect on performance since the operations with the UPDATE etas are dominated by those with the INVERT etas. The effect of exploiting hyper-sparsity during INVERT has a marked effect on the performance of a few problems, notably DFL001.

2.10.1 Comparison with OSL simplex and barrier

All of the techniques described above, with the exception of the hyper-sparse CHUZC have been implemented within EMSOL, whose performance is compared in Table 7 with that of the revised simplex solver and barrier solver in OSL [8], for the test problems which exhibit hyper-sparsity. The results are given and show that, with the exception of the largest two PDS problems, EMSOL is clearly superior—by an order of magnitude for some problems. The extent to which the computational cost of complete CHUZC now dominates the solution time for the problem may be seen in the column headed 'CHUZC', where the absence of a value indicates that the overall solution time is too short for the

Problem	Dimensions			Solution time CPU (s)		Speed-up
	Rows	Columns	Nonzeros	Sparse	Hyper-sparse	
80BAU3B	2262	9799	21002	52.01	19.17	2.7
CYCLE	1903	2857	20720	6.96	1.83	3.8
CZPROB	929	3523	10669	3.27	1.72	1.9
FIT1P	627	1677	9868	1.89	1.24	1.5
FIT2P	3000	13525	50284	199.84	87.79	2.3
GANGES	1309	1681	6912	1.31	0.68	1.9
MAROS	846	1443	9614	3.10	1.70	1.8
SCFXM3	990	1371	7777	1.92	1.04	1.8
SCTAP3	1480	2480	8874	1.48	0.52	2.8
SHELL	536	1775	3556	0.43	0.24	1.8
SHIP08L	778	4283	12802	2.72	0.83	3.3
SHIP12L	1151	5427	16170	5.31	1.51	3.5
SHIP12S	1151	2763	8178	1.24	0.48	2.6
SIERRA	1227	2036	7302	1.20	0.44	2.7
STOCFOR2	2157	2031	8343	3.63	1.97	1.8
STOCFOR3	16675	15695	64875	315.99	108.71	2.9
WOODW	1098	8405	37474	14.94	4.77	3.1
DCP1	4950	3007	93853	65.60	20.62	3.2
DCP2	32388	21087	559390	3484.95	911.70	3.8
DETEQ8	20678	56227	128968	495.65	99.94	5.0
DETEQ27	68672	186928	429472	6470.06	1131.32	5.7
PDS-02	2953	7535	16390	10.65	2.19	4.9
PDS-06	9881	28655	62524	311.26	55.25	5.6
PDS-10	16558	48763	106436	1341.13	255.33	5.3
PDS-20	33874	105728	230200	19092.51	3220.69	5.9

Table 5: Solution time for EMSOL with sparse and hyper-sparse techniques when applied to problems exhibiting hyper-sparsity

proportion of time accounted for by CHUZC to be determined reliably.

For the PDS problems, the initial vertex following the crash is feasible. A phase II version of hyper-sparse CHUZC is much more straightforward to implement than a phase I version since it is not necessary to handle the situation where reduced costs change as a result of basic cost changes. For these problems, the performance of EMSOL with hyper-sparse CHUZC is compared to that of OSL in Table 8. For the largest of the PDS problems, EMSOL is still not faster than the OSL simplex solver. This is due to the fact that OSL requires fewer than one tenth of the number of iterations required by EMSOL. For problems such as the PDS problems which are highly degenerate, it is often highly advantageous to first solve a perturbation of the problem and then recover a solution to the original problem. When solving the PDS problems, OSL reports that it is solving a perturbed problem and this may well account for the dramatically smaller number of iterations required to solve the problem. EMSOL has no such perturbation strategy and this will be the subject of future research. This technique has no effect on the computational requirements in simplex iterations so, if an effective perturbation strategy were implemented in EMSOL its performance on the PDS problems would reflect its clear superiority

Problem	Dimensions			Solution time CPU (s)		Speed-up
	Rows	Columns	Nonzeros	Sparse	Hyper-sparse	
25FV47	821	1571	10400	15.74	11.65	1.4
BNL1	643	1175	5121	2.15	1.44	1.5
BNL2	2324	3489	13999	17.72	11.30	1.6
D2Q06C	2171	5167	32417	151.84	116.04	1.3
D6CUBE	415	6184	37704	195.90	154.34	1.3
DEGEN2	444	534	3978	1.11	0.89	1.2
DEGEN3	1503	1818	24646	26.50	15.39	1.7
DFL001	6071	12230	35632	7339.64	4243.50	1.7
GREENBEA	2392	5405	30877	89.64	43.47	2.1
GREENBEB	2392	5405	30877	76.60	37.35	2.1
GROW15	300	645	5620	1.80	1.91	0.9
GROW22	440	946	8252	5.78	5.19	1.1
MAROS-R7	3136	9408	144848	283.42	267.65	1.1
MODSZK1	687	1620	3168	3.79	2.92	1.3
NESM	662	2923	13288	8.05	5.48	1.5
PEROLD	625	1376	6018	7.06	6.73	1.0
PILOT	1441	3652	43167	254.89	270.51	0.9
PILOT.JA	940	1988	14698	18.14	16.10	1.1
PILOT.WE	722	2789	9126	13.01	10.12	1.3
PILOT4	410	1000	5141	2.37	2.48	1.0
PILOT87	2030	4883	73152	623.54	601.13	1.0
PILOTNOV	975	2172	13057	8.03	5.92	1.4
QAP8	912	1632	7296	56.21	56.66	1.0
SCSD8	397	2750	8584	4.06	2.27	1.8
STAIR	356	467	3856	1.48	0.81	1.8
TRUSS	1000	8806	27836	90.76	42.03	2.2
WOOD1P	244	2594	70215	5.15	7.36	0.7

Table 6: Solution time for EMSOL with sparse and hyper-sparse techniques when applied to problems not exhibiting hyper-sparsity

over OSL in terms of iteration speed.

When a phase I version of the hyper-sparse CHUZC is implemented in EMSOL it may be expected that, as has been seen for the PDS problems, there will be a performance improvement for the remaining problems which exhibit hyper-sparsity which corresponds to the proportion of CPU time attributable to complete CHUZC given in Table 7.

3 Comparison with a network solver

A class of LP problems which are well known to maintain sparsity are those with a near or complete network structure. It is, therefore, of interest to see how the performance of a general revised simplex solver with techniques to exploit hyper-sparsity compares with the network simplex method. In this section, EMSOL and OSL are compared with NETFLO, the efficient implementation of the network simplex method due to Kennington [9]). This is done using the NETGEN test set [10] of network problems which are now very modest in size,

Problem	Speed-up of hyper-sparse EMSOL over			CHUZC CPU %
	Sparse EMSOL	OSL simplex	OSL barrier	
80BAU3B	2.7	1.9	1.3	53.8
CYCLE	3.8	12.1	5.1	17.7
CZPROB	1.9	1.2	2.2	
FIT1P	1.5	2.8	21.9	8.9
FIT2P	2.3	1.3	28.0	24.1
GANGES	1.9	2.5	3.1	1.7
MAROS	1.8	4.7	2.0	21.1
SCFXM3	1.8	2.5	2.0	12.4
SCTAP3	2.8	4.7	5.1	
SHELL	1.8	1.9	4.6	
SHIP08L	3.3	1.9	3.0	
SHIP12L	3.5	2.2	2.2	49.2
SHIP12S	2.6	2.8	3.9	
SIERRA	2.7	2.2	5.4	
STOCFOR2	1.8	4.5	2.1	6.8
STOCFOR3	2.9	7.2	0.5	13.2
WOODW	3.1	2.1	2.1	33.1
DCP1	3.2	12.2	2.6	12.9
DCP2	3.8	8.4	1.0	16.6
DETEQ8	5.0	11.2	2.5	63.8
DETEQ27	5.7	11.6	2.6	68.3
PDS-02	4.9	1.4	4.4	58.9
PDS-06	5.6	1.0	2.3	67.8
PDS-10	5.3	0.7	2.2	69.9
PDS-20	5.9	0.4	2.3	65.6

Table 7: Speed-up of EMSOL over OSL

and a larger problem generated by the authors using the NETGEN program.

Initial results when applying EMSOL to the NETGEN set using the techniques for exploiting hyper-sparsity developed in Section 2 showed that the PRICE operation dominated the solution time overwhelmingly. This suggested that, as in NETFLO, it is preferable not to maintain a full set of reduced costs and use a partial pricing strategy. This is a standard technique, used particularly when solving LP problems with very many more columns than rows. The technique, and the consequences of using it in the presence of hyper-sparsity is outlined below.

3.1 Partial and multiple PRICE/CHUZC

Rather than incur the cost of maintaining the value of, and choosing from, a full set of reduced costs, it may be preferable to work with a small subset of the nonbasic variables in a given simplex iteration. There are two classical approaches: (pure) partial PRICE where reduced costs are formed for a different small set of variables each iteration, and multiple PRICE where reduced costs for a small set of variables are updated until none is attractive.

Whilst a partial or multiple PRICE strategy greatly reduces the cost of the

Problem	OSL simplex		OSL barrier
	Iteration count	CPU time	CPU time
PDS-02	3.8	2.3	7.5
PDS-06	7.2	1.6	3.7
PDS-10	8.0	1.7	5.3
PDS-20	10.8	0.8	5.3

Table 8: Increase in iteration count and solution speed of hyper-sparse EMSOL over OSL for the PDS problems

PRICE operation, it can lead to a significant increase in the number of iterations required to solve the problem. This is due to the fact that the 'best' candidate to enter the basis is not normally chosen, either because its reduced cost has not been computed, or because the possibilities for using even a simple edge-weight based strategy such as Devex are severely restricted. However, for problems where the tableau row is usually sparse, the reduced cost and Devex weight for most attractive candidates are unchanged from one iteration to the next so there may be no significant impact on the number of iterations required. For problems with very many more columns than rows, a complete PRICE is so expensive that a large increase in the number of iterations can be accepted.

In EMSOL, what is referred to below as partial PRICE is a combination of multiple and pure partial PRICE. Reduced costs are updated for a small set of variables, but those which enter the basis or become unattractive are replaced each iteration. Periodically the current 'pool' is discarded and a full re-PRICE is performed.

3.2 Updating 'complete' $\boldsymbol{\pi}$

In order to calculate reduced costs directly it is necessary to have the 'complete' $\boldsymbol{\pi}$ vector

$$\boldsymbol{\pi}^T = \boldsymbol{c}_B^T B^{-1},$$

where $\boldsymbol{c}_B$ is the vector of basic costs. This vector may be dense or even full, particularly in phase I when there are many infeasibilities, or in phase II when solving network problems for which there is a cost on each arc. In this case, the complete $\boldsymbol{\pi}$ vector has no sparsity which may be exploited during BTRAN. However, given $\boldsymbol{\pi}$, the vector $\boldsymbol{\pi}'$ required for the subsequent iteration is given by

$$\begin{aligned}
\boldsymbol{\pi}'^T = \boldsymbol{c}_{B'}^T B'^{-1} &= (\boldsymbol{c}_B + \delta_p \boldsymbol{e}_p)^T B'^{-1} \\
&= \boldsymbol{c}_B^T \left(I - \frac{1}{\eta_p}(\boldsymbol{\eta} - \boldsymbol{e}_p)\boldsymbol{e}_p^T \right) B^{-1} + \delta_p \boldsymbol{e}_p^T B'^{-1} \\
&= \boldsymbol{c}_B^T B^{-1} - \frac{1}{\eta_p}\boldsymbol{c}_B^T(\boldsymbol{\eta} - \boldsymbol{e}_p)\boldsymbol{e}_p^T B^{-1} + \delta_p \boldsymbol{e}_p^T B'^{-1} \\
&= \boldsymbol{\pi}^T + (\delta_p - \boldsymbol{c}_B^T(\boldsymbol{\eta} - \boldsymbol{e}_p))\boldsymbol{e}_p^T B'^{-1}
\end{aligned}$$

	NETGN101–150			NETGN201
	Minimum	Average	Maximum	
NETFLO	1	1	1	1
EMSOL	2.9	6.1	14	2.4
OSL	13	30	79	24

Table 9: Solution time for EMSOL and OSL simplex relative to NETFLO

since $\eta_p \boldsymbol{e}_p^T B'^{-1} = \boldsymbol{e}_p^T B^{-1}$. Thus the complete $\boldsymbol{\pi}$ vector may be updated using the unit $\boldsymbol{\pi}$ vector $\boldsymbol{e}_p^T B^{-1}$. Observing further that since $\hat{c}_q = c_q - \boldsymbol{c}_B^T \boldsymbol{\eta}$,

$$\begin{aligned} \delta_p - \boldsymbol{c}_B^T(\boldsymbol{\eta} - \boldsymbol{e}_p) &= \delta_p + c_{B_p} - \boldsymbol{c}_B^T \boldsymbol{\eta} \\ &= c_{B'_p} + (\hat{c}_q - c_q) \end{aligned}$$

and it follows that even the inner product $\boldsymbol{c}_B^T \boldsymbol{\eta}$ may be avoided.

3.3 Results

In comparing EMSOL and OSL against NETFLO, it is worth observing that all use some form of partial pricing and, in phase I, add a multiple of the phase II objective to the L_1-penalty function which penalises infeasibilities. At the expense of a larger number of phase I iterations, this reciprocal of the 'big-M' method aims to find a first feasible vertex which is much closer to an optimal solution than would be achieved if the L_1-penalty function were minimized regardless of the phase II objective. Note also that brief practical experience showed OSL's network solver to be very much less efficient than its revised simplex solver! The standard NETGEN problems, numbered NETGN101–150, have between 1000 and 10000 rows, and between 12500 and 75000 columns. For the purposes of the results given below, these problems, which are small by today's standards, are supplemented by a larger problem generated by the authors using the Netgen program. This problem, named NETGN201 has 25000 rows and 100000 columns.

As the results in Table 9 clearly show, for the smaller problems EMSOL is faster than OSL by about the same factor that it is slower than NETFLO. However, for NETGN201 the performance of EMSOL is somewhat closer to that of NETFLO than OSL is to EMSOL.

Although it would be remarkable if the performance of EMSOL were similar to that of a well-written network solver, it is worth considering why NETFLO is faster than EMSOL. The main reason is that NETFLO exploits fully the fact that the basis of a network problem corresponds to a spanning tree by combining BTRAN with UPDATE, FTRAN with CHUZR, and INVERT is avoided since the spanning tree triangularisation of the basis matrix is updated each iteration. Finally, NETFLO solves problems with only integer-valued bounds and costs so no floating-point operations are required, whereas EMSOL treats the positive and negative unit entries in the constraint matrix as floating-point numbers.

Another factor which may explain the relatively good performance of EMSOL on NETGN201 is the fact that for most of the small problems, the data structures required by NETFLO sit entirely within the cache on the SUN UltraSPARC 140 chip. NETFLO requires six arrays of size equal to the number

of nodes (rows) and five arrays of size equal to the number of arcs. For a problem such as NETGN101 with 5000 nodes and 25000 arcs, this corresponds to a memory requirement of 620 Kbytes: little more than the chip cache size of 512 Kbytes. Since NETFLO performs partial pricing, it is likely that only the data for the nonbasic variables not priced for many iterations would need to be read from memory into cache. Contrast this with EMSOL which, with its general revised simplex data structures, requires far more data per iteration than could fit into the cache.

4 Conclusions and extensions

This paper has presented a number of techniques for exploiting hyper-sparsity in the revised simplex method. These techniques have been shown to improve the performance of the authors' solver, EMSOL, to such an extent that, for problems which exhibit hyper-sparsity, it is many times faster than a major commercial solver. As important are the comparisons which show EMSOL to be several times faster than a commercial barrier solver. Although this performance gain may only be achieved for a subset of LP problems, the amenable problems in this paper are large (albeit not particularly so) and or of genuine practical value.

As indicated in Section 2, the performance of EMSOL is likely to improve further when hyper-sparse CHUZC is fully implemented. In addition if techniques, such as perturbation, are implemented in EMSOL which reduce the number of iterations required to solve a problem without compromising the iteration speed, further performance gains may be achieved.

References

[1] J. Bisschop and A. J. Meeraus. Matrix augmentation and partitioning in the updating of the basis inverse. *Mathematical Programming*, 13:241–254, 1977.

[2] W. J. Carolan, J. E. Hill, J. L. Kennington, S. Niemi, and S. J. Wichmann. An empirical evaluation of the KORBX algorithms for military airlift applications. *Operations Research*, 38(2):240–248, 1990.

[3] G. B. Dantzig and W. Orchard-Hays. The product form for the inverse in the simplex method. *Math. Comp.*, 8:64–67, 1954.

[4] I. S. Duff. On algorithms for obtaining a maximum transversal. Technical Report CSS 49, AERE, Harwell, 1978.

[5] J. J. H. Forrest and J. A. Tomlin. Updated triangular factors of the basis to maintain sparsity in the product form simplex method. *Mathematical Programming*, 2:263–278, 1972.

[6] D. M. Gay. Electronic mail distribution of linear programming test problems. *Mathematical Programming Society COAL Newsletter*, 13:10–12, 1985.

[7] P. M. J. Harris. Pivot selection methods of the Devex LP code. *Mathematical Programming*, 5:1–28, 1973.

[8] IBM. *Optimization Subroutine Library, guide and reference, release 2*, 1993.

[9] L. J. Kennington and R. V. Helgason. *Algorithms for Network Programming*, pages 244–256. John Wiley and Sons, New York, 1980.

[10] D. Klingman, A. Napier, and J. Stutz. NETGEN - a program for generating large scle (un) capacitated assignment, transportation, and minimum cost network flow problems. *Management Science*, 20:814–822, 1974.

[11] I. Maros and G. Mitra. Finding better starting bases for the simplex method. In P. Kleinschmidt, *et al.*, editor, *Operations Research Proceedings 1995*, pages 7–12. Springer Verlag, 1996.

[12] C. E. Pfefferkorn and J. A. Tomlin. Design of a linear programming system for the ILLIAC IV. Technical Report SOL 76-8, Systems Optimization Laboratory, Stanford University, 1976.

[13] R. Tarjan. Depth first search and linear graph algorithms. *SIAM J. Comput.*, 1:146–160, 1972.