Journal of Computational Mathematics
Vol.31, No.4, 2013, 335–354.

http://www.global-sci.org/jcm
doi:10.4208/jcm.1301-m4106

ON AN EFFICIENT IMPLEMENTATION OF THE FACE ALGORITHM FOR LINEAR PROGRAMMING*

Lei-Hong Zhang
Department of Applied Mathematics, Shanghai University of Finance and Economics, Shanghai 200433, China
Email: longzlh@gmail.com
Wei Hong Yang
School of Mathematical Sciences, Fudan University, Shanghai 200433, China
Email: whyang@fudan.edu.cn
Li-Zhi Liao
Department of Mathematics and Institute of Computational and Theoretical Studies, Hong Kong Baptist University, Hong Kong, China
Email: liliao@hkbu.edu.hk

Abstract

In this paper, we consider the solution of the standard linear programming (LP). A remarkable result in LP claims that all optimal solutions form an optimal face of the underlying polyhedron. In practice, many real-world problems have infinitely many optimal solutions and pursuing the optimal face, not just an optimal vertex, is quite desirable. The face algorithm proposed by Pan [19] targets at the optimal face by iterating from face to face, along an orthogonal projection of the negative objective gradient onto a relevant null space. The algorithm exhibits a favorable numerical performance by comparing the simplex method. In this paper, we further investigate the face algorithm by proposing an improved implementation. In exact arithmetic computation, the new algorithm generates the same sequence as Pan's face algorithm, but uses less computational costs per iteration, and enjoys favorable properties for sparse problems.

Mathematics subject classification: 62H20, 15A12, 65F10, 65K05.
Key words: Linear programming, Level face, Optimal face, Rank-one correction.

1. Introduction

In this paper, we consider the solution of the linear programming (LP) in the standard form:

$$\begin{array}{ll} \min & \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} & A\mathbf{x} = \mathbf{b}, \quad \mathbf{x} \geq \mathbf{0}, \end{array} \tag{1.1}$$

where $A \in \mathbb{R}^{m\times n}$ $(m < n)$, and $\mathbf{c}, \mathbf{x} \in \mathbb{R}^n$ and $\mathbf{b} \in \mathbb{R}^m$.

There are two basic classes of algorithms for solving LP in the literature. The first milestone is the well-known simplex method founded by Dantzig [2], in which the philosophy is to move on the underlying polyhedron, from vertex to an adjacent vertex, until reaching an optimal vertex. Since then, extensive theoretical analysis, numerical implementations as well as numerous variants (see e.g., [1,3,4,6,7,9,13–18,20–24]) have been developed. To date, the simplex algorithm is accepted as one of the most famous and widely used mathematical tools in the world [13]. The

* Received March 22, 2012 / Revised version received December 29, 2012 / Accepted January 31, 2013 / Published online July 9, 2013 /

other type of methods distinguishes the simplex method by approaching an optimal solution (not necessarily an optimal vertex) from inside of the polyhedron, and they are usually categorized as the interior point methods. Karmarkar's projective algorithm [10] is one of successful and practical interior point methods and stimulates this trend (see e.g., [11–13, 23]). Both the simplex method (and its variants) and the interior point methods have their own advantages and disadvantages; in particular, some interior point methods can reach an optimal solution in polynomial time theoretically, yet neither of them in general shows a dominant performance to the other numerically.

One of the remarkable results for LP (1.1) is that whenever (1.1) has a solution, then at least one vertex of the polyhedron of (1.1) is an optimal solution (see e.g., [13]). This serves as the theoretical fundamental for Dantzig's simplex method. Another well-known result of (1.1) claims that all the optimal solutions form an optimal *face* (see Definition 2.1). In practice, many real-world problems usually have infinitely many optimal solutions and pursuing an optimal face, not just a vertex (a vertex is a special face), is quite desirable. This motives the face algorithm proposed by Pan [19, Chapter 8][1)] . The basic idea behind the face algorithm is to move from face to face, along an orthogonal projection of the negative objective gradient onto a relevant null space (the search direction), to reach an optimal face. The search direction can be efficiently computed via the Cholesky factorization. Preliminary computational testing is carried out and the results show that the face algorithm has a favorable numerical performance [19, Chapter 8].

In this paper, we further investigate the face algorithm by proposing an improved implementation. The basic idea of our new implementation is to *update* the search direction in a similar manner as the *eta-matrix* technique proposed by Dantzig and Orchard-Hayes [3] for the simplex method. The key for our implementation is that the kth search direction of the face algorithm is the solution of a relevant linear system whose coefficient matrix only has a rank-one correction upon that of $(k-1)$th. This fact, with the aid of the Sherman-Morrison formula (see Lemma 4.1), then makes it possible to reuse the previous information and then update the search direction in a similar fashion to the *eta-matrix* technique of Dantzig and Orchard-Hayes to improve the efficiency of the face algorithm. In exact arithmetic computation, we will see that our improved face algorithm generates the same sequence as the face algorithm of Pan [19, Chapter 8], but the new implementation uses less computational costs per iteration and enjoys favorable properties for sparse problems. The detailed computational gains and performance tradeoffs will be discussed in Section 5.

The remaining paper is organized in the following way. In Section 2, some basic concepts and results related to the face algorithm are provided. Section 3 then outlines the original face algorithm proposed in [19, Chapter 8]. Our new implementation, together with some new and further properties of the face algorithm, is presented in Section 4. The computational gains as well as performance tradeoffs of our new algorithm are then discussed in Section 5. Finally, we report computational results in Section 6 and draw a conclusion in Section 7.

Throughout the paper, all vectors are column vectors and are typeset in bold. For a given vector $\mathbf{x}$, we use x_i to denote the i-th component of $\mathbf{x}$. For a matrix $A \in \mathbb{R}^{n\times m}$, $A^\top$ denotes its transpose, and $\mathrm{Null}(A)$ stands for the null space of A.

[1)] The face algorithm is also presented and available at: P.-Q. Pan, *A face algorithm for linear programming,* preprint. http://www.optimization-online.org/DB_HTML/2007/10/1806.html.

2. Some Basic Concepts Related to the Face Algorithm

To implement the face algorithm in a more efficient way, we first take a preparatory step to transform the standard program (1.1) to an equivalent one. Assume a feasible point $\mathbf{x}^{(0)}$ of (1.1) is available, then by introducing a new nonnegative variable x_{n+1}, we solve

$$\begin{array}{ll}\min & -x_{n+1}\\ \text{s.t.} & \begin{pmatrix} A & 0\\ \mathbf{c}^\top & 1\end{pmatrix}\begin{pmatrix}\mathbf{x}\\ x_{n+1}\end{pmatrix}=\begin{pmatrix}\mathbf{b}\\ \mathbf{c}^\top\mathbf{x}^{(0)}\end{pmatrix},\qquad [\mathbf{x}^\top, x_{n+1}]^\top\geq\mathbf{0}.\end{array}$$

Now by redefining

$$A:=\begin{pmatrix} A & 0\\ \mathbf{c}^\top & 1\end{pmatrix},\quad \mathbf{b}:=\begin{pmatrix}\mathbf{b}\\ \mathbf{c}^\top\mathbf{x}^{(0)}\end{pmatrix},\quad \mathbf{x}:=\begin{pmatrix}\mathbf{x}\\ x_{n+1}\end{pmatrix},$$

$$n:=n+1,\ m:=m+1,\ \mathbf{c}:=-\mathbf{e}_n,$$

the standard problem (1.1) is equivalent to the following model:

$$\begin{array}{ll}\min & -x_n\\ \text{s.t.} & A\mathbf{x}=\mathbf{b},\quad \mathbf{x}\geq\mathbf{0}.\end{array}\tag{2.1}$$

It is clear then that $A\mathbf{e}_n=\mathbf{e}_m$ and $[(\mathbf{x}^{(0)})^\top,0]^\top$ is a feasible solution of (2.1), where $\mathbf{e}_i$ stands for the unit vector with i-th component 1. Moreover, the assumption $\operatorname{rank}(A)=m$ is not substantial as the case $\operatorname{rank}(A)<m$ could be simply tackled in the face algorithm (see [19, Chapter 8]), and therefore, in our following discussions, we assume $\operatorname{rank}(A)=m$.

In the traditional simplex method (see e.g., [2, 4, 13, 21]), each iteration is related to a partition $[B,N]$ of A, where $B\in\mathbb{R}^{m\times m}$ is a basis matrix. The basic solution associated with B is obtained by setting $\mathbf{x}_N$ equal to zero: $\mathbf{x}_N=\mathbf{0}$ and $\mathbf{x}_B=B^{-1}b$. If $\mathbf{x}_B\geq\mathbf{0}$, then we say that $\mathbf{x}$ is a basic feasible solution. For each j, we define the reduced cost $\bar{z}_j$ related to the variable x_j by the formula

$$\bar{z}_j=c_j-\mathbf{c}_B^TB^{-1}\mathbf{a}_j,\tag{2.2}$$

and the following result is well known:

Lemma 2.1. *Consider a basic feasible solution* $\mathbf{x}$ *associated with a basis matrix* B*, and let* $\bar{\mathbf{z}}$ *be the corresponding vector of reduced costs. If* $\bar{\mathbf{z}}\geq\mathbf{0}$*, then* $\mathbf{x}$ *is optimal.*

For the face algorithm [19, Chapter 8], on the other hand, each iteration is also linked with a partition $[B,N]$ of A, but now $B\in\mathbb{R}^{m\times k}$ $(m\leq k\leq n)$ and $\operatorname{rank}(B)=m$. The number k of columns of B varies from iteration to iteration. To simplify our presentation, in what follows, we will use $\mathcal{B}$ and $\mathcal{N}$ to denote the associated columns index sets of the matrices B and N, respectively. Related to such partition $[B,N]$ of A, we have several definitions (see [19, Chapter 8]).

Definition 2.1. *(i). Let* $[B,N]$ *be a partition of* A*, where* $B\in\mathbb{R}^{m\times k}$ *and* $m\leq k\leq n$*. The nonempty convex subset*

$$D_k=\{\mathbf{x}\mid B\mathbf{x}_B=\mathbf{b},\ \mathbf{x}_B\geq 0,\quad \mathbf{x}_N=\mathbf{0}\}\tag{2.3}$$

is called a $(k-m)$*-dimensional* face*. The matrices* B *and* N *are called the* face and nonface matrices*, respectively. A point* $\mathbf{x}\in D_k$ *is called a* face point*.*

(ii). A face is a level face if the objective value is constant over it.

(iii). A level face is an optimal face [2] *if objective value is equal to the optimal value over it.*

Different from the traditional simplex method which only pursues an optimal vertex (i.e., a zero-dimensional face), the face algorithm of Pan [19, Chapter 8] targets at an optimal face. This is achieved by the so-called *face subproblem* (2.4) associated with a face D_k in each iteration:

$$\begin{array}{ll} \min & -x_n \\ \text{s.t.} & B\mathbf{x}_B = \mathbf{b}, \quad \mathbf{x}_B \geq \mathbf{0}, \end{array} \qquad \text{where } B \in \mathbb{R}^{m\times k},\ n \in \mathcal{B} \text{ and } B\mathbf{e}_k = \mathbf{e}_m. \tag{2.4}$$

The face algorithm does not attempt to solve the face subproblem (2.4) exactly as it is still a linear programming; what we are interested is the search direction came out of (2.4). As one can easily see that a good search direction for (2.4) is the projection of the negative gradient onto the null space Null(B) of B given by

$$-\Delta_B := P_{N(B)}\mathbf{e}_k \in \mathbb{R}^k, \tag{2.5}$$

where $P_{N(B)} = I - B^\top(BB^\top)^{-1}B$ is the orthogonal projection onto Null(B). One can think Δ_B as a sub-vector of an n-dimensional direction $\Delta \in \mathbb{R}^n$ indexed by the index set $\mathcal{B}$, and thus the other sub-vector of Δ indexed by $\mathcal{N}$ is $\Delta_N = \mathbf{0} \in \mathbb{R}^{n-k}$. Interestingly, it turns out that if D_k is a level face, it can be identified by Δ_B as the following lemma (Lemma 5, [19, Chapter 8]) shows.

Lemma 2.2. *If $\Delta_B = \mathbf{0}$, then D_k is a level face of* (2.1)*; vice versa, if D_k is a level face and if $\mathbf{x}_B > \mathbf{0}$ for some $[\mathbf{x}_B^\top, \mathbf{0}^\top]^\top \in D_k$, then $\Delta_B = \mathbf{0}$.*

The basic idea behind the face algorithm is to iterate from one face to another, guided by the projected negative gradient $-\Delta_B$ of the associated face subproblem, until an optimal face is reached. In the next section, we begin with the search direction $-\Delta_B$ and discuss the original implementation of the face algorithm of Pan [19, Chapter 8].

3. The Original Implementation of the Face Algorithm

Suppose $[\mathbf{x}_B^\top, \mathbf{0}^\top]^\top \in D_k$ is a current iterate, where D_k is a face and the related face subproblem is given by (2.4). To make the paper self-contained, in this section, we will first describe a complete step between successive iterates of the face algorithm proposed in [19, Chapter 8].

3.1. Search direction

Recall that the search direction suggested by (2.4) is $-\Delta_B = P_{N(B)}\mathbf{e}_k \in \text{Null}(B)$, which is a descent direction for (2.4), and satisfies the following properties (see [19, Chapter 8]):

Proposition 3.1 ([19]) *(i). The following are equivalent: $\Delta_B \neq \mathbf{0} \Leftrightarrow \Delta_B^\top \mathbf{e}_k < 0 \Leftrightarrow \mathbf{e}_k \notin$ Range(B^T);*

(ii). If $\Delta_B \neq \mathbf{0}$, then

$$-\mathbf{e}_k^\top \Delta_B / \|\Delta_B\|_2 \geq -\mathbf{e}_k^\top \mathbf{v}/\|\mathbf{v}\|_2, \quad \forall\ \mathbf{0} \neq \mathbf{v} \in \text{Null}(B).$$

[2] It is a well-known result that all optimal solutions of (1.1) or (2.1) form a face.

Proposition 3.1 basically says that $-\Delta_B$ is the steepest downhill for (2.4), and whenever $\Delta_B \neq \mathbf{0}$, it always leads to strictly decrease in $-x_n$.

To compute Δ_B, from (2.5), we observe that

$$\begin{aligned}\Delta_B &= -P_{N(B)}\mathbf{e}_k = -\mathbf{e}_k + B^\top (BB^\top)^{-1} B\mathbf{e}_k \\ &= -\mathbf{e}_k + B^\top (BB^\top)^{-1}\mathbf{e}_m := -\mathbf{e}_k - B^\top \mathbf{y}, \end{aligned} \tag{3.1}$$

where

$$\mathbf{y} := -(BB^\top)^{-1}\mathbf{e}_m. \tag{3.2}$$

Therefore, if we have the QR decomposition

$$B^\top = Q \begin{pmatrix} L^\top \\ 0 \end{pmatrix}, \tag{3.3}$$

where $L \in \mathbb{R}^{m\times m}$ and $L^\top$ is upper triangular, then the Cholesky factorization of $BB^\top$ turns out to be

$$BB^\top = LL^\top \quad (L \text{ is called the } \textit{Cholesky factor} \text{ of } BB^\top), \tag{3.4}$$

and therefore, $\mathbf{y} = -(L^{-\top}L^{-1}\mathbf{e}_m) = -L^{-\top}\mathbf{e}_m/\nu$, where ν is the m-th diagonal of L; that is, $\mathbf{y}$ can be simply obtained via solving a *single* triangular system: $L^\top \mathbf{y} = -\mathbf{e}_m/\nu$.

3.2. Optimality test

Once the search direction $-\Delta_B$ at the current iterate is on hand, we can first check the optimal condition to either identify the lower unboundedness of problem (2.4) or lead the algorithm to different terminating criteria. The following theorem (see [19, Chapter 8]) summarizes two cases for which the iteration can be terminated, either achieving optimal solutions or detecting the lower unboundedness of problem (2.4).

Theorem 3.1. ([19]) *(i) If $\Delta_B \leq \mathbf{0}$, problem* (2.1) *and hence* (1.1) *is unbounded below; (ii) If $\Delta_B = \mathbf{0}$ and $\mathbf{z}_N := -N^\top \mathbf{y} \geq \mathbf{0}$, where $\mathbf{y}$ is defined by* (3.2)*, then $[\mathbf{x}_B^\top, \mathbf{0}^\top]^\top$ and $[\mathbf{0}^\top, \mathbf{z}_N^\top]^\top$ with $\mathbf{y}$, are a pair of primal and dual optimal solutions to* (2.1)*; moreover, D_k is an optimal face of problem* (2.1)*.*

Theorem 3.1, with the assumption that problem (2.1) is bounded below, suggests that the algorithm would encounter two possible cases: (i) $\Delta_B \not\leq \mathbf{0}$, and (ii) $\Delta_B = \mathbf{0}$ but $\mathbf{z}_N = -N^\top \mathbf{y} \not\geq \mathbf{0}$. Treatment of case (i) and case (ii) leads to two different procedures: contracting face D_k and expanding face D_k, respectively.

Now we show the relationship between $\mathbf{z}_N$ defined in Theorem 3.1 and the reduced cost vector $\bar{\mathbf{z}}$, defined by (2.2), in the standard simplex method.

Lemma 3.1. *Let $[B, N]$ be one partition of A, where $B \in \mathbb{R}^{m\times k}$ and $m \leq k \leq n$, and let $[\bar{B}, \bar{N}]$ be another partition of A, where $\bar{B}$ is a basis matrix such that $n \in \bar{\mathcal{B}} \subseteq \mathcal{B}$. If $\Delta_B = \mathbf{0}$, then for each $j \in \mathcal{N}$, it is true that $z_j = \bar{z}_j$, where $\bar{z}_j$ is the reduced cost related to the variable x_j.*

Proof. Since $\mathbf{c} = -\mathbf{e}_n$ and $n \in \bar{\mathcal{B}}$, we have $\mathbf{c}_{\bar{B}} = -\mathbf{e}_m$ and $\mathbf{c}_{\bar{N}} = \mathbf{0}$. Fix $j \in \mathcal{N}$. It is obvious that $j \in \bar{\mathcal{N}}$. Let $\mathbf{w} = \bar{B}^{-\top}\mathbf{e}_m$. From (2.2), it follows that

$$\bar{z}_j = -\mathbf{e}_m^\top \bar{B}^{-1}\mathbf{a}_j = -(\bar{B}^{-\top}\mathbf{e}_m)^\top \mathbf{a}_j = -\mathbf{w}^\top \mathbf{a}_j. \tag{3.5}$$

Since $\Delta_B = \mathbf{0}$, by (3.1), there exists a vector $\mathbf{y} \in \mathbb{R}^m$ such that $B^\top \mathbf{y} = -\mathbf{e}_k$ and therefore $z_j = -\mathbf{a}_j^\top \mathbf{y}$. Because $n \in \bar{\mathcal{B}} \subseteq \mathcal{B}$ and $B^\top \mathbf{y} = -\mathbf{e}_k$, we must have $\bar{B}^\top \mathbf{y} = -\mathbf{e}_m$. Thus we have $\mathbf{w} = \mathbf{y}$, which together with (3.5) and $z_j = -\mathbf{a}_j^\top \mathbf{y}$ implies $z_j = \bar{z}_j$. □

3.3. Contracting face D_k

We first deal with the case $\Delta_B \not\leq \mathbf{0}$. In this case, we have known from Proposition 3.1 that $-\Delta_B$ is a downhill for (2.4), and we can implement a line search along $-\Delta_B$, until one of the nonnegative constraints of $\mathbf{x}_B$ is violated. The largest step-length then is determined by

$$\alpha = x_s/\Delta_s = \min\{x_j/\Delta_j \mid \Delta_j > 0,\ j \in \mathcal{B}\}. \tag{3.6}$$

Because $\mathbf{e}_k^\top \Delta_B < 0$ (see Proposition 3.1), we know $s \neq n$. Consequently, $\mathbf{x}_B$ is updated by $\mathbf{x}_B - \alpha\Delta_B$. It is said [19, Chapter 8] that α could vanish if $\mathbf{x}_B$ is *degenerate* where some of its components are zero. Interestingly, it is observed numerically that significantly less degeneracy occurs than that in the traditional simplex method [19, Chapter 8]. Under the nondegenerate assumption, it is clear that the next iterate is an improvement of $[\mathbf{x}_B^\top, \mathbf{0}^\top]^\top$. Moreover, because x_s becomes zero, the corresponding face and nonface indexes should be updated accordingly:

$$\widetilde{\mathcal{B}} = \mathcal{B}\backslash\{s\} \quad \text{and} \quad \widetilde{\mathcal{N}} = \mathcal{N} \cup \{s\}, \tag{3.7}$$

and hence the dimension of the new face is reduced by one. Let $\widetilde{B} \in \mathbb{R}^{m\times(k-1)}$ be the matrix resulting from B by removing the corresponding column $\mathbf{a}_s$ and let $\mathbf{x}_{\widetilde{B}}$ be the new iterate. It is shown [19, Chapter 8] that $\mathrm{rank}(\widetilde{B}) = m$ if $\mathrm{rank}(B) = m$. Corresponding to the new iterate $\mathbf{x}_{\widetilde{B}}$, the next face subproblem is

$$\begin{array}{ll} \min & -x_n \\ \text{s.t.} & \widetilde{B}\mathbf{x}_{\widetilde{B}} = \mathbf{b}, \quad \mathbf{x}_{\widetilde{B}} \geq \mathbf{0}, \end{array} \qquad \text{where } \widetilde{B} \in \mathbb{R}^{m\times(k-1)},\ n \in \widetilde{\mathcal{B}} \text{ and } \widetilde{B}\mathbf{e}_{k-1} = \mathbf{e}_m,$$

and the next search direction will be

$$\Delta_{\widetilde{B}} = -P_{N(\widetilde{B})}\mathbf{e}_{k-1} = -\mathbf{e}_{k-1} + \widetilde{B}^\top(\widetilde{B}\widetilde{B}^\top)^{-1}\mathbf{e}_m := -\mathbf{e}_{k-1} - \widetilde{B}^\top\widetilde{\mathbf{y}}, \tag{3.8}$$

where $\widetilde{\mathbf{y}}$ is obtained from $(\widetilde{B}\widetilde{B}^\top)\widetilde{\mathbf{y}} = -\mathbf{e}_m$. [19, Chapter 8] provides the following downdating procedure to update the Cholesky factor $\widetilde{L}$ of $\widetilde{B}\widetilde{B}^\top$ (i.e., $\widetilde{B}\widetilde{B}^\top = \widetilde{L}\widetilde{L}^\top$) based on the information of the previous Cholesky factorization $BB^\top = LL^\top$.

Algorithm 3.1 (Downdating) *:*

1. *Solve $m \times m$ lower triangular system $L\mathbf{p} = \mathbf{a}_s$ ($\|\mathbf{p}\|_2 < 1$ holds);*
2. *$\beta := \sqrt{1 - \|\mathbf{p}\|_2^2}$;*
3. *Determine Givens rotations $J_1, \cdots, J_m$ with J_i in the $(m+1, i)$-plane such that*

$$J_1 \cdots J_m \begin{pmatrix} \mathbf{p} \\ \beta \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ 1 \end{pmatrix};$$

4. *Calculate $J_1 \cdots J_m \begin{pmatrix} L^\top \\ \mathbf{0}^\top \end{pmatrix} \equiv \begin{pmatrix} \widetilde{L}^\top \\ \mathbf{a}_s^\top \end{pmatrix}$ to obtain $\widetilde{L}$.*

This procedure is from Saunders [20]. In Section 5, we will take a close look at its computational cost and its performance for large and sparse problems.

3.4. Expanding face D_k

Now we discuss the other case: $\Delta_B = \mathbf{0}$ but $\mathbf{z}_N = -N^\top \mathbf{y} \ngeq \mathbf{0}$. This would happen if D_k is a level face but not the optimal (see Lemma 2.2 and Theorem 3.1). In this case, the face D_k is expanded by bringing an index t from $\mathcal{N}$ to $\mathcal{B}$, where the index t is determined from

$$t \in \arg\min\{z_j \mid j \in \mathcal{N}\}. \tag{3.9}$$

Similarly, the new face and nonface indexes are $\widetilde{\mathcal{B}} = \mathcal{B} \cup \{t\}$ and $\widetilde{\mathcal{N}} = \mathcal{N} \backslash \{t\}$, respectively. Let $\widetilde{B} \in \mathbb{R}^{m \times (k+1)}$ be the corresponding new face matrix, and the Cholesky factor $\widetilde{L}$ of $\widetilde{B}\widetilde{B}^\top$ now is updated according to the following step [19, Chapter 8]:

Algorithm 3.2 (Updating) *: Find Givens rotations $G_m, \cdots, G_1$ where G_i is in the $(i, m+1)$-plane such that*

$$G_m \cdots G_1 \begin{pmatrix} L^\top \\ \mathbf{a}_t^\top \end{pmatrix} = \begin{pmatrix} \widetilde{L}^\top \\ \mathbf{0}^\top \end{pmatrix},$$

where $\widetilde{L}$ is lower triangular and is the Cholesky factor of $\widetilde{B}\widetilde{B}^\top$.

This updating procedure simply follows [5] and we will also provide some discussion on its performance in Section 5.

Consequently, starting from the current iterate $[\mathbf{x}_B^\top, \mathbf{0}^\top]^\top \in D_k$, we can summarize briefly a basic iteration of the face algorithm [19, Chapter 8] as follows:

(1). compute the search direction Δ_B defined by (2.5) according to Section 3.1;

(2). test optimality according to Section 3.2; and

(3). contract (according to Section 3.3) or expand (according to Section 3.4) the current face D_k and update $[\mathbf{x}_B^\top, \mathbf{0}^\top]^\top$.

For a complete pseudo-code of the face algorithm as well as the **Phase-1** method for an initial feasible solution of (2.1), we refer to the detailed discussions in [19, Chapter 8].

4. The Improved Implementation

By investigating one basic step of the face algorithm presented in Section 3, one can easily find that the dominant computation lies in solving the search direction Δ_B (2.5). The proposed implementation in [19, Chapter 8] (i.e., downdating and updating procedures in Section 3) is one of efficient ways which updates the Cholesky factor of $BB^\top$ by using the previous information. However, there are still rooms to improve this implementation. In this section, we will propose an alternative to obtain the search direction with less computational costs (detailed comparison of the computational complexity will be discussed in Section 5). This alternative realization of obtaining the search direction is based on the observation that either the contracting face procedure or the expanding face procedure only changes the face matrix B by removing or adding *one* column respectively. This fact, with the aid of the Sherman-Morrison formula, then makes it possible to update $(BB^\top)^{-1}$ directly. Consequently, our new implementation

turns out to be in a similar fashion to the *eta-matrix* technique first described by Dantzig and Orchard-Hayes [3] for the simplex method.

Lemma 4.1 (Sherman-Morrison formula) *Let $\widetilde{M} = M + \mathbf{u}\mathbf{v}^\top$, where $M \in \mathbb{R}^{m\times m}$ is nonsingular and $\mathbf{u}, \mathbf{v} \in \mathbb{R}^m$. If $\widetilde{M}$ is nonsingular, then*

$$\widetilde{M}^{-1} = M^{-1} - \frac{M^{-1}\mathbf{u}\mathbf{v}^\top M^{-1}}{1 + \mathbf{v}^\top M^{-1}\mathbf{u}}. \tag{4.1}$$

4.1. Downdating in the contracting face procedure

Using the same notation in Section 3, we assume that $[\mathbf{x}_B^\top, \mathbf{0}^\top]^\top \in D_k$ is a current iterate (or the initial iterate) and the Cholesky factorization (3.4) is available. Suppose $\Delta_B \not\leq \mathbf{0}$, then we should contract the current face D_k and thus, according to (3.6) and (3.7), a column $\mathbf{a}_s$ will be removed from B. To simplify our presentation, we can do a permutation to B, $\mathcal{B}$, $\mathbf{x}_B$ and Δ_B accordingly, so that $\mathbf{a}_s$ appears as the first column of B; that is,

$$B = [\mathbf{a}_s, \widetilde{B}], \tag{4.2}$$

and hence

$$\widetilde{B}\widetilde{B}^\top = BB^\top - \mathbf{a}_s\mathbf{a}_s^\top. \tag{4.3}$$

Now by $\operatorname{rank}(\widetilde{B}) = m$ (see [19, Chapter 8]) and Sherman-Morrison formula (4.1), it follows that

$$\begin{aligned}(\widetilde{B}\widetilde{B}^\top)^{-1} &= (BB^\top)^{-1} + \frac{(BB^\top)^{-1}\mathbf{a}_s\mathbf{a}_s^\top(BB^\top)^{-1}}{1 - \mathbf{a}_s^\top(BB^\top)^{-1}\mathbf{a}_s} \\ &:= (BB^\top)^{-1} + \frac{\mathbf{h}^{(1)}(\mathbf{h}^{(1)})^\top}{1+\eta^{(1)}},\end{aligned} \tag{4.4}$$

where

$$\mathbf{h}^{(1)} := (BB^\top)^{-1}\mathbf{a}_s \quad \text{and } \eta^{(1)} := -\mathbf{a}_s^\top\mathbf{h}^{(1)} = -\mathbf{a}_s^\top(BB^\top)^{-1}\mathbf{a}_s < 0.$$

Since the Cholesky factorization of $BB^\top$ is available, the calculation of $\mathbf{h}^{(1)}$ is simplified as solving two triangular systems. Formula (4.4) is very important because the next search direction $-\Delta_{\widetilde{B}}$ (3.8) can be expressed as

$$\Delta_{\widetilde{B}} = -\mathbf{e}_{k-1} + \widetilde{B}^\top(\widetilde{B}\widetilde{B}^\top)^{-1}\mathbf{e}_m \tag{4.5}$$

$$= -\mathbf{e}_{k-1} + \widetilde{B}^\top(BB^\top)^{-1}\mathbf{e}_m + \widetilde{B}^\top\mathbf{h}^{(1)}\frac{\mathbf{e}_m^\top\mathbf{h}^{(1)}}{1+\eta^{(1)}}. \tag{4.6}$$

Moreover, from (3.1) and (4.2), it follows that

$$\begin{aligned}\Delta_B = -\mathbf{e}_k + B^\top(BB^\top)^{-1}\mathbf{e}_m &= -\begin{pmatrix} 0 \\ \mathbf{e}_{k-1}\end{pmatrix} + \begin{pmatrix} \mathbf{a}_s^\top(BB^\top)^{-1}\mathbf{e}_m \\ \widetilde{B}^\top(BB^\top)^{-1}\mathbf{e}_m\end{pmatrix} \\ &= \begin{pmatrix} \mathbf{e}_m^\top\mathbf{h}^{(1)} \\ -\mathbf{e}_{k-1} + \widetilde{B}^\top(BB^\top)^{-1}\mathbf{e}_m\end{pmatrix},\end{aligned}$$

which implies that $-\mathbf{e}_{k-1} + \widetilde{B}^\top(BB^\top)^{-1}\mathbf{e}_m$ in (4.6) is the subvector $\Delta_B(2:k)$ of Δ_B by removing its first component $\mathbf{e}_m^\top\mathbf{h}^{(1)} = h_m^{(1)}$. Based on this fact, one has from (4.6) that

$$\Delta_{\widetilde{B}} = \Delta_B(2:k) + \widetilde{B}^\top\mathbf{h}^{(1)}\frac{h_m^{(1)}}{1+\eta^{(1)}}, \quad \text{where} \quad h_m^{(1)} = \mathbf{e}_m^\top\mathbf{h}^{(1)}. \tag{4.7}$$

Consequently, we know that *computing the next search direction* $-\Delta_{\widetilde{B}}$ *only requires a vector* $\mathbf{h}^{(1)}$ *and a scalar* $\eta^{(1)}$. As pointed out earlier, $\mathbf{h}^{(1)}$ can be achieved by solving two triangular systems of order m and the flops are $2m^2$, while $\eta^{(1)} = -\mathbf{a}_s^\top \mathbf{h}^{(1)}$ only requires $2m$ flops.

One may argue that for the new face matrix $\widetilde{B}$, we do not have the Cholesky factorization for $\widetilde{B}\widetilde{B}^\top$ any more, and hence, the next search direction could not be obtained similarly from our previous economic computation. In fact, we point out that the inverse formula (4.4) of $\widetilde{B}\widetilde{B}^\top$ carries the Cholesky factorization of $BB^\top$ and hence the computation for the next search direction could still be very economic. Indeed, we suppose that in the next iteration, another column say $\mathbf{a}_p$ is removed from $\widetilde{B}$, and as before, we can do a permutation to make $\mathbf{a}_p$ the first column of $\widetilde{B}$, i.e., $\widetilde{B} = [\mathbf{a}_p, \widehat{B}]$. Following the same argument as (4.4), it is true that

$$\begin{aligned}(\widehat{B}\widehat{B}^\top)^{-1} &= (\widetilde{B}\widetilde{B}^\top)^{-1} + \frac{(\widetilde{B}\widetilde{B}^\top)^{-1}\mathbf{a}_p\mathbf{a}_p^\top(\widetilde{B}\widetilde{B}^\top)^{-1}}{1-\mathbf{a}_p^\top(\widetilde{B}\widetilde{B}^\top)^{-1}\mathbf{a}_p} \\ &= (\widetilde{B}\widetilde{B}^\top)^{-1} + \frac{\mathbf{h}^{(2)}(\mathbf{h}^{(2)})^\top}{1+\eta^{(2)}} \\ &= (BB^\top)^{-1} + \frac{\mathbf{h}^{(1)}(\mathbf{h}^{(1)})^\top}{1+\eta^{(1)}} + \frac{\mathbf{h}^{(2)}(\mathbf{h}^{(2)})^\top}{1+\eta^{(2)}},\end{aligned} \tag{4.8}$$

where

$$\mathbf{h}^{(2)} := (\widetilde{B}\widetilde{B}^\top)^{-1}\mathbf{a}_p \quad \text{and } \eta^{(2)} := -\mathbf{a}_p^\top \mathbf{h}^{(2)} = -\mathbf{a}_p^\top(\widetilde{B}\widetilde{B}^\top)^{-1}\mathbf{a}_p < 0. \tag{4.9}$$

On the other hand, with the help of (4.8), the new search direction $-\Delta_{\widehat{B}}$ can be computed as

$$\begin{aligned}\Delta_{\widehat{B}} &= -P_{N(\widehat{B})}\mathbf{e}_{k-2} = -\mathbf{e}_{k-2} + \widehat{B}^\top(\widehat{B}\widehat{B}^\top)^{-1}\mathbf{e}_m \\ &= -\mathbf{e}_{k-2} + \widehat{B}^\top(\widetilde{B}\widetilde{B}^\top)^{-1}\mathbf{e}_m + \widehat{B}^\top\mathbf{h}^{(2)}\frac{\mathbf{e}_m^\top\mathbf{h}^{(2)}}{1+\eta^{(2)}}.\end{aligned} \tag{4.10}$$

Analogously, we note from (4.5) and $\widetilde{B} = [\mathbf{a}_p, \widehat{B}]$ that

$$\begin{aligned}\Delta_{\widetilde{B}} &= -\mathbf{e}_{k-1} + \widetilde{B}^\top(\widetilde{B}\widetilde{B}^\top)^{-1}\mathbf{e}_m = -\begin{pmatrix} 0 \\ \mathbf{e}_{k-2}\end{pmatrix} + \begin{pmatrix} \mathbf{a}_p^\top(\widetilde{B}\widetilde{B}^\top)^{-1}\mathbf{e}_m \\ \widehat{B}^\top(\widetilde{B}\widetilde{B}^\top)^{-1}\mathbf{e}_m\end{pmatrix} \\ &= \begin{pmatrix} \mathbf{e}_m^\top\mathbf{h}^{(2)} \\ -\mathbf{e}_{k-2} + \widehat{B}^\top(\widetilde{B}\widetilde{B}^\top)^{-1}\mathbf{e}_m\end{pmatrix},\end{aligned}$$

and therefore from (4.10)

$$\Delta_{\widehat{B}} = \Delta_{\widetilde{B}}(2:k-1) + \widehat{B}^\top\mathbf{h}^{(2)}\frac{h_m^{(2)}}{1+\eta^{(2)}}, \quad \text{where} \quad h_m^{(2)} = \mathbf{e}_m^\top\mathbf{h}^{(2)}. \tag{4.11}$$

The formulation (4.11) implies that the search direction $-\Delta_{\widehat{B}}$ can be obtained if $\mathbf{h}^{(2)}$ is available, which indeed could be computed very simply and economically because by (4.9) and (4.4), one has

$$\mathbf{h}^{(2)} = (\widetilde{B}\widetilde{B}^\top)^{-1}\mathbf{a}_p = (BB^\top)^{-1}\mathbf{a}_p + \mathbf{h}^{(1)}\frac{(\mathbf{h}^{(1)})^\top\mathbf{a}_p}{1+\eta^{(1)}}.$$

Therefore, if we store the Cholesky factor L of $BB^\top$, the vector $\mathbf{h}^{(1)}$ and the scalar $\eta^{(1)}$, the search direction $-\Delta_{\widehat{B}}$ is achievable by only solving two triangular systems (for $(\widetilde{B}\widetilde{B}^\top)^{-1}\mathbf{a}_p$), plus a matrix-vector product $\widehat{B}^\top\mathbf{h}^{(2)}$ and an inner product $(\mathbf{h}^{(1)})^\top\mathbf{a}_p$.

Our previous procedure can proceed recursively, and if l contracting face steps have been implemented, the current face matrix, namely $\check{B}$, enjoys the following relationship:

$$(\check{B}\check{B}^\top)^{-1} = (BB^\top)^{-1} + \frac{\mathbf{h}^{(1)}(\mathbf{h}^{(1)})^\top}{1+\eta^{(1)}} + \frac{\mathbf{h}^{(2)}(\mathbf{h}^{(2)})^\top}{1+\eta^{(2)}} + \cdots + \frac{\mathbf{h}^{(l)}(\mathbf{h}^{(l)})^\top}{1+\eta^{(l)}}. \tag{4.12}$$

Therefore, we conclude that

(1). to represent $(\check{B}\check{B}^\top)^{-1}$, we only need to store a series of vectors $[\mathbf{h}^{(1)}, \cdots, \mathbf{h}^{(l)}]$ and the corresponding negative scalars $[\eta^{(1)}, \cdots, \eta^{(l)}]$;

(2). computing $(\check{B}\check{B}^\top)^{-1}\mathbf{a}_q$ for some column $\mathbf{a}_q \in \mathbb{R}^m$ only needs to solve two triangular systems of order m, plus l inner products; and

(3). the search direction $-\Delta_{\check{B}}$ can be updated from the previous search direction (similarly to (4.7)) with additionally solving two triangular systems of order m.

4.2. Updating in the expanding face procedure

Now suppose from the iterate $[\mathbf{x}_B^\top, \mathbf{0}^\top]^\top \in D_k$, l contracting face steps have been taken (l could be 0), and the current face matrix $\check{B}$ satisfies (4.12). In this subsection, we will discuss the corresponding updating step to expand the face $\check{B}$. As we have pointed out in Section 3.4 that this could only happen if $\Delta_{\check{B}} = \mathbf{0}$. According to (3.9), a new column $\mathbf{a}_t$ will be added to $\check{B}$ and without loss of generality, we let it be the first column of the new face matrix $\bar{B}$; that is,

$$\bar{B} = [\mathbf{a}_t, \check{B}]. \tag{4.13}$$

Note from $\bar{B}\bar{B}^\top = \check{B}\check{B}^\top + \mathbf{a}_t\mathbf{a}_t^\top$, the Sherman-Morrison formula and (4.12), we have

$$\begin{aligned}
(\bar{B}\bar{B}^\top)^{-1} &= (\check{B}\check{B}^\top)^{-1} - \frac{(\check{B}\check{B}^\top)^{-1}\mathbf{a}_t\mathbf{a}_t^\top(\check{B}\check{B}^\top)^{-1}}{1+\mathbf{a}_t^\top(\check{B}\check{B}^\top)^{-1}\mathbf{a}_t} \\
&= (\check{B}\check{B}^\top)^{-1} - \frac{\mathbf{h}^{(l+1)}(\mathbf{h}^{(l+1)})^\top}{1+\eta^{(l+1)}} \qquad (4.14) \\
&= (BB^\top)^{-1} + \frac{\mathbf{h}^{(1)}(\mathbf{h}^{(1)})^\top}{1+\eta^{(1)}} + \cdots + \frac{\mathbf{h}^{(l)}(\mathbf{h}^{(l)})^\top}{1+\eta^{(l)}} - \frac{\mathbf{h}^{(l+1)}(\mathbf{h}^{(l+1)})^\top}{1+\eta^{(l+1)}}, \qquad (4.15)
\end{aligned}$$

where

$$\mathbf{h}^{(l+1)} := (\check{B}\check{B}^\top)^{-1}\mathbf{a}_t \quad \text{and } \eta^{(l+1)} := \mathbf{a}_t^\top\mathbf{h}^{(l+1)} = \mathbf{a}_t^\top(\check{B}\check{B}^\top)^{-1}\mathbf{a}_t > 0.$$

Here, we should point out that the scalar $\eta^{(l+1)}$ is defined differently from $\eta^{(1)}, \cdots, \eta^{(l)}$, because we want to distinguish the expanding face procedure from the contracting face procedure. Moreover, we have mentioned in Section 4.1 that computing $\mathbf{h}^{(l+1)}$ only requires to solve two triangular systems, plus l inner products, yielding $2m^2 + 2ml$ flops. Therefore, by (4.13) and (4.14), the new search direction $-\Delta_{\bar{B}}$ could be obtained from

$$\begin{aligned}
\Delta_{\bar{B}} &= -P_{N(\bar{B})}\mathbf{e}_{k-l+1} = -\mathbf{e}_{k-l+1} + \bar{B}^\top(\bar{B}\bar{B}^\top)^{-1}\mathbf{e}_m \\
&= -\mathbf{e}_{k-l+1} + \bar{B}^\top(\check{B}\check{B}^\top)^{-1}\mathbf{e}_m - \bar{B}^\top\mathbf{h}^{(l+1)}\frac{\mathbf{e}_m^\top\mathbf{h}^{(l+1)}}{1+\eta^{(l+1)}} \\
&= \begin{pmatrix} 0 \\ -\mathbf{e}_{k-l} \end{pmatrix} + \begin{pmatrix} \mathbf{a}_t^\top(\check{B}\check{B}^\top)^{-1}\mathbf{e}_m \\ \check{B}^\top(\check{B}\check{B}^\top)^{-1}\mathbf{e}_m \end{pmatrix} - \bar{B}^\top\mathbf{h}^{(l+1)}\frac{\mathbf{e}_m^\top\mathbf{h}^{(l+1)}}{1+\eta^{(l+1)}} \\
&= \begin{pmatrix} (\mathbf{h}^{(l+1)})^\top\mathbf{e}_m \\ -\mathbf{e}_{k-l} + \check{B}^\top(\check{B}\check{B}^\top)^{-1}\mathbf{e}_m \end{pmatrix} - \bar{B}^\top\mathbf{h}^{(l+1)}\frac{\mathbf{e}_m^\top\mathbf{h}^{(l+1)}}{1+\eta^{(l+1)}}. \qquad (4.16)
\end{aligned}$$

On the other hand, since

$$\Delta_{\check{B}} = -P_{N(\check{B})}\mathbf{e}_{k-l} = -\mathbf{e}_{k-l} + \check{B}^\top(\check{B}\check{B}^\top)^{-1}\mathbf{e}_m = \mathbf{0},$$

we know from (4.16) that

$$\Delta_{\bar{B}} = \begin{pmatrix} h_m^{(l+1)} \\ \mathbf{0} \end{pmatrix} - \bar{B}^\top \mathbf{h}^{(l+1)} \frac{h_m^{(l+1)}}{1+\eta^{(l+1)}}, \tag{4.17}$$

where $h_m^{(l+1)} = \mathbf{e}_m^\top \mathbf{h}^{(l+1)}$ is the last component of $\mathbf{h}^{(l+1)}$. This completes the updating.

To sum up, in order to implement an updating procedure, one only needs to compute and store one new vector $\mathbf{h}^{(l+1)}$ and one new positive scalar $\eta^{(l+1)}$. These terms are necessary for the search direction $\Delta_{\bar{B}}$ and $(\bar{B}\bar{B}^\top)^{-1}$ which is always updated by a rank-one matrix. Finally, we point out that $(\bar{B}\bar{B}^\top)^{-1}$ in (4.15) can also be rewritten as

$$\begin{aligned}(\bar{B}\bar{B}^\top)^{-1} &= (BB^\top)^{-1} + \mathrm{sgn}(-\eta^{(1)})\frac{\mathbf{h}^{(1)}(\mathbf{h}^{(1)})^\top}{1+\eta^{(1)}} + \cdots + \mathrm{sgn}(-\eta^{(l+1)})\frac{\mathbf{h}^{(l+1)}(\mathbf{h}^{(l+1)})^\top}{1+\eta^{(l+1)}} \\ &= (BB^\top)^{-1} + \sum_{i=1}^{l+1} \mathrm{sgn}(-\eta^{(i)})\frac{\mathbf{h}^{(i)}(\mathbf{h}^{(i)})^\top}{1+\eta^{(i)}},\end{aligned} \tag{4.18}$$

where $\mathrm{sgn}(-\eta^{(i)})$ denotes the sign of $-\eta^{(i)}$. The practical computation will benefit from this uniform expression, because we only store the vectors $[\mathbf{h}^{(1)}, \cdots, \mathbf{h}^{(l+1)}]$ and the corresponding scalars $[\eta^{(1)}, \cdots, \eta^{(l+1)}]$, and the formulation (4.18) will automatically distinguish the contracting and expanding procedures. Moreover, this formulation also facilitates us to express the vector $\overline{\mathbf{y}}$ (see (3.2)) defined as

$$\begin{aligned}\overline{\mathbf{y}} &= -(\bar{B}\bar{B}^\top)^{-1}\mathbf{e}_m = -(BB^\top)^{-1}\mathbf{e}_m - \sum_{i=1}^{l+1} \mathrm{sgn}(-\eta^{(i)})\frac{h_m^{(i)}}{1+\eta^{(i)}}\mathbf{h}^{(i)} \\ &= [-(BB^\top)^{-1}\mathbf{e}_m - \sum_{i=1}^{l} \mathrm{sgn}(-\eta^{(i)})\frac{h_m^{(i)}}{1+\eta^{(i)}}\mathbf{h}^{(i)}] - \mathrm{sgn}(-\eta^{(l+1)})\frac{h_m^{(l+1)}}{1+\eta^{(l+1)}}\mathbf{h}^{(l+1)},\end{aligned} \tag{4.19}$$

associated with the current face matrix $\bar{B}$, which, according to Theorem 3.1, is needed to determine the dual information $\mathbf{z}_N = -N^\top\overline{\mathbf{y}}$ for optimality test. The expression (4.19) suggests a recursive updating for the vector $\mathbf{y}$.

4.3. The new face algorithm for linear programming

Based on previous discussion on the contracting and expanding face procedures, we are now able to summarize the overall steps of our new face algorithm for (2.1) in Algorithm 4.1.

As our new face algorithm (Algorithm 4.1) is only an alternative realization to the face algorithm proposed in [19, Chapter 8], hence the following convergence property holds:

Theorem 4.1. *Under the nondegeneracy assumption, Algorithm* 4.1 *terminates at either*

1. *Step* 4*, detecting lower unboundedness of program* (2.1)*, or*
2. *Step* 13*, reaching an optimal face together with a pair of primal and dual optimal solutions.*

4.4. Further properties of the face algorithm

In this subsection, we shall take another close look at the expanding procedure, and in particular, we will show that, if in one iteration, the number of elements in the face index set is increased by one, then in the next iteration the number of elements in the face index set will not be increased again.

Theorem 4.2. *Assume that $\check{B}$ is updated by $\bar{B} = [\mathbf{a}_t, \check{B}]$ in the previous iteration. Then in the current iteration, we have $\Delta_{\bar{B}} \neq \mathbf{0}$.*

Proof. Since $\check{\mathcal{B}}$ is updated by $\bar{\mathcal{B}} = \check{\mathcal{B}} \cup \{t\}$ in the previous iteration, according to the face algorithm, we must have $\Delta_{\check{B}} = \mathbf{0}$. From (3.1), it follows that $\check{B}^\top \mathbf{y} = -\mathbf{e}_k$, where $\mathbf{y} := -(\check{B}\check{B}^\top)^{-1}\mathbf{e}_m$. By the algorithm again, we have $z_t = -\mathbf{a}_t^\top \mathbf{y} < 0$. Let $\mathbf{h} := (\check{B}\check{B}^\top)^{-1}\mathbf{a}_t$ and $\eta := \mathbf{a}_t^\top \mathbf{h} = \mathbf{a}_t^\top (\check{B}\check{B}^\top)^{-1}\mathbf{a}_t$. Then $\eta > 0$. By (4.17), we have

$$\Delta_{\bar{B}} = \begin{pmatrix} h_m \\ \mathbf{0} \end{pmatrix} - \bar{B}^\top \mathbf{h} \frac{h_m}{1+\eta}, \tag{4.20}$$

where $h_m = \mathbf{e}_m^\top \mathbf{h}$. By the definition of $\mathbf{h}$, we have

$$h_m = \mathbf{e}_m^\top \mathbf{h} = ((\check{B}\check{B}^\top)^{-1}\mathbf{e}_m)^\top \mathbf{a}_t = -\mathbf{y}^\top \mathbf{a}_t = z_t < 0.$$

Note that

$$\bar{B}^\top \mathbf{h} = \begin{pmatrix} \mathbf{a}_t^\top \\ \check{B}^\top \end{pmatrix} \mathbf{h} = \begin{pmatrix} \mathbf{a}_t^\top \mathbf{h} \\ \check{B}^\top \mathbf{h} \end{pmatrix} = \begin{pmatrix} \eta \\ \check{B}^\top \mathbf{h} \end{pmatrix}. \tag{4.21}$$

By (4.20) and (4.21), we have

$$\Delta_{\bar{B}} = \begin{pmatrix} h_m \\ \mathbf{0} \end{pmatrix} - \begin{pmatrix} \eta h_m/(1+\eta) \\ \mathbf{q} \end{pmatrix} = \begin{pmatrix} h_m/(1+\eta) \\ -\mathbf{q} \end{pmatrix}, \tag{4.22}$$

where $\mathbf{q} = h_m \check{B}^\top \mathbf{h}/(1+\eta)$. Since $\eta > 0$ and $h_m < 0$, we have $\Delta_{\bar{B}} \neq \mathbf{0}$. □

Theorem 4.3. *Assume that, in the current iteration, the face matrix $\check{B}$ is a basis matrix. Then the face algorithm and the standard simplex algorithm generate the same iterates onwards.*

Proof. Since $\check{B}$ is a basis matrix, by (2.3), the corresponding face $\check{D}_k$ is a point, and the current iterate $[\mathbf{x}_{\check{B}}^\top, \mathbf{0}^\top]^\top \in \check{D}_k$ is a basic feasible solution in the standard simplex method. We also have $\Delta_{\check{B}} = \mathbf{0}$. Let $\mathbf{z}_{\check{N}}$ be defined as in Lemma 3.1. One has that $\mathbf{z}_{\check{N}} = \bar{\mathbf{z}}_{\check{N}}$, where $\bar{\mathbf{z}}$ is the reduced cost vector associated with $\check{B}$ in the standard simplex algorithm. Let $t \in \check{\mathcal{N}}$ be such that $z_t = \min\{z_j \mid j \in \check{\mathcal{N}}\}$ and let $\bar{B} = [\mathbf{a}_t, \check{B}]$. Since $\check{B}$ is nonsingular, by (4.22), we have

$$\Delta_{\bar{B}} = \frac{-z_t}{1 + \mathbf{a}_t^\top (\check{B}\check{B}^\top)^{-1}\mathbf{a}_t} \begin{pmatrix} -1 \\ \check{B}^{-1}\mathbf{a}_t \end{pmatrix}. \tag{4.23}$$

Let s be such that $x_s/\Delta_s = \min\{x_j/\Delta_j \mid \Delta_j > 0,\ j \in \bar{\mathcal{B}}\}$. Let B be the next face matrix and $\mathcal{B}$ be the corresponding face index set. Then $\mathcal{B} = \bar{\mathcal{B}} \backslash \{s\}$. By [19, Chapter 8], $\operatorname{rank}(B) = m$ and therefore the corresponding face D_k is a point and the next iterate in the face algorithm is $[\mathbf{x}_B^\top, \mathbf{x}_N^\top]^\top = [(B^{-1}\mathbf{b})^\top, \mathbf{0}^\top]^\top$. Let $\check{\mathcal{B}}_j$ denote the j-th element in the index set $\check{\mathcal{B}}$. Since $z_t < 0$

and $\Delta_s > 0$, by (4.23), we know that $s \in \check{\mathcal{B}}$ and so $s = \check{\mathcal{B}}_i$ for some $i \in \{1, \cdots, m\}$. Thus, we have

$$x_{\check{\mathcal{B}}_i}/(\check{B}^{-1}\mathbf{a}_t)_i = \min\Big\{x_{\check{\mathcal{B}}_j}/(\check{B}^{-1}\mathbf{a}_t)_j \mid (\check{B}^{-1}\mathbf{a}_t)_j > 0,\ j = 1, \cdots, m\Big\}. \tag{4.24}$$

It is clear that (4.24) is exactly the rule for selecting the leaving variable in the standard simplex, which implies that B is the next basis in the standard simplex algorithm. The proof is complete. □

Algorithm 4.1. (NFALP)*:*

*Let $k = n, l = 1, B = A$ and $\mathcal{N} = \varnothing$ (l denotes the total number of iterations). Given a feasible solution (by the **Phase-1** procedure in* [19, Chapter 8]*) $\mathbf{x}^{(0)} = \mathbf{x}_B$ of* (2.1) *and the Cholesky factorization $BB^\top = L^\top L$, this algorithm solves program (2.1).*

1. *Solve $L^\top \mathbf{y} = -\mathbf{e}_m/\nu$ for $\mathbf{y}$, where ν is the m-th diagonal of L; see* (3.2).
2. *Compute $\Delta_B = -\mathbf{e}_k - B^\top \mathbf{y}$; see* (3.1).
3. *Goto Step* 13 *if $\Delta_B = \mathbf{0}$.*
4. *Stop if $\Delta_B \leq \mathbf{0}$ (unbounded problem).*

Contracting face procedure

5. *Determine steplength $\alpha = x_s/\Delta_s = \min\{x_j/\Delta_j \mid \Delta_j > 0,\ j \in \mathcal{B}\}$; see* (3.6).
6. *Permutate B, $\mathcal{B}$, $\mathbf{x}_B$ and Δ_B accordingly so that $\mathbf{a}_s$ is the first column of B.*
7. *Update $\mathbf{x}_B = \mathbf{x}_B - \alpha\Delta_B$ (line search).*
8. *Update $\mathcal{B} = \mathcal{B}\backslash\{s\}, \mathcal{N} = \mathcal{N} \cup \{s\}$ and the corresponding matrices B and N; see* (3.7).
9. *Solve $L^\top \mathbf{u} = \mathbf{a}_s$ and $L\mathbf{v} = \mathbf{u}$ for $\mathbf{v}$, and compute (see Section* 4.1)

$$\mathbf{h}^{(l)} = \mathbf{v} + \sum_{i=1}^{l-1} \operatorname{sgn}(-\eta^{(i)}) \frac{\mathbf{a}_s^\top \mathbf{h}^{(i)}}{1 + \eta^{(i)}} \mathbf{h}^{(i)} \quad \text{and} \quad \eta^{(l)} = -\mathbf{a}_s^\top \mathbf{h}^{(l)} < 0.$$

10. *Update $\Delta_B = \Delta_B(2:k) + B^\top \mathbf{h}^{(l)} \frac{h_m^{(l)}}{1+\eta^{(l)}}$; see* (4.7).
11. *Update $\mathbf{y} = \mathbf{y} - \operatorname{sgn}(-\eta^{(l)}) \frac{h_m^{(l)}}{1+\eta^{(l)}} \mathbf{h}^{(l)}$; see* (4.19).
12. *Update $k = k - 1, l = l + 1$ and goto Step* 3.

Expanding face procedure

13. *Stop if $\mathbf{z}_N = -N^\top \mathbf{y} \geq \mathbf{0}$ (optimality achieved, see Theorem* 3.1*), and return*

$$[\mathbf{x}_B^\top, \mathbf{0}^\top]^\top \quad \text{and} \quad [\mathbf{0}^\top, \mathbf{z}_N^\top]^\top \text{with } \mathbf{y},$$

as a pair of primal and dual optimal solutions for (2.1).

14. *Determine $t \in \arg\min\{z_j \mid j \in \mathcal{N}\}$; see* (3.9).

15. *Update* $\mathcal{B} = \mathcal{B} \cup \{t\}, \mathcal{N} = \mathcal{N} \backslash \{t\}$, *and update accordingly the face matrix* $B = [\mathbf{a}_t, B]$ *and the nonface matrix* N; *see* (3.7).

16. *Solve* $L^\top \mathbf{u} = \mathbf{a}_t$ *and* $L\mathbf{v} = \mathbf{u}$ *for* $\mathbf{v}$, *and compute* (*see Section* 4.2)

$$\mathbf{h}^{(l)} = \mathbf{v} + \sum_{i=1}^{l-1} \operatorname{sgn}(-\eta^{(i)}) \frac{\mathbf{a}_t^\top \mathbf{h}^{(i)}}{1+\eta^{(i)}} \mathbf{h}^{(i)} \quad \text{and} \quad \eta^{(l)} = \mathbf{a}_t^\top \mathbf{h}^{(l)} > 0.$$

17. *Update* (*see Section* 4.2)

$$\Delta_B = \begin{pmatrix} h_m^{(l)} \\ \mathbf{0} \end{pmatrix} - B^\top \mathbf{h}^{(l)} \frac{h_m^{(l)}}{1+\eta^{(l)}}.$$

18. *Update* $\mathbf{y} = \mathbf{y} - \operatorname{sgn}(-\eta^{(l)}) \frac{h_m^{(l)}}{1+\eta^{(l)}} \mathbf{h}^{(l)}$; *see* (4.19).

19. *Update* $k = k+1, l = l+1$ *and goto Step* 3.

5. Computational Gains and Performance Tradeoffs

As we have presented two different implementations to realize the face algorithm in the previous sections, we are able to make a comparison between their computational performances. The most time-consuming step in the face algorithm is the computation of the search direction Δ_B, which involves updating the inverse of $BB^\top$. The method proposed in [19, Chapter 8] gets round this bottleneck by updating the Cholesky factor, while our strategy is to update the inverse in every iteration via a rank-one correction. By taking a close investigation on the computational complexity of the contracting face D_k procedure (see **Downdating** process in Section 3.3), we find that m^2 flops are required in Step 1 for solving a triangular system and $3m^2$ (dominant) flops are required in Step 4 to obtain the updated Cholesky factor $\widetilde{L}$; additionally, to calculate the new search direction $\Delta_{\widetilde{B}}$ by (3.8), another triangular system of order m needs to be solved, together with a matrix-vector product (flops $2(k-1)m$); consequently, a contracting face procedure requires totally

$$flops: 5m^2 + 2(k-1)m.$$

By contrast, it is not difficult to count the dominant flops used in contracting procedure in Algorithm 4.1: only two triangular systems and l inner products of order m in Step 9, and a matrix-vector product (flops $2(k-1)m$) in Step 10 are necessary, yielding totally

$$flops: \ 2m^2 + 2ml + 2(k-1)m.$$

Therefore, it is clear that the new implementation saves order of m^2 flops in the contracting face procedure. Similarly, a careful counting on the computational costs shows that the expanding process in Section 3.4 needs $4m^2 + 2(k-1)m$ (dominant) flops, while our new implementation in Algorithm 4.1 requires $2m^2 + 2ml + 2(k-1)m$ (dominant) flops.

Recall that in order to reconstruct $(BB^\top)^{-1}$ for the current face matrix B, all we need to save is a series of vectors $[\mathbf{h}^{(1)}, \cdots, \mathbf{h}^{(l)}]$ and the corresponding scalars $[\eta^{(1)}, \cdots, \eta^{(l)}]$. In actual implementations, these can be stored in lists. In history, the *eta-matrix* technique by Dantzig and Orchard-Hayes [3] for the simplex method also needs to store a similar series of vectors in lists, which is called the *eta-file* ([23, Chapter 8]). As l gets large, storage is not a serious problem since computer memory is relatively cheap nowadays; what we really concern is that the amount of work required to go through the entire eta-file begins to dominate the amount of work. This problem can be well settled by periodically implementing Cholesky factorization of $BB^\top$ (and accompanied purging of the eta-file). In other words, whenever l gets large, we can simply purge the eta-file but compute and store the Cholesky factor of $BB^\top$ for the current face matrix, while leaving other steps in Algorithm 4.1 unchanged. This strategy is of some advantages as now we are free to use the Cholesky factor related to *any* face matrix during the iteration. In fact, from (3.3), we can also employ row and column permutations to the face matrix B to make the related Cholesky factor L as sparse as possible. In contrast, because the downdating and updating procedures described in Section 3 use the Givens rotatons to update the Cholesky factor, it will introduce nonzero elements and make the sequential Cholesky factors much denser.

The next question then is: how often should we recompute the Cholesky factor? As motivated by the strategy for the eta-matrix technique in simplex method (see e.g., [23, Chapter 8]), we can choose the period of refactorization so that the average number of arithmetic operations per iteration is minimized. This strategy tells us that for dense problems, refactorization should roughly take place every m iterations or so (the derivation follows the same as that for the eta-matrix technique in [23, Chapter 8]). For sparse problems, however, one may recompute the Cholesky factor less than every m iterations (for instance every $\sqrt{m}$ iterations or so). In practical computation, we can make this period as a user-settable parameter and recommend, for example 100, as the default value (see e.g., [23]).

Refactorization is also a necessary procedure for the numerical stability of the face algorithm (Algorithm 4.1). As l gets large, rounding-off errors accumulate, which may deteriorate the subsequent computations. For some problems, especially for those badly modeled, significant errors can occur and be amplified. Monitoring the numerical status and stabilizing the computations is very important for a stable and robust algorithm. For Algorithm 4.1, the condition number $\mathrm{cond}(BB^\top)$ for the current face matrix plays a crucial role. Though it is not easy to check $\mathrm{cond}(BB^\top)$ in every iteration, the value $\eta^{(l)}$ defined in the expanding or the contracting procedure reveals $\mathrm{cond}(BB^\top)$ to some extent. Moreover, $\eta^{(l)}$ can also serve as a monitor for the accumulation of the errors. From this point of view, we think it is reasonable to trigger the refactorization procedure of $BB^\top = LL^\top$ whenever $|\eta^{(l)}|$ is too small or too large. Particularly, in our numerical testing presented in the next section, we choose to recompute the Cholesky factor if $|\eta^{(l)}| > 10^6$ or $|\eta^{(l)}| < 10^{-3}$ or $l > [\frac{m}{3}]$.

6. Computational Results

In this section, we will carry out numerical testing and investigate the practical behavior of our new implementation of the face algorithm. In exact arithmetic computation, it is clear that the new realization presented in Algorithm 4.1 produces the same sequence as that in [19, Chapter 8], which has demonstrated a superior performance compared to the standard simplex method. Therefore, our main purpose in this section is to conduct a comparison between the

Table 6.1: The performance of `FALP` on Group 1.

Problem size	Total		Phase-1	
(m,n)	Avg. Iter. #	Avg. CPU(s)	Avg. Iter. #	Avg. CPU(s)
$(50,100)$	84.2	0.0596	2.4	0.0087
$(100,150)$	72.0	0.1140	4.6	0.0140
$(100,200)$	208.3	0.4005	3.6	0.0296
$(150,250)$	172.4	0.4722	5.5	0.0601
$(200,300)$	159.0	0.8134	8.2	0.1164

Table 6.2: The performance of `NFALP` on Group 1.

Problem size	Total		Phase-1	
(m,n)	Avg. Iter. #	Avg. CPU(s)	Avg. Iter. #	Avg. CPU(s)
$(50,100)$	84.2	0.0568	2.4	0.0030
$(100,150)$	72.0	0.0516	4.6	0.0101
$(100,200)$	208.3	0.2291	3.6	0.0193
$(150,250)$	172.4	0.2817	5.5	0.0423
$(200,300)$	159.0	0.4266	8.2	0.0886

Table 6.3: The performance of `SIMPLEX` on Group 1.

Problem size (m,n)	$(50,100)$	$(100,150)$	$(100,200)$	$(150,250)$	$(200,300)$
Avg. CPU(s)	0.4011	3.1891	6.8457	27.5783	77.5893

two implementations of the face algorithm. For this purpose, in particular, we code the two implementations:

- `FALP`: the original face algorithm proposed in [19, Chapter 8], and
- `NFALP`: our new algorithm (Algorithm 4.1)

on the MATLAB 7.1 (R14) platform on a PC with Pentium(R) Dual-Core CPU E5300 @2.60GHz. To give a more clear picture of their performances, we also provide the consuming CPU(s) times used in the simplex method (labelled as `SIMPLEX`[3] below) and the interior-point method (labelled as `INTERIOR`[4] below) incorporated in MATLAB 7.1 (R14). For both `FALP` and `NFALP`, the initial feasible point $\mathbf{x}^{(0)}$ was obtained based on the Phase-1 procedure described in [19, Chapter 8], in which components of the starting point are all ones. Harris' two-pass practical tactic [8] was used for pivot selection. The value 10^{-6} was set as primal and dual feasibility tolerance while $\|\Delta_B\|_\infty < 5 \times 10^{-6}$ was used in place of $\Delta_B = \mathbf{0}$. For `FALP`, furthermore, we also recompute the Cholesky factor related with the face matrix whenever $1 - \|\mathbf{p}\|_2 < 10^{-8}$, where $\mathbf{p}$ is the solution of $L\mathbf{p} = \mathbf{a}_s$ given in the **Downdating** procedure in Section 3.3.

The four algorithms were tested on two groups of linear programming problems. The first group contains many randomly generated LP problems of given sizes: m and n. In particular, for each fixed pair (m,n), we first randomly generated the coefficient matrix $A \in \mathbb{R}^{m\times n}$, a nonnegative vector $\mathbf{x} \in \mathbb{R}^n$ and the cost vector $\mathbf{c} \in \mathbb{R}^n$ with elements normally distributed, and then we set $\mathbf{b} = A\mathbf{x}$ as the right hand side. It is easy to see that the feasible set $\{\mathbf{x}|A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq$

[3] To call the simplex method in MATLAB, we can simply set the options: `options=optimset('LargeScale', 'off', 'Simplex', 'on')` and then use it in the function: `linprog`.

[4] To call the interior point method in MATLAB, we can simply set the following option: `options=optimset('LargeScale', 'on')` and then use it in the function: `linprog`.

Table 6.4: The performance of INTERIOR on Group 1.

Problem size (m, n)	$(50, 100)$	$(100, 150)$	$(100, 200)$	$(150, 250)$	$(200, 300)$
Avg. CPU(s)	0.0560	0.1726	0.2349	0.5227	1.0131

Table 6.5: Ratios of the CPU(s) on Group 1.

Problem size	Total			Phase-1
(m, n)	FALP/NFALP	SIMPLEX/NFALP	INTERIOR/NFALP	FALP/NFALP
$(50, 100)$	1.0493	7.0616	0.9859	2.9000
$(100, 150)$	2.2093	61.8043	3.3450	1.3861
$(100, 200)$	1.7481	29.8808	1.0253	1.5337
$(150, 250)$	1.6763	97.8995	1.8555	1.4208
$(200, 300)$	1.9067	181.8783	2.3748	1.3138
Average	1.7179	75.7049	1.9173	1.7109

$\mathbf{0}\}$ is nonempty, and therefore the corresponding LP is either solvable or unbounded below. For every given (m, n), we generate 1000 such LP problems, and then report the average performance of each algorithm over the solvable problems. In Tables 6.1, 6.2, 6.3 and 6.4, we report the average numbers of iterations (labelled as 'Avg. Iter. #') as well as their average CPU(s) times (labelled as 'Avg. CPU(s)') of FALP, NFALP, SIMPLEX and INTERIOR, respectively. In addition, in Table 6.5, we report the ratios of their average CPU(s) between different algorithms. From these results, one can easily find that the numbers of iterations for FALP and NFALP are all the same, indicating that they both generate the same iterative sequence.

The second group includes 16 standard LP problems from NETLIB[5)] that do not have BOUNDS and RANGES sections in their MPS files. Test statistics from these algorithms are reported in Tables 6.6, 6.7 and 6.8. From these tables, we observed that the numbers of iterations for the FALP and NFALP are all the same in the Phase-1, and for the Phase-2, they

[5)] http://www.netlib.org/lp/data/

Table 6.6: The performance of FALP on Group 2.

	Total		Phase-1		Optimal value
Problem	Iter. #	CPU(s)	Iter. #	CPU(s)	
AFIRO	25	0.0781	13	0.0313	-4.6475314286E+02
SC50B	33	0.0625	3	0.0156	-7.0000000000E+01
SC50A	39	0.0625	33	0.0314	-6.4575077059E+01
ADLITTLE	137	0.1406	28	0.0313	2.2549496322E+05
BLEND	105	0.1562	25	0.0468	-3.0812149846E+01
SHARE2B	111	0.2188	60	0.1094	-4.1573224074E+02
SC105	107	0.2188	16	0.0313	-5.2202061212E+01
STOCFOR1	75	0.1719	40	0.0937	-4.1131976220E+04
SCAGR7	169	0.4063	35	0.1094	-2.3313898243E+06
ISRAEL	579	2.2344	113	0.4219	-8.9664482186E+05
SHARE1B	305	0.6094	145	0.2969	-7.6589318579E+04
SC205	271	1.4531	38	0.2969	-5.2202061214E+01
BEACONFD	159	0.7813	80	0.3281	3.3592485852E+04
LOTFI	360	1.3125	85	0.3594	-2.5264706096E+02
BRANDY	363	1.6406	196	0.8281	1.5185098965E+03
E226	1065	6.8906	178	1.3750	-1.8751928983E+01

Table 6.7: The performance of NFALP on Group 2.

	Total		Phase-1		Optimal value
Problem	Iter. #	CPU(s)	Iter. #	CPU(s)	
AFIRO	25	0.0469	13	0.0156	-4.6475314286E+02
SC50B	33	0.0469	3	0.0156	-7.0000000000E+01
SC50A	39	0.0469	33	0.0156	-6.4575077059E+01
ADLITTLE	146	0.0625	28	0.0156	2.2549496287E+05
BLEND	105	0.0937	25	0.0203	-3.0812149846E+01
SHARE2B	111	0.1093	60	0.0625	-4.1573224074E+02
SC105	107	0.1093	16	0.0313	-5.2202061212E+01
STOCFOR1	75	0.1406	40	0.0625	-4.1131976220E+04
SCAGR7	169	0.2812	35	0.0625	-2.3313898246E+06
ISRAEL	562	2.2344	113	0.3125	-8.9664475363E+05
SHARE1B	305	0.4218	145	0.2187	-7.6589318579E+04
SC205	279	0.7813	38	0.1406	-5.2202061212E+01
BEACONFD	159	0.4531	80	0.1875	3.3592485826E+04
LOTFI	360	1.2656	85	0.1718	-2.5264706062E+02
BRANDY	363	1.2968	199	0.6875	1.5185098965E+03
E226	1092	7.7968	178	0.8125	-1.8751929066E+01

Table 6.8: The performances of SIMPLEX and INTERIOR on Group 2.

	SIMPLEX		INTERIOR	
Problem	CPU(s)	Optimal value	CPU(s)	Optimal value
AFIRO	0.0313	-4.6475314286E+02	0.0313	-4.6475314265E+02
SC50B	0.0313	-7.0000000000E+01	0.0156	-7.0000000000E+01
SC50A	0.0625	-6.4575077059E+01	0.0156	-6.4575076981E+01
ADLITTLE	0.3750	2.2549496316E+05	0.0781	2.2549496316E+05
BLEND	0.5937	-3.0812149846E+01	0.0469	-3.0812149846E+01
SHARE2B	0.3750	-4.1573224074E+02	0.0156	-4.1573224023E+02
SC105	0.07812	-5.2202061211E+01	0.0468	-5.2202061211E+01
STOCFOR1	0.4531	-4.1131976219E+04	0.0313	-4.1131976219E+04
SCAGR7	0.18750	-2.3313898243E+06	0.0468	-2.3313898243E+06
ISRAEL	6.7031	-8.9664482186E+05	1.0468	-8.9664482160E+05
SHARE1B	2.9687	-7.6589318579E+04	1.7500	-7.6588658198E+04
SC205	2.3281	-5.2202061212E+01	0.6250	-5.2202061212E+01
BEACONFD	1.9375	3.3592485807E+04	0.5937	3.3592485807E+04
LOTFI	7.5781	-2.5264706062E+01	1.0000	-2.5264706062E+01
BRANDY	fail	fail	0.8281	1.5185098965E+03
E226	fail	fail	2.0781	-1.8751929066E+02

are all the same except for slight differences in the examples ADLITTLE, ISRAEL, SC205 and E226, which is mainly due to round-off errors. In Table 6.9, we list ratios of CPU(s) between four algorithms. It is observed that the SIMPLEX failed in solving BRANDY and E226, and therefore, the average ratio SIMPLEX/NFALP in the bottom line in Table 6.9 excludes those from BRANDY and E226.

Table 6.9: Ratios of the CPU(s) on Group 2.

	Total			Phase-1
Problem	FALP/NFALP	SIMPLEX/NFALP	INTERIOR/NFALP	FALP/NFALP
AFIRO	1.6652	0.6674	0.6674	2.0064
SC50B	1.3326	0.6674	0.3326	1.0000
SC50A	1.3326	1.3326	0.3326	2.0128
ADLITTLE	2.2496	6.0000	1.2496	2.0064
BLEND	1.6670	6.3362	0.5005	2.3054
SHARE2B	2.0018	3.4309	0.1427	1.7504
SC105	2.0018	0.7147	0.4282	1.0000
STOCFOR1	1.2226	3.2226	0.2226	1.4992
SCAGR7	1.4449	0.6668	0.1664	1.7504
ISRAEL	1.0000	3.0000	0.4685	1.3501
SHARE1B	1.4448	7.0382	4.1489	1.3576
SC205	1.8598	2.9798	0.7999	2.1117
BEACONFD	1.7243	4.2761	1.3103	1.7499
LOTFI	1.0371	5.9878	0.7901	2.0920
BRANDY	1.2651	-	0.6386	1.2045
E226	0.8838	-	0.2665	1.6923
Average	1.5083	3.3086	0.7791	1.6806

These numerical results overall show that (i) the original face algorithm and our new implementation generate the same sequence numerically, and (ii) the new algorithm improves the efficiency of the face algorithm by saving more than half of CPU times needed in the our new implementation, which coincides with our analysis on computational complex in Section 5.

7. Concluding Remarks

In this paper, we have put some efforts in improving the face algorithm proposed in [19, Chapter 8]. A new and efficient implementation has been developed, which follows the same framework of the original face algorithm but realizes it in a different way. The principal of the new algorithm is to update the inverse of $BB^\top$ for each face matrix B via a rank-one correction. Such updating procedure can be computed economically and enjoys favorable properties for sparse problems. We also took a close look at the behavior of the face algorithm and explored some further properties. Preliminary numerical testings on dense problems show the efficiency of the new algorithm.

Acknowledgments. We would like to thank the Editor and the anonymous referees for their helpful suggestions that have improved the presentation of this paper. We are also very grateful to Professor Ping-Qi Pan for his valuable comments and suggestions on this paper. The work of the first author was supported in part by the National Natural Science Foundation of China NSFC-11101257 and the Basic Academic Discipline Program, the 11th five year plan of 211 Project for Shanghai University of Finance and Economics. The work of the third author was supported in part by GRF grant HKBU201611 from the Research Grant Council of Hong Kong.

References

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

[2] G.B. Dantzig, Programming in a linear structure, Comptroller, *USAF*, Washington, D.C. (February 1948).

[3] G.B. Dantzig and W. Orchard-Hayes, The product form for the inverse in the simplex method, *Mathematical Tables and Other Aids to Computation,* **8** (1954), 64-67.

[4] J.J.H. Forrest and D. Goldfarb, Steepest-edge simplex algorithms for linear programming, *Math. Program.,* **57** (1992), 341-374.

[5] G.H. Golub, Numerical methods for solving linear least squares problems, *Numer. Math.* **7** (1965), 206-216.

[6] W.W. Hager, The LP dual active set algorithm, *in High-Performance Algorithms and Software in Nonlinear Optimization,* Dordrecht, Klower, 1998, 243-254.

[7] W.W. Hager, The dual active set algorithm and its application to linear programming, *Comput. Optim. Appl.,* **21** (2002), 263-275.

[8] P.M.J. Harris, Pivot selection methods of the DEVEX LP code, *Math. Program.,* **5** (1973), 1-28.

[9] J.-F. Hu and Pan P.-Q, An efficient approach to updating simplex multipliers in the simplex algorithm, *Math. Program.,* **114** (2008), 235-248.

[10] N. Karmarkar, A new plolynomial time algorithm for linear programming, *Combinatorica,* **4** (1984), 373-395.

[11] I.J. Lustig, R.E. Marsten and D.F. Shanno, On implementing Mehrotra's predictor-corrector interior-point method for linear programming, *SIAM J. Optim.,* **2** (1992), 435-440.

[12] S. Mehrotra, On the implementation of a primal-dual interior point method, *SIAM J. Optim.,* **2**, 575-601.

[13] J. Nocedal and S.J. Wright, Numerical Optimization, 2nd ed., Springer Verlag, New York, 2006.

[14] P.-Q. Pan, A simplex-like method with bisection for linear programming, *Optimization,* **22** (1991), 717-743.

[15] P.-Q. Pan, A basis-deficiency-allowing variation of the simplex method, *Computers and Mathematics with Applications,* **36** (1998), 33-53.

[16] P.-Q. Pan, A projective simplex method for linear programming, *Linear Algebra and Its Applications,* **292** (1999), 99-125.

[17] P.-Q. Pan, A dual projective pivot algorithm for linear programming, *Comput. Optim. Appl.,* **29** (2004), 333-344.

[18] P.-Q. Pan, A revised dual projective pivot algorithm for linear programming, *SIAM J. on Optimization,* **16** (2005), 49-68.

[19] P.-Q. Pan, Linear Programming Computation (in Chinese), Science Press, Beijing, 2012. NOTE: The face algorithm is also available at: P.-Q. Pan, A face algorithm for linear programming, http://www.optimization-online.org/DB_HTML/2007/10/1806.html.

[20] M.A. Saunders, Large Scale Linear Programming Using the Cholesky Factorization, *Stanford University Report STAN-CS-72-152*, 1972.

[21] A. Schrijver, Theory of Linear and Integer Programming, John Wiley & Sons, Chichester, England, 1986.

[22] T. Terlaky, A covergent criss-cross method, *Optimization,* **16** (1985), 683-690.

[23] R.J. Vanderbei, Linear Programming: Fundations and Extensions, 2nd ed., 2001.

[24] S. Zionts, The criss-cross method for solving linear programming problems, *Management Science,* **15** (1969), 426-445.