Novel update techniques for the revised simplex method

Novel update techniques for the revised simplex method

Qi Huangfu · **J. A. Julian Hall**

7 July 2013

Technical Report ERGO-13-001

Abstract This paper introduces three novel techniques for updating the invertible representation of the basis matrix when solving practical sparse linear programming (LP) problems using a high performance implementation of the dual revised simplex method, being of particular value when suboptimization is used. Two are variants of the product form update and the other permits multiple Forrest-Tomlin updates to be performed. Computational results show that one of the product form variants is significantly more efficient than the traditional approach, with its performance approaching that of the Forrest-Tomlin update for some problems. The other is less efficient, but valuable in the context of the dual revised simplex method with suboptimization. Results show that the multiple Forrest-Tomlin updates are performed with no loss of serial efficiency.

Keywords Linear programming · dual revised simplex method · sparse matrices · basis inverse update techniques

Mathematics Subject Classification (2000) 90C05 · 65F05

1 Introduction

For particular classes of linear programming (LP) problems and families of related LP problems, the revised simplex method is generally the preferred solution technique. The opportunities and limitations of modern computer architectures, together with the desire to solve ever larger problem instances, continue to drive developments in high performance computing techniques for the simplex method [14, 19].

Qi Huangfu · J. A. Julian Hall
School of Mathematics, University of Edinburgh
JCMB, King's Buildings, Edinburgh EH9 3JZ, UK
E-mail: J.A.J.Hall@ed.ac.uk

The principal computational challenge when implementing the revised simplex method is the efficient solution of linear systems of equations whose matrix of coefficients is the simplex basis matrix or its transpose. There are standard efficient techniques [22,24] to obtain an invertible representation of a particular basis matrix and many more have been developed to update this representation by exploiting the relation between successive basis matrices, whereby one column is replaced as a result of each simplex iteration. The original and simplest technique for updating the invertible representation is the product form update of Dantzig and Orchard-Hays [6]. The class of updates that is widely accepted as being the most efficient derive from the work of Bartels and Golub [1], most notable being the techniques of Forrest and Tomlin [9], Reid [20] and Suhl and Suhl [23]. Several other schemes have been proposed, and of particular relevance to this paper is the work by Saunders *et al.* [8,12] and Hall [13] on generalising the Schur complement update of Bisschop and Meeraus [3]. A recent review of simplex update procedures is given by Elble and Sahinidis [7].

Motivated by the requirements of the high performance implementation of the revised simplex method (`hsol`) developed by Huangfu and Hall [16], this paper introduces three novel procedures for updating the invertible representation of the simplex basis matrix. Section 2 introduces the background necessary to discuss the new update procedures in Section 3. Results in Section 4 show that the serial performance of two of them is competitive with traditional techniques and conclusions are offered in Section 5.

2 Background

An LP problem in standard form is

$$\text{minimize } c^T x \quad \text{subject to } Ax = b, \quad x \geq 0,$$

where $x \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$. It may be assumed that the matrix A is of full rank. In the simplex method, the indices of variables are partitioned into sets $\mathscr{B}$ corresponding to m basic variables and $\mathscr{N}$ corresponding to $n-m$ nonbasic variables such that the basis matrix B formed from the columns of A corresponding to $\mathscr{B}$ is nonsingular. Each iteration of the simplex method chooses an index $q \in \mathscr{N}$ to exchange with the pth entry of $\mathscr{B}$. The corresponding change in the basis matrix B may be expressed as

$$\bar{B} = B + (a_q - Be_p)e_p^T. \tag{1}$$

In the (primal or dual) revised simplex method the indices p and q are identified by algorithmic techniques dependent on data obtained by solving linear systems of equations of the form

$$B\hat{x} = x \tag{2}$$

$$\hat{x}^T B = x^T \ (\text{or } B^T\hat{x} = x), \tag{3}$$

each at least once per iteration. The linear systems (2) and (3) are referred to as the *forward* and *transposed* system respectively. In particular, each revised simplex iteration requires the solution of the forward system

$$B\hat{a}_q = a_q \tag{4}$$

and the transposed system

$$\hat{e}_p^T B = e_p^T. \tag{5}$$

Such linear systems of equations are solved using an invertible representation of B whose efficient identification and use represent the principal computational challenge when implementing the revised simplex method.

There are standard efficient techniques [22,24] to obtain an invertible representation of a particular basis matrix directly, all of which use Gaussian elimination to identify an LU decomposition of B. The operation of finding the invertible representation directly in this way is referred to as `INVERT` and the matrix to which it is applied is denoted by B_0. Although `INVERT` involves row and column permutations, for convenience and with no loss of generality, they may be ignored so $B_0 = L_0 U_0$. This decomposition is represented as a product of elementary *eta* matrices of the form

$$E = \begin{bmatrix} 1 & & \eta_1 & & \\ & \ddots & \vdots & & \\ & & \eta_p & & \\ & & \vdots & \ddots & \\ & & \eta_m & & 1 \end{bmatrix}, \tag{6}$$

where η_p is referred to as the *pivotal entry* or simply the *pivot*, and the remaining entries in the p^{th} *pivotal* column form the *eta vector* η, for which the p^{th} entry is zero.

The solution procedure for the forward system (2) is referred to as `FTRAN` and, within it, the operation with E^{-1} is given by

$$E^{-1}x: \quad x_p := x_p/\eta_p \quad \text{and then} \quad x := x - x_p \eta. \tag{7}$$

For the backward system (3), the solution procedure is referred to as `BTRAN` and the corresponding operation with E^{-1} is given by

$$x^T E^{-1}: \quad y = x^T \eta \quad \text{and then} \quad x_p := (x_p - y)/\eta_p. \tag{8}$$

When the vector x is sparse the `FTRAN` operation (7) is easily skipped when x_p is zero. However, for `BTRAN` (8), the corresponding null operation occurs when y is zero. This is just as likely to occur but cannot be identified so easily. However, it is possible to represent the LU factors as a product of row eta matrices, each of which is of the form

$$R = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ r_1 & \dots & r_p & \dots & r_m \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}. \tag{9}$$

Thus the backward system (3) may be solved via a sequence of `FTRAN` operations (7) using the row-wise representation of the LU factors. This is observed, for example, by Hall and McKinnon [15].

2.1 Updating the invertible representation of B

Obtaining an invertible representation of the basis matrix following the column replacement (1) by updating the existing invertible representation is essential for efficiency reasons and is an operation referred to as `UPDATE`. The first and simplest such operation is the product form update of Dantzig and Orchard-Hays [6]. Many others have been proposed, but the update of Forrest and Tomlin [9] and the related Suhl update [23] are widely accepted as being the most efficient.

The product form update

The product form (PF) update rearranges (1) so that

$$\bar{B} = B + (a_q - Be_p)e_p^T = B\big(I + (\hat{a}_q - e_p)e_p^T\big) = BE$$

where $E = I + (\hat{a}_q - e_p)e_p^T$ is an eta matrix whose vector $\hat{a}_q$ is naturally available as the result of the forward system (4). Applying this k times yields the following representation of the basis matrix and its inverse.

$$B_k = B_0E_1E_2\ldots E_k \quad \Rightarrow \quad B_k^{-1} = E_k^{-1}\ldots E_2^{-1}E_1^{-1}B_0^{-1} \tag{10}$$

Thus the original decomposition $B_0 = L_0U_0$ remains unaltered when the basis matrix changes.

The Forrest-Tomlin update

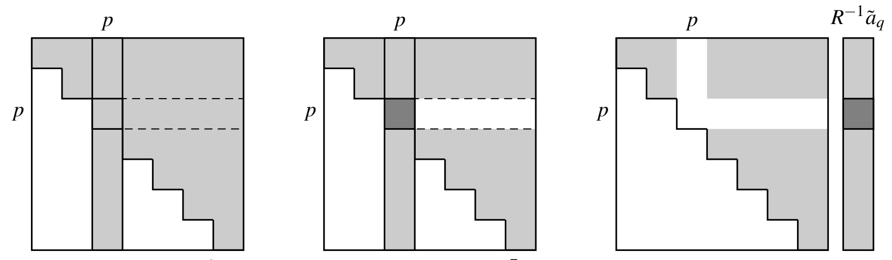
By allowing the decomposition $B = LU$ to be modified, the Forrest-Tomlin (FT) update [9] generally achieves greater efficiency with respect to sparsity than the PF update. This is done by working on the following rearrangement of the basis matrix update equation.

$$\begin{aligned} \bar{B} &= B + (a_q - Be_p)e_p^T \\ \Rightarrow \quad L^{-1}\bar{B} &= U + (L^{-1}a_q - Ue_p)e_p^T \\ &= U + (\tilde{a}_q - u_p)e_p^T = U' \end{aligned} \tag{11}$$

Whilst the basis update equation (1) replaces column p of basis matrix B by a_q in (11), column p of the factor U is replaced by the partial `FTRAN` result $\tilde{a}_q = L^{-1}a_q$. As illustrated in Figure 1(a), the replacement yields a spiked upper factor U'.

From U', the Forrest-Tomlin update restores triangularity by elimination. Specifically, it uses other rows to eliminate the off-diagonal entries of row p, yielding a permuted triangular matrix $\bar{U}$ as shown in Figure 1(b).

The elimination process can be represented by a single row transformation R^{-1}, so that $\bar{U} = R^{-1}U'$. Note that this row eta matrix R is a special case of (9) since it has pivotal entry $r_p = 1$ so R and its inverse can be expressed as $R = I + e_pr^T$ and $R^{-1} = I - e_pr^T$ respectively. The computation of the eta vector r was identified by Forrest and Tomlin [9] as an additional partial `BTRAN` operation $r^T = \bar{u}_p^TU^{-1}$, where



(a) Spiked upper: U' (b) After elimination: $\bar{U}$ (c) Actual implementation

Fig. 1 Forrest-Tomlin update: column spike and elimination

$\bar{u}_p^T$ is row p of U without the diagonal entry u_{pp}. It can easily be verified that applying R^{-1} to U as $R^{-1}U = U - e_p\bar{u}_p^T$ modifies only the entries in row p, eliminating all its off-diagonal entries. Since U and U' differ only in their pth column, applying R^{-1} to U' also eliminates its off-diagonal entries in row p. Meanwhile, applying R^{-1} to column p of U' modifies only the pth entry of the newly inserted vector $\tilde{a}_q$ which becomes $\tilde{a}_{pq} := \tilde{a}_{pq} - r^T\tilde{a}_q$. Therefore, applying the row transformation R^{-1} to U' yields the permuted triangular matrix $\bar{U}$ as shown in Figure 1(b).

As identified later by Tomlin [25], the additional partial BTRAN operation can be avoided by forming r from the intermediate result $\tilde{e}_p^T = e_p^T U^{-1}$ of the regularly solved transposed system (5). This can be identified by observing that $\bar{u}_p^T = e_p^T U - u_{pp}e_p^T$ so

$$r^T = \bar{u}_p^T U^{-1} = (e_p^T U - u_{pp}e_p^T)U^{-1} = e_p^T - u_{pp}\tilde{e}_p^T. \qquad (12)$$

Since its pth entry is zero, it follows that r is given by scaling $\tilde{e}_p$ by $-u_{pp}$ and setting the pth entry to zero. Thus, if the partial BTRAN result $\tilde{e}_p$ is stored, it can be assumed that the eta vector r is available at negligible cost.

Clearly $\bar{U} = R^{-1}U'$ is not a triangular matrix, but it can be put into this form via a symmetric cyclic permutation of rows and columns p to m. However, in practice, no permutations are performed since all that is required is an invertible representation of $\bar{U}$ rather than an explicit triangular matrix. Within an implementation, triangular matrices in basis matrix decompositions are represented in product form as sequences of eta vectors, pivotal entries and pivotal indices. Thus the representation of $\bar{U}$ is obtained by deleting the eta vector corresponding to column p of U, setting all entries corresponding to row p of U to zero and appending a new eta vector $R^{-1}\tilde{a}_q$, pivotal entry $\tilde{a}_{pq}$ and index p to the sequence, as illustrated in Figure 1(c).

Combining $\bar{B} = LU'$ (11) and $\bar{U} = R^{-1}U'$ yields the following updated representation of the basis matrix and its inverse.

$$\bar{B} = LR\bar{U} \quad \text{and} \quad \bar{B}^{-1} = \bar{U}R^{-1}L^{-1}$$

Repeating these operations, after k updates, the basis matrix B_k and its inverse can be expressed as

$$B_k = LR_1R_2\ldots R_kU_k \quad \Rightarrow \quad B_k^{-1} = U_k^{-1}R_k^{-1}\ldots R_2^{-1}R_1^{-1}L^{-1}. \qquad (13)$$

The essential difference between the FT and PF updates is that the former stores two partially transformed results $\tilde{a}_q$ and $\tilde{e}_p$ and deletes one row and one column from the eta file, whereas the PF update simply stores the final `FTRAN` result $\hat{a}_q$. These deletions and the sparsity of the partially transformed results relative to $\hat{a}_q$ is such that the FT update frequently has a significantly lower storage requirement than the PF update. This, in turn, leads to generally superior performance when the invertible representation based on the FT update is used to solve linear systems. However, the operations of the FT update require dynamic data structures to accommodate deletion and insertion, making it significantly more difficult to implement than the PF update.

For several reasons, the FT update is used as a benchmark for the experiments in Section 4 and as the basis of the CFT update introduced in Section 3.2. It is preferred to other FT-like updates such as that of Suhl and Suhl [23] since the latter's efficiency is compromised by the requirement for the `BTRAN` operation which, as observed above, can be avoided with the FT update. The FT update is chosen over the Reid update [20] since the former is preferable when solving hyper-sparse LP problems. The results in Section 4 comparing Huangfu and Hall's FT-based solver `hsol` to `clp` and `cplex` give an empirical justification for this choice of update technique.

3 New update techniques

This section introduces two novel variants of the product form update and an extension of the Forrest-Tomlin update. Although applicable in general, all are motivated by the requirements of the high performance implementation by Hall and Huangfu [14, 16] of the dual revised simplex method with suboptimization introduced by Rosander [21].

3.1 Variants of the product form update

Although the variants of the product form update set out below are relatively simple, the authors are unaware of them having been described before. This is possibly because, until now, the scope for them to be useful has not arisen.

The elementary matrix for a general rank-one update

The updates introduced below are based on elementary matrices for rank-one updates which are more general than the eta matrices of LU factors or the PF and FT updates. This general elementary matrix can be expressed as

$$T = I + uv^T, \tag{14}$$

where u and v are arbitrary compatible vectors. When $v = e_p$, T is an eta matrix E (6) and when $u = e_p$, T is a row-wise eta matrix R (9). When T is nonsingular,

$$T^{-1} = I - \frac{1}{\mu}uv^T, \tag{15}$$

where $\mu = 1 + v^T u$. Clearly T and its inverse can be represented by u and v. Although strictly unnecessary, when operating with T^{-1} it is also convenient to record the value of μ. Solving linear systems with T is straightforward, the operations with T^{-1} for `FTRAN` being

$$T^{-1}x = x - \frac{v^T x}{\mu} u, \tag{16}$$

and those for `BTRAN`

$$x^T T^{-1} = x^T - \frac{x^T u}{\mu} v^T. \tag{17}$$

3.1.1 Alternate product form update

Working from the basis update expression (1), if B is taken out as a factor on the *right* then

$$\bar{B} = (I + (a_q - Be_p)\hat{e}_p^T)B = (I + (a_q - a_{p'})\hat{e}_p^T)B,$$

where p' is the index within A of column p of B. The matrix $T = I + (a_q - a_{p'})\hat{e}_p^T$ is in the general form (14) so, using (15), the inverse of the updated basis matrix is given by

$$\bar{B}^{-1} = B^{-1}(I - \frac{1}{\mu}(a_q - a_{p'})\hat{e}_p^T),$$

where $\mu = 1 + \hat{e}_p^T(a_q - a_{p'}) = \hat{a}_{pq}$ is the p^{th} entry of $\hat{a}_q$. In contrast to the PF update, which requires the `FTRAN` result $\hat{a}_q$, the new update formula uses the `BTRAN` result $\hat{e}_p$ and thus is called the *alternate* product form (APF) update. It follows from (16) that the operation with T^{-1} for `FTRAN` is

$$T^{-1}x: \quad y = \hat{e}_p^T x/\mu \quad \text{and then} \quad x := x - y(a_q - a_{p'})$$

and, from (17), that the corresponding operation for `BTRAN` is

$$x^T T^{-1}: \quad y = x^T(a_q - a_{p'})/\mu \quad \text{and then} \quad x^T := x^T - y\hat{e}_p^T.$$

After k APF updates, the basis matrix and its inverse can be expressed as

$$B_k = T_k T_{k-1} \ldots T_1 B_0 \quad \Rightarrow \quad B_k^{-1} = B_0^{-1} T_1^{-1} \ldots T_{k-1}^{-1} T_k^{-1}.$$

Note that, in contrast to the PF update, the elementary matrices which constitute the update are applied before rather than after the LU decomposition of B_0 when solving forward systems with B_k^{-1}.

Discussion

The relation between the APF and PF updates may be viewed as being analogous to that between the *alternative block LU* update (introduced and implemented by Hall [13]) and the *Block LU* (BLU) update of Eldersveld and Saunders [8]. However, the APF is distinctive since it avoids the requirement for an invertible representation of a Schur complement. This yields a significant overhead when large numbers of updates are performed.

Relative to the PF update, trading operations with x_p for operations with $a_q - a_{p'}$ makes the APF update appear unattractive. However, there is minimal additional storage overhead since a_q and $a_{p'}$ are columns from the coefficient matrix A so may be represented by the indices q and p'. Hall and McKinnon [15] also identify (classes of) LP problems where it is typical for $\hat{a}_q$ to be dense but $\hat{e}_p$ to be sparse, in which case the APF update will require significantly less storage. Performance-wise, unless $\hat{e}_p$ is significantly more sparse than $\hat{a}_q$, the overhead of the operations with $a_q - a_{p'}$ is such that using the APF rather than the PF update can be expected to be less efficient.

Since the cost of using the APF update corresponds to the density of the final `BTRAN` result $\hat{e}_p$, rather than partial results $\tilde{a}_q$ and $\tilde{e}_p$ that underpin the FT update, it is expected that, like the PF update, the storage requirement and performance of the APF update will be inferior to those of the FT update. However, it is shown by Huangfu and Hall in [16] that the APF is particularly valuable in the context of a high performance parallel scheme for the dual simplex method since it permits particular multiple `FTRAN` operations to be performed as a single `FTRAN`. The number of elementary APF operations that must be performed is of the same order as the number of `FTRAN` operations. Thus the saving of all but one of these `FTRAN`s that is achieved by using the APF update is not expected to be compromised by the overhead of maintaining or applying APF updates rather than an alternative scheme.

3.1.2 Middle product form update

The *middle* product form (MPF) update inserts the updates in product form into the middle of factors L and U. In detail, the MPF update is derived as follows from the basis update expression (1), assuming that $B = LU$.

$$\begin{aligned}\bar{B} &= LU + (a_q - Be_p)e_p^T \\ &= LU + LL^{-1}(a_q - Be_p)e_p^T U^{-1}U \\ &= L(I + (\tilde{a}_q - Ue_p)\tilde{e}_p^T)U \\ &= L(I + (\tilde{a}_q - u_p)\tilde{e}_p^T)U\end{aligned}$$

where $\tilde{a}_q = L^{-1}a_q$ and $\tilde{e}_p^T = e_p^T U^{-1}$ are partial `FTRAN` and `BTRAN` results respectively, and $u_p = Ue_p$ is the p^{th} column of U. The matrix $T = I + (\tilde{a}_q - u_p)\tilde{e}_p^T$ is in the general form (14) so, using (15), the inverse of the updated basis matrix is given by

$$\bar{B}^{-1} = U^{-1}(I - \frac{1}{\mu}(\tilde{a}_q - u_p)\tilde{e}_p^T)L^{-1},$$

where $\mu = 1 + \tilde{e}_p^T(\tilde{a}_q - u_p) = \hat{a}_{pq}$ is the pth entry of $\hat{a}_q$. It follows from (16) that the operation with T^{-1} for `FTRAN` is

$$T^{-1}x: \quad y = \tilde{e}_p^T x/\mu \quad \text{and then} \quad x := x - y(\tilde{a}_q - u_p)$$

and, from (17), that the corresponding operation for `BTRAN` is

$$x^T T^{-1}: \quad y = x^T(\tilde{a}_q - u_p)/\mu \quad \text{and then} \quad x^T := x^T - y\tilde{e}_p^T.$$

Note that T is inserted after L so the invertible representation of $\bar{B}$ is given by $\bar{L} = LT$ and $\bar{U} = U$. Since U remains unchanged, its pth column is always available directly. After k updates, the basis matrix and its inverse can be expressed as

$$B_k = L_0 T_1 T_2 \ldots T_k U_0 \quad \Rightarrow \quad B_k^{-1} = U_0^{-1} T_k^{-1} \ldots T_2^{-1} T_1^{-1} L_0^{-1}.$$

Discussion

The relation between the MPF and PF updates is analogous to that between the *partitioned LU* update (introduced by Gill *et al.* [12] but not implemented) and the BLU update of Eldersveld and Saunders [8]. However, like the APF, the MPF is distinctive since it avoids a Schur complement.

The MPF update is comparable with the FT update since both are based on the partial `FTRAN` result $\tilde{a}_q$ and partial `BTRAN` result $\tilde{e}_p$. However the MPF has no deletion corresponding to that of the FT update so will have greater storage overhead. In terms of update efficiency, the MPF update is preferable since it only involves storing operations, whereas the FT update needs additional deletion and insertion work to update the factor U. In terms of solving efficiency, it is expected to be preferable to operate with the FT update since it replaces u_p by $\tilde{a}_q$ and stores only $\tilde{e}_p$ as an eta matrix R. Thus solving with the FT update requires only additional simple eta matrix operations with R, while solving with the MPF requires slightly more complicated and time-consuming operations with T.

Concerns over the numerical stability of product form representations of B^{-1} motivated the introduction of LU decompositions into simplex implementations by Bartels and Golub [2] and partial pivoting for stability when updating the LU decomposition [1]. Although some FT-like update procedures derived from the work of Bartels and Golub have incorporated threshold pivoting, modern descriptions of such update procedures and codes based on them [5,18,20,23] include no numerical pivoting. Thus the superior numerical performance of FT-like updates must be assumed to derive from the following two observations. The first is the fact that FT-like updates may delete etas corresponding to ill-conditioned bases, whereas PF-like updates retain all etas. The second follows from the observation (see results in Section 4) that FT-like updates have lower storage requirements, implying fewer floating-point operations and, hence, a lower probability of numerical instability actually occurring. In practice, numerical difficulties with PF-like updates are not encountered when solving most classes of LP problem, particularly those which are hyper-sparse since little numerical elimination is necessary. Any poor numerical behaviour is readily identified and rectified by reinversion so it is very rare for a good PF-based solver to fail to solve an LP problem. Finally, in the context of Huangfu and Hall's FT-based simplex solver `hsol` [16], PF-like updates are only used for very limited numbers of basis changes and no numerical instability has been observed.

3.2 Collective Forrest-Tomlin update

The collective Forrest-Tomlin (CFT) update is designed to perform a set of t Forrest-Tomlin operations simultaneously to obtain the FT invertible representation of B_{k+t} directly from that of B_k. This requirement arises naturally within the dual simplex method with suboptimization.

The dual simplex method with suboptimization

The details of this simplex variant are given by Huangfu and Hall [16]. However, it is necessary to set out some of its data requirements to motivate the collective Forrest-Tomlin update described below. Suboptimization in the dual simplex method requires the vectors $\hat{e}_p^T = e_p^T B_k^{-1}$ for p in a small subset $\mathscr{P}$ of the rows. Iterations of suboptimization then identify t basis changes given by $\{(p_i,\, q_i)\}_{i=0}^{t-1}$, where $\{p_i\}_{i=0}^{t-1} \subseteq \mathscr{P}$. When, during the course of suboptimization, the vector $e_{p_i}^T B_{k+i}^{-1}$ is required, it is obtained from $\hat{e}_{p_i}$ by a few APF operations. Following the suboptimization iterations, updating data for the whole LP problem requires the pivotal columns $\hat{a}_{q_i} = B_{k+i}^{-1} a_{q_i}$ for $i = 0, \ldots, t-1$, which are also obtained in two steps as $\hat{a}_q = B_k^{-1} a_q$ and then regular PF operations.

The Forrest-Tomlin update after suboptimization

To perform Forrest-Tomlin updates after suboptimization requires

$$\tilde{a}_{q_i} = R_{k+i}^{-1} \ldots R_k^{-1} \ldots R_1^{-1} L^{-1} a_{q_i} \quad \text{and} \quad \tilde{e}_{p_i}^T = e_{p_i}^T U_{k+i}^{-1}, \tag{18}$$

for $i = 0, \ldots, t-1$, so that the next elimination matrix R_{k+i+1} can be constructed using $\tilde{e}_{p_i}^T$ and the new column transformation can be formed as $R_{k+i+1}^{-1} \tilde{a}_{q_i}$. In the sequential simplex method, they are naturally available as partial results of the routinely solved forward (4) and transposed (5) systems. In suboptimization, initialisation of $\hat{e}_p^T = e_p^T B_k^{-1}$ and the subsequent computation of $\hat{a}_q = B_k^{-1} a_q$ yield only

$$\bar{a}_{q_i} = R_k^{-1} \ldots R_1^{-1} L^{-1} a_{q_i} \quad \text{and} \quad \bar{e}_{p_i}^T = e_{p_i}^T U_k^{-1}. \tag{19}$$

The challenge, therefore, is to obtain the partial results for the appropriate basis (18) from the partial results (19) obtained naturally, for $i = 1, \ldots, t-1$. The results for $i = 0$ are known. The rest of this section will discuss how to achieve this efficiently using simple linear algebra operations.

3.2.1 *Updating the partial* FTRAN *results*

Updating the partial FTRAN results $\bar{a}_{q_i}$ to $\tilde{a}_{q_i}$ is a relatively straightforward task. By comparing the available (19) and the required (18) results, the updating operation is readily identified as applying new row eta transformations after R_k,

$$\tilde{a}_{q_i} = R_{k+i}^{-1} \dots R_{k+2}^{-1} R_{k+1}^{-1} \bar{a}_{q_i},$$

where each new row transformation $R_{k+j}, (0 < j \leq i)$ is available once $\tilde{e}_{p_{j-1}}$ is computed. Thus the computation of $\tilde{a}_{q_i}$ is scheduled after the computation of $\tilde{e}_{p_{i-1}}$, which corresponds to the latest required row transformation R_{k+i}.

3.2.2 *Updating the partial* BTRAN *results*

Compared to updating the partial FTRAN results, updating the partial BTRAN results from

$$\bar{e}_{p_i}^T = e_{p_i}^T U_k^{-1} \quad \text{to} \quad \tilde{e}_{p_i}^T = e_{p_i}^T U_{k+i}^{-1}$$

is more complicated because the difference between U_k and U_{k+i} when $i \geq 1$ involves (multiple) replacements and eliminations. However, by carefully rearranging the replacement and elimination involved, computation of $\tilde{e}_{p_i}$ can still be achieved by simple linear algebra operations. Their derivation is explored in detail by updating $\bar{e}_{p_1}$ to $\tilde{e}_{p_1}$, with reference to the first two upper factors U_k and U_{k+1}.

Recall that the Forrest-Tomlin update derives U_{k+1} via the following elimination operations on the spiked matrix U_k'

$$U_{k+1} = R_{k+1}^{-1} U_k', \tag{20}$$

where $U_k' = U_k + (\tilde{a}_{q_0} - U_k e_{p_0}) e_{p_0}^T$. Rearranging U_k' by taking out U_k as a factor on the left, it follows that

$$\begin{aligned} U_k' &= U_k + U_k (U_k^{-1} \tilde{a}_{q_0} - e_{p_0}) e_{p_0}^T \\ &= U_k \left(I + (\hat{a}_{q_0} - e_{p_0}) e_{p_0}^T \right) \\ &= U_k E_{k+1}, \end{aligned}$$

where $E_{k+1} = I + (\hat{a}_{q_0} - e_{p_0}) e_{p_0}^T$ is the eta matrix used in the PF update (10). By substituting $U_k' = U_k E_{k+1}$ into (20), U_{k+1} and its inverse can be represented by

$$U_{k+1} = R_{k+1}^{-1} U_k E_{k+1} \quad \text{and} \quad U_{k+1}^{-1} = E_{k+1}^{-1} U_k^{-1} R_{k+1}. \tag{21}$$

By using this new representation for U_{k+1}^{-1}, the calculation of $\tilde{e}_{p_1}$ can be considered in three steps applying E_{k+1}^{-1}, U_k^{-1} and R_{k+1} respectively. Using y to represent intermediate results, starting from $y = e_{p_1}$, the following three-step process is given by (21).

(1) Applying E_{k+1}^{-1} updates y by

$$y^T := y^T E_{k+1}^{-1} = e_{p_1}^T E_{k+1}^{-1} = \begin{cases} e_{p_1}^T + \mu e_{p_0}^T & p_1 \neq p_0 \\ \alpha e_{p_1}^T & p_1 = p_0 \end{cases}$$

where $\alpha = 1/\hat{a}_{p_0q_0}$ and $\mu = -\hat{a}_{p_1q_0}/\hat{a}_{p_0q_0}$ are the pivotal entry and the p_1^{th} entry of the eta vector of E_{k+1}^{-1}. The first situation $p_1 \neq p_0$ is the common case, which is implied in the suboptimization framework, as each row in $\mathscr{P}$ is used only once. In more general applications, it is possible to have $p_1 = p_0$. However, since this case corresponds to $\mu = 0$ and a scaling of y, it is reasonable and convenient to omit it from the following analysis.

(2) Applying U_k^{-1} to y gives

$$y^T := y^T U_k^{-1} = (e_{p_1}^T + \mu e_{p_0}^T)U_k^{-1} = \bar{e}_{p_1}^T + \mu \tilde{e}_{p_0}^T.$$

This expresses y in terms of the available partial `BTRAN` results $\tilde{e}_{p_0}$ and $\bar{e}_{p_1}$, where both are computed with U_k^{-1}.

(3) Applying R_{k+1} to y completes the process, giving the required partial `BTRAN` result $\tilde{e}_{p_1}$ thus.

$$\tilde{e}_{p_1}^T = y^T R_{k+1} = \bar{e}_{p_1}^T R_{k+1} + \mu \tilde{e}_{p_0}^T R_{k+1} \tag{22}$$

When transposed as $R_{k+1}^T \bar{e}_{p_1}$, calculation of the first term in (22) is seen to be a standard `FTRAN` operation (7) which adds a multiple $[\bar{e}_{p_1}]_{p_0}$ of r_{k+1} to $\bar{e}_{p_1}$ so

$$\bar{e}_{p_1}^T R_{k+1} = \bar{e}_{p_1}^T + [\bar{e}_{p_1}]_{p_0} r_{k+1}^T. \tag{23}$$

As for the second term in (22), by substituting for R_{k+1} using (9) and r_{k+1} using (12),

$$\begin{aligned} \mu \tilde{e}_{p_0}^T R_{k+1} &= \mu \tilde{e}_{p_0}^T (I + e_{p_0} r_{k+1}^T) \\ &= \mu \tilde{e}_{p_0}^T + \mu (e_{p_0}^T - u_{p_0p_0} \tilde{e}_{p_0}^T)/u_{p_0p_0} \\ &= \frac{\mu}{u_{p_0p_0}} e_{p_0}^T \end{aligned} \tag{24}$$

so the second term in (22) is seen as adding the scalar value $\mu/u_{p_0p_0}$ to y_{p_0}. Substituting (23) and (24) for the two terms in (22) yields the following update formula.

$$\tilde{e}_{p_1}^T = \bar{e}_{p_1}^T + [\bar{e}_{p_1}]_{p_0} r_{k+1}^T + \frac{\mu}{u_{p_0p_0}} e_{p_0}^T \tag{25}$$

In practice, to obtain $\mu = -\hat{a}_{p_1q_0}/\hat{a}_{p_0q_0}$ requires only one inner product, $\hat{a}_{p_1q_0} = \bar{e}_{p_1}^T \tilde{a}_{q_0}$, since $\hat{a}_{p_0q_0}$ is available as the simplex pivotal entry. Thus updating the partial `BTRAN` result from $\bar{e}_{p_1}$ to $\tilde{e}_{p_1}$ involves only one vector addition and the evaluation of one inner product.

The partial `BTRAN` result $\bar{e}_{p_2}^T = e_{p_2}^T U_k^{-1}$ corresponding to the third pivot choice, can be updated in two steps, firstly to $e_{p_2}^T U_{k+1}^{-1}$ by using (25), and then to $e_{p_2}^T U_{k+2}^{-1}$ by repeating the same operations with partial results $\tilde{e}_{p_1}$ and $\tilde{a}_{q_1}$. Using this stepwise updating procedure for $i = 1, \ldots, t-1$, the partial `BTRAN` result $\bar{e}_{p_i}$ corresponding to the $(i+1)^{\text{st}}$ pivot choice can be updated to the required form $\tilde{e}_{p_i}$ in i steps of the form (25). Note that to apply (25) requires the previous partial `FTRAN` results $\tilde{a}_{p_j}$ for $j = 0, \ldots, i-1$. Thus the updating of $\bar{e}_{p_i}$ is scheduled after the computation of last required partial `FTRAN` result $\tilde{a}_{q_{i-1}}$. Therefore, together with the data requirement of operations with $\tilde{a}_i$ discussed previously, the calculation for updating the two types of partial transformed results is interlaced as the sequence $\tilde{a}_{q_1}, \tilde{e}_{p_1}, \tilde{a}_{q_2}, \tilde{e}_{p_2}$, etc.

3.3 New techniques and hyper-sparsity

For hyper-sparse LP problems, it is generally possible to skip most of the operations with elementary matrices in the invertible representation of B_k, so it is important to identify the operations which must be performed at a cost proportional to the number of resulting floating-point operations. This is readily achieved by established techniques [11,15] if the elementary matrices are column (6) or row (9) eta matrices. Thus hyper-sparsity may be exploited fully with the PF or FT updates. However, in the case of the APF and MPF updates, the form of the general elementary matrix (14) prohibits the extension of these techniques within the phase of `FTRAN` and `BTRAN` associated with `UPDATE`.

4 Results

The serial efficiency of the novel updates introduced in Section 3 is assessed in this section using the set of 30 LP test problems in Table 1.

The LP problems and testing environment

The LP test problems are drawn from the Netlib [10] and Kennington [4] test sets, as well as other publicly-available collections. Although not particularly large, the range of algebraic properties exhibited by these problems is representative of practical and large scale problems. The experiments are performed using modules of Huangfu's dual revised simplex solver `hsol` which uses the Forrest-Tomlin update. As a measure of the computational efficiency of these modules, results in Table 2 give the solution time, iteration count and iteration time of `hsol` and the dual revised simplex solvers of `clp` 1.15 [5] and `cplex` 12.4 [17]. These results are illustrated using the performance profiles in Figures 2 and 3 which show, for each solver, the proportion of problems for which the particular measure relative to the best solver is within the factor on the horizontal axis. It is clear from these results that `hsol` is competitive with `cplex` and `clp`, both overall and per iteration, so the observations on the results presented below for the update techniques developed in this paper are applicable to the most efficient of simplex solvers.

Of particular relevance to the experiments discussed below is the average relative size of the matrix B_0 and its invertible representation, given in the column of Table 1 headed "Fill-in", and the proportion of the results of `FTRAN` and `BTRAN` with density less than 10%, given in the corresponding columns of Table 1. Hall and McKinnon [15] introduced this measure of an LP problem's hyper-sparsity and discuss reasons for the extreme variance between these values for some problems. The claim that the LP test set is representative is based on the range of values for these three measures, as well as problem size. Unsurprisingly there is clear correlation between the fill-in during `INVERT` and the measures of hyper-sparsity.

For a given LP problem and simplex update technique there is an optimal interval between successive `INVERT` operations which balances the cost of `INVERT` and the overheads associated with having an updated invertible representation. The particular

Name	Rows	Nz/Col	Fill-in	FTRAN	BTRAN	Interval	PF	APF	MPF	FT	CFT
CRE-B	9649	4.53	1.01	100	83	175	0.44	0.86	**0.37**	0.59	0.40
DANO3MIP_LP	3203	5.74	1.42	1	7	101	24.99	26.77	**10.89**	7.93	7.26
DCP2	32388	26.53	1.02	100	97	437	13.63	7.49	**6.23**	2.96	3.00
DFL001	6072	3.42	1.29	38	62	153	11.59	15.91	**5.07**	4.01	3.61
FOME12	24285	3.42	1.29	50	63	312	82.62	126.04	**40.23**	28.26	25.55
GEN4	1538	25.64	2.22	1	15	123	1.13	**0.77**	1.20	0.78	0.78
KEN-18	105128	3.31	1.00	100	100	942	9.61	9.95	**4.03**	2.36	2.34
L30	2702	4.21	2.06	10	8	106	6.95	6.60	**2.69**	2.10	2.00
LINF_520C	93326	8.21	2.04	10	11	548	17367.06	12358.58	**3530.39**	1353.97	1278.56
LP22	2959	5.88	1.82	16	25	132	12.47	12.51	**6.89**	4.73	4.40
MAROS-R7	3137	16.06	2.00	5	13	136	7.37	**5.35**	5.69	3.58	3.52
MOD2	35665	6.94	1.03	46	68	274	70.84	58.26	**19.03**	15.99	13.75
NS1688926	32768	103.22	1.00	72	100	346	44.74	**8.62**	9.28	8.94	8.90
OSA-60	10281	7.00	1.00	100	99	247	0.05	0.09	**0.04**	0.03	0.03
PDS-20	33875	2.88	1.01	100	98	452	2.93	5.56	**1.80**	1.35	1.20
PDS-40	66845	2.85	1.01	100	98	641	16.88	33.82	**8.42**	5.90	5.46
PDS-100	156244	2.75	1.00	100	99	988	47.50	156.00	**26.39**	15.36	14.67
PILOT87	2031	15.11	2.06	10	19	144	4.12	4.04	**2.95**	1.79	1.74
QAP12	3193	5.00	4.13	2	16	214	151.22	167.04	**74.47**	41.82	39.94
SELF	960	156.01	1.68	0	2	131	**5.02**	6.00	5.09	4.14	4.15
SGPF5Y6	246077	2.68	1.00	100	100	1444	24.87	20.39	**11.49**	5.17	6.00
STAT96V1	5995	2.98	1.58	24	4	119	22.12	23.41	**5.36**	4.60	4.19
STAT96V4	3173	7.88	1.25	74	31	189	15.39	28.97	**9.03**	6.90	6.41
STORMG2-125	66186	3.36	1.00	100	100	730	4.98	2.06	**1.76**	0.93	1.10
STORMG2-1000	528186	3.36	1.00	100	100	2020	386.03	**55.56**	60.43	29.78	30.89
STP3D	159488	3.23	1.04	97	71	487	461.42	906.36	**128.10**	124.00	103.14
TRUSS	1001	4.16	1.37	41	2	68	1.20	2.12	**0.92**	0.73	0.63
WATSON_1	201156	2.74	1.00	100	100	1531	31.08	16.00	**11.17**	4.08	4.75
WATSON_2	352013	2.74	1.00	100	100	2132	45.93	16.70	**15.89**	5.20	6.49
WORLD	35511	6.74	1.04	41	61	256	81.94	73.98	**22.50**	19.30	16.33

Table 1 LP test problems, fill-in factor for the representation of B_0^{-1}, sparsity of FTRAN and BTRAN results, reinversion interval and CPU time performing FTRAN and BTRAN and UPDATE operations for different simplex update techniques

	`hsol`			`clp`			`cplex`		
	Solution	Iteration		Solution	Iteration		Solution	Iteration	
Problem	Time	Count	Time	Time	Count	Time	Time	Count	Time
CRE-B	6.4	11599	551	4.2	11489	368	2.1	16030	130
DANO3MIP_LP	53.2	60161	885	31.8	42506	747	8.8	23608	372
DCP2	12.9	25360	508	15.3	24456	626	5.6	23432	237
DFL001	16.0	26322	607	17.9	26866	667	9.8	20255	483
FOME12	101.1	103005	982	66.9	84818	789	78.6	90732	867
GEN4	5.3	1107	4757	16.7	4157	4025	16.5	5276	3125
KEN-18	13.6	107471	127	19.3	106158	182	8.5	96533	88
L30	11.2	10290	1091	11.5	10233	1124	8.1	10857	742
LINF_520C	3646.8	132244	27576	3349.1	140119	23902	8242.8	117755	70000
LP22	22.2	25080	883	21.0	22963	913	14.9	24308	612
MAROS-R7	11.5	6025	1914	12.0	5531	2172	3.9	6585	596
MOD2	53.2	43386	1226	31.3	35308	886	26.2	47995	547
NS1688926	25.0	13849	1804	4424.7	196353	22535	52.8	23819	2217
OSA-60	6.5	4694	1394	2.1	4792	434	0.8	5014	158
PDS-20	6.9	38872	177	8.6	40183	213	4.5	22825	199
PDS-40	28.8	94914	304	36.5	104217	350	15.6	56126	278
PDS-100	87.4	234184	373	194.7	319231	610	56.6	163076	347
PILOT87	7.0	7240	970	7.2	7179	1009	4.7	7285	645
QAP12	158.3	128131	1235	146.7	89194	1645	139.9	140184	998
SELF	40.2	4738	8483	30.2	4713	6406	54.4	15281	3559
SGPF5Y6	180.2	348115	518	242.1	360809	671	7.9	97918	81
STAT96V1	97.6	16904	5777	82.5	17984	4588	51.9	14226	3646
STAT96V4	157.5	72531	2172	52.3	34657	1510	79.0	43124	1832
STORMG2-125	9.5	81869	116	23.5	89988	262	4.4	92755	47
STORMG2-1000	530.7	658534	806	1876.8	730471	2569	228.2	873329	261
STP3D	602.5	130689	4610	350.0	115329	3035	328.7	114533	2870
TRUSS	8.2	18929	433	5.5	18817	290	4.2	19693	213
WATSON_1	50.4	238973	211	65.8	223522	294	9.6	87148	110
WATSON_2	55.1	334733	165	12976.9	498860	26013	35.3	358621	99
WORLD	66.1	47104	1402	41.0	41279	993	22.2	50538	438

Table 2 Solution time (s), iteration count and iteration time (μs) for the `hsol`, `cplex` and `clp` dual revised simplex solvers

optimal interval can be expected to vary as the simplex method solves a given LP. In theory this interval should be chosen so that the time associated with `INVERT` equals the total time forming and operating with the etas associated with `UPDATE`. To ensure deterministic performance, in `hsol` the interval is determined using a flop-based pseudo-clock and practical experience shows that performance is not sensitive to the parameters in the model. The sequence of optimal reinversion intervals was estimated for the FT update, and the average value is given in the column headed "Interval".

Experiments with the PF, APF, MPF, FT and CFT updates were performed using the same sequence of basis changes from a "logical" basis $B = I$ to an optimal solution of the LP. This eliminates variance in the results due to the fact that a change in update technique typically leads to the simplex method taking a different path to an optimal solution. The same sequence of optimal (FT) reinversion intervals was also used for all experiments with a given problem. This facilitates some comparisons between methods and ensures that the FT update is not disfavoured. Results for each update

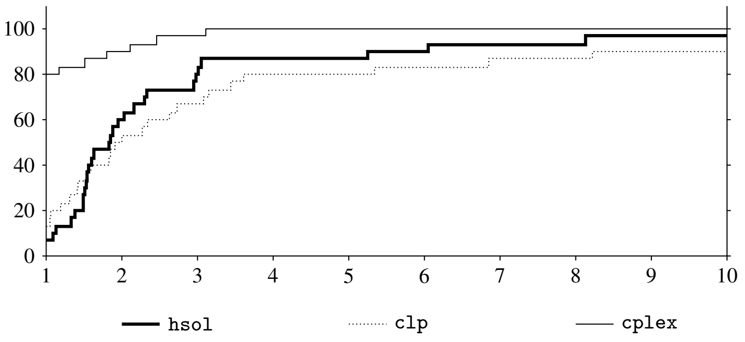


Fig. 2 Performance profiles for the solution time of `hsol`, `clp` and `cplex` on the test problems

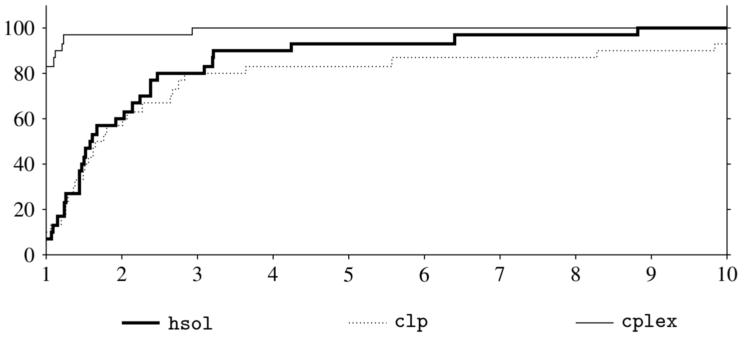


Fig. 3 Performance profiles for the iteration time of `hsol`, `clp` and `cplex` on the test problems

technique's particular sequence of optimal reinversion intervals were obtained but did not lead to deeper insight or any alternative conclusions so are omitted.

Results and analysis

The principal measure of the efficiency of a particular update procedure is the total CPU time to perform the update and then perform FTRAN and BTRAN for each basis in the sequence. This is given in the columns headed PF, APF, MPF, FT and CFT in Table 1 where, for each LP problem, the best performance of the three product form updates is highlighted in bold.

The superiority of the MPF update over the PF and APF updates is clear: it is the best for 26 of the 30 problems. Of the other four problems, the APF update is the best for three and in all of these cases the proportion of sparse BTRAN results is at least that of FTRAN. However, on 16 of the 30 problems the APF update is the worst, with the PF update being the worst for all but one of the others. A further measure of the superiority of the MPF update is obtained by considering the CPU time relative to the better of the CPU time for the PF and APF updates. The geometric mean of this ratio shows the use of the MPF update to be 40% more efficient.

The FT update is better than the best of the product form updates for 27 of the problems and, on average, is 61% more efficient than the better of the PF and APF updates. However, the FT update is only 34% more efficient than the MPF update. Thus the efficiency of the MPF update is rather closer to that of the FT update than it is to the other product form updates.

The performance when using the collective FT update (CFT) is very similar to that when using the standard FT update, the former being 4% faster. This demonstrates that the CFT update incurs no significant overhead.

A secondary measure of efficiency is the storage requirement so, for each LP problem and each update technique, the relative size of B_k and its invertible representation was calculated. The relative performances of the update procedures was similar to the CPU results, so details are not given. Using the MPF update was 31% more efficient than the better of the PF and APF updates. Using the FT update was 16% more efficient than the MPF update and 42% more efficient than the better of the PF and APF updates. There was no significant difference between the average fill-in for the FT and CFT updates.

Although the deletion and insertion operations of the FT update result in a CPU overhead that is not shared with the product form updates, the overall computational efficiency when using the FT update rather than the MPF or the best of the product form updates is greater than the storage efficiency. This is due to the fact that hyper-sparsity may be exploited fully when using the FT update, but only when operating with B_0^{-1} in the case of the APF and MPF updates.

Further insight into why the MPF update is significantly more efficient than the PF and APF updates is given in the results presented in Table 3. Although it is unsurprising that the average density of the vectors $\tilde{a}_q$ and $\tilde{e}_p$ is lower than the average density of $\hat{a}_q$ and $\hat{e}_p$ for the PF and APF updates respectively, the relative densities are generally so much lower that it is much more efficient to store and apply $\tilde{a}_q$ and $\tilde{e}_p$ rather than just one of $\hat{a}_q$ and $\hat{e}_p$. An intuitive explanation follows from the observation that $\tilde{a}_q$ and $\tilde{e}_p$ are the solution of triangular rather than square systems of equations. Specifically, the initial fill-in which occurs when forming $\tilde{a}_q = L^{-1}a_q$ and $\tilde{e}_p^T = e_p^T U^{-1}$ is likely to imply the need to apply etas in the representation of U^{-1} and L^{-1} when forming $\hat{a}_q = U^{-1}\tilde{a}_q$ and $\hat{e}_p^T = \tilde{e}_p^T L^{-1}$, resulting in significant further fill-in. However, the initial fill-in is very much less likely to require the application of further etas in the representation of L^{-1} and U^{-1} when forming $\tilde{a}_q = L^{-1}a_q$ and $\tilde{e}_p^T = e_p^T U^{-1}$ since much of the fill-in occurs in components for which any corresponding eta occurs earlier in the representation so need not be applied. Table 3 also illustrates the greater efficiency of the FT update with respect to sparsity.

5 Conclusions

Three novel update procedures have been introduced in this paper. Two of them are variants of the product form and one is an extension of the Forrest-Tomlin (FT) update.

Of the two product form variants, the middle product form (MPF) update is generally much more efficient than the alternate product form (APF) update. The APF is very inefficient for some problems and is not recommended as a valuable general technique. However, its limited use in a variant of the dual revised simplex method

	Average density					
	PF	APF	MPF		FT	
Model	$\hat{a}_q$	$\hat{e}_p$	$\tilde{a}_q$	$\tilde{e}_p$	$\tilde{a}_q$	$\tilde{e}_p$
CRE-B	0.313	4.237	0.137	0.022	0.229	0.038
DANO3MIP_LP	47.206	56.684	10.373	0.721	2.524	1.332
DCP2	3.194	0.877	1.223	0.026	0.141	0.062
DFL001	26.926	31.374	4.888	0.213	0.667	0.441
FOME12	6.893	8.503	1.776	0.056	0.197	0.151
GEN4	97.135	31.433	86.840	16.533	26.823	20.709
KEN-18	0.030	0.104	0.010	0.002	0.009	0.002
L30	88.009	75.486	17.018	3.224	3.876	4.506
LINF_520C	67.949	26.318	7.101	0.575	0.645	0.789
LP22	62.458	42.579	19.856	2.679	2.952	3.582
MAROS-R7	79.525	36.413	45.516	10.664	9.487	11.569
MOD2	9.101	7.441	0.802	0.030	0.125	0.088
NS1688926	15.797	0.329	1.127	0.059	0.249	0.140
OSA-60	0.123	0.489	0.112	0.019	0.086	0.036
PDS-20	0.239	0.947	0.103	0.008	0.051	0.013
PDS-40	0.232	0.825	0.085	0.005	0.034	0.008
PDS-100	0.071	0.387	0.036	0.002	0.016	0.003
PILOT87	66.639	47.045	32.635	5.465	8.173	5.949
QAP12	88.641	73.825	29.858	3.930	4.271	4.650
SELF	98.875	85.018	96.353	26.404	49.351	28.758
SGPF5Y6	0.004	0.012	0.003	0.001	0.002	0.001
STAT96V1	67.349	74.287	5.261	0.754	1.218	1.219
STAT96V4	20.358	40.531	4.483	0.912	0.980	0.870
STORMG2-125	0.034	0.021	0.012	0.004	0.007	0.005
STORMG2-1000	0.028	0.003	0.003	0.001	0.001	0.001
STP3D	2.235	4.943	0.134	0.006	0.021	0.015
TRUSS	24.768	77.844	10.812	1.132	3.661	1.966
WATSON_1	0.021	0.016	0.006	0.001	0.004	0.001
WATSON_2	0.007	0.002	0.002	0.000	0.001	0.000
WORLD	9.945	8.938	0.849	0.032	0.140	0.097

Table 3 Average density of eta vectors for the PF, MPF and APF updates

that is amenable to parallelisation yields a valuable computational saving. The performance of the MPF is generally very much better than that of the APF and original product form update. Indeed, its performance frequently approaches that of the FT update. Since the MPF update does not require any elimination operations or dynamic data structures it is very much easier to implement than the FT update so is an attractive update procedure when developing a simple, relatively efficient implementation of the revised simplex method for either research studies or in an industrial context where an independent code-base is necessary.

The collective Forrest-Tomlin (CFT) update organises the calculations required to perform multiple FT updates with the same efficiency as the corresponding sequence of standard FT updates. This ensures that there is no loss of serial efficiency when the CFT is used in the context of the dual revised simplex method with suboptimization.

References

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

2. Bartels, R.H., Golub, G.H.: The simplex method of linear programming using LU decomposition. Commun. ACM **12**(5), 266–268 (1969)
3. Bisschop, J., Meeraus, A.J.: Matrix augmentation and partitioning in the updating of the basis inverse. Mathematical Programming **13**, 241–254 (1977)
4. Carolan, W.J., Hill, J.E., Kennington, J.L., Niemi, S., Wichmann, S.J.: An empirical evaluation of the KORBX algorithms for military airlift applications. Operations Research **38**(2), 240–248 (1990)
5. COIN-OR: Clp. http://www.coin-or.org/projects/Clp.xml. Accessed: 26/06/2013
6. Dantzig, G.B., Orchard-Hays, W.: The product form for the inverse in the simplex method. Math. Comp. **8**, 64–67 (1954)
7. Elble, J.M., Sahinidis, N.V.: A review of the LU update in the simplex algorithm. International Journal of Mathematics in Operational Research **4**(4), 366–399 (2012)
8. Eldersveld, S.K., Saunders, M.A.: A block-LU update for large-scale linear programming. SIAM Journal on Matrix Analysis and Applications **13**, 191–201 (1992)
9. Forrest, J.J.H., Tomlin, J.A.: Updated triangular factors of the basis to maintain sparsity in the product form simplex method. Mathematical Programming **2**, 263–278 (1972)
10. Gay, D.M.: Electronic mail distribution of linear programming test problems. Mathematical Programming Society COAL Newsletter **13**, 10–12 (1985)
11. Gilbert, J.R., Peierls, T.: Sparse partial pivoting in time proportional to arithmetic operations. SIAM J. Sci. Stat. Comput. **9**(5), 862–874 (1988)
12. Gill, P.E., Murray, W., Saunders, M.A., Wright, M.H.: Sparse matrix methods in optimization. SIAM J. Sci. Stat. Comput. **5**, 562–589 (1984)
13. Hall, J.A.J.: Sparse matrix algebra for active set methods in linear programming. Ph.D. thesis, University of Dundee Department of Mathematics and Computer Science (1991)
14. Hall, J.A.J., Huangfu, Q.: A high performance dual revised simplex solver. In: R.W. et al. (ed.) PPAM 2011, Part I, *LNCS*, vol. 7203, pp. 143–151. Springer, Heidelberg (2012)
15. Hall, J.A.J., McKinnon, K.I.M.: Hyper-sparsity in the revised simplex method and how to exploit it. Computational Optimization and Applications **32**(3), 259–283 (2005)
16. Huangfu, Q., Hall, J.A.J.: Parallelizing the dual revised simplex method with suboptimization. Tech. rep., School of Mathematics, University of Edinburgh (2013). In preparation
17. IBM: ILOG CPLEX Optimizer. http://www.ibm.com/software/products/gb/en/ibmilogcpleoptistud/
18. Koberstein, A.: Progress in the dual simplex algorithm for solving large scale LP problems: techniques for a fast and stable implementation. Computational Optimization and Applications **41**(2), 185–204 (2008)
19. Lubin, M., Hall, J.A.J., Petra, C.G., Anitescu, M.: Parallel distributed-memory simplex for large-scale stochastic LP problems. Tech. Rep. ANL/MCS-P2075-0412, Mathematics and Computer Science Division, Argonne National Laboratory (2012). Accepted for publication in Computational Optimization and Applications
20. Reid, J.K.: A sparsity-exploiting variant of the Bartels-Golub decomposition for linear programming bases. Mathematical Programming **24**(1), 55–69 (1982)
21. Rosander, R.R.: Multiple pricing and suboptimization in dual linear programming algorithms. Mathematical Programming Study **4**, 108–117 (1975)
22. Suhl, U.H., Suhl, L.M.: Computing sparse LU factorizations for large-scale linear programming bases. ORSA Journal on Computing **2**(4), 325–335 (1990)
23. Suhl, U.H., Suhl, L.M.: A fast LU update for linear programming. Annals of Operations Research **43**(1), 33–47 (1993)
24. Tomlin, J.A.: Pivoting for size and sparsity in linear programming inversion routines. J. Inst. Maths. Applics **10**, 289–295 (1972)
25. Tomlin, J.A.: On pricing and backward transformation in linear programming. Mathematical Programming **6**, 42–47 (1974)