Hyper-sparsity in the revised simplex method and how to exploit it

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

08^{th} October 2002

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. Each iteration of the revised simplex method requires the solution of two linear systems and a matrix vector product. For a significant number of practical problems the result of one or more of these operations is usually sparse, a property we call *hyper-sparsity*. Analysis of the commonly-used techniques for implementing each step of the revised simplex method shows them to be inefficient when hyper-sparsity is present. Techniques to exploit hyper-sparsity are developed and their performance is compared with the standard techniques. For the subset of our test problems that exhibits hyper-sparsity, the average speedup in solution time is 5.2 when these techniques are used. For this problem set our implementation of the revised simplex method which exploits hyper-sparsity is shown to be competitive with the leading commercial solver and significantly faster than the leading public-domain solver.

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 constraint matrices for most practical LP problems are sparse and, for an implementation of the revised simplex method to be efficient, it is crucial that only non-zeros in the coefficient matrix are stored and operated on. Each iteration of the revised simplex method requires the solution of two linear systems, the computation of which are commonly referred to as FTRAN and BTRAN, and a matrix vector product computed as the result of an operation referred to as PRICE. The matrices involved in these operations are submatrices of the coefficient matrix so are normally sparse. For many problems these three operations yield dense vectors, in which case there is limited scope for improving the performance by fully exploiting any zeros in the vectors. However, there is

a significant number of practical problems where the results of one or more of these operations are usually sparse, a property referred to in this paper as *hyper-sparsity*. This phenomenon has been reported independently by Bixby [3] and Bixby *et al* [4]. A number of classes of problems are known to exhibit hyper-sparsity, in particular those with a significant network structure such as multicommodity flow problems. Specialist variants of the revised simplex method which exploit this structure explicitly have been developed, in particular by McBride and Mamer [16, 17]. This paper identifies hyper-sparsity in a wider range of more general LP problems. Techniques are developed which can be applied within a general revised simplex solver when hyper-sparsity is identified during the solution process and, for problems exhibiting hyper-sparsity, significant performance improvements are demonstrated.

The computational components of the revised simplex method and standard techniques for FTRAN and BTRAN are introduced in Section 2 of this paper. Section 3 describes hyper-sparsity and gives statistics on its occurrence in a test set of LPs drawn from the standard Netlib set [10], the Kennington test set [5] and the authors' personal collection [13]. Analysis given in Section 4 shows the commonly-used techniques for each of the computational components of the revised simplex method to be inefficient when hyper-sparsity is present and techniques to exploit hyper-sparsity are described. Section 5 presents a computational comparison of the authors' revised simplex solver, EMSOL, with and without the techniques for exploiting hyper-sparsity. For those problems which exhibit hyper-sparsity, a comparison is also made between EMSOL, SOPLEX 1.2 [21] and the CPLEX 6.5 [15] primal simplex solver. Conclusions are offered in Section 6.

Although it contains no new ideas, this paper represents a significant reworking of [12]. The presentation has been greatly improved and the comparisons between EMSOL, SOPLEX and CPLEX have been added.

2 The revised simplex method

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

$$\begin{array}{rl} \text{minimize} & \boldsymbol{c}^T\boldsymbol{x} \\ \text{subject to} & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0}, \end{array}$$

where $\boldsymbol{x} \in I\!R^n$ and $\boldsymbol{b} \in I\!R^m$.

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. If any basic variables are infeasible they are given costs of -1, with other variables being given costs of zero. Phase I iterations are then performed in order to force all variables to

be non-negative, if possible. An account of the details of the revised simplex method is given by Chvátal in [6] and the computational components of each iteration are summarised in Figure 1. Note that although the reduced costs may be computed directly using the following BTRAN and PRICE operations

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

it is more efficient computationally to update them by calculating the pivotal row $\hat{\boldsymbol{a}}_p^T = \boldsymbol{e}_p^T B^{-1} N$, where $\boldsymbol{e}_p$ is column p of the identity matrix, using the BTRAN and PRICE operations defined in Figure 1. The only significant computational requirement which is not indicated in Figure 1 occurs when, in a phase I iteration, one or more variables which remain basic become feasible so their cost coefficients increase by one to zero. In order to update the reduced costs, it is necessary to compute the corresponding linear combination of tableau rows. This composite row is formed by the following BTRAN and PRICE operations

$$\boldsymbol{\pi}_\delta^T = \boldsymbol{\delta}^T B^{-1}$$
$$\hat{\boldsymbol{a}}_\delta^T = \boldsymbol{\pi}_\delta^T N,$$

where the nonzeros in $\boldsymbol{\delta}$ are equal to one for each variable which remains basic and becomes feasible. In this paper, techniques are discussed which are applicable to all BTRAN and PRICE operations. This is done with reference to an unsubscripted vector $\boldsymbol{\pi}$ corresponding to a generic right-hand-side vector $\boldsymbol{r}$ for the system ${B_0}^T \boldsymbol{\pi} = \boldsymbol{r}$ solved by BTRAN. When techniques are only applicable to a particular class of BTRAN or PRICE operations, the specific notation for the right-hand-side and solution is used.

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

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

Efficient implementations of the revised simplex method generally weight each reduced cost in CHUZC by some measure of the magnitude of the corresponding column of the standard simplex tableau, commonly referred to as an *edge weight*. Many solvers, including the authors' solver EMSOL, use the Harris *Devex* strategy [14]. This requires only the pivotal row to update

the edge weights so carries no significant computational overhead beyond that outlined in Figure 1.

2.1 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 respectively forwards and backwards through the data structure corresponding to a factored representation of B^{-1}. There are a number of procedures for updating the factored representation of B^{-1}, the original and simplest being the product form update of Dantzig and Orchard-Hays [7]. This approach is used by EMSOL and the techniques in this paper are developed with the product form update in mind.

If B_0^{-1} is used to denote the factored representation obtained by INVERT, and E_U represents the transformation of B_0 corresponding to subsequent basis changes such that $B = B_0 E_U$, it follows that B^{-1} may be expressed as

$$B^{-1} = E_U^{-1} B_0^{-1}.$$

In a typical implementation of the revised simplex method for large sparse LP problems, B_0^{-1} is represented as a product of K_I elimination operations derived directly from the nontrivial columns of the matrices L and U which form an LU decomposition of (a row and column permutation of) B_0. This invertible representation allows B_0^{-1} to be expressed algebraically as $B_0^{-1} = \prod_{k=K_I}^{1} E_k^{-1}$, where

$$E_k^{-1} = \begin{bmatrix} 1 & & & -\eta_1^k & & & \\ & \ddots & & \vdots & & & \\ & & 1 & -\eta_{p_k-1}^k & & & \\ & & & 1 & & & \\ & & & -\eta_{p_k+1}^k & 1 & & \\ & & & \vdots & & \ddots & \\ & & & -\eta_m^k & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & \mu^k & & & \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{bmatrix}. \quad (1)$$

Within an implementation, the nonzeros in the 'eta' vector

$$\boldsymbol{\eta}_k = [\, \eta_1^k \quad \ldots \quad \eta_{p_k-1}^k \quad 0 \quad \eta_{p_k+1}^k \quad \ldots \quad \eta_m^k \,]^T$$

are stored as value-index pairs and the data structure $\{p_k, \mu_k, \boldsymbol{\eta}_k\}_{k=1}^{K_I}$ is referred to as an eta file. Eta vectors associated with the matrices L and U coming from Gaussian elimination are referred to as L-etas and U-etas respectively. Note that the number of nonzeros in the eta file is no more that the number of nonzeros in the LU decomposition.The operations with p_k, μ_k and $\boldsymbol{\eta}_k$ required when forming $E_k^{-1}\boldsymbol{r}$ during the standard algorithms for FTRAN and $E_k^{-T}\boldsymbol{r}$ during BTRAN are illustrated in Figure 2.

The product form update leaves the factored form of B_0^{-1} unaltered and represents E_U^{-1} as a product of pairs of elementary matrices of the form (1). The representation of each UPDATE operation is obtained directly from the pivotal column and is given by $p_k = p$, $\mu_k = 1/\hat{a}_{pq}$ and $\boldsymbol{\eta}_k = \hat{\boldsymbol{a}}_q - \hat{a}_{pq}\boldsymbol{e}_p$.

In solvers based on the product form, the representation of the UPDATE operations can be appended to the eta file following INVERT, resulting in a

```
if (r_{p_k} ≠ 0) then
   r_{p_k} := μ_k r_{p_k}
   r := r − r_{p_k} η_k
end if
```

(a) FTRAN

$r_{p_k} := \mu_k(r_{p_k} - \boldsymbol{r}^T\boldsymbol{\eta}_k)$

(b) BTRAN

Figure 2: Standard operations in FTRAN and BTRAN

single homogeneous data structure. However, in this paper and in EMSOL, the particular properties of the INVERT and UPDATE etas and the nature of the operations with them are exploited, so FTRAN is performed as the pair of operations

$$\tilde{\boldsymbol{a}}_q = B_0^{-1}\boldsymbol{a}_q \qquad \text{(I-FTRAN)}$$

followed by

$$\hat{\boldsymbol{a}}_q = E_U^{-1}\tilde{\boldsymbol{a}}_q. \qquad \text{(U-FTRAN)}$$

Conversely, BTRAN is performed as

$$\tilde{\boldsymbol{\pi}}^T = \boldsymbol{r}^T E_U^{-1} \qquad \text{(U-BTRAN)}$$

followed by

$$\boldsymbol{\pi}^T = \tilde{\boldsymbol{\pi}}^T B_0^{-1}. \qquad \text{(I-BTRAN)}$$

Note that the term RHS is used to refer, not just to the vector on the right hand side of the system to be solved, but to the vector at any stage in the process of transforming it into the solution of the system.

3 What is hyper-sparsity?

Each iteration of the revised simplex method performs FTRAN to obtain the pivotal column $\hat{\boldsymbol{a}}_q = B^{-1}\boldsymbol{a}_q$ as the solution of one linear system, BTRAN to obtain $\boldsymbol{\pi}_p^T = \boldsymbol{e}_p^T B^{-1}$ as the solution of a second linear system, and PRICE to form the pivotal row $\hat{\boldsymbol{a}}_p^T = \boldsymbol{\pi}_p^T N$ as the result of a matrix-vector product. For many LP problems, even those for which the constraint matrix is sparse, there are few zero values in the results of these operations. However, as this paper demonstrates, there is a significant set of practical problems for which the proportion of zero values in the results of these operations is such that exploiting this yields very large computational savings.

For the sake of classifying test problems, in this paper the result of an FTRAN, BTRAN or PRICE operation is considered to be sparse if no more than 10% of its entries are nonzero and if at least 60% of the results of that particular operation are sparse then this is taken to be a clear majority. An LP problem is considered to exhibit hyper-sparsity if a clear majority of the results of one or more of these operations is sparse. These thresholds are used for the presentation of results only and are not parameters in the implementation of the methods which exploit hyper-sparsity.

The extent to which hyper-sparsity exists in LP problems was investigated for a subset of the standard Netlib test set [10], the Kennington test set [5]

and the authors' personal collection [13]. Those problems from the Netlib set whose solution requires less than ten seconds of CPU time were excluded, as were FIT2D and the OSA problems from the Kennington set. For the latter problems, the number of columns is particularly large relative to the number of rows so the the solution techniques developed in this paper are inappropriate. Note that a simple standard scaling algorithm is applied to each of the problems for reasons of numerical stability and each problem is solved from the initial basis obtained using the crash procedure in CPLEX [15].

For each problem the density of each pivotal column $\hat{\boldsymbol{a}}_q$ following FTRAN, of $\boldsymbol{\pi}_p$ and $\boldsymbol{\pi}_\delta$ following BTRAN and of $\hat{\boldsymbol{a}}_p^T$ and $\hat{\boldsymbol{a}}_\delta^T$ following PRICE was determined, and the total number of each which was found to be sparse was counted. The problems for which there is a clear majority of sparse results for at least one of these three operations are listed in Table 1 and referred to as test set $\mathcal{H}$. The remaining problems, those which exhibit no hyper-sparsity in any of FTRAN, BTRAN or PRICE, are listed in Table 2 and referred to as test set $\mathcal{H}'$. For each of the three operations, Tables 1 and 2 give the percentage of the results which are sparse. Subsequent columns give, for each of the three operations, the average density of those results which are sparse and the average density of those results which are not. The final column of Table 1 summarises the extent to which the problem exhibits hyper-sparsity by giving the initial letter of the operation(s) for which a clear majority of the results is sparse.

The first thing to note from the results in Table 1 is that all but two of the problems in $\mathcal{H}$ exhibit hyper-sparsity for BTRAN and most do so for all three operations. It is interesting to consider why this is the case and why there are exceptions.

Recall that the result of FTRAN is the pivotal column $\hat{\boldsymbol{a}}_q$ and the result of (most) PRICE operations is the pivotal row. Since these are a column and row from the same matrix (the standard simplex tableau $B^{-1}N$) it might be expected that a problem would exhibit hyper-sparsity in both FTRAN and PRICE or in neither. Also, since the $\boldsymbol{\pi}_p$ is just a single row of B^{-1}, and the pivotal column is usually a linear combination of several columns of B^{-1}, it might be expected that $\boldsymbol{\pi}_p$ would be less dense than $\hat{\boldsymbol{a}}_q$. These arguments would lead us to expect all problems in $\mathcal{H}$ to have property B, and that problems in $\mathcal{H}$ would be either FBP or B. The reasons for the exceptions are now explained.

There are two problems, DCP1 and DCP2, of type F, *i.e.* the pivotal columns are typically sparse but the results of BTRAN and PRICE are not. 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.

Conversely there are five problems of the opposite type, BP, *i.e.* the pivotal rows are typically sparse but the pivotal columns are not. The most remarkable of these is FIT2P: 81% of pivotal columns are essentially full and all but one of the pivotal rows are sparse. For this problem, most columns of the constraint matrix have only one nonzero entry, with the remaining columns being very dense. Thus B^{-1} is largely diagonal with a small number of essentially full columns. For this model it turns out that most variables chosen to enter the basis have a single nonzero entry such that the pivotal column is (a multiple of) one of these dense columns of B^{-1}. Each $\boldsymbol{\pi}_p$ is a row of B^{-1} and its resulting

							Mean density (%)						
	Dimensions			Sparse results (%)			Sparse results			Non-sparse results			Hyper-
Problem	Rows	Columns	Nonzeros	FTRAN	BTRAN	PRICE	FTRAN	BTRAN	PRICE	FTRAN	BTRAN	PRICE	sparse
80BAU3B	2262	9799	21002	97	72	70	2.40	0.61	0.66	10.84	42.41	50.85	FBP
FIT2P	3000	13525	50284	19	100	100	0.17	0.65	0.91	91.10	16.93	19.46	BP
GREENBEA	2392	5405	30877	15	60	60	4.62	0.49	0.81	24.73	75.54	80.75	BP
GREENBEB	2392	5405	30877	16	61	60	4.55	0.48	0.77	27.81	77.75	83.78	BP
STOCFOR3	16675	15695	64875	29	100	75	2.09	2.26	2.86	62.18	—	14.97	BP
WOODW	1098	8405	37474	32	73	71	3.90	1.01	1.14	22.60	57.55	59.91	BP
DCP1	4950	3007	93853	98	56	47	4.52	1.33	1.26	11.00	13.90	82.60	F
DCP2	32388	21087	559390	100	59	53	1.25	0.68	0.45	—	15.08	82.34	F
CRE-A	3516	4067	14987	100	82	80	1.91	0.39	0.61	—	28.17	56.06	FBP
CRE-C	3068	3678	13244	100	83	80	2.86	0.45	0.59	—	33.71	58.43	FBP
KEN-11	14694	21349	49058	100	98	97	0.10	0.27	0.28	—	11.88	13.79	FBP
KEN-13	28632	42659	97246	100	92	92	0.12	0.16	0.17	—	35.85	42.07	FBP
KEN-18	105127	154699	358171	100	93	93	0.05	0.21	0.20	—	39.58	41.94	FBP
PDS-06	9881	28655	62524	100	97	97	0.94	0.34	0.46	—	18.95	17.71	FBP
PDS-10	16558	48763	106436	100	96	96	1.22	0.24	0.31	—	23.60	25.19	FBP
PDS-20	33874	105728	230200	100	94	94	2.55	0.21	0.29	—	41.87	41.11	FBP

Table 1: Problems exhibiting hyper-sparsity (set $\mathcal{H}$): dimensions, percentage of the results of FTRAN, BTRAN and PRICE which are sparse, average density of those results which are sparse, average density of those results which are not and summary of those operations for which more than 60% of the results are sparse.

							Mean density (%)					
	Dimensions			Sparse results (%)			Sparse results			Non-sparse results		
Problem	Rows	Columns	Nonzeros	FTRAN	BTRAN	PRICE	FTRAN	BTRAN	PRICE	FTRAN	BTRAN	PRICE
BNL2	2324	3489	13999	29	36	35	2.28	0.61	0.79	28.56	40.03	70.22
D2Q06C	2171	5167	32417	12	25	25	1.84	0.48	0.86	49.32	66.32	87.24
D6CUBE	415	6184	37704	2	9	9	5.10	0.64	1.11	65.98	94.43	97.91
DEGEN3	1503	1818	24646	15	43	42	4.42	1.22	1.47	25.55	62.05	84.97
DFL001	6071	12230	35632	2	37	37	1.57	0.22	0.37	51.57	83.45	92.51
MAROS-R7	3136	9408	144848	24	1	1	0.54	0.59	1.43	81.25	30.78	62.30
PILOT	1441	3652	43167	9	27	24	3.89	1.59	2.06	60.86	76.31	88.51
PILOT87	2030	4883	73152	8	19	15	1.33	0.99	0.89	74.93	72.50	91.08
QAP8	912	1632	7296	0	11	11	1.46	0.55	1.92	83.92	75.51	98.45
TRUSS	1000	8806	27836	8	39	38	4.03	0.81	1.28	47.71	85.56	80.62
CRE-B	9648	72447	256095	58	55	55	4.15	0.19	0.37	14.01	55.81	89.16
CRE-D	8926	69980	242646	57	56	55	3.61	0.22	0.39	13.82	54.43	89.20

Table 2: Problems not exhibiting hyper-sparsity (set $\mathcal{H}'$): dimensions, percentage of the results of FTRAN, BTRAN and PRICE which are sparse, average density of those results which are sparse and average density of those results which are not.

sparsity is inherited by the pivotal row since most columns of N in the PRICE operation have only one nonzero entry.

Several observations can be made from the columns in Tables 1 and 2. For those problems in $\mathcal{H}'$ the density of the majority of results is such that the techniques developed for hyper-sparsity are seen to be of limited value. For the problems in $\mathcal{H}$, those results which are not sparse may have a very high proportion of nonzero entries. It is important therefore that techniques for exploiting hyper-sparsity are 'switched off' when such cases are identified during a particular FTRAN, BTRAN or PRICE operation. If this is not done then any computational savings obtained when exploiting hyper-sparsity may be compromised.

4 Exploiting hyper-sparsity

Each computational component of the revised simplex method either forms, or operates with, the result of FTRAN, BTRAN or PRICE. Each of these components is considered below and it is shown that typical computational techniques are inefficient in the presence of hyper-sparsity. In each case, equivalent computational techniques are developed which exploit hyper-sparsity.

4.1 Relative cost of computational components

Table 3 gives, for test set $\mathcal{H}$, the percentage of solution time which can be attributed to each of the major computational components in the revised simplex method. This allows the value of exploiting hyper-sparsity in a particular component to be assessed. Although PRICE and CHUZC together are dominant for most problems, each of the other computational components, with the exception of I-FTRAN and I-BTRAN, constitute at least 10% of the solution time for some of the problems. Thus, to obtain general improvement in computational performance requires hyper-sparsity to be exploited in all computational components of the revised simplex method.

For the problems in test set $\mathcal{H}'$, only U-BTRAN, PRICE and INVERT benefit noticeably from the techniques developed below. The percentages of the solution time for these computational components are given as part of Table 6.

4.2 Hyper-sparse FTRAN

For most problems which exhibit hyper-sparsity, Table 3 shows that the dominant computational cost of FTRAN is I-FTRAN, particularly so for the larger problems. When the pivotal column $\hat{\boldsymbol{a}}_q$ computed by FTRAN is sparse, only a very small proportion of the INVERT (and UPDATE) eta vectors, needs to be applied (unless there is an improbable amount of cancellation). Indeed 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 $\hat{\boldsymbol{a}}_q$. Gilbert and Peierls [11] identified this property in the context of Gaussian elimination when the pivotal column is formed as required and is expected to be sparse. It follows that the cost of I-FTRAN (and hence FTRAN) will be dominated by the test for zero if the INVERT etas are applied using the standard operation illustrated in Figure 2(a).

Problem	Solution CPU (s)	Percentage of solution time							
		CHUZC	I-FTRAN	U-FTRAN	CHUZR	I-BTRAN	U-BTRAN	PRICE	INVERT
80BAU3B	31.51	20.26	5.52	0.65	3.35	6.46	1.09	60.02	1.58
FIT2P	117.77	9.00	4.01	1.65	13.11	15.57	0.76	44.90	8.32
GREENBEA	66.71	4.74	9.34	3.59	5.92	16.87	5.17	42.00	11.12
GREENBEB	47.78	4.68	9.57	3.79	6.15	17.02	5.31	40.41	11.83
STOCFOR3	372.43	4.29	7.84	7.95	13.91	17.13	4.20	23.07	18.34
WOODW	10.52	9.94	4.48	0.68	2.77	7.86	1.70	68.60	2.99
DCP1	47.51	3.86	4.71	1.94	3.33	17.79	2.33	59.74	5.64
DCP2	2572.45	4.58	3.83	2.27	2.39	17.11	2.46	60.38	6.87
CRE-A	13.71	12.13	9.08	0.98	6.38	16.81	2.46	47.52	3.00
CRE-C	10.12	12.58	8.35	1.10	7.15	15.22	2.56	47.71	3.53
KEN-11	209.86	9.00	10.58	0.20	3.27	21.84	0.96	53.26	0.59
KEN-13	1221.09	8.60	9.96	0.15	2.80	20.08	0.66	57.03	0.59
KEN-18	21711.40	9.10	10.57	0.08	2.79	20.72	0.36	55.93	0.42
PDS-06	143.33	12.76	7.93	0.22	3.11	14.41	1.28	58.48	1.47
PDS-10	868.28	12.45	7.01	0.18	2.68	13.51	1.24	61.05	1.70
PDS-20	10967.10	11.88	5.58	0.27	2.66	11.01	2.02	63.80	2.65
Mean		9.37	7.40	1.61	5.11	15.59	2.16	52.74	5.04

Table 3: Total solution time and percentage of solution time for computational components when not exploiting hyper-sparsity for test set $\mathcal{H}$.

The aim of this subsection is to develop a computational technique which identifies the INVERT etas which have to be applied without passing through the whole INVERT eta file and testing each value of r_{p_k} for zero. The Gilbert and Peierls approach finds this minimal set of etas at a cost proportional to their total number of nonzeros.

A limitation of the Gilbert and Peierls approach is that the entire minimal set of etas has to be found before any of it can be used. If $\tilde{\boldsymbol{a}}_q$ is not sparse then the cost of this approach could be significantly greater than the standard I-FTRAN. This drawback is avoided by the hyper-sparse I-FTRAN algorithm given as pseudo-code in Figure 3, and explained in the following paragraph.

$\mathcal{K} = \{k : r_{p_k} \neq 0\}$
`While` $\mathcal{K} \neq \emptyset$
  $k_0 = \min_{k \in \mathcal{K}}$
  $r_{p_{k_0}} := \mu_{k_0} r_{p_{k_0}}$
  `for` $i \in \mathcal{E}_{k_0}$ `do`
    `if` $(r_i \neq 0)$ `then`
      $r_i := r_i - r_{p_{k_0}} [\boldsymbol{\eta}_{k_0}]_i$
    `else`
      $r_i := -r_{p_{k_0}} [\boldsymbol{\eta}_{k_0}]_i$
      `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`
  $\mathcal{K} := \mathcal{K} \backslash \{k_0\}$
`end while`

Figure 3: Hyper-sparse I-FTRAN algorithm

For a given RHS vector $\boldsymbol{r}$ and set of indices of nonzeros $\mathcal{R} = \{i : r_i \neq 0\}$ (which is always available without a search for the nonzeros), the hyper-sparse I-FTRAN algorithm is initialised by forming a set $\mathcal{K}$ of indices k of etas for which r_{p_k} is nonzero. This is done by passing through the indices in $\mathcal{R}$ and using the arrays $P^{(1)}$ and $P^{(2)}$. These record, for each row, the index of the first and second eta which have a pivot in the particular row. Note that, because the etas used in I-FTRAN correspond to the LU factors, there can be at most two etas with a pivot in any particular row. Unless cancellation occurs, all the etas with indices in $\mathcal{K}$ will have to be applied. If $\mathcal{K}$ is empty then no more etas need to be applied so I-FTRAN is complete. Otherwise, the least index $k_0 \in \mathcal{K}$ identifies the next eta which needs to be applied. In applying eta k_0 the algorithm steps through the nonzeros in $\boldsymbol{\eta}_{k_0}$ (whose indices are in $\mathcal{E}_{k_0}$). For each fill-in row the algorithm checks if there are etas k with $k > k_0$ whose pivot is in this row, and any such are added to $\mathcal{K}$. (The check only requires two lookups using the $P^{(1)}$ and $P^{(2)}$ arrays.) Finally the k_0 entry is removed.

The set $\mathcal{K}$ must be searched to determine the next eta to be applied, and there is some scope for variation in the way that this is achieved. In EMSOL, $\mathcal{K}$ is maintained as an unordered list and, if the number of entries in $\mathcal{K}$ becomes large, there comes a point at which the cost of the search exceeds the cost of the tests for zero which it seeks to avoid. To prevent this happening the average

skip through the eta file which has been achieved during the current FTRAN is compared with a multiple of $|\mathcal{K}|$ to determine the point at which it is preferable to complete I-FTRAN using the standard algorithm.

Although the set of possible nonzeros in the RHS is not required by the algorithm in Figure 3, it is maintained as the set $\mathcal{R}$. There is no real scope for exploiting hyper-sparsity in U-FTRAN which, as a consequence, is performed using the standard algorithm with the modification that $\mathcal{R}$ is maintained so long as the RHS is sparse. The availability of set $\mathcal{R}$ which, on completion of FTRAN, gives the possible positions of nonzeros in the pivotal column, allows hyper-sparsity to be exploited in CHUZR, as indicated below.

4.3 Hyper-sparse CHUZR

CHUZR performs a small number of floating-point operations for each of the nonzero entries in the pivotal column. If the indices of the nonzeros are not known *a priori* then all entries in the pivotal column will have to be tested for zero, and for problems when this vector is typically sparse, the cost of CHUZR will be dominated by the cost of performing the tests for zero. If a list of indices of entries in the pivotal column which are (or may be) nonzero is known, then this overhead is eliminated. The nonzero entries in the pivotal column are also required both to update the values of the basic variables following CHUZR and, as described in Section 4.8, 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.

4.4 Hyper-sparse BTRAN

When performing BTRAN using the standard operation illustrated in Figure 2(b), most of the work comes from the evaluation of the inner product $\boldsymbol{r}^T\boldsymbol{\eta}_k$. However, when the RHS is sparse, it will usually be the case that there is no intersection of the nonzeros in $\boldsymbol{r}^T$ and $\boldsymbol{\eta}_k$ so that the result is structurally zero. Unfortunately, checking directly whether there is a non-empty intersection is slower than evaluating the inner product directly. Better techniques are discussed in the remainder of this subsection.

4.4.1 Avoiding structurally zero inner products and operations with zero

When using the product form update, it is valuable to consider U-BTRAN separately from I-BTRAN. When forming $\boldsymbol{\pi}_p^T = \boldsymbol{e}_p^T B^{-1}$, which constitutes the overwhelming majority of BTRAN operations, it is possible in U-BTRAN to eliminate all the structurally zero inner products and significantly reduce the number of operations with zero. For *all* BTRAN operations it is possible to eliminate a significant number of the structurally zero inner products in I-BTRAN.

Hyper-sparse U-BTRAN when forming $\boldsymbol{\pi}_p$

Let K_U denote the number of UPDATE operations which have been performed since INVERT and let $\mathcal{P}$ denote the set of indices of those rows which have been pivotal. Note that the inequality $|\mathcal{P}| \leq K_U$ is strict if a particular row has been pivotal more than once. Since the RHS of $B^T\boldsymbol{\pi}_p = \boldsymbol{e}_p$ has only one nonzero in the row which has just been pivotal and fill-in during U-BTRAN can only occur in components of the RHS corresponding to pivots, it follows that the nonzeros in $\tilde{\boldsymbol{\pi}}_p^T = \boldsymbol{e}_p^T E_U^{-1}$ are restricted to the components with indices in $\mathcal{P}$. Thus, when applying the k^{th} UPDATE eta, only the nonzeros with indices in $\mathcal{P}$ contribute to $\boldsymbol{r}^T\boldsymbol{\eta}_k$. Since $|\mathcal{P}|$ is very much smaller than the dimension of B, it follows that unless this observation is exploited, most of the floating point operations when applying the UPDATE etas involve zero. A significant improvement in efficiency is achieved by maintaining a rectangular array $\mathtt{E_P}$ of dimension $|\mathcal{P}| \times K_U$ which holds the values of the entries corresponding to $\mathcal{P}$ in the UPDATE etas, allowing $\tilde{\boldsymbol{\pi}}_p$ to be formed as a sequence of K_U inner products. These are computed by indirection into $\mathtt{E_P}$ using a list of indices of nonzeros in the RHS which is simple and cheap to maintain.

When using this technique, if the update etas are sparse then $\mathtt{E_P}$ will be largely zero. As a consequence, most of the inner products $\boldsymbol{r}^T\boldsymbol{\eta}_k$ will be structurally zero, and most of the values of r_{p_k} will be zero. The first (next) eta for which this is not the case, and so must be applied, is identified by searching for the first (next) nonzero in the rows of $\mathtt{E_P}$ for which the corresponding component of $\boldsymbol{r}$ is nonzero. 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 needs to be applied.

If K_U is sufficiently large then a prohibitively large amount of storage is required to store $\mathtt{E_P}$ in a rectangular array. However $\mathtt{E_P}$ may be represented by ensuring that in the UPDATE eta file the indices and values of nonzeros in $\boldsymbol{\eta}_k$ for rows in $\mathcal{P}$ are stored before any remaining indices and values. It is still possible to perform row-wise searches of $\mathtt{E_P}$ by using an additional, more compact, data structure from which the eta index of nonzeros in each row of $\mathtt{E_P}$ may be deduced. The overhead of maintaining this data structure and searching it is usually much less than the floating-point operations with zero that would otherwise be performed.

Note that since $\tilde{\boldsymbol{\pi}}_p$ is sparse for *any* LP problem, the technique for exploiting this is of benefit whether or not the update etas are sparse. However it is seen in Table 6 that, for problems in test set $\mathcal{H}'$, the saving is a small proportion of the overall cost of performing a simplex iteration.

Hyper-sparse I-BTRAN using a column-wise INVERT eta file

As in the U-BTRAN case above we wish to find a way of only applying those INVERT eta vectors which require at least one nonzero operation. Assume we have available a list, $Q^{(1)}$, of the indices of the last INVERT eta with a nonzero in each row, 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 $Q^{(l)}$ of the index of the l^{th} last INVERT eta with a nonzero in each row is recorded, for l 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, and the backward steps are likely to become smaller for larger l, so in EMSOL only $Q^{(1)}$ and $Q^{(2)}$ are recorded.

4.4.2 Hyper-sparse I-BTRAN using a row-wise INVERT eta file

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

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

During BTRAN, when using the standard operation illustrated in Figure 2(b), it is simple and cheap to maintain a list of the indices of the possible nonzeros in the RHS: if r_{p_k} is zero and $\boldsymbol{r}^T\boldsymbol{\eta}_k$ is nonzero then the index p_k is added to the end of a list. When performing I-BTRAN by applying the algorithm given in Figure 3 with a row-wise INVERT eta file, the list of the indices of the possible nonzeros in the RHS is maintained as set $\mathcal{R}$. For problems when $\boldsymbol{\pi}$ is frequently sparse, knowing the indices of those elements which are (or may be) nonzero allows a valuable saving to be made in PRICE.

4.5 Row-wise (hyper-sparse) PRICE

For the problems in test set $\mathcal{H}$, it is clear from Table 3 that PRICE accounts for about half of the CPU time required for most problems and significantly more than that for others. The matrix-vector product $\boldsymbol{\pi}^T N$ is commonly formed as a sequence of inner products between $\boldsymbol{\pi}$ and (the packed form of) the appropriate columns of the constraint matrix. In the case when $\boldsymbol{\pi}$ is full, there will be no floating-point operations with zero 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 floating point operations with zero are avoided. Although the cost of maintaining the row-wise representation of N is non-trivial, this is far outweighed by the efficiency with which $\boldsymbol{\pi}^T N$ may then be formed.

Within reason, even for problems when $\boldsymbol{\pi}$ is not sparse, performing PRICE with a row-wise representation of N is advantageous. For example, even if $\boldsymbol{\pi}$ is on average half full, the overhead of maintaining the row-wise representation of N is small compared to the half of the work of column-wise PRICE which is saved.

For problems when $\boldsymbol{\pi}$ is particularly sparse, the time to test all entries for zero dominates the time to do the small number of floating point operations which involve the nonzeros in $\boldsymbol{\pi}$. The cost of testing can be avoided if a list of the indices of the non-zeros in $\boldsymbol{\pi}$ is known and, as identified in Section 4.4.3, such a list can be constructed at negligible cost during BTRAN. After each nonzero entry in $\boldsymbol{\pi}$ is used it is zeroed, so that by the end of BTRAN the entire $\boldsymbol{\pi}$ workspace is zeroed in preparation for the next BTRAN. Thus, 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.

Provided $\boldsymbol{\pi}^T N$ remains sparse, it is advantageous to maintain a list of the indices of its nonzeros. This list can be used to avoid having to search for the nonzeros in $\boldsymbol{\pi}^T N$, which would otherwise be the dominant cost when using this vector. Once again the list of indices of the non-zeros can be use to zero the workspace so, provided the $\boldsymbol{\pi}^T N$ vectors remain sparse, the only full pass through this workspace that is needed is to zero it before the first simplex iteration.

4.6 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, $\boldsymbol{e}_p^T B^{-1} N$, is sparse, the number of reduced costs which *change* each iteration is small and this can be exploited to improve the efficiency of CHUZC.

The aim of the hyper-sparse CHUZC algorithm is to maintain a set, C, consisting of a small number, s, of variables which is guaranteed to include the variable with the most negative reduced cost. At the start of an LP iteration let g be the most negative reduced cost of any variable in C and let h be the lowest reduced cost of those non-basic variables not in C.

The steps of CHUZC for one LP iteration are:

- If $h < g$ then reinitialise C by performing a full CHUZC: pass through all the non-basic variables and find the $s+1$ most attractive (*i.e.* those with the lowest reduced costs). Store those with the lowest s reduced costs in C, set g to the lowest reduced cost and h to the reduced cost of the remaining variable.

- Find and remove the variable with the best reduced cost from C.

  This provides the candidate to enter the basis.

- Whilst updating the reduced costs

  - form a set D of the variables (not in C) with the lowest s reduced costs which have changed;

– update h each time a variable is not included in D and has a lower reduced cost than the current value of h.

Note that hyper-sparse PRICE generates a list of the non-zeros in the pivot row, which is used to drive this step and simultaneously zero the workspace for the pivot row.

- Update C to correspond to the lowest s reduced costs in $C \cup D$, set g to the lowest reduced cost in the resultant C and update h if a variable which is not included in C has a lower reduced cost than the current value of h.

Note that the set operations with C and D can be performed in a time proportional to their length. Also, observe that the technique for hyper-sparse CHUZC described above extends naturally, and at no additional cost, when the column selection strategy incorporates edge weights. Since the edge weight for a column changes only if the corresponding entry in the pivotal row is nonzero, the list D still contains the most attractive candidates not in C whose reduced cost and weight have changed.

4.7 Hyper-sparse Tomlin INVERT

The default INVERT used in EMSOL is based on the procedure described by Tomlin [19]. 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 submatrix is then factorised using Gaussian elimination with the order of the columns determined prior to the numerical calculation by merit counts based on fill-in estimates. 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. This procedure requires much simpler data structure management and pivot search strategy compared to a Markowitz-based procedure, which maintains and selects the pivot from the whole up-to-date active submatrix.

The pivotal column in a given stage of the Tomlin INVERT is formed by passing forwards through the L-etas for the residual block which have been computed up to that stage. Even for problems which do not exhibit hyper-sparsity, the pivotal column of the active submatrix during Gaussian elimination is very likely to be sparse. This is the situation where what is referred to in this paper as hyper-sparsity was identified by Gilbert and Peierls [11]. This partial FTRAN operation is particularly amenable to the exploitation of hyper-sparsity using the algorithm illustrated in Figure 3. Note that the data structures required to exploit hyper-sparsity during this stage in INVERT, as well as during I-FTRAN itself, are generated at almost no cost during INVERT.

For the problems in test set $\mathcal{H}$, the Tomlin INVERT yields factors which, in the worst case, have an average of 4% more entries than those generated by a Markowitz-based INVERT. The average fill-in with the Tomlin INVERT is no more than 10%. For problems with such low fill-in, the Tomlin INVERT (when exploiting hyper-sparsity) is at least as fast as a Markowitz-based INVERT. For many problems in $\mathcal{H}$ the dimension of the residual block in the Tomlin INVERT is very small (no more than a few percent) relative to the dimension of B_0 so

the scope for exploiting hyper-sparsity is negligible. However, for others, the average dimension of the residual block is between 10 and 25 percent of the dimension of B_0. Since there is little fill-in, the scope for exploiting hyper-sparsity during INVERT for these problems is significant. Indeed, if hyper-sparsity is not exploited, the Tomlin INVERT may be significantly slower than a Markowitz-based INVERT.

For some of the problems in test set $\mathcal{H}'$, using the Tomlin INVERT results in significantly more fill-in than would occur with a Markowitz-based INVERT, in which case it would generally be preferable to use a Markowitz-based INVERT. For the remaining problems, the Tomlin INVERT is at least competitive when exploiting hyper-sparsity.

4.8 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 explained in Section 4.3 this packed form is produced at negligible cost during the course of CHUZR.

4.9 Hyper-sparsity for other update procedures

The product form update is commonly criticised for its lack of numerical stability and inefficiency with regard to sparsity. For this reason, some implementations of the revised simplex method are based on the Forrest-Tomlin [9] or Bartels-Golub [1] update procedures which modify the U (but not the L) etas in the representation of B_0^{-1} in order to gain numerical stability and efficiency with regard to sparsity. If such a procedure were used, the data structure which enables hyper-sparsity to be exploited when applying U-etas during BTRAN and FTRAN would have to be modified after each UPDATE. The overhead of doing this is likely to limit severely the value of exploiting hyper-sparsity. Also, the advantage of the Forrest-Tomlin and Bartels-Golub update procedures with respect to sparsity is small for problems which exhibit hyper-sparsity in the product form UPDATE etas. If greater numerical stability is required than is offered by the product form update, the Schur complement update [2] may be used. Like the product form update, the representation of B_0^{-1} is unaltered so the data structures for exploiting hyper-sparsity when applying the INVERT etas remain static. Techniques analogous to those described above for the product form update may be used to exploit hyper-sparsity during U-BTRAN when using a Schur complement update.

4.10 Controlling the use of hyper-sparsity techniques

All the hyper-sparse techniques described above are less efficient than the standard versions in the absence of hyper-sparsity and so should only be applied when hyper-sparsity is present. 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. The technique would then be switched off 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 can be achieved by monitoring the density of the result during the operation and switching on some tolerance. Practical experience has shown that performance is very insensitive to changes in these tolerances around the optimal value.

5 Results

Computational results in this section demonstrate the value of the techniques for exploiting hyper-sparsity described in the previous section. A detailed analysis is given of the speedup of the authors' revised simplex solver, EMSOL, as a result of exploiting hyper-sparsity. In addition, for problems which exhibit hyper-sparsity, EMSOL is compared with SOPLEX 1.2 [21] and the CPLEX 6.5 [15] primal simplex solver. All results were obtained on a SunBlade 100 with 512Mb of memory.

5.1 Speedup of EMSOL when exploiting hyper-sparsity

The efficiency of the techniques for exploiting hyper-sparsity is demonstrated by the results in Table 4 for the problems in test set $\mathcal{H}$ and Table 6 for the problems in test set $\mathcal{H}'$. These tables give the speedup of both overall solution time and the time attributed to each computational component where significant hyper-sparsity may be exploited. In this paper, mean values of speedup are geometric since this avoids bias when favourable and unfavourable speedups are being combined.

5.1.1 Problems which exhibit hyper-sparsity

For test set $\mathcal{H}$, Table 4 shows clearly the value of exploiting hyper-sparsity. The solution time of all problems improves, by more than an order of magnitude in the case of the larger problems, and all computational components show a significant mean speedup. Note that, particularly for the larger problems, the speedup in PRICE and CHUZC underestimates the efficiency of these operations when the pivotal row is sparse: although only a small percentage of the pivotal rows are not sparse, they dominate the time required for PRICE, and in addition, if the pivotal row is not sparse, the set C in the hyper-sparse CHUZC must be reinitialised, requiring a full CHUZC.

Although U-BTRAN exhibits the greatest mean speedup, it is seen in Table 3 that, when not exploiting hyper-sparsity, this operation makes a relatively small contribution to overall solution time. It is the speedups in I-FTRAN, I-BTRAN, PRICE and CHUZC which are of greatest value. However, the speedup in all operations is of some value.

We now consider whether there is scope for further improvement. Table 5 gives the percentage of solution time attributable to the major computational components when exploiting hyper-sparsity. The column headed 'Hyper-sparsity' is the percentage of the solution time which is attributable to creating and maintaining the data structures required to exploit hyper-sparsity in the

Problem	speedup in total solution time and computational components							
	Solution	CHUZC	I-FTRAN	CHUZR	I-BTRAN	U-BTRAN	PRICE	INVERT
80BAU3B	3.34	3.05	5.13	1.93	3.51	6.72	6.06	1.34
FIT2P	1.75	18.91	1.30	0.93	12.22	3.59	13.47	0.87
GREENBEA	2.71	1.33	1.30	1.13	3.60	19.87	3.45	2.83
GREENBEB	2.44	1.39	1.35	1.21	3.69	21.88	3.44	2.78
STOCFOR3	1.85	4.47	1.14	0.96	7.26	56.99	7.61	3.16
WOODW	3.40	1.82	1.70	1.30	4.17	11.72	5.14	1.53
DCP1	3.25	1.58	2.36	1.19	3.87	6.25	6.71	1.70
DCP2	5.32	1.60	8.24	2.51	6.21	13.99	6.20	8.63
CRE-A	3.05	2.54	4.00	1.72	3.64	6.50	4.48	1.14
CRE-C	2.89	2.88	4.67	1.97	3.58	6.53	4.97	1.08
KEN-11	22.84	19.36	98.04	13.93	27.22	9.90	66.36	1.02
KEN-13	12.12	6.31	104.09	7.31	12.87	9.19	17.60	0.94
KEN-18	15.27	6.63	263.94	15.27	13.91	13.07	19.92	1.01
PDS-06	17.48	15.25	24.07	3.57	21.58	35.76	28.18	1.02
PDS-10	10.36	10.67	11.24	1.85	16.60	49.99	17.55	0.96
PDS-20	10.35	8.58	5.96	1.68	14.33	189.19	15.40	1.44
Mean	5.21	4.38	7.03	2.28	7.64	15.44	9.71	1.55

Table 4: Speedup for test set $\mathcal{H}$ when exploiting hyper-sparsity.

computational components. PRICE and CHUZC are still the major cost for many problems and some form of partial/multiple pricing might reduce the time per iteration attributable to PRICE and CHUZC. However, the saving may be more than offset by an increase in the number of iterations required to solve these problems.

The one operation where no techniques for exploiting hyper-sparsity have been developed is U-FTRAN. The contribution of this operation to overall solution time has increased from an average of 1.61% when not exploiting hyper-sparsity in other components (see Table 3) to a far from dominant 6.11%.

5.1.2 Problems which do not exhibit hyper-sparsity

As identified in Section 4, for problems in test set $\mathcal{H}'$ the only significant scope for exploiting hyper-sparsity is in U-BTRAN, PRICE and INVERT. Table 6 gives the percentage of solution time attributable to these three operations when not exploiting hyper-sparsity. Although the overhead of INVERT is higher than for problems in set $\mathcal{H}$, the dominant operation is, again, PRICE. With the exception of PILOT, all other problems show some speedup in solution time when exploiting hyper-sparsity, with a modest but not insignificant mean speedup of 1.45. Despite the significant speedup in U-BTRAN, much of the overall performance gain can be put down to halving the time attributable to PRICE. Although the mean speed of INVERT is doubled, it should be born in mind that a Markowitz-based INVERT procedure may well be preferable for these problems. For the other computational components, there is an mean speedup of between 1.02 and 1.10, indicating that the hyper-sparse techniques are not significant relative to the rest of the calculation. However, the overhead associated with creating and maintaining the data structures required to exploit hyper-sparsity is not significant and for the only problems where it accounts for more than 1% of the solution time, it yields a significant overall speedup.

5.2 Comparison with representative simplex solvers

In this section the performance of EMSOL is compared with other simplex solvers for the problems in test set $\mathcal{H}$. CPLEX [15] is commonly held to be the leading commercial simplex solver, a view supported by benchmark tests performed by Mittelmann [18]. SOPLEX [21] is a public-domain simplex solver developed by Wunderling [20] which outperforms other such solvers in tests performed by Mittelmann [18]. In the results presented below, EMSOL is compared with the most recent version of CPLEX available to the authors (version 6.5) and SOPLEX version 1.2. Although SOPLEX version 1.2.1 is available, it shows no noticeable performance improvement over version 1.2.

It is important when comparing the performance of computational components of different solvers that all solvers start from the same (or similar) advanced basis and any algorithmic differences which affect the computational requirements are reduced to a minimum. In the comparisons below, EMSOL is started from the basis obtained using the CPLEX crash procedure. SOPLEX cannot be started from a given advanced basis. However, since the SOPLEX crash procedure described by Wunderling [20] appears to be closely related to that of CPLEX, the SOPLEX initial basis may be expected to be comparable to that of CPLEX, and hence EMSOL.

Problem	Percentage of solution time								
	CHUZC	I-FTRAN	U-FTRAN	CHUZR	I-BTRAN	U-BTRAN	PRICE	INVERT	Hyper-sparsity
80BAU3B	26.24	4.23	4.00	6.89	7.34	1.13	39.22	2.81	4.83
FIT2P	1.21	7.78	4.66	35.34	3.21	0.51	8.39	18.56	13.01
GREENBEA	8.24	16.51	7.94	12.03	10.82	0.79	28.10	7.48	4.98
GREENBEB	7.97	16.87	7.98	12.14	10.95	0.90	27.89	7.37	4.90
STOCFOR3	1.97	14.18	13.91	29.68	4.85	0.21	6.23	13.55	9.07
WOODW	17.79	8.45	3.51	7.02	6.17	1.17	43.90	3.18	4.70
DCP1	8.16	6.69	8.92	9.42	15.39	1.80	29.85	8.73	7.93
DCP2	13.54	2.20	5.93	4.51	13.05	0.67	46.16	6.26	6.99
CRE-A	14.03	6.58	5.87	10.92	13.43	1.65	31.08	4.47	8.32
CRE-C	13.60	5.60	5.69	11.41	13.32	2.02	30.31	4.21	9.50
KEN-11	12.02	2.74	5.22	6.08	20.64	4.61	20.72	8.51	15.14
KEN-13	17.06	1.20	3.85	4.81	19.53	1.81	40.58	3.86	5.85
KEN-18	20.75	0.60	2.59	2.76	22.51	1.01	42.42	2.62	4.32
PDS-06	12.29	4.80	5.90	12.77	9.75	1.50	30.38	6.03	12.86
PDS-10	11.89	6.35	5.99	14.74	8.29	0.77	35.43	5.18	9.27
PDS-20	12.55	8.49	5.75	14.35	6.96	0.30	37.54	5.48	7.16
Mean	12.46	7.08	6.11	12.18	11.64	1.30	31.14	6.77	8.05

Table 5: Percentage of solution time for computational components and overhead attributable to exploiting hyper-sparsity for test set $\mathcal{H}$.

	Not exploiting hyper-sparsity			Exploiting hyper-sparsity				
	Percentage of solution time			Speedup				Hyper-sparsity
Problem	U-BTRAN	PRICE	INVERT	Solution	U-BTRAN	PRICE	INVERT	CPU (%)
BNL2	6.29	32.61	14.74	1.87	14.61	2.43	2.80	4.79
D2Q06C	2.69	36.65	14.57	1.33	8.34	1.84	2.20	1.12
D6CUBE	0.45	72.35	3.83	1.65	1.37	2.04	1.37	0.23
DEGEN3	6.51	27.12	14.14	2.32	11.58	3.15	2.11	6.65
DFL001	2.90	24.22	22.52	1.69	10.93	1.97	5.21	0.59
MAROS-R7	1.79	18.50	26.22	1.05	2.14	2.06	0.98	0.51
PILOT	1.40	22.15	13.03	0.96	1.28	1.43	0.92	0.26
PILOT87	1.19	19.51	17.39	1.00	1.39	1.50	0.85	0.26
QAP8	1.25	15.56	16.68	1.05	1.29	1.24	1.48	0.40
TRUSS	2.62	54.50	10.26	1.80	6.47	2.49	4.40	0.88
CRE-B	1.31	74.61	2.97	1.82	50.74	2.33	2.93	0.60
CRE-D	1.25	74.95	2.82	1.52	41.97	2.16	3.14	0.65
Mean	2.47	39.39	13.26	1.45	5.80	1.99	2.01	1.41

Table 6: Percentage of solution time when not exploiting hyper-sparsity, speedup when exploiting hyper-sparsity and percentage of solution time attributable to exploiting hyper-sparsity for test set $\mathcal{H}'$.

When CPLEX is run, the default approach to pricing is to start with an inexpensive strategy and switch to Devex. For the test problems in this paper, this approach leads to a speedup of 1.22 (1.14 for the problems in $\mathcal{H}$) over the performance when using only Devex pricing. However, since EMSOL uses only Devex pricing, it is compared with CPLEX using only Devex pricing.

By default, SOPLEX uses the dual simplex method, although it often switches to the primal, particularly when close to optimality and occasionally significantly before. Although it is suggested that SOPLEX can run as a primal simplex solver, in practice it soon switches to the dual simplex method for the test problems in this paper. Thus, for the comparisons below, SOPLEX is run using its default choice of method. The default pricing strategy used by SOPLEX is steepest edge which is described by Forrest and Goldfarb in [8]. Even for the dual simplex method, steepest edge carries a higher computational overhead than Devex since it requires an additional BTRAN operation in the case of the primal (FTRAN in the dual) as well as some pricing in the primal. SOPLEX can be forced to use Devex pricing which has the same computational requirements in the dual as in the primal. Thus, in the results below, SOPLEX is forced to use only Devex pricing. Although the use of the primal or dual simplex method may significantly affect the number of iterations required to solve a problem, comparing the average time per iteration of SOPLEX and EMSOL gives a fair measure of the efficiency of the computational components of the respective solvers.

SOPLEX has a presolve which is used by default. Since the presolve improves the solution time by only about 10% and incorporates scaling, in the results for SOPLEX given below it is run with the presolve. As a result, the relative iteration speed of SOPLEX may give a slight overestimate of the efficiency of the techniques used in its computational components.

The results of the comparisons of EMSOL with CPLEX and SOPLEX, when run in the modes discussed above, are given in Table 7. Values of speedup which are greater than unity indicate that EMSOL is faster.

5.2.1 Comparison with CPLEX

Table 7 shows that for the $\mathcal{H}$ problems EMSOL is faster on seven and CPLEX is faster on nine. On average, CPLEX is slightly faster: the mean speedup for EMSOL is 0.84. There is no consistent reason for the difference, with neither code dominating the other in time per iteration or number of iterations. Because the different solvers do not follow the same paths to the solution, it is difficult to make precise comparisons. Also we have noticed that making an arbitrary perturbation to the pivot choice can double or half the solution time (though the result presented in this paper are all for the EMSOL default settings). This difference is due both to the variation in number of iterations taken and to variation in the average amount of hyper-sparsity encountered on the solution path.

Brief comments by Bixby [3] and by Bixby *et al* [4] indicate that methods to exploit hyper-sparsity in FTRAN, BTRAN and PRICE have been developed independently for use in CPLEX [15]. These techniques, for which no details have been published, were introduced to CPLEX between versions 6 and 6.5. There may be other unpublished features of CPLEX unconnected with hyper-sparsity which contribute the the differences in performance when compared

with EMSOL.

5.2.2 Comparison with SOPLEX

In the comparison of EMSOL and SOPLEX, two values are given in Table 7 for each model: the relative speedup in total solution time when EMSOL is used and the relative speedup in the time required by EMSOL to perform a simplex iteration. In terms of solution speed, EMSOL is clearly far superior, dramatically so for FIT2P and DCP1 for which SOPLEX takes an excessive number of iterations. However, even after excluding these problems, the mean speedup when using EMSOL is 2.72, which is considerable. Note that, for the problems in test set $\mathcal{H}$, SOPLEX with Devex pricing is 1.14 times faster than with all its defaults when it uses steepest edge.

For the reasons set out above, the values for the speedup in iteration time provide the most reliable comparison of the techniques used within the computational components of EMSOL and SOPLEX. With the exception of STOCFOR3, the iteration speed of EMSOL exceeds that of SOPLEX, significantly so for the larger problems.

	EMSOL	CPLEX (Devex)		SOPLEX (Devex)		
Problem	Solution time CPU (s)	Solution time CPU (s)	EMSOL solution speedup	Solution time CPU (s)	EMSOL solution speedup	EMSOL iteration speedup
80BAU3B	9.43	14.39	1.53	20.50	2.17	1.78
FIT2P	67.34	42.08	0.62	1058.79	15.72	1.30
GREENBEA	24.63	21.18	0.86	150.53	6.11	1.38
GREENBEB	19.55	18.31	0.94	76.36	3.91	1.56
STOCFOR3	201.21	41.15	0.20	154.11	0.77	0.64
WOODW	3.09	3.99	1.29	21.41	6.93	2.58
DCP1	14.64	16.30	1.11	330.35	22.56	1.69
DCP2	483.64	1412.89	2.92	1507.40	3.12	1.24
CRE-A	4.49	6.29	1.40	5.07	1.13	1.09
CRE-C	3.50	5.91	1.69	3.39	0.97	1.06
KEN-11	9.19	13.57	1.48	46.99	5.11	2.20
KEN-13	100.77	91.41	0.91	417.26	4.14	2.05
KEN-18	1421.50	786.08	0.55	5167.60	3.64	1.91
PDS-06	8.20	5.49	0.67	44.22	5.39	3.37
PDS-10	83.83	26.92	0.32	158.79	1.89	2.37
PDS-20	1059.19	238.48	0.23	1754.17	1.66	2.73
Mean			0.84		3.47	1.67

Table 7: Performance of EMSOL relative to CPLEX 6.5 and SOPLEX 1.2

6 Conclusions and extensions

This paper has identified hyper-sparsity as a property of a significant number of LP problems when solved by the revised simplex method. Techniques for exploiting this property in each of the computational components of the revised simplex method have been described. Although this has been done in the

context of the primal simplex method, the techniques developed in this paper can be applied immediately to the dual simplex method since it has comparable computational components.

For the subset of our test problems that do not exhibit hyper-sparsity ($\mathcal{H}$), the geometric mean speedup is 1.45, and for those problems which do exhibit hyper-sparsity ($\mathcal{H}'$), the speedup is substantial, increases with problem size and has a mean value of 5.38.

For this latter subset of problems our implementation of the revised simplex which exploits hyper-sparsity has been shown to be comparable to the leading commercial simplex solver and several times faster than the leading public-domain solver. Although this performance gain is substantial for only a subset of LP problems, the amenable problems in this paper are of genuine practical value and the techniques yield some performance improvement in all but one of the test problems.

The authors would like to thank John Reid who brought the Gilbert-Peierls algorithm to their attention and made valuable comments on an earlier version of this paper.

References

[1] R. H. Bartels. A stabilization of the simplex method. *Numer. Math.*, 16:414–434, 1971.

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

[3] R. E. Bixby. Solving real-world linear programs: A decade and more of progress. *Operations Research*, 50(1):3–15, 2002.

[4] R. E. Bixby, M. Fenelon, Z. Gu, E. Rothberg, and R. Wunderling. MIP: Theory and practice closing the gap. In M. J. D. Powell and S. Scholtes, editors, *System Modelling and Optimization: Methods, Theory and Applications*, pages 19–49. Kluwer, The Netherlands, 2000.

[5] 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.

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

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

[8] J. J. Forrest and D. Goldfarb. Steepset-edge simplex algorithms for linear programming. *Mathematical Programming*, 57:341–374, 1992.

[9] 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.

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

[11] J. R. Gilbert and T. Peierls. Sparse partial pivoting in time proportional to arithmetic operations. *SIAM J. Sci. Stat. Comput.*, 9(5):862–874, 1988.

[12] J. A. J. Hall and K. I. M. McKinnon. Hyper-sparsity in the revised simplex method and how to exploit it. Technical Report MS00-015, Department of Mathematics and Statistics, University of Edinburgh, 2000. Submitted to SIAM Journal of Optimization.

[13] J. A. J. Hall and K. I. M. McKinnon. LP test problems. http://www.maths.ed.ac.uk/hall/PublicLP/, 2002.

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

[15] ILOG. *CPLEX 6.5 Reference Manual*, 1999.

[16] R. D. McBride and J. W. Mamer. Solving multicommodity flow problems with a primal embedded network simplex algorithm. *INFORMS Journal on Computing*, 9(2):154–163, Spring 1997.

[17] R. D. McBride and J. W. Mamer. A decomposition-based pricing procedure for large-scale linear programs: an application to the linear multicommodity flow problem. *INFORMS Journal on Computing*, 46(5):693–709, May 2000.

[18] H. D. Mittelmann. Benchmarks for optimization software. http://www.plato.la.asu.edu/bench.html, April 2002.

[19] J. A. Tomlin. Pivoting for size and sparsity in linear programming inversion routines. *J. Inst. Maths. Applics*, 10:289–295, 1972.

[20] R. Wunderling. Paralleler und objektorientierter simplex. Technical Report TR-96-09, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1996.

[21] R. Wunderling, A. Bley, T. Pfender, and T. Koch. *SOPLEX 1.2.0*, 2002.