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

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

August 2000

MS 00-015

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

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

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

10^{th} August 2000

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 many problems, even those for which the constraint matrix is sparse, the results of these operations are usually dense. However it is shown in this paper that there is a significant number of practical problems where the results 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 5.61 when these techniques are used. For this problem set our implementation of the revised simplex method which exploits hyper-sparsity is shown to be many times faster that a commercial implementation of both the simplex and barrier method. When applied to network problems, our implementation is shown to approach the speed of 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 constraint matrices for most practical LP problems are sparse: on average there are only about six non-zeros per column and this number does not increase with problem size. 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 (called FTRAN and BTRAN) and a matrix vector product (called PRICE). The matrices involved in these operations

are submatrices of the coefficient matrix so are normally sparse. However, 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 it is shown in this paper that there is a significant number of practical problems where the results of one or more of these operations are usually sparse, a property we call *hyper-sparsity*, and for these problems significant performance improvements are possible.

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 [8] and larger problems from the Kennington test set [3] and the authors' personal collection. 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 and the barrier and simplex solvers in Version 2 of IBM's Optimization Subroutine Library, OSL [11]. For network problems, EMSOL is compared with NETFLO, Kennington's efficient implementation of the network simplex method [12]. Conclusions are offered in Section 6.

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

$$\begin{array}{rl} \text{minimize} & \boldsymbol{c}^T\boldsymbol{x} \\ \text{subject to} & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \leq \boldsymbol{0}, \end{array} \tag{1}$$

where $\boldsymbol{x} \in I\!R^n$ and $\boldsymbol{b} \in I\!R^m$. The matrix A may be assumed to contain the negation of the identity matrix. 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. An efficient solver should also be able to handle problems with more general bounds on the variables and constraints without the duplication of constraint rows and columns required to put such problems into the form (1).

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. An account of the details of the revised simplex method is given by Chvátal in [4] 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

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

$$\hat{\boldsymbol{c}}_N^T = \boldsymbol{c}_N^T - \boldsymbol{\pi}^T N$$

it is more efficient computationally to update them by calculating the pivotal row $\hat{\boldsymbol{a}}_p^T$ as indicated in Figure 1. The pivotal row is also required in order to update the weights used in Harris' *Devex* strategy [10] which is commonly used to select the entering variable in efficient implementations of the revised simplex method. The only significant computational requirement which is not indicated in Figure 1 occurs when, in 'phase I', the step results in one or more non-pivotal basic cost changes so the corresponding linear combination of tableau rows must be computed in order to update the reduced costs. This composite row is formed by the following BTRAN and PRICE operations

$$\hat{\boldsymbol{\delta}}^T = \boldsymbol{\delta}^T B^{-1}$$

$$\hat{\boldsymbol{a}}_\delta^T = \hat{\boldsymbol{\delta}}^T N,$$

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 'composite' BTRAN and PRICE constitute additional computation.

`CHUZC:` Use $\hat{\boldsymbol{c}}_N$ and the Devex weights to find good candidate q to enter basis.
`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. 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 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 with Devex pricing

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 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 of which is the product form update of Dantzig and Orchard-Hays [5]. This approach is used by the authors' solver EMSOL and the techniques in this paper are developed with the product form update in mind. Their application to alternative update procedures is discussed in Section 4.

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

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

Thus FTRAN may be performed as

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

followed by

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

Conversely, BTRAN is performed as

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

followed by

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

In a typical implementation of the revised simplex method for large sparse LP problems, B_0^{-1} is represented as a product of elementary matrices derived 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 $B_0 = \prod_{k=1}^{r} E_k$, where

$$E_k = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & \boldsymbol{\eta}_k & & & \\ & & 1 & & & & \\ & & & \eta_k & & & \\ & & & & 1 & & \\ & & & \boldsymbol{\eta}_k & & \ddots & \\ & & & & & & 1 \end{bmatrix} \leftarrow \text{row } p_k \qquad (2)$$

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 an **eta file**.

The product form update leaves the factored form of B_0^{-1} unaltered and represents E_U^{-1} as a product of elementary matrices of the form (2). 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 solvers based on the product form, the representation of the UPDATE operations can be appended to the eta file following INVERT, resulting in a 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 considered as the pair of operations I-FTRAN followed by U-FTRAN, and BTRAN as U-BTRAN followed by I-BTRAN.

When the product form update is used, the I-FTRAN and U-FTRAN operations transform $\boldsymbol{a}_q$ into the pivotal column $\hat{\boldsymbol{a}}_q$ by first scattering $\boldsymbol{a}_q$ from its packed form in the constraint matrix into a zeroed workspace vector $\boldsymbol{b}$ and then passing forwards though the INVERT and UPDATE eta files in turn, both 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, 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.

```
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

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 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.

The U-BTRAN and I-BTRAN operations form the 'unit' $\boldsymbol{\pi}$ vector by first setting to unity the appropriate component of a zeroed workspace vector $\boldsymbol{b}$ and then transforming it into $\boldsymbol{\pi}$ by passing backwards through the UPDATE and INVERT eta files, both according to the algorithm represented as pseudo-code in Figure 2(b). Unlike the FTRAN algorithm, there is no simple way to exploit sparsity in the RHS since testing for individual zeros to avoid the corresponding multiplication is slower than doing the multiplication itself.

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 the unit $\boldsymbol{\pi}$ vector $\boldsymbol{\pi}^T = \boldsymbol{e}_p^T B^{-1}$ as the solution of a second linear system, and PRICE to form the pivotal row $\hat{\boldsymbol{a}}_p^T = \boldsymbol{\pi}^T N$ as the result of a matrix-vector product. For many LP problems, even those for which the constraint matrix is sparse, the results of these operations are usually dense and this is assumed in many implementations of the revised simplex method. 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 taken 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 [8] and larger problems from the Kennington test set [3] and the authors' personal collection. Those problems from the Netlib set whose solution requires less than one second of CPU 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.

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. As a result, techniques for exploiting hyper-sparsity would be very effective at speeding up these early iterations. However it is still usually faster to generate an initial 'crash' basis, so in the computational comparisons in this paper, a crash has been used in all cases. The EMSOL crash basis is obtained using a stabilisation of the algorithm described by Maros [14].

The density of each pivotal column $\hat{\boldsymbol{a}}_q$ following FTRAN, unit $\boldsymbol{\pi}$ 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. The problems for which at least one of these three operations has a clear majority of the sparse results are listed in Table 1 and referred to as test set $\mathcal{H}$. The remaining problems, those which exhibit no hyper-sparsity in 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 were sparse. Note that these tables also include results for the composite 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 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 of the problems that exhibit hyper-sparsity do so 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 and the result of PRICE is the pivotal row. Since these are a column and row from the same matrix (the standard simplex tableau $B^{-1}N$) one might expect that a problem would exhibit hyper-sparsity in both FTRAN and PRICE or in neither. Also, since the unit $\boldsymbol{\pi}$ 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 the $\boldsymbol{\pi}$ vector would be less dense than $\hat{\boldsymbol{a}}_q$. These arguments would lead us to expect all problems to have property B, and that problems would be either FBP or B. The reasons for the exception are now explained.

There are two problems, DCP1 and DCP2, of type FB, in which the pivotal columns are typically sparse but the pivotal rows are not. These problems are decentralised planning problems for which a typical standard simplex tableau is

Problem	Dimensions			Sparse results (%)			Hyper-sparse operations
	Rows	Columns	Nonzeros	FTRAN	BTRAN	PRICE	
80BAU3B	2262	9799	21002	97	78	76	FBP
CYCLE	1903	2857	20720	49	92	55	B
CZPROB	929	3523	10669	100	66	62	FBP
FIT2P	3000	13525	50284	15	100	100	BP
GREENBEA	2392	5405	30877	12	64	62	BP
GREENBEB	2392	5405	30877	13	62	62	BP
MAROS	846	1443	9614	31	74	69	BP
MAROS-R7	3136	9408	144848	14	84	15	B
SHIP12L	1151	5427	16170	100	94	94	FBP
STOCFOR2	2157	2031	8343	23	97	63	BP
STOCHFOR	16675	15695	64875	53	100	100	BP
WOODW	1098	8405	37474	52	81	79	BP
DCP1	4950	3007	93853	97	75	53	FB
DCP2	32388	21087	559390	100	65	54	FB
DETEQ8	20678	56227	128968	95	100	100	FBP
DETEQ27	68672	186928	429472	94	100	100	FBP
CRE-A	3516	4067	14987	100	84	80	FBP
CRE-C	3068	3678	13244	100	82	81	FBP
KEN-07	2426	3602	8404	100	100	100	FBP
KEN-11	14694	21349	49058	100	99	97	FBP
KEN-13	28632	42659	97246	100	91	90	FBP
KEN-18	105127	154699	358171	100	90	90	FBP
PDS-02	2953	7535	16390	100	99	99	FBP
PDS-06	9881	28655	62524	100	97	97	FBP
PDS-10	16558	48763	106436	100	97	97	FBP
PDS-20	33874	105728	230200	100	94	94	FBP

Table 1: Problems exhibiting hyper-sparsity (set $\mathcal{H}$): dimensions, percentage of the results of FTRAN, BTRAN and PRICE which are sparse and summary of those operations for which more than 60% of the results are sparse.

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 seven problems of the opposite type, BP, in which the pivotal rows are typically sparse but the the pivotal columns are not. The most remarkable of these is FIT2P: almost all pivotal columns are essentially full and all pivotal rows are sparse. For this problem, 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 whose pivot is in one of these dense columns, so the pivotal column is (a multiple of) one of these dense columns of B^{-1}. Each unit $\boldsymbol{\pi}$ 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.

The partition of the test problems into sets $\mathcal{H}$ and $\mathcal{H}'$ is not clear-cut: some of the problems in set $\mathcal{H}'$ will still benefit from exploiting hyper-sparsity in the minority of FTRAN, BTRAN and PRICE operations for which the result is sparse.

	Dimensions			Sparse results (%)		
Problem	Rows	Columns	Nonzeros	FTRAN	BTRAN	PRICE
25FV47	821	1571	10400	3	15	16
BNL2	2324	3489	13999	34	45	42
D2Q06C	2171	5167	32417	16	29	28
D6CUBE	415	6184	37704	2	12	13
DEGEN3	1503	1818	24646	14	58	56
DFL001	6071	12230	35632	1	34	36
GROW22	440	946	8252	1	13	11
MODSZK1	687	1620	3168	14	34	33
NESM	662	2923	13288	26	23	17
PEROLD	625	1376	6018	6	27	28
PILOT	1441	3652	43167	7	13	10
PILOT.JA	940	1988	14698	4	20	21
PILOT.WE	722	2789	9126	13	34	32
PILOT4	410	1000	5141	4	25	23
PILOT87	2030	4883	73152	6	18	15
PILOTNOV	975	2172	13057	11	32	29
QAP8	912	1632	7296	0	9	11
SCSD8	397	2750	8584	8	55	50
TRUSS	1000	8806	27836	5	45	53
WOOD1P	244	2594	70215	0	51	53
WORLD	35664	31728	198250	38	53	51
CRE-B	9648	72447	256095	53	55	55
CRE-D	8926	69980	242646	48	53	53

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

In addition, for some of the problems in set $\mathcal{H}'$, it is identified in Section 4 that there is scope for exploiting hyper-sparsity during INVERT and U-BTRAN and that, in the case of INVERT, this scope does not exist for most problems in set $\mathcal{H}$.

3.1 Hyper-sparsity and the optimal block triangular form of the basis matrix

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 [6] to determine a row permutation P such that PB_0 has a transversal, and then using Tarjan's algorithm [15], to identify a permutation Q such that Q^TPB_0Q 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 usually 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.

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 this block will fill in entirely (if no zeros are created as a result of cancellation), in which case the dimension of this block is a lower bound on the number of nonzeros in the solution. A consequence of this observation is that if the result of FTRAN or BTRAN is typically sparse and cancellation is not occurring, then the diagonal blocks in the optimal block triangular ordering of the matrix must be small.

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

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 total solution time. Table 3 gives, for test set $\mathcal{H}$, the percentage of CPU time which can be attributed to each of the major computational components in 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 5. Note that the decision to perform INVERT is made optimally by EMSOL according to a pseudo-clock. For some problems PRICE and CHUZC are dominant. However, the even spread of cost across computational components for some problems and variance in the dominant cost between other problems suggests that to obtain general improvement in computational performance requires hyper-sparsity to be exploited in all computational components of the revised simplex method.

4.2 Hyper-sparse FTRAN

For problems with sparse pivotal columns, since there is unlikely to be more than a few hundred UPDATE etas, the dominant computational cost of FTRAN is I-FTRAN. 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.

Problem	Total	CHUZC	I-FTRAN	U-FTRAN	CHUZR	I-BTRAN	U-BTRAN	PRICE	INVERT
80BAU3B	16.51	21.78	5.79	0.61	3.40	7.52	0.72	52.09	2.95
CYCLE	2.48	2.95	20.98	1.97	4.26	22.30	2.95	23.28	15.74
CZPROB	1.04	10.68	4.85	0.00	2.91	18.45	0.97	52.43	1.94
FIT2P	57.45	6.99	4.99	3.75	14.59	13.39	10.73	32.61	8.40
GREENBEA	39.57	5.77	7.75	3.71	5.83	16.93	4.35	32.29	19.44
GREENBEB	32.10	5.47	7.80	3.82	5.74	17.02	4.40	31.80	19.97
MAROS	1.54	5.34	6.87	3.82	6.87	15.27	3.82	35.11	13.74
MAROS-R7	107.70	1.57	14.84	9.04	3.46	17.79	5.12	25.89	20.77
SHIP12L	1.62	13.58	3.70	0.00	2.47	9.88	0.62	61.73	1.23
STOCFOR2	2.11	3.19	7.98	4.26	14.36	18.62	6.91	14.36	19.68
STOCHFOR	113.56	4.75	8.36	1.96	11.81	15.60	8.05	20.01	24.83
WOODW	5.90	10.55	3.12	0.59	2.54	8.40	0.98	66.60	3.71
DCP1	29.37	3.35	4.08	4.64	5.05	12.67	3.81	54.06	9.09
DCP2	1971.60	4.79	3.62	3.08	2.15	14.21	3.00	58.95	9.63
DETEQ8	198.27	12.02	8.20	0.30	3.43	16.08	2.07	53.52	2.62
DETEQ27	2074.33	9.99	6.79	0.24	3.27	16.95	2.32	57.25	2.08
CRE-A	5.81	11.07	5.71	0.71	6.07	19.64	1.43	46.61	4.11
CRE-C	6.22	10.90	10.73	0.85	6.64	18.40	1.53	41.57	3.75
KEN-07	1.39	6.57	12.41	0.00	5.84	30.66	1.46	32.12	2.92
KEN-11	89.00	8.29	15.32	0.18	3.50	22.81	0.86	46.05	1.36
KEN-13	580.56	9.30	11.56	0.19	2.87	24.44	0.97	48.31	1.28
KEN-18	13311.43	8.68	11.04	0.07	2.45	23.04	0.53	53.08	0.71
PDS-02	3.48	14.24	8.90	0.30	4.15	18.40	1.19	44.81	2.08
PDS-06	136.37	11.53	9.76	0.20	2.77	13.93	1.69	56.42	2.03
PDS-10	513.23	15.38	8.04	0.17	2.77	14.51	1.54	54.26	2.06
PDS-20	6875.61	13.86	5.99	0.21	2.24	12.07	2.51	58.91	3.56
Average		9.25	7.35	1.49	4.42	14.71	2.49	49.93	6.68

Table 3: Total solution time (CPU seconds) and percentage of solution time for computational components of the revised simplex method for test set $\mathcal{H}$.

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). 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.

The algorithm is illustrated as pseudo-code in Figure 3. 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. Also, since there is at most one L-eta and at most one U-eta with a pivot 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 I-FTRAN 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.

$\mathcal{R} = \{i : b_i \neq 0\}$
$\mathcal{K} = \{k : b_{p_k} \neq 0\}$
`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

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 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 $|\mathcal{K}|$ to determine the point at which it is preferable to complete FTRAN using the standard algorithm.

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.

4.3 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 CHUZR and, 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.

4.4 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.

4.4.1 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 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.

4.4.2 Reducing trivial inner products and operations with zero

When using the product form update, it is valuable to consider U-BTRAN separately from I-BTRAN. 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 since INVERT and let $\mathcal{P}$ denote the set of indices of those rows which have been pivotal. Note that the inequality $|\mathcal{P}| \leq K$ 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.

If the number of updates is very large then it may be prohibitively expensive to store $\mathtt{E_P}$ in a rectangular array. However, at the expense of the ability to search it row-wise, it may be represented by ensuring that in the UPDATE eta file the indices and values of nonzeros in $\boldsymbol{\eta}_k$ for rows in set $\mathcal{P}$ are stored before any remaining indices and values.

Note that since $\tilde{\boldsymbol{\pi}}$ 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 5 that in the non-sparse case any saving is relatively small 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, it is shown above that there is a significant reduction in the cost of U-BTRAN. 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.

4.4.3 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 I-BTRAN 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 achieved when applying it.

4.5 Hyper-sparse PRICE

As is clear from Table 3, for the problems in test set $\mathcal{H}$, PRICE accounts for about half of the CPU time required to solve the problem in general, and is significantly more than that for some problems. The matrix-vector product $\boldsymbol{\pi}^T N$ is commonly 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 N is non-trivial, this is far outweighed by the efficiency with which $\boldsymbol{\pi}^T N$ may then be formed.

For problems when $\boldsymbol{\pi}$ is not sparse, performing PRICE with a row-wise representation of N is advantageous. Even if $\boldsymbol{\pi}$ is typically half full, the overhead of maintaining the row-wise representation of N is sufficiently small that saving half the work of PRICE is worthwhile.

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.

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 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 as attractive as 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 thus 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. This

event is used to trigger a reset of C_0 ensuring that the hyper-sparse CHUZC finds as good a candidate as a complete CHUZC.

4.7 Hyper-sparse (preordered) INVERT

The default INVERT in EMSOL is based on the procedure described by Tomlin [16]. 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 similar in size to 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 using the algorithm illustrated in Figure 3. Since problems which exhibit hyper-sparsity do not have a large residual diagonal block, there is usually less scope for speed-up in INVERT than for problems which do not exhibit hyper-sparsity. Note that the data structures required to exploit hyper-sparsity during FTRAN itself, are generated at almost no cost during the course of INVERT.

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 identified above, the former is readily achieved during CHUZR and the latter is a scalar operation performed afterwards.

4.9 Exploiting hyper-sparsity in 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 [7] or Bartels-Golub [1] update procedures which modify the representation of B_0^{-1} with respect to subsequent UPDATEs in order to gain numerical stability and efficiency with regard to sparsity. If such a procedure were used, the data structures which enable hyper-sparsity to be exploited

during BTRAN and FTRAN would have to be modified after each UPDATE to correspond to the changes in the representation of B_0^{-1}. The overhead of doing this may severely limit the value of exploiting hyper-sparsity. The greater efficiency of the Forrest-Tomlin and Bartels-Golub update procedures with respect to sparsity is not seen when solving 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 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 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

The efficacy of the techniques described in the previous section may be judged from the results presented in this section. These were obtained on a Sun UltraSPARC 10 with 256Mb 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 dynamically 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.

Note that the computational techniques which exploit hyper-sparsity result in small differences in numerical rounding and changes in the order of (otherwise arbitrary) tie-breaking in CHUZC and CHUZR. Whilst these minor differences are of no theoretical consequence, they do result in the simplex method taking a different path and number of iterations before reaching an optimal vertex. For a given problem, since exploiting hyper-sparsity generally changes the balance between the cost of INVERT and the work associated with operations dependent

on the number of UPDATE operations performed, EMSOL's dynamic reinversion strategy will result in a change in the typical frequency of INVERT.

5.1 Speedup of EMSOL when exploiting hyper-sparsity

For test set $\mathcal{H}$, Table 4 gives the speedup in the CPU time attributable to exploiting hyper-sparsity and Table 5 indicates the speedup in U-BTRAN, INVERT, PRICE as well as solution time for the problems in test set $\mathcal{H}'$. Note that differences in performance due to the number of iterations required to solve a problem and the reinversion frequency have been filtered out in the results for individual computational components but not in the overall speedup of solution time.

For test set $\mathcal{H}$, exploiting hyper-sparsity has less effect on INVERT than on operations associated with UPDATE so the dynamic reinversion strategy leads to a decrease in the number of INVERT operations. As a consequence of filtering out this effect, the high speedup of U-BTRAN in Table 4 has marginally less effect on the overall performance than it might suggest. Despite variances caused by differences in the number of iterations required to solve problems, the speed-up in solution time for all of the problems is greater than unity and dramatically so for some of the larger problems.

For many problems in test set $\mathcal{H}'$, the scope for performance improvement when exploiting hyper-sparsity in INVERT is significantly greater than for U-BTRAN, despite the latter's high speed-up. Thus the dynamic reinversion strategy leads to an increase in the number of INVERT operations. As a consequence of filtering out this effect, the speedup of U-BTRAN in Table 5 has marginally more effect on the overall performance than it might suggest. For the other computational components, there is an average speedup of between 0.94 and 1.12, indicating that the hyper-sparse techniques are generally not being used. Despite the variances caused by differences in the number of iterations required to solve problems, the overall solution time shows an average speedup which is significantly greater than unity.

Despite the performance improvements for problems in test set $\mathcal{H}$ demonstrated in Table 4, it is interesting to consider the scope for further performance improvement. Table 6 gives the percentage of CPU 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 computational components. Although PRICE and CHUZC are still the major cost for some problems, they are by no means dominant. Some form of partial/multiple pricing might reduce the time per iteration attributable to PRICE and CHUZC but this may well be offset by the likely increase in the number of iterations required to solve these problems. The overhead associated with the current techniques suggests that significant investment in more sophisticated techniques will result in little, if any, return.

5.2 Comparison with OSL simplex and barrier

Having established the efficacy of the techniques for exploiting hyper-sparsity by comparing the relative performance improvement of EMSOL, it is interesting to compare the performance of EMSOL with the revised simplex solver and barrier

Problem	Total	CHUZC	I-FTRAN	U-FTRAN	CHUZR	I-BTRAN	U-BTRAN	PRICE	INVERT
80BAU3B	1.69	1.42	2.91	0.49	1.27	2.55	2.91	3.53	1.31
CYCLE	2.34	0.92	2.85	0.62	0.84	5.38	7.49	4.21	2.66
CZPROB	1.96	1.63	5.92	–	1.78	4.50	2.17	3.55	1.29
FIT2P	2.16	19.91	1.08	1.43	0.97	10.74	125.97	15.50	0.52
GREENBEA	2.00	1.12	1.10	1.02	1.13	3.12	21.18	2.33	3.33
GREENBEB	2.24	1.14	1.12	1.06	1.14	3.17	20.95	2.37	3.48
MAROS	2.23	1.28	0.94	1.17	1.10	2.92	4.69	3.74	1.46
MAROS-R7	1.04	1.13	1.05	0.92	1.05	0.58	36.57	2.40	0.71
SHIP12L	5.40	7.01	5.74	–	3.83	7.65	2.07	15.94	0.88
STOCFOR2	1.79	1.67	1.04	1.30	1.02	3.65	23.31	2.82	1.60
STOCHFOR	3.19	28.73	2.24	1.92	1.00	14.06	114.50	14.68	2.71
WOODW	2.72	1.48	1.16	0.98	1.20	3.97	8.20	5.02	1.33
DCP1	3.52	1.44	1.82	1.84	1.99	3.94	8.33	7.39	2.39
DCP2	3.93	2.09	2.25	0.97	3.85	4.50	9.09	5.15	9.75
DETEQ8	12.50	27.81	8.87	1.82	3.67	29.67	27.55	59.17	1.00
DETEQ27	10.71	18.96	9.06	1.03	2.69	29.60	63.81	68.26	0.93
CRE-A	1.96	2.12	2.31	0.57	2.09	5.01	4.98	4.40	1.10
CRE-C	2.29	1.83	4.96	0.56	2.30	4.08	4.34	4.04	0.90
KEN-07	5.56	9.82	18.55	–	8.73	22.92	5.28	24.01	1.80
KEN-11	14.40	18.20	85.03	1.51	13.77	18.24	6.81	40.62	1.02
KEN-13	9.19	7.12	61.10	1.59	14.64	10.81	8.63	11.28	1.02
KEN-18	8.91	4.54	84.14	0.98	26.43	9.36	10.55	12.15	1.01
PDS-02	5.44	12.17	12.68	1.16	4.44	15.72	4.62	13.67	0.97
PDS-06	14.70	14.47	24.76	2.22	4.96	23.24	39.53	32.08	1.09
PDS-10	12.74	19.03	18.04	1.64	4.64	26.28	45.98	29.75	1.17
PDS-20	11.34	11.66	14.88	1.19	2.81	17.45	101.93	19.50	2.07
Average	5.61	8.41	14.45	1.08	4.36	10.89	27.36	15.67	1.83

Table 4: Speed-up in total solution time and computational components of the revised simplex method for test set $\mathcal{H}$ when exploiting hyper-sparsity. Note that – indicates that the time required for U-FTRAN increased from a value less than the resolution of the timing mechanism.

	Percentage of solution CPU time			Speedup			
Problem	U-BTRAN	PRICE	INVERT	U-BTRAN	PRICE	INVERT	Total
25FV47	4.22	22.36	24.89	7.87	1.42	1.64	1.07
BNL2	7.91	27.30	18.11	16.54	1.97	2.30	1.28
D2Q06C	4.99	27.67	26.64	34.32	1.62	2.74	1.39
D6CUBE	1.21	63.91	8.28	6.15	1.68	1.24	1.16
DEGEN3	5.54	18.43	21.45	13.36	2.16	1.83	1.72
DFL001	12.27	13.65	36.61	105.38	1.40	5.72	1.56
GROW22	3.74	12.15	31.78	6.21	0.99	1.27	1.24
MODSZK1	4.46	20.54	25.89	+	1.39	2.14	1.12
NESM	2.20	50.34	9.29	3.20	1.86	1.20	1.17
PEROLD	4.51	17.21	27.46	10.15	1.40	1.55	0.98
PILOT	4.98	15.52	28.19	14.28	1.22	1.21	1.05
PILOT.JA	4.62	16.27	30.31	12.93	1.20	1.43	0.96
PILOT.WE	4.05	26.08	26.58	11.86	1.48	1.86	1.05
PILOT4	4.00	18.00	24.00	4.25	1.58	1.17	1.06
PILOT87	4.33	14.35	28.26	18.42	1.19	1.09	1.07
PILOTNOV	4.45	22.26	26.71	6.79	1.53	1.36	1.04
QAP8	5.14	9.74	35.07	19.82	1.26	1.66	1.18
SCSD8	3.08	50.00	11.54	2.95	2.43	2.23	1.36
TRUSS	3.73	49.15	16.24	19.88	2.76	6.17	2.29
WOOD1P	1.73	61.27	7.51	5.18	1.48	1.20	1.19
WORLD	10.23	25.46	23.96	74.39	3.01	7.95	2.57
CRE-B	1.70	72.04	3.89	41.16	2.32	3.33	1.68
CRE-D	1.77	72.06	4.17	43.33	2.79	4.61	2.21
Average	4.37	33.98	20.72	20.84	1.71	2.41	1.34

Table 5: Percentage of solution time attributable to U-BTRAN, PRICE and INVERT when not exploiting hyper-sparsity, speedup for these components and total solution time when exploiting hyper-sparsity for test set $\mathcal{H}'$. Note that for MODSZK1, the improvement in the time for U-BTRAN was to a value less than the resolution of the timing mechanism.

Problem	CHUZC	I-FTRAN	U-FTRAN	CHUZR	I-BTRAN	U-BTRAN	PRICE	INVERT	Hyper-sparsity
80BAU3B	31.36	4.07	3.48	5.46	6.04	0.70	30.20	3.37	3.69
CYCLE	7.29	16.67	8.33	11.46	9.38	1.04	12.50	11.46	6.60
CZPROB	15.09	1.89	1.89	3.77	9.43	1.89	33.96	1.89	7.55
FIT2P	0.80	10.54	15.04	34.39	2.85	0.49	4.80	14.68	4.70
GREENBEA	9.92	13.51	7.74	9.92	10.46	0.44	26.70	10.19	3.44
GREENBEB	9.39	13.71	7.80	9.92	10.53	0.45	26.36	10.23	3.49
MAROS	7.27	12.73	7.27	10.91	9.09	1.82	16.36	12.73	5.80
MAROS-R7	1.42	14.54	14.40	3.39	31.24	0.20	11.07	21.18	0.54
SHIP12L	10.71	3.57	3.57	3.57	7.14	3.57	21.43	3.57	6.67
STOCFOR2	2.97	11.88	10.89	21.78	7.92	0.99	7.92	8.91	6.78
STOCHFOR	0.53	12.03	10.05	38.02	3.58	0.69	4.40	9.70	6.13
WOODW	19.79	7.49	2.67	5.88	5.88	0.53	36.90	4.81	3.69
DCP1	7.91	7.63	9.50	8.63	10.94	1.73	24.89	11.65	6.47
DCP2	9.05	6.34	7.26	2.20	12.43	0.76	45.09	6.68	6.34
DETEQ8	5.45	11.65	6.51	11.77	6.83	2.94	11.40	10.58	16.71
DETEQ27	6.89	9.81	9.34	15.89	7.49	1.47	10.97	9.48	15.27
CRE-A	12.68	5.99	4.58	7.04	9.51	1.06	25.70	5.99	10.81
CRE-C	14.16	5.15	5.58	6.87	10.73	1.29	24.46	6.44	9.19
KEN-07	3.70	3.70	3.70	3.70	7.41	3.70	7.41	3.70	16.00
KEN-11	6.53	2.58	4.41	3.65	17.93	4.71	16.26	7.29	14.72
KEN-13	11.72	1.70	2.75	1.76	20.28	2.64	38.41	4.27	7.63
KEN-18	17.89	1.23	1.71	0.87	23.04	1.13	40.89	2.74	4.92
PDS-02	7.14	4.29	2.86	5.71	7.14	2.86	20.00	7.14	18.75
PDS-06	10.50	5.20	3.43	7.38	7.90	1.66	23.18	8.32	16.81
PDS-10	10.87	6.00	4.05	8.03	7.43	1.31	24.54	8.16	15.71
PDS-20	13.05	4.42	4.93	8.75	7.59	0.67	33.17	7.57	10.96
Average	9.77	7.63	6.30	9.64	10.39	1.57	22.27	8.18	8.82

Table 6: Percentage of solution time for computational components of the revised simplex method and computational overhead attributable to exploiting hyper-sparsity for test set $\mathcal{H}$.

solver in OSL [11]. Each code was run for the problems in test set $\mathcal{H}$ with a workspace limit of 64Mb. To eliminate differences due to the different crash routines in the two simplex solvers, they were both started from the same basis. To avoid accusations of favouritism, the type 1 OSL crash basis was chosen. Other OSL crash types produced similar results. Note that the OSL barrier code failed to solve FIT2P within the workspace limit.

The results given in Table 7 show that, with the exception of MAROS-R7, EMSOL when exploiting hyper-sparsity is uniformly faster than the OSL simplex solver, and by more than an order of magnitude for some problems. This is despite the fact that OSL generally requires fewer iterations than EMSOL, significantly so for some of the larger problems. We believe that it should be possible to introduce algorithmic techniques into EMSOL to reduce the number of iterations required to solve a problem without increasing the computational cost per iteration. For example, OSL has a strategy for perturbing the problem when solving highly degenerate problems which can significantly reduce the number of iterations required to solve the problem.

Although the average speedup of EMSOL relative to OSL barrier is similar to the average speedup relative to OSL simplex, it is not uniformly so. However, for the larger problems, EMSOL is generally faster.

5.3 Comparison with OSL simplex and NETFLO for the NETGEN test set

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. To this end, EMSOL is compared with NETFLO, the efficient implementation of the network simplex method due to Kennington [12]. This is done using the NETGEN test set [13] of network problems which are now very modest in size, and a larger problem generated by the authors using the NETGEN program. The OSL simplex solver was also compared with NETFLO. Note that brief practical experience showed the OSL network solver to be very much less efficient than its simplex solver!

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 '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. 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 8 clearly show, for the smaller problems EMSOL is faster than OSL by about the same factor that EMSOL is slower than NETFLO. However, for NETGN201 the performance of EMSOL is somewhat closer to that of NETFLO than OSL is to EMSOL.

Problem	Simplex			Barrier
	Total speedup	Iteration count decrease	Iteration time speedup	Total speedup
80BAU3B	2.40	1.25	1.92	3.24
CYCLE	1.60	0.55	2.92	3.37
CZPROB	2.56	1.31	1.96	3.25
FIT2P	1.04	0.57	1.84	—
GREENBEA	1.27	0.62	2.05	0.43
GREENBEB	1.29	0.60	2.18	0.60
MAROS	1.44	0.72	2.02	1.81
MAROS-R7	0.61	0.64	0.96	1.60
SHIP12L	3.40	0.86	3.96	10.08
STOCFOR2	2.24	0.82	2.75	1.09
STOCHFOR	3.40	0.86	3.97	0.23
WOODW	2.12	0.64	3.33	2.90
DCP1	3.39	1.05	3.23	1.56
DCP2	3.86	0.81	4.79	0.59
DETEQ8	7.50	0.67	11.23	1.41
DETEQ27	7.75	0.61	12.68	1.58
CRE-A	3.09	0.65	4.73	1.33
CRE-C	2.38	0.63	3.80	1.80
KEN-07	7.08	1.00	7.09	7.12
KEN-11	26.65	0.99	26.88	3.07
KEN-13	11.33	0.95	11.89	0.58
KEN-18	19.51	0.98	19.98	0.45
PDS-02	3.67	0.81	4.53	26.04
PDS-06	4.29	0.61	7.04	14.82
PDS-10	4.94	0.58	8.45	16.23
PDS-20	1.74	0.28	6.29	9.39
Average	5.02	0.77	6.25	4.58

Table 7: Performance of EMSOL relative to OSL simplex and barrier.

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.

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.

	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 8: Solution time for EMSOL and OSL simplex relative to NETFLO.

Variants on the hyper-sparse FTRAN algorithm illustrated in Figure 3 may improve its practical performance. 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. There may also be some value in using the algorithm of Gilbert and Peierls [9] which determines the etas to be applied in a time proportional to the number of arithmetic operations which must be performed. This is better than the worst case performance of our algorithm without the mechanism for giving up exploiting hyper-sparsity when $|\mathcal{K}|$ becomes too large. However, since the Gilbert and Peierls algorithm must be run to completion for it to be of any use, it may approach its worst case behaviour if it is not abandoned when the solution of the system is not sparse. Experiments with these variants of our algorithm and a practical comparison of it with the Gilbert and Peierls algorithm will be the subject of future research.

For the subset of our test problems that do not exhibit significant hyper-sparsity in FTRAN, BTRAN or PRICE, the average speedup due to exploiting hyper-sparsity in other components is 1.34, and for those problems which do exhibit hyper-sparsity the average speedup is 5.61. For this latter subset of problems our implementation of the revised simplex which exploits hyper-sparsity has been shown to be many times faster that a commercial implementation of both the simplex and barrier method. When applied to network problems our implementation has been shown to approach the speed of an efficient implementation of the network simplex method. 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.

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] 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.

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

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

[6] I. S. Duff. On algorithms for obtaining a maximum transversal. *ACM Trans. Math. Softw.*, 7:315–330, 1981.

[7] 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.

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

[9] 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.

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

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

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

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

[14] 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.

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

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