Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2012, Article ID 482193, 28 pages
doi:10.1155/2012/482193

Research Article

A Two-Phase Support Method for Solving Linear Programs: Numerical Experiments

Mohand Bentobache[1,2] and Mohand Ouamer Bibi[2]

[1] *Department of Technology, University of Laghouat, 03000, Algeria*
[2] *Laboratory of Modelling and Optimization of Systems (LAMOS), University of Bejaia, 06000, Algeria*

Correspondence should be addressed to Mohand Bentobache, mbentobache@yahoo.com

Received 6 September 2011; Accepted 7 February 2012

Academic Editor: J. Jiang

We develop a single artificial variable technique to initialize the primal support method for solving linear programs with bounded variables. We first recall the full artificial basis technique, then we will present the proposed algorithm. In order to study the performances of the suggested algorithm, an implementation under the MATLAB programming language has been developed. Finally, we carry out an experimental study about CPU time and iterations number on a large set of the NETLIB test problems. These test problems are practical linear programs modelling various real-life problems arising from several fields such as oil refinery, audit staff scheduling, airline scheduling, industrial production and allocation, image restoration, multisector economic planning, and data fitting. It has been shown that our approach is competitive with our implementation of the primal simplex method and the primal simplex algorithm implemented in the known open-source LP solver LP_SOLVE.

1. Introduction

Linear programming is a mathematical discipline which deals with solving the problem of optimizing a linear function over a domain delimited by a set of linear equations or inequations. The first formulation of an economical problem as a linear programming problem is done by Kantorovich (1939, [1]), and the general formulation is given later by Dantzig in his work [2]. LP is considered as the most important technique in operations research. Indeed, it is widely used in practice, and most of optimization techniques are based on LP ones. That is why many researchers have given a great interest on finding efficient methods to solve LP problems. Although some methods exist before 1947 [1], they are restricted to solve some particular forms of the LP problem. Being inspired from the work of Fourier on linear inequalities, Dantzig (1947, [3]) developed the simplex method which is

known to be very efficient for solving practical linear programs. However, in 1972, Klee and Minty [4] have found an example where the simplex method takes an exponential time to solve it.

In 1977, Gabasov and Kirillova [5] have generalized the simplex method and developed the primal support method which can start by any basis and any feasible solution and can move to the optimal solution by interior points or boundary points. The latter is adapted by Radjef and Bibi to solve LPs which contain two types of variables: bounded and nonnegative variables [6]. Later, Gabasov et al. developed the adaptive method to solve, particularly, linear optimal control problems [7]. This method is extended to solve general linear and convex quadratic problems [8–18]. In 1979, Khachian developed the first polynomial algorithm which is an interior point one to solve LP problems [19], but it's not efficient in practice. In 1984, Karmarkar presented for the first time an interior point algorithm competitive with the simplex method on large-scale problems [20].

The efficiency of the simplex method and its generalizations depends enormously on the first initial point used for their initialization. That is why many researchers have given a new interest for developing new initialization techniques. These techniques aim to find a good initial basis and a good initial point and use a minimum number of artificial variables to reduce memory space and CPU time. The first technique used to find an initial basic feasible solution for the simplex method is the full artificial basis technique [3]. In [21, 22], the authors developed a technique using only one artificial variable to initialize the simplex method. In his experimental study, Millham [23] shows that when the initial basis is available in advance, the single artificial variable technique can be competitive with the full artificial basis one. Wolfe [24] has suggested a technique which consists of solving a new linear programming problem with a piecewise linear objective function (minimization of the sum of infeasibilities). In [25–31], crash procedures are developed to find a good initial basis.

In [32], a two-phase support method with one artificial variable for solving linear programming problems was developed. This method consists of two phases and its general principle is the following: in the first phase, we start by searching an initial support with the Gauss-Jordan elimination method, then we proceed to the search of an initial feasible solution by solving an auxiliary problem having one artificial variable and an obvious feasible solution. This obvious feasible solution can be an interior point of the feasible region. After that, in the second phase, we solve the original problem with the primal support method [5].

In [33, 34], we have suggested two approaches to initialize the primal support method with nonnegative variables and bounded variables: the first approach consists of applying the Gauss elimination method with partial pivoting to the system of linear equations corresponding to the main constraints and the second consists of transforming the equality constraints to inequality constraints. After finding the initial support, we search a feasible solution by adding only one artificial variable to the original problem, thus we get an auxiliary problem with an evident support feasible solution. An experimental study has been carried out on some NETLIB test problems. The results of the numerical comparison revealed that finding the initial support by the Gauss elimination method consumes much time, and transforming the equality constraints to inequality ones increases the dimension of the problem. Hence, the proposed approaches are competitive with the full artificial basis simplex method for solving small problems, but they are not efficient to solve large problems.

In this work, we will first extend the full artificial basis technique presented in [7], to solve problems in general form, then we will combine a crash procedure with a single artificial variable technique in order to find an initial support feasible solution for the initialization of the support method. This technique is efficient for solving practical problems. Indeed, it

takes advantage of sparsity and adds a reduced number of artificial variables to the original problem. Finally, we show the efficiency of our approach by carrying out an experimental study on some NETLIB test problems.

The paper is organized as follows: in Section 2, the primal support method for solving linear programming problems with bounded variables is reviewed. In Section 3, the different techniques to initialize the support method are presented: the full artificial basis technique and the single artificial variable one. Although the support method with full artificial basis is described in [7], it has never been tested on NETLIB test problems. In Section 4, experimental results are presented. Finally, Section 5 is devoted to the conclusion.

2. Primal Support Method with Bounded Variables

2.1. State of the Problem and Definitions

The linear programming problem with bounded variables is presented in the following standard form:

$$\max \quad z = c^T x, \tag{2.1}$$

$$\text{s.t.} \quad Ax = b, \tag{2.2}$$

$$l \le x \le u, \tag{2.3}$$

where c and x are n-vectors; b an m-vector; A an $(m \times n)$-matrix with $\operatorname{rank}(A) = m < n$; l and u are n-vectors. In the following sections, we will assume that $\|l\| < \infty$ and $\|u\| < \infty$. We define the following index sets:

$$\begin{gathered} I = \{1, 2, \ldots, m\}, \quad J = \{1, 2, \ldots, n\}, \quad J = J_N \cup J_B, \\ J_N \cap J_B = \emptyset, \quad |J_B| = m, \quad |J_N| = n - m. \end{gathered} \tag{2.4}$$

So we can write and partition the vectors and the matrix A as follows:

$$x = x(J) = (x_j, j \in J), \quad x = \begin{pmatrix} x_N \\ x_B \end{pmatrix}, \quad x_N = x(J_N) = (x_j, j \in J_N),$$

$$x_B = x(J_B) = (x_j, j \in J_B); \quad c = c(J) = (c_j, j \in J), \quad c = \begin{pmatrix} c_N \\ c_B \end{pmatrix},$$

$$c_N = c(J_N) = (c_j, j \in J_N), \quad c_B = c(J_B) = (c_j, j \in J_B),$$

$$l = l(J) = (l_j, j \in J), \quad u = u(J) = (u_j, j \in J),$$

$$A = A(I, J) = (a_{ij}, i \in I, j \in J) = (a_1, \ldots, a_j, \ldots, a_n) = \begin{pmatrix} A_1^T \\ \vdots \\ A_i^T \\ \vdots \\ A_m^T \end{pmatrix},$$

$$a_j = \begin{pmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{pmatrix}, \quad j = \overline{1,n}; \quad A_i^T = (a_{i1}, a_{i2}, \ldots, a_{in}), \quad i = \overline{1,m},$$

$$A = (A_N \mid A_B), \quad A_N = A(I, J_N),\ A_B = A(I, J_B). \tag{2.5}$$

(i) A vector x verifying constraints (2.2) and (2.3) is called a *feasible solution* for the problem (2.1)–(2.3).

(ii) A feasible solution x^0 is called *optimal* if $z(x^0) = c^T x^0 = \max c^T x$, where x is taken from the set of all feasible solutions of the problem (2.1)–(2.3).

(iii) A feasible solution x^ϵ is said to be *ϵ-optimal* or *suboptimal* if

$$z(x^0) - z(x^\epsilon) = c^T x^0 - c^T x^\epsilon \leq \epsilon, \tag{2.6}$$

where x^0 is an optimal solution for the problem (2.1)–(2.3), and ϵ is a positive number chosen beforehand.

(iv) We consider the set of indices $J_B \subset J$ such that $|J_B| = |I| = m$. Then J_B is called a *support* if $\det(A_B) = \det(A(I, J_B)) \neq 0$.

(v) The pair $\{x, J_B\}$ comprising a feasible solution x and a support J_B will be called a *support feasible solution* (SFS).

(vi) An SFS is called *nondegenerate* if $l_j < x_j < u_j,\ j \in J_B$.

Remark 2.1. An SFS is a more general concept than the basic feasible solution (BFS). Indeed, the nonsupport components of an SFS are not restricted to their bounds. Therefore, an SFS may be an interior point, a boundary point or an extreme point, but a BFS is always an extreme point. That is why we can classify the primal support method in the class of interior search methods within the simplex framework [35].

(i) We define the Lagrange multipliers vector π and the reduced costs vector Δ as follows:

$$\pi^T = c_B^T A_B^{-1}, \quad \Delta^T = \Delta^T(J) = \pi^T A - c^T = \left(\Delta_N^T, \Delta_B^T\right), \tag{2.7}$$

where $\Delta_N^T = c_B^T A_B^{-1} A_N - c_N^T,\ \Delta_B^T = c_B^T A_B^{-1} A_B - c_B^T = 0$.

Theorem 2.2 (the optimality criterion [5]). *Let $\{x, J_B\}$ be an SFS for the problem* (2.1)–(2.3). *So the relations:*

$$\begin{aligned} &\Delta_j \geq 0 \quad for\ x_j = l_j, \\ &\Delta_j \leq 0 \quad for\ x_j = u_j, \\ &\Delta_j = 0 \quad for\ l_j < x_j < u_j,\ j \in J_N \end{aligned} \tag{2.8}$$

are sufficient and, in the case of nondegeneracy of the SFS $\{x, J_B\}$*, also necessary for the optimality of the feasible solution* x*.*

The quantity $\beta(x, J_B)$ defined by:

$$\beta(x, J_B) = \sum_{\Delta_j > 0,\, j \in J_N} \Delta_j (x_j - l_j) + \sum_{\Delta_j < 0,\, j \in J_N} \Delta_j (x_j - u_j), \tag{2.9}$$

is called the *suboptimality estimate*. Thus, we have the following theorem [5].

Theorem 2.3 (sufficient condition for suboptimality). *Let* $\{x, J_B\}$ *be an SFS for the problem* (2.1)–(2.3) *and* ϵ *an arbitrary positive number. If* $\beta(x, J_B) \leq \epsilon$*, then the feasible solution* x *is* ϵ*-optimal.*

2.2. The Primal Support Method

Let $\{x, J_B\}$ be an initial SFS and ϵ an arbitrary positive number. The scheme of the primal support method is described in the following steps:

(1) Compute $\pi^T = c_B^T A_B^{-1}$; $\Delta_j = \pi^T a_j - c_j$, $j \in J_N$.

(2) Compute $\beta(x, J_B)$ with the formula (2.9).

(3) If $\beta(x, J_B) = 0$, then the algorithm stops with $\{x, J_B\}$, an optimal SFS.

(4) If $\beta(x, J_B) \leq \epsilon$, then the algorithm stops with $\{x, J_B\}$, an ϵ-optimal SFS.

(5) Determine the nonoptimal index set:

$$J_{NNO} = \{ j \in J_N : [\Delta_j < 0,\ x_j < u_j] \text{ or } [\Delta_j > 0,\ x_j > l_j] \}. \tag{2.10}$$

(6) Choose an index j_0 from J_{NNO} such that $|\Delta_{j_0}| = \max_{j \in J_{NNO}} |\Delta_j|$.

(7) Compute the search direction d using the relations:

$$\begin{aligned} d_{j_0} &= -\operatorname{sign} \Delta_{j_0}, \\ d_j &= 0, \quad j \neq j_0,\ j \in J_N, \\ d(J_B) &= -A_B^{-1} A_N d(J_N) = -A_B^{-1} a_{j_0} d_{j_0}. \end{aligned} \tag{2.11}$$

(8) Compute $\theta_{j_1} = \min_{j \in J_B} \theta_j$, where θ_j is determined by the formula:

$$\theta_j = \begin{cases} \dfrac{(u_j - x_j)}{d_j}, & \text{if } d_j > 0; \\ \dfrac{(l_j - x_j)}{d_j}, & \text{if } d_j < 0; \\ \infty, & \text{if } d_j = 0. \end{cases} \tag{2.12}$$

(9) Compute θ_{j_0} using the formula

$$\theta_{j_0} = \begin{cases} x_{j_0} - l_{j_0}, & \text{if } \Delta_{j_0} > 0; \\ u_{j_0} - x_{j_0}, & \text{if } \Delta_{j_0} < 0. \end{cases} \tag{2.13}$$

(10) Compute $\theta^0 = \min\{\theta_{j_1}, \theta_{j_0}\}$.

(11) Compute $\overline{x} = x + \theta^0 d$, $\overline{z} = z + \theta^0 |\Delta_{j_0}|$.

(12) Compute $\beta(\overline{x}, J_B) = \beta(x, J_B) - \theta^0 |\Delta_{j_0}|$.

(13) If $\beta(\overline{x}, J_B) = 0$, then the algorithm stops with $\{\overline{x}, J_B\}$, an optimal SFS.

(14) If $\beta(\overline{x}, J_B) \leq \epsilon$, then the algorithm stops with $\{\overline{x}, J_B\}$, an ϵ-optimal SFS.

(15) If $\theta^0 = \theta_{j_0}$, then we put $\overline{J}_B = J_B$.

(16) If $\theta^0 = \theta_{j_1}$, then we put $\overline{J}_B = (J_B \setminus \{j_1\}) \cup \{j_0\}$.

(17) We put $x = \overline{x}$ and $J_B = \overline{J}_B$. Go to the step (1).

3. Finding an Initial SFS for the Primal Support Method

Consider the linear programming problem written in the following general form:

$$\begin{aligned} \max \quad & z = c^T x, \\ \text{s.t.} \quad & A^1 x \leq b^1, \\ & A^2 x = b^2, \\ & l \leq x \leq u, \end{aligned} \tag{3.1}$$

where $c = (c_1, c_2, \ldots, c_p)^T$, $x = (x_1, x_2, \ldots, x_p)^T$, $l = (l_1, l_2, \ldots, l_p)^T$, $u = (u_1, u_2, \ldots, u_p)^T$ are vectors in $\mathbb{R}^p$; A^1 is a matrix of dimension $(m_1 \times p)$, A^2 is a matrix of dimension $(m_2 \times p)$, $b^1 \in \mathbb{R}^{m_1}$, $b^2 \in \mathbb{R}^{m_2}$. We assume that $\|l\| < \infty$ and $\|u\| < \infty$.

Let $m = m_1 + m_2$ be the number of constraints of the problem (3.1) and $I = \{1, 2, \ldots, m\}$, $J_0 = \{1, 2, \ldots, p\}$ are, respectively, the constraints indices set and the original variables indices set of the problem (3.1). We partition the set I as follows: $I = I_1 \cup I_2$, where I_1 and I_2 represent, respectively, the indices set of inequality and equality constraints. We note by $e^{(j)}$ the j-vector of ones, that is, $e^{(j)} = (1, 1, \ldots, 1) \in \mathbb{R}^j$ and e_j the jth vector of the identity matrix I_m of order m.

After adding $m_1 = |I_1|$ slack variables to the problem (3.1), we get the following problem in standard form:

$$\max \quad z = c^T x, \tag{3.2}$$

$$\text{s.t.} \quad Ax + H^e x^e = b, \tag{3.3}$$

$$l \leq x \leq u, \tag{3.4}$$

$$0 \leq x^e \leq u^e, \tag{3.5}$$

where $A = (a_{ij}, i \in I, j \in J_0) = \begin{pmatrix} A^1 \\ A^2 \end{pmatrix}$, $b = (b_i, i \in I) = \begin{pmatrix} b^1 \\ b^2 \end{pmatrix}$, $H^e = \begin{pmatrix} I_{m_1} \\ 0_{(m_2 \times m_1)} \end{pmatrix} = (e_i, i \in I_1) = (e_1, e_2, \ldots, e_{m_1})$, $x^e = (x_{p+i}, i \in I_1) = (x_{p+1}, x_{p+2}, \ldots, x_{p+m_1})^T$ is the vector of the added slack variables and $u^e = (u_{p+i}, i \in I_1) = (u_{p+1}, u_{p+2}, \ldots, u_{p+m_1})^T$ its upper bound vector. In order to work with finite bounds, we set $u_{p+i} = M$, $i \in I_1$, that is, $u^e = Me^{(m_1)}$, where M is a finite, positive and big real number chosen carefully.

Remark 3.1. The upper bounds, u_{p+i}, $i \in I_1$, of the added slack variables can also be deduced as follows: $u_{p+i} = b_i - A_i^T h^i, i \in I_1$, where h^i is a p-vector computed with the formula

$$h_j^i = \begin{cases} l_j, & \text{if } a_{ij} > 0; \\ u_j, & \text{if } a_{ij} < 0; \\ 0, & \text{if } a_{ij} = 0. \end{cases} \tag{3.6}$$

Indeed, from the system (3.3), we have $x_{p+i} = b_i - \sum_{j=1}^{p} a_{ij} x_j$, $i \in I_1$. By using the bound constraints (3.4), we get $x_{p+i} \leq b_i - \sum_{j=1}^{p} a_{ij} h_j^i = b_i - A_i^T h^i$, $i \in I_1$. However, the experimental study shows that it's more efficient to set $u_{p+i}, i \in I_1$, to a given finite big value, because for the large-scale problems, the deduction formula (3.6) given above takes much CPU time to compute bounds for the slack variables.

The initialization of the primal support method consists of finding an initial support feasible solution for the problem (3.2)–(3.5). In this section, being inspired from the technique used to initialize interior point methods [36] and taking into account, the fact that the support method can start with a feasible point and a support which are independent, we suggest a single artificial variable technique to find an initial SFS. Before presenting the suggested technique, we first extend the full artificial basis technique, originally presented in [7] for standard form, to solve linear programs presented in the general form (3.1).

3.1. The Full Artificial Basis Technique

Let x^+ be a p-vector chosen between l and u and w an m-vector such that $w = b - Ax^+$. We consider the following subsets of I_1: $I_1^+ = \{i \in I_1 : w_i \geq 0\}$, $I_1^- = \{i \in I_1, w_i < 0\}$, and we assume without loss of generality that $I_1^+ = \{1, 2, \ldots, k\}$, with $k \leq m_1$ and $I_1^- = \{k+1, k+2, \ldots, m_1\}$. Remark that I_1^+ and I_1^- form a partition of I_1; $|I_1^+| = k$ and $|I_1^-| = m_1 - k$.

We make the following partition for x^e, u^e and H^e: $x^e = \begin{pmatrix} x^{e+} \\ x^{e-} \end{pmatrix}$, where

$$\begin{aligned} x^{e+} &= \left(x_{p+i}, i \in I_1^+\right) = \left(x_{p+1}, x_{p+2}, \ldots, x_{p+k}\right)^T, \\ x^{e-} &= \left(x_{p+i}, i \in I_1^-\right) = \left(x_{p+k+1}, x_{p+k+2}, \ldots, x_{p+m_1}\right)^T; \end{aligned} \tag{3.7}$$

$u^e = \begin{pmatrix} u^{e+} \\ u^{e-} \end{pmatrix}$, where

$$\begin{aligned} u^{e+} &= \left(u_{p+i}, i \in I_1^+\right) = \left(u_{p+1}, u_{p+2}, \ldots, u_{p+k}\right)^T, \\ u^{e-} &= \left(u_{p+i}, i \in I_1^-\right) = \left(u_{p+k+1}, u_{p+k+2}, \ldots, u_{p+m_1}\right)^T, \end{aligned} \tag{3.8}$$

and $H^e = (H^{e+}, H^{e-})$, where

$$\begin{aligned} H^{e+} &= \left(e_i, i \in I_1^+\right) = I_m\left(I, I_1^+\right) = (e_1, e_2, \ldots, e_k), \\ H^{e-} &= \left(e_i, i \in I_1^-\right) = I_m\left(I, I_1^-\right) = (e_{k+1}, e_{k+2}, \ldots, e_{m_1}). \end{aligned} \tag{3.9}$$

Hence, the problem (3.2)–(3.5) becomes

$$\max \quad z = c^T x, \tag{3.10}$$

$$\text{s.t.} \quad Ax + H^{e-}x^{e-} + H^{e+}x^{e+} = b, \tag{3.11}$$

$$l \le x \le u, \tag{3.12}$$

$$0 \le x^{e+} \le u^{e+}, \quad 0 \le x^{e-} \le u^{e-}. \tag{3.13}$$

After adding $s = m - k$ artificial variables to the equations $k+1, k+2, \ldots, m$ of the system (3.11), where s is the number of elements of the artificial index set $I^a = I_1^- \cup I_2 = \{k+1, k+2, \ldots, m_1, m_1+1, \ldots, m\}$, we get the following auxiliary problem:

$$\begin{aligned} \max \quad & \psi = -e^{(s)T}x^a, \\ \text{s.t.} \quad & Ax + H^{e-}x^{e-} + H^{e+}x^{e+} + H^a x^a = b, \\ & l \le x \le u, \\ & 0 \le x^{e+} \le u^{e+}, \quad 0 \le x^{e-} \le u^{e-}, \\ & 0 \le x^a \le u^a, \end{aligned} \tag{3.14}$$

where

$$x^a = \left(x_{p+m_1+i}, i = \overline{1,s}\right) = \left(x_{p+m_1+1}, \ldots, x_{p+m_1+s}\right)^T \tag{3.15}$$

represents the artificial variables vector,

$$\begin{aligned} H^a &= \left(\operatorname{sign}(w_i)e_i, i \in I^a\right) = \left(\operatorname{sign}(w_{k+1})e_{k+1}, \operatorname{sign}(w_{k+2})e_{k+2}, \ldots, \operatorname{sign}(w_m)e_m\right), \\ u^a &= |w(I^a)| + \delta e^{(s)} = \left(|w_i| + \delta, i \in I^a\right), \end{aligned} \tag{3.16}$$

where δ is a real nonnegative number chosen in advance.

If we put $x_N = \begin{pmatrix} x \\ x^{e-} \end{pmatrix} \in \mathbb{R}^{p+m_1-k}$, $x_B = \begin{pmatrix} x^{e+} \\ x^a \end{pmatrix} \in \mathbb{R}^m$, $A_N = (A, H^{e-})$, $A_B = (H^{e+}, H^a)$, $l_N = \begin{pmatrix} l \\ 0_{\mathbb{R}^{m_1-k}} \end{pmatrix}$, $u_N = \begin{pmatrix} u \\ u^{e-} \end{pmatrix}$, $l_B = 0_{\mathbb{R}^{k+s}} = 0_{\mathbb{R}^m}$, $u_B = \begin{pmatrix} u^{e+} \\ u^a \end{pmatrix}$, $c_B = \begin{pmatrix} 0_{\mathbb{R}^k} \\ -e^{(s)} \end{pmatrix}$, then we get the following auxiliary problem:

$$\max \quad \psi = c_B^T x_B, \tag{3.17}$$

$$\text{s.t.} \quad A_N x_N + A_B x_B = b, \tag{3.18}$$

$$l_N \le x_N \le u_N, \quad l_B \le x_B \le u_B. \tag{3.19}$$

The variables indices set of the auxiliary problem (3.17)–(3.19) is

$$\begin{aligned} J &= J_0 \cup \{p+i,\ i \in I_1\} \cup \left\{p+m_1+i,\ i=\overline{1,s}\right\} \\ &= \{1,\ldots,p,p+1,\ldots,p+m_1,p+m_1+1,\ldots,p+m_1+s\}. \end{aligned} \tag{3.20}$$

Let us partition this set as follows: $J = J_N \cup J_B$, where

$$\begin{aligned} J_N &= J_0 \cup \{p+i, i \in I_1^-\} = \{1,\ldots,p,p+k+1,\ldots,p+m_1\}, \\ J_B &= \{p+i, i \in I_1^+\} \cup \left\{p+m_1+i, i=\overline{1,s}\right\} \\ &= \{p+1,\ldots,p+k,p+m_1+1,\ldots,p+m_1+s\}. \end{aligned} \tag{3.21}$$

Remark that the pair $\{y, J_B\}$, where

$$y = \begin{pmatrix} y_N \\ y_B \end{pmatrix} = \begin{pmatrix} x_N \\ x_B \end{pmatrix} = \begin{pmatrix} x \\ x^{e-} \\ x^{e+} \\ x^a \end{pmatrix} = \begin{pmatrix} x^+ \\ 0_{\mathbb{R}^{m_1-k}} \\ w(I_1^+) \\ |w(I^a)| \end{pmatrix}, \tag{3.22}$$

with $w(I_1^+) = (w_i, i \in I_1^+)$ and $w(I^a) = (w_i, i \in I^a)$, is a support feasible solution (SFS) for the auxiliary problem (3.17)–(3.19). Indeed, y lies between its lower and upper bounds: for $\delta \geq 0$ we have

$$\begin{pmatrix} l \\ 0_{\mathbb{R}^{m_1-k}} \\ 0_{\mathbb{R}^k} \\ 0_{\mathbb{R}^s} \end{pmatrix} \leq y = \begin{pmatrix} x^+ \\ 0_{\mathbb{R}^{m_1-k}} \\ w(I_1^+) \\ |w(I^a)| \end{pmatrix} \leq \begin{pmatrix} u \\ Me^{(m_1-k)} \\ Me^{(k)} \\ |w(I^a)| + \delta e^{(s)} \end{pmatrix}. \tag{3.23}$$

Furthermore, the $(m \times m)$-matrix

$$A_B = (e_1, e_2, \ldots, e_k, \operatorname{sign}(w_{k+1})e_{k+1}, \operatorname{sign}(w_{k+2})e_{k+2}, \ldots, \operatorname{sign}(w_m)e_m) \tag{3.24}$$

is invertible because

$$\det(A_B) = \prod_{i \in I^a} \operatorname{sign}(w_i) = \prod_{i=k+1}^{m} \operatorname{sign}(w_i) = \pm 1, \tag{3.25}$$

and y verifies the main constraints: by replacing x, x^{e-}, x^{e+}, and x^a with their values in the system (3.18), we get

$$\begin{aligned} A_N x_N + A_B x_B &= \left(A, H^{e-}\right)\begin{pmatrix} x^+ \\ 0_{\mathbb{R}^{m_1-k}} \end{pmatrix} + \left(H^{e+}, H^a\right)\begin{pmatrix} w\left(I_1^+\right) \\ |w(I^a)| \end{pmatrix} \\ &= Ax^+ + H^{e+}w\left(I_1^+\right) + H^a|w(I^a)| \\ &= Ax^+ + \sum_{i=1}^{k} w_i e_i + \sum_{i=k+1}^{m} w_i e_i \\ &= Ax^+ + I_m w \\ &= Ax^+ + w \\ &= Ax^+ + b - Ax^+ \\ &= b. \end{aligned} \tag{3.26}$$

Therefore, the primal support method can be initialized with the SFS $\{y, J_B\}$ to solve the auxiliary problem. Let $\{y^*, J_B^*\}$ be the obtained optimal SFS after the application of the primal support method to the auxiliary problem (3.17)–(3.19), where

$$y^* = \begin{pmatrix} x^* \\ x^{e-*} \\ x^{e+*} \\ x^{a*} \end{pmatrix}, \quad \varphi^* = -e^{(s)T} x^{a*}. \tag{3.27}$$

If $\varphi^* < 0$, then the original problem (3.2)–(3.5) is infeasible.

Else, when J_B^* does not contain any artificial index, then $\{\begin{pmatrix} x^* \\ x^{e*} \end{pmatrix}, J_B^*\}$ will be an SFS for the original problem (3.2)–(3.5). Otherwise, we delete artificial indices from the support J_B^*, and we replace them with original or slack appropriate indices, following the algebraic rule used in the simplex method.

In order to initialize the primal support method for solving linear programming problems with bounded variables written in the standard form, in [7], Gabasov et al. add m artificial variables, where m represents the number of constraints. In this work, we are interested in solving the problem written in the general form (3.1), so we have added artificial variables only for equality constraints and inequality constraints with negative components of the vector w.

Remark 3.2. We have $I = I_1 \cup I_2 = I_1^+ \cup I_1^- \cup I_2 = I_1^+ \cup I^a$. Since in the relationship (3.22), we have $y_B = x_B = \begin{pmatrix} w(I_1^+) \\ |w(I^a)| \end{pmatrix}$ and $w(I_1^+) \geq 0$, we get $y_B = |w(I_1^+ \cup I^a)| = |w(I)| = |w|$.

Remark 3.3. If we choose x^+ such that $x^+ = l$ or $x^+ = u$, then the vector y, given by the relationship (3.22), will be a BFS. Hence, the simplex algorithm can be initialized with this point.

Remark 3.4. If $b^1 \geq 0_{\mathbb{R}^{m_1}}$, $l \leq 0_{\mathbb{R}^p}$ and $u \geq 0_{\mathbb{R}^p}$, two cases can occur.

Case 1. If $b^2 \geq 0_{\mathbb{R}^{m_2}}$, then we put $x^+ = 0_{\mathbb{R}^p}$.

Case 2. If $b^2 < 0_{\mathbb{R}^{m_2}}$, then we put $A^2 = -A^2$, $b^2 = -b^2$ and $x^+ = 0_{\mathbb{R}^p}$.

In the two cases, we get $b \geq 0_{\mathbb{R}^m}$, $w = b - Ax^+ = b \geq 0$, therefore $I_1^- = \emptyset$. Hence $I^a = I_1^- \cup I_2 = I_2$, so we add only m_2 artificial variables for the equality constraints.

3.2. The Single Artificial Variable Technique

In order to initialize the primal support method using this technique, we first start by searching an initial support, then we proceed to the search of an initial feasible solution for the original problem.

The application of the Gauss elimination method with partial pivoting to the system of equations (3.3) can give us a support $J_B = \{j_1, j_2, \ldots, j_r\}$, where $r \leq m$. However, the experimental study realized in [33, 34] reveals that this approach takes much time in searching the initial support, that is, why it's important to take into account the sparsity property of practical problems and apply some procedure to find a triangular basis among the columns corresponding to the original variables, that is, the columns of the matrix A. In this work, on the base of the crash procedures presented in [26, 28–30], we present a procedure to find an initial support for the problem (3.2)–(3.5).

Procedure 1 (searching an initial support). (1) We sort the columns of the $(m \times p$-) matrix A according to the increasing order of their number of nonzero elements. Let L be the list of the sorted columns of A.

(2) Let $a_{j_0} = L(r)$ (the rth column of L) be the first column of the list L verifying: $\exists i_0 \in I$ such that

$$\left|a_{i_0 j_0}\right| = \max_{i \in I} \left|a_{i j_0}\right|, \quad \left|a_{i_0 j_0}\right| > \text{pivtol}, \tag{3.28}$$

where pivtol is a given tolerance. Hence, we put $J_B = \{j_0\}$, $I_{\text{piv}} = \{i_0\}$, $k = 1$.

(3) Let j_k be the index corresponding to the column $(r + k)$ of the list L, that is, $a_{j_k} = L(r + k)$. If a_{j_k} has zero elements in all the rows having indices in I_{piv} and if $\exists i_k \in I$ such that

$$\left|a_{i_k j_k}\right| = \max_{i \in I \setminus I_{\text{piv}}} \left|a_{i j_k}\right|, \quad \left|a_{i_k j_k}\right| > \text{pivtol}, \tag{3.29}$$

then we put $J_B = J_B \cup \{j_k\}$, $I_{\text{piv}} = I_{\text{piv}} \cup \{i_k\}$.

(4) We put $k = k + 1$. If $r + k \leq p$, then go to step (3), else go to step (5).

(5) We put $s = 0$, $I^a = \emptyset$, $J^a = \emptyset$.

(6) For all $i \in I \setminus I_{\text{piv}}$, if the ith constraint is originally an inequality constraint, then we add to J_B an index corresponding to the slack variable $p + i$, that is, $J_B = J_B \cup \{p + i\}$. If the ith constraint is originally an equality constraint, then we put $s = s + 1$ and add to this latter an artificial variable x_{p+m_1+s} for which we set the lower bound to zero and the upper bound to a big and well-chosen value M. Thus, we put $J_B = J_B \cup \{p + m_1 + s\}$, $J^a = J^a \cup \{p + m_1 + s\}$, $I^a = I^a \cup \{i\}$.

Remark 3.5. The inputs of Procedure 1 are:

(i) the $(m \times p)$-matrix of constraints, A;

(ii) a pivoting tolerance fixed beforehand, pivtol;

(iii) a big and well chosen value M.

The outputs of Procedure 1 are:

(i) $J_B = \{j_0, j_1, \ldots, j_{m-1}\}$: the initial support for the problem (3.2)–(3.5);

(ii) I^a: the indices of the constraints for which we have added artificial variables;

(iii) J^a: the indices of artificial variables added to the problem (3.2)–(3.5);

(iv) $s = |I^a| = |J^a|$: the number of artificial variables added to the problem (3.2)–(3.5).

After the application of the procedure explained above (Procedure 1) for the problem (3.2)–(3.5), we get the following linear programming problem:

$$\begin{aligned} \max \quad & z = c^T x, \\ \text{s.t.} \quad & Ax + H^e x^e + H^a x^a = b, \\ & l \le x \le u, \\ & 0 \le x^e \le M e^{(m_1)}, \\ & 0 \le x^a \le M e^{(s)}, \end{aligned} \tag{3.30}$$

where $x^a = (x_{p+m_1+1}, x_{p+m_1+2}, \ldots, x_n)^T$ is the s-vector of the artificial variables added during the application of Procedure 1, with $n = p + m_1 + s$, $H^a = (e_i, i \in I^a)$ is an $(m \times s)$-matrix.

Since we have got an initial support, we can start the procedure of finding a feasible solution for the original problem.

Procedure 2 (searching an initial feasible solution). Consider the following auxiliary problem:

$$\begin{aligned} \max \quad & \psi = -x_{n+1} - \sum_{j \in J^a} x_j, \\ \text{s.t.} \quad & Ax + H^e x^e + H^a x^a + \rho x_{n+1} = b, \\ & l \le x \le u, \\ & 0 \le x^e \le M e^{(m_1)}, \\ & 0 \le x^a \le M e^{(s)}, \\ & 0 \le x_{n+1} \le 1, \end{aligned} \tag{3.31}$$

where x_{n+1} is an artificial variable, $\rho = b - Ax^+$, and x^+ is a p-vector chosen between l and u. We remark that

$$y = \begin{pmatrix} x \\ x^e \\ x^a \\ x_{n+1} \end{pmatrix} = \begin{pmatrix} x^+ \\ 0_{\mathbb{R}^{m_1}} \\ 0_{\mathbb{R}^s} \\ 1 \end{pmatrix} \tag{3.32}$$

is an obvious feasible solution for the auxiliary problem. Indeed,

$$Ax^+ + H^e 0_{\mathbb{R}^{m_1}} + H^a 0_{\mathbb{R}^s} + b - Ax^+ = b. \tag{3.33}$$

Hence, we can apply the primal support method to solve the auxiliary problem (3.31) by starting with the initial SFS $\{y, J_B\}$, where $J_B = \{j_0, j_1, \ldots, j_{m-1}\}$ is the support obtained with Procedure 1. Let

$$y^* = \begin{pmatrix} x^* \\ x^{e*} \\ x^{a*} \\ x^*_{n+1} \end{pmatrix}, \; J_B^*, \; \psi^* \tag{3.34}$$

be, respectively, the optimal solution, the optimal support, and the optimal objective value of the auxiliary problem (3.31).

If $\psi^* < 0$, then the original problem (3.2)–(3.5) is infeasible.

Else, when J_B^* does not contain any artificial index, then $\{\binom{x^*}{x^{e*}}, J_B^*\}$ will be an SFS for the original problem (3.2)–(3.5). Otherwise, we delete the artificial indices from the support J_B^* and we replace them with original or slack appropriate indices, following the algebraic rule used in the simplex method.

Remark 3.6. The number of artificial variables $n_a = s+1$ of the auxiliary problem (3.31) verifies the inequality: $1 \leq n_a \leq m_2 + 1$.

The auxiliary problem will have only one artificial variable, that is, $n_a = 1$, when the initial support J_B found by Procedure 1 is constituted only by the original and slack variable indices ($J^a = \emptyset$), and this will hold in two cases.

Case 1. When all the constraints are inequalities, that is, $I_2 = \emptyset$.

Case 2. When $I_2 \neq \emptyset$ and step (4) of Procedure 1 ends with $I_{\text{piv}} = I_2$.

The case $n_a = m_2 + 1$ holds when the step (4) of Procedure 1 stops with $I_{\text{piv}} = I_1$.

Remark 3.7. Let's choose a p-vector x^+ between l and u and two nonnegative vectors $v^e \in R^{m_1}$ and $v^a \in R^s$. If we put in the auxiliary problem (3.31), $\rho = b - v - Ax^+$, with $v = H^e v^e + H^a v^a$, then the vector

$$y = \begin{pmatrix} x \\ x^e \\ x^a \\ x_{n+1} \end{pmatrix} = \begin{pmatrix} x^+ \\ v^e \\ v^a \\ 1 \end{pmatrix} \tag{3.35}$$

is a feasible solution for the auxiliary problem. Indeed,

$$Ax + H^e x^e + H^a x^a + \rho x_{n+1} = Ax^+ + H^e v^e + H^a v^a + b - H^e v^e - H^a v^a - Ax^+ = b. \tag{3.36}$$

We remark that if we put $v^e = 0_{\mathbb{R}^{m_1}}$, $v^a = 0_{\mathbb{R}^s}$, we get $v = 0_{\mathbb{R}^m}$, then $\rho = b - Ax^+$ and we obtain the evident feasible point that we have used in our numerical experiments, that is,

$$y = \begin{pmatrix} x \\ x^e \\ x^a \\ x_{n+1} \end{pmatrix} = \begin{pmatrix} x^+ \\ 0_{\mathbb{R}^{m_1}} \\ 0_{\mathbb{R}^s} \\ 1 \end{pmatrix}. \tag{3.37}$$

It's important here, to cite two other special cases.

Case 1. If we choose the nonbasic components of y equal to their bounds, then we obtain a BFS for the auxiliary problem, therefore we can initialize the simplex algorithm with it.

Case 2. If we put $v^e = e^{(m_1)}$ and $v^a = e^{(s)}$, then $v = H^e e^{(m_1)} + H^a e^{(s)} = \sum_{i\in I_1} e_i + \sum_{i\in I^a} e_i$. Hence, the vector

$$y = \begin{pmatrix} x \\ x^e \\ x^a \\ x_{n+1} \end{pmatrix} = \begin{pmatrix} x^+ \\ e^{(m_1)} \\ e^{(s)} \\ 1 \end{pmatrix} = \begin{pmatrix} x^+ \\ e^{(m_1+s+1)} \end{pmatrix} \tag{3.38}$$

is a feasible solution for the auxiliary problem (3.31), with $\rho = b - v - Ax^+$.

Numerical Example

Consider the following LP problem:

$$\left\{\max\ c^T x,\ \text{s.t. } A^1 x \le b^1,\ A^2 x = b^2,\ l \le x \le u\right\}, \tag{3.39}$$

where

$$\begin{gathered} A^1 = \begin{pmatrix} 0 & 0 & 1 & 2 \\ 2 & 0 & 0 & -3 \end{pmatrix}, \quad b^1 = \begin{pmatrix} 3 \\ 3 \end{pmatrix}, \quad A^2 = \begin{pmatrix} 0 & 3 & 0 & 2 \\ 1 & 0 & 2 & 3 \end{pmatrix}, \quad b^2 = \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \\ c = (2, -3, -1, 1)^T, \quad l = 0_{\mathbb{R}^4}, \quad u = (10, 10, 10, 10)^T, \quad x = (x_1, x_2, x_3, x_4)^T. \end{gathered} \tag{3.40}$$

We put $M = 10^{10}$, pivtol $= 10^{-6}, x^+ = 0_{\mathbb{R}^4}$, and we apply the two-phase primal support method using the single artificial variable technique to solve the problem (3.39).

Phase 1. After adding the slack variables x_5 and x_6 to the problem (3.39), we get the following problem in standard form:

$$\left\{\max\ z = c^T x,\ \text{s.t. } Ax + H^e x^e = b,\ l \le x \le u,\ 0_{\mathbb{R}^2} \le x^e \le u^e\right\}, \tag{3.41}$$

where

$$A = \begin{pmatrix} 0 & 0 & 1 & 2 \\ 2 & 0 & 0 & -3 \\ 0 & 3 & 0 & 2 \\ 1 & 0 & 2 & 3 \end{pmatrix}, \quad H^e = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad b = \begin{pmatrix} 3 \\ 3 \\ 2 \\ 2 \end{pmatrix}, \quad x^e = \begin{pmatrix} x_5 \\ x_6 \end{pmatrix}, \quad u^e = \begin{pmatrix} M \\ M \end{pmatrix}. \tag{3.42}$$

The application of Procedure 1 to the problem (3.41) gives us the following initial support: $J_B = \{2,1,3,5\}$. In order to find an initial feasible solution to the original problem, we add an artificial variable x_7 to problem (3.41), and we compute the vector ρ: $\rho = b - Ax^+ = b$. Thus, we obtain the following auxiliary problem:

$$\left\{\max \psi = -x_7,\ \text{s.t. } Ax + H^e x^e + \rho x_7 = b,\ l \le x \le u,\ 0_{\mathbb{R}^2} \le x^e \le u^e,\ 0 \le x_7 \le 1\right\}. \tag{3.43}$$

Remark that the SFS $\{y, J_B\}$, where $y = (x_1, x_2, x_3, x_4, x_5, x_6, x_7)^T = (0,0,0,0,0,0,1)^T$, and $J_B = \{2,1,3,5\}$ is an obvious support feasible solution for the auxiliary problem. Hence, the primal support method can be applied to solve the problem (3.43) starting with the SFS $\{y, J_B\}$.

Iteration 1. The initial support is $J_B = \{2,1,3,5\}$, $J_N = \{4,6,7\}$; the initial feasible solution and the corresponding objective function value are: $y = (0,0,0,0,0,0,1)^T$ and $\psi = -1$.

The vector of multipliers is $\pi^T = (0,0,0,0)$, and $\Delta_N^T = (0,0,1)$. Hence, the reduced costs vector is $\Delta = (0,0,0,0,0,0,1)^T$.

The suboptimality estimate is $\beta(x, J_B) = \Delta_7(x_7 - l_7) = 1 > 0$. So the current solution is not optimal. The set of nonoptimal indices is $J_{NNO} = \{7\} \Rightarrow j_0 = 7$.

In order to improve the objective function, we compute the search direction d: we have $d_7 = -\operatorname{sign} \Delta_7 = -1$, so $d_N = (d_4, d_6, d_7)^T = (0,0,-1)^T$; $d_B = -A_B^{-1}A_N d_N = (2/3, 3/2, 1/4, 11/4)^T$. Hence, the search direction is $d = (3/2, 2/3, 1/4, 0, 11/4, 0, -1)^T$.

Since $d_j > 0,\ \forall j \in J_B$, $\theta_j = (u_j - x_j)/d_j,\ \forall j \in J_B$. So $\theta_2 = 15$, $\theta_1 = 20/3$, $\theta_3 = 40$ and $\theta_5 = 4M/11 \Rightarrow \theta_{j_1} = \theta_1 = 20/3$, and $\theta_{j_0} = \theta_7 = x_7 - l_7 = 1$. Hence, the primal step length is $\theta^0 = \min\{\theta_1, \theta_7\} = 1 = \theta_7$. The new solution is $\overline{y} = y + \theta^0 d = (3/2, 2/3, 1/4, 0, 11/4, 0, 0)^T$, and the new objective value is $\overline{\psi} = \psi + \theta^0|\Delta_7| = 0 \Rightarrow y$ is optimal for the auxiliary problem. Therefore, the pair $\{\mathbf{x}, J_B\}$, where $\mathbf{x} = (3/2, 2/3, 1/4, 0, 11/4, 0)^T$ and $J_B = \{2,1,3,5\}$, is an SFS for the problem (3.41).

Phase 2.

Iteration 1. The initial support is $J_B = \{2,1,3,5\}$, and $J_N = \{4,6\}$.

The initial feasible solution is $\mathbf{x} = (3/2, 2/3, 1/4, 0, 11/4, 0)^T$, and the objective value is $z = 3/4$.

The multipliers vector and the reduced costs vector are: $\pi^T = (0, 5/4, -1, -1/2)$ and $\Delta = (0,0,0,-33/4,0,5/4)^T$. The suboptimality estimate is $\beta(\mathbf{x}, J_B) = \Delta_4(x_4 - u_4) + \Delta_6(x_6 - l_6) = 165/2 > 0$. So the initial solution is not optimal.

The set of nonoptimal indices is $J_{NNO} = \{4\} \Rightarrow$ the entering index is $j_0 = 4$.

The search direction is $d = (3/2, -2/3, -9/4, 1, 1/4, 0)^T$.

The primal step length is $\theta^0 = \min\{\theta_3, \theta_4\} = \min\{1/9, 1\} = 1/9 = \theta_3$. So the leaving index is $j_1 = 3$. The new solution is $\overline{\mathbf{x}} = \mathbf{x} + \theta^0 d = (5/3, 16/27, 0, 1/9, 25/9, 0)^T$, and $\overline{z} = z + \theta^0|\Delta_4| = 5/3$.

Iteration 2. The current support is $J_B = \{2,1,4,5\}$, and $J_N = \{3,6\}$.

The current feasible solution is $\mathbf{x} = (5/3, 16/27, 0, 1/9, 25/9, 0)^T$, and the objective value is $z = 5/3$.

The multipliers vector and the reduced costs vector are: $\pi^T = (0, 1/3, -1, 4/3)$ and $\Delta = (0,0,11/3,0,0,1/3)^T$. The suboptimality estimate is $\beta(\mathbf{x}, J_B) = \Delta_3(x_3 - l_3) + \Delta_6(x_6 - l_6) = 0$.

Therefore, the optimal solution and the optimal objective value of the original problem (3.39) are

$$\mathbf{x}^* = \left(\frac{5}{3}, \frac{16}{27}, 0, \frac{1}{9}\right)^T, \quad z^* = \frac{5}{3}. \tag{3.44}$$

4. Experimental Results

In order to perform a numerical comparison between the simplex method and the different variants of the support method, we have programmed them under the MATLAB programming language version 7.4.0 (R2007a).

Since the test problems available in the NETLIB library present bounds on the variables which can be infinite, we have built a sample of 68 test problems having finite bounds and a same optimal objective value as those of NETLIB. The principle of the building process is as follows: let xl and xu be the obtained bounds after the conversion of a given test problem from the "mps" format to the "mat" format and x^* the optimal solution. We put $x\min = \min\{x_j^*, j = \overline{1,p}\}$ and $x\max = \max\{x_j^*, j = \overline{1,p}\}$. Thus, the constraint matrix and the objective coefficients vector of the new problem remains the same as those of NETLIB, but the new lower bounds $\tilde{l}$ and upper bounds $\tilde{u}$ will be changed as follows:

$$\begin{aligned} \tilde{l}_j &= \begin{cases} x\min, & \text{if } xl_j = -\infty; \\ xl_j, & \text{if } xl_j > -\infty, \end{cases} \\ \tilde{u}_j &= \begin{cases} x\max, & \text{if } xu_j = +\infty; \\ xu_j, & \text{if } xu_j < +\infty. \end{cases} \end{aligned} \tag{4.1}$$

Table 1 represents the listing of the main characteristics of the considered 68 test problems, where NC, NV, NEC, and D represent, respectively, the number of constraints, the number of variables, the number of equality constraints, and the density of the constraints matrix (the ratio between the number of nonzero elements and the number of total elements of the constraints matrix, multiplied by 100).

There exist many LP solvers which include a primal simplex algorithm package for solving LP problems, we cite: commercial solvers such as CPLEX [37], MINOS [28] and open-source solvers such as LP_SOLVE [38], GLPK [39], and LPAKO [30]. The LP_SOLVE solver is well documented and widely used in applications and numerical experiments [40, 41]. Moreover, the latter has a mex interface called mxlpsolve which can be easily installed and used with the MATLAB language. That is why we compare our algorithm with the primal simplex algorithm implemented in LP_SOLVE. However, MATLAB is an interpreted language, so it takes much time in solving problems than the native language C++ used for the implementation of LP_SOLVE. For this reason, the numerical comparison carried out between our method and LP_SOLVE concerns only the iterations number. In order to compare our algorithm with the primal simplex algorithm in terms of CPU time, we have developed our own simplex implementation. The latter is based on the one given in [42].

In order to compare the different solvers, we have given them the following names.

Table 1: Characteristics of the test problems.

LP	NC	NV	NEC	D (%)	LP	NC	NV	NEC	D (%)
(1) afiro	27	32	8	9,61	(35) fffff800	524	854	350	1,39
(2) kb2	43	41	16	16,20	(36) grow22	440	946	440	1,98
(3) sc50a	50	48	20	5,42	(37) standata	359	1075	160	0,79
(4) sc50b	50	48	20	4,92	(38) scsd6	147	1350	147	2,17
(5) adlittle	56	97	15	7,05	(39) standmps	467	1075	268	0,73
(6) blend	74	83	43	7,99	(40) standgub	361	1184	162	0,73
(7) share2b	96	79	13	9,15	(41) scfxm2	660	914	374	0,86
(8) sc105	105	103	45	2,59	(42) scrs8	490	1169	384	0,56
(9) stocfor1	117	111	63	3,44	(43) gfrd-pnc	616	1092	548	0,35
(10) scagr7	129	140	84	2,33	(44) bnl1	643	1175	232	0,68
(11) israel	174	142	0	9,18	(45) ship04s	402	1458	354	0,74
(12) share1b	117	225	89	4,37	(46) fit1p	627	1677	627	0,94
(13) vtpbase	198	203	55	2,26	(47) modszk1	687	1620	687	0,29
(14) sc205	205	203	91	1,32	(48) shell	536	1775	534	0,37
(15) beaconfd	173	262	140	7,45	(49) scfxm3	990	1371	561	0,57
(16) grow7	140	301	140	6,20	(50) 25fv47	821	1571	516	0,81
(17) lotfi	153	308	95	2,29	(51) ship04l	402	2118	354	0,74
(18) brandy	220	249	166	3,92	(52) wood1p	244	2594	243	11,10
(19) e226	223	282	33	4,10	(53) sctap2	1090	1880	470	0,33
(20) bore3d	233	315	214	1,95	(54) ganges	1309	1681	1284	0,31
(21) capri	271	353	142	1,85	(55) scsd8	397	2750	397	0,79
(22) agg	488	163	36	3,03	(56) ship08s	778	2387	698	0,38
(23) scorpion	388	358	280	1,03	(57) ship12s	1151	2763	1045	0,26
(24) bandm	305	472	305	1,73	(58) ctap3	1480	2480	620	0,24
(25) sctap1	300	480	120	1,18	(59) stocfor2	2157	2031	1143	0,19
(26) scfxm1	330	457	187	1,72	(60) czprob	929	3523	890	0,33
(27) agg2	516	302	60	2,75	(61) ship08l	778	4283	698	0,38
(28) agg3	516	302	60	2,76	(62) bnl2	2324	3489	1327	0,17
(29) stair	356	467	209	2,32	(63) ship12l	1151	5427	1045	0,26
(30) scsd1	77	760	77	4,08	(64) d2q06c	2171	5167	1507	0,29
(31) grow15	300	645	300	2,90	(65) woodw	1098	8405	1085	0,41
(32) scagr25	471	500	300	0,66	(66) truss	1000	8806	1000	0,32
(33) fit1d	24	1026	1	54,40	(67) fit2d	25	10500	1	49,10
(34) finnis	497	614	47	0,76	(68) maros-r7	3136	9408	3136	0,49

Solver1. SupportSav

The two-phase support method, where we use the suggested single artificial variable technique in the first phase, that is, the initial support is found by applying Procedure 1, and the feasible solution is found by applying Procedure 2.

Solver2. SupportFab

The two-phase support method, where we use the full artificial basis technique presented above in the first phase.

Table 2: CPU time and iterations number for Solver1 (SupportSav).

LP	$cput_1$	cput	nit_1	nit	LP	$cput_1$	cput	nit_1	nit
afiro	0,00	0,02	15	18	fffff800	0,14	4,75	1339	1707
kb2	0,00	0,05	29	95	grow22	0,42	2,70	18	917
sc50a	0,00	0,02	27	38	standata	0,11	2,19	635	806
sc50b	0,03	0,05	19	33	scsd6	0,05	1,17	87	425
adlittle	0,02	0,08	25	118	standmps	0,16	3,17	998	1064
blend	0,00	0,05	59	81	standgub	0,11	2,09	616	741
share2b	0,00	0,11	134	170	scfxm2	0,27	4,02	990	1344
sc105	0,02	0,06	61	84	scrs8	0,44	2,27	173	707
stocfor1	0,02	0,06	56	74	gfrd-pnc	0,66	2,92	510	929
scagr7	0,02	0,11	114	138	bnl1	0,31	13,78	3541	3902
israel	0,02	0,59	272	589	ship04s	0,05	1,44	324	455
share1b	0,02	0,38	300	450	fit1p	0,88	3,36	4	673
vtpbase	0,03	1,03	942	989	modszk1	0,91	6,83	322	1759
sc205	0,08	0,23	128	169	shell	0,47	2,20	280	525
beaconfd	0,06	0,23	133	161	scfxm3	0,61	9,20	1546	2137
grow7	0,05	0,27	18	229	25fv47	0,33	62,47	5376	12998
lotfi	0,03	0,27	79	228	ship04l	0,08	2,20	324	521
brandy	0,03	0,58	419	461	wood1p	0,05	3,17	383	450
e226	0,05	0,89	165	690	sctap2	1,48	6,09	220	1015
bore3d	0,06	0,23	97	140	ganges	1,83	8,66	1311	1645
capri	0,06	1,05	636	741	scsd8	0,34	3,50	212	643
agg	0,05	0,22	72	127	ship08s	0,17	4,42	656	848
scorpion	0,09	0,64	288	316	ship12s	0,31	8,30	968	1309
bandm	0,13	1,03	360	630	sctap3	2,73	10,08	251	1237
sctap1	0,13	0,61	129	340	stocfor2	4,08	17,31	1325	2061
scfxm1	0,09	1,05	455	620	czprob	0,13	23,55	2477	3373
agg2	0,09	0,38	40	169	ship08l	0,17	8,23	656	1026
agg3	0,09	0,41	36	178	bnl2	4,19	88,17	3256	8840
stair	0,13	1,33	452	701	ship12l	0,38	17,36	979	1679
scsd1	0,02	0,31	48	188	d2q06c	5,14	643,03	6625	46311
grow15	0,20	1,23	18	573	woodw	1,31	49,59	2107	3003
scagr25	0,19	0,89	309	414	truss	2,25	153,02	448	9241
fit1d	0,03	3,36	14	1427	fit2d	0,16	290,91	11	13593
finnis	0,17	2,41	343	994	maros-r7	21,30	100,34	1757	3480

Solver3. SimplexFab

The two-phase primal simplex method, where we use the full artificial basis technique presented above in the first phase (we use the BFS presented in Remark 3.3, with $x^+ = \widetilde{l}$).

Solver4. LP_Solve_PSA

The primal simplex method implemented in the open-source LP solver LP_SOLVE. The different options are (Primal-Primal, PRICER_DANTZIG, No Scaling, No Presolving). For setting these options, the following MATLAB commands are used.

Table 3: CPU time and iterations number for Solver2 (SupportFab).

LP	cput	nit_1	nit	LP	cput	nit_1	nit
afiro	0,02	10	18	fffff800	4,48	1334	1659
kb2	0,06	84	143	grow22	3,59	442	1427
sc50a	0,02	26	49	standata	1,22	427	479
sc50b	0,02	26	52	scsd6	1,50	203	554
adlittle	0,09	35	127	standmps	2,00	667	718
blend	0,06	92	118	standgub	1,28	419	470
share2b	0,08	96	129	scfxm2	5,16	1384	1772
sc105	0,06	56	112	scrs8	3,09	620	1112
stocfor1	0,05	64	82	gfrd-pnc	3,11	719	1095
scagr7	0,19	242	285	bnl1	11,23	2752	3235
israel	0,36	9	346	ship04s	1,67	479	535
share1b	0,33	241	394	fit1p	12,47	1625	3067
vtpbase	1,02	951	1001	modszk1	8,59	1038	2325
sc205	0,22	112	232	shell	2,78	568	757
beaconfd	0,17	138	170	scfxm3	10,39	1853	2464
grow7	0,36	142	337	25fv47	50,19	3562	10694
lotfi	0,28	150	261	ship04l	3,27	706	789
brandy	0,66	520	566	wood1p	3,06	351	426
e226	0,94	160	757	sctap2	7,78	896	1452
bore3d	0,33	226	282	ganges	9,58	1637	1979
capri	0,91	527	678	scsd8	7,48	556	1432
agg	0,19	77	118	ship08s	5,73	913	1125
scorpion	0,73	378	423	ship12s	9,48	1334	1478
bandm	2,22	1095	1339	sctap3	13,50	1183	1902
sctap1	0,59	292	375	stocfor2	11,23	1144	1655
scfxm1	1,17	552	723	czprob	17,91	1647	2544
agg2	0,28	78	169	ship08l	14,63	1404	1806
agg3	0,33	80	181	bnl2	123,72	7209	12614
stair	2,17	860	1128	ship12l	28,77	2463	2719
scsd1	0,38	81	226	d2q06c	644,20	9240	45670
grow15	1,53	301	843	woodw	45,80	1916	2752
scagr25	4,05	1834	2129	truss	180,47	1320	11069
fit1d	3,52	10	1469	fit2d	297,03	11	13574
finnis	2,41	395	1083	maros-r7	137,19	4519	5612

(i) mxlpsolve("set_simplextype", lp, 5); % (Phase 1: Primal, Phase 2: Primal).

(ii) mxlpsolve("set_pivoting", lp, 1); % (Dantzig's rule).

(iii) mxlpsolve("set_scaling", lp, 0); % (No scaling).

The Lagrange multipliers vector and the basic components of the search direction in the first three solvers are computed by solving the two systems of linear equations: $A_B^T \pi = c_B$ and $A_B d_B = -a_{j_0} d_{j_0}$, using the LU factorization of A_B [43]. The updating of the L and U factors is done, in each iteration, using the Sherman-Morrison-Woodbury formula [42].

In the different implementations, we have used the Dantzig's rule to choose the entering variable. To prevent cycling, the EXPAND procedure [44] is used.

Table 4: CPU time and iterations number for Solver3 (SimplexFab).

LP	cput	nit_1	nit	LP	cput	nit_1	nit
afiro	0,02	10	18	fffff800	5,38	1513	1920
kb2	0,05	68	125	grow22	3,38	441	1366
sc50a	0,02	26	51	standata	5,45	2609	2646
sc50b	0,03	26	52	scsd6	1,31	203	524
adlittle	0,06	33	123	standmps	7,19	3217	3232
blend	0,06	91	117	standgub	5,81	2610	2627
share2b	0,08	96	131	scfxm2	4,88	1245	1602
sc105	0,08	56	112	scrs8	2,69	529	975
stocfor1	0,05	64	82	gfrd-pnc	3,05	672	1035
scagr7	0,23	276	318	bnl1	11,06	2718	3100
israel	0,34	9	348	ship04s	1,69	479	534
share1b	0,33	234	383	fit1p	11,98	1716	3001
vtpbase	3,03	3921	4145	modszk1	8,47	1100	2275
sc205	0,22	111	224	shell	4,72	646	1507
beaconfd	0,17	138	170	scfxm3	10,55	1901	2420
grow7	0,34	141	317	25fv47	41,23	3350	8974
lotfi	0,22	155	235	ship04l	3,23	709	784
brandy	0,75	501	569	wood1p	3,11	355	426
e226	0,89	156	713	sctap2	7,88	894	1494
bore3d	0,52	431	486	ganges	9,95	1657	1991
capri	4,19	3769	3917	scsd8	6,27	536	1276
agg	0,20	88	133	ship08s	5,77	912	1121
scorpion	0,73	373	417	ship12s	9,66	1333	1477
bandm	2,13	1034	1251	sctap3	13,22	1176	1879
sctap1	0,63	295	374	stocfor2	11,55	1144	1635
scfxm1	1,08	513	656	czprob	20,03	2164	3027
agg2	0,30	76	167	ship08l	14,36	1404	1800
agg3	0,30	79	179	bnl2	131,38	6782	12918
stair	2,78	1216	1475	ship12l	28,69	2462	2715
scsd1	0,34	81	236	d2q06c	539,45	8708	39227
grow15	1,47	301	816	woodw	41,14	1844	2550
scagr25	3,08	1299	1563	truss	150,53	1359	9845
fit1d	3,06	10	1420	fit2d	275,03	11	13535
finnis	2,53	520	1190	maros-r7	132,11	4434	5477

We have solved the 68 NETLIB test problems listed in Table 1 with the solvers mentioned above on a personal computer with Intel (R) Core (TM) 2 Quad CPU Q6600 2.4 GHz, 4 GB of RAM, working under the Windows XP SP2 operating system, by setting the different tolerances appropriately. We recall that the initial point x^+, needed in the three methods (Solvers 1, 2 and 3), must be located between its lower and upper bounds, so after giving it some values and observing the variation of the CPU time, we have concluded that it's more efficient to set $x^+ = \widetilde{l}$. The upper bounds for the slack and artificial variables added for the methods SupportSav, SupportFab, and SimplexFab are put to $M = 10^{10}$.

Numerical results are reported in Tables 2, 3, 4, and 5, where nit_1, nit, and cput represent respectively the phase one iterations number, the total iterations number, and the

Table 5: Iterations number for Solver4 (LP_SOLVE_PSA).

LP	nit	LP	nit	LP	nit	LP	nit
afiro	17	brandy	1215	fffff800	1771	wood1p	352
kb2	96	e226	909	grow22	1369	sctap2	1465
sc50a	50	bore3d	193	standata	221	ganges	1893
sc50b	51	capri	835	scsd6	594	scsd8	6165
adlittle	172	agg	583	standmps	454	ship08s	702
blend	129	scorpion	386	standgub	210	ship12s	1249
share2b	185	bandm	1040	scfxm2	1272	sctap3	1864
sc105	105	sctap1	359	scrs8	1170	stocfor2	3317
stocfor1	124	scfxm1	592	gfrd-pnc	895	czprob	3422
scagr7	184	agg2	597	bnl1	3810	ship08l	1153
israel	606	agg3	634	ship04s	485	bnl2	9579
share1b	379	stair	936	fit1p	2873	ship12l	2197
vtpbase	782	scsd1	249	modszk1	1736	d2q06c	39828
sc205	250	grow15	987	shell	754	woodw	2179
beaconfd	110	scagr25	898	scfxm3	1962	truss	14046
grow7	328	fit1d	2196	25fv47	12767	fit2d	14249
lotfi	296	finnis	999	ship04l	733	maros-r7	4474

CPU time in seconds of each method necessary to find the optimal objective values presented in [25] or [45]; cput_1, shown in columns 2 and 7 of Table 2, represents the CPU time necessary to find the initial support with Procedure 1. The number of artificial variables added to the original problem in our implementations is listed in Table 6.

In order to compare the different solvers, we have ordered the problems according to the increasing order of the sum of the constraints and variables numbers (NC+NV), as they are presented in the different tables and we have computed the CPU time ratio (Table 7) and the iteration ratio (Table 8) of the different solvers over the solver SupportSav (Solver1), where for each test problem, we have

$$\begin{aligned} \text{cputr}_{1j} &= \frac{\text{cput}(\text{Solver}j)}{\text{cput}(\text{Solver1})}, \quad j = \overline{2,3}, \\ \text{nitr}_{1j} &= \frac{\text{nit}(\text{Solver}j)}{\text{nit}(\text{Solver1})}, \quad j = \overline{2,4}. \end{aligned} \tag{4.2}$$

The above ratios (see [46]) indicate how many times SupportSav is better than the other solvers. Ratios greater than one mean that our method is more efficient for the considered problem: let S be one of the solvers 2, 3, and 4. If $\text{cputr}_{1S} \geq 1$, (resp., $\text{nitr}_{1S} \geq 1$), then our solver (Solver1) is competitive with SolverS in terms of CPU time, (resp., in terms of iterations number). Ratios greater or equal to one are mentioned with the bold character in Tables 7 and 8.

We plot the CPU time ratios of the solvers SupportFab and SimplexFab over SupportSav (Figures 1(a) and 1(c)), and the iteration ratios of each solver over SupportSav (Figures 1(b), 1(d), and 1(e)). Ratios greater than one correspond to the points which are above the line $y = 1$ in the graphs of Figure 1.

Table 6: The number of artificial variables added in Phase one of the three Solvers.

LP	Solver1	Solver2	Solver3	LP	Solver1	Solver2	Solver3
afiro	7	8	8	fffff800	304	384	384
kb2	7	16	16	grow22	17	440	440
sc50a	19	20	20	standata	121	171	171
sc50b	15	20	20	scsd6	20	147	147
adlittle	9	16	16	standmps	229	279	279
blend	33	43	43	standgub	122	173	173
share2b	10	13	13	scfxm2	247	374	374
sc105	41	45	45	scrs8	65	386	386
stocfor1	50	63	63	gfrd-pnc	89	548	548
scagr7	47	91	91	bnl1	171	239	239
israel	1	8	8	ship04s	355	354	354
share1b	62	89	89	fit1p	1	627	627
vtpbase	40	81	81	modszk1	112	687	687
sc205	82	91	91	shell	151	534	534
beaconfd	100	140	140	scfxm3	370	561	561
grow7	17	140	140	25fv47	417	520	520
lotfi	34	95	95	ship04l	355	354	354
brandy	127	166	166	wood1p	234	244	244
e226	28	46	46	sctap2	1	521	521
bore3d	103	214	214	ganges	678	1284	1284
capri	125	170	170	scsd8	45	397	397
agg	1	63	63	ship08s	699	698	698
scorpion	245	333	333	ship12s	1046	1045	1045
bandm	138	305	305	sctap3	1	682	682
sctap1	1	154	154	stocfor2	889	1143	1143
scfxm1	124	187	187	czprob	870	890	890
agg2	1	60	60	ship08l	699	698	698
agg3	3	60	60	bnl2	864	1335	1335
stair	86	301	301	ship12l	1046	1045	1045
scsd1	13	77	77	d2q06c	558	1533	1533
grow15	17	300	300	woodw	608	1089	1089
scagr25	173	325	325	truss	62	1000	1000
fit1d	2	1	1	fit2d	2	1	1
finnis	24	129	129	maros-r7	1	3136	3136
Sum					13234	27389	27389

The analysis of the different tables and graphs leads us to make the following observations.

(i) In terms of iterations number, SupportSav is competitive with SupportFab in solving 74% of the test problems with an average iteration ratio of 1.33. In terms of CPU time, SupportSav is competitive with SupportFab in solving 66% of the test problems with an average CPU time ratio of 1.20. Particularly, for the LP "scagr25" (Problem number 32), SupportSav is 5.14 times faster than SupportFab in terms of iterations number and solves the problem 4.55 times faster than SupportFab in terms of CPU time.

Table 7: CPU time ratio of the different solvers over SupportSav.

LP\Ratio	$cputr_{12}$	$cputr_{13}$	LP\Ratio	$cputr_{12}$	$cputr_{13}$
afiro	**1,000**	**1,000**	fffff800	0,943	**1,133**
kb2	**1,200**	**1,000**	grow22	**1,330**	**1,252**
sc50a	**1,000**	**1,000**	standata	0,557	**2,489**
sc50b	0,400	0.600	scsd6	**1,282**	**1,120**
adlittle	**1,125**	0,750	standmps	0,631	**2,268**
blend	**1,200**	**1,200**	standgub	0,612	**2,780**
share2b	0,727	0,727	scfxm2	**1,284**	**1,214**
sc105	**1,000**	**1,333**	scrs8	**1,361**	**1,185**
stocfor1	0,833	0,833	gfrd-pnc	**1,065**	**1,045**
scagr7	**1,727**	**2,091**	bnl1	0,815	0,803
israel	0,610	0,576	ship04s	**1,160**	**1,174**
share1b	0,868	0,868	fit1p	**3,711**	**3,565**
vtpbase	0,990	**2,942**	modszk1	**1,258**	**1,240**
sc205	0,957	0,957	shell	**1,264**	**2,145**
beaconfd	0,739	0,739	scfxm3	**1,129**	**1,147**
grow7	**1,333**	**1,259**	25fv47	0,803	0,660
lotfi	**1,037**	0,815	ship04l	**1,486**	**1,468**
brandy	**1,138**	**1,293**	wood1p	0,965	0,981
e226	**1,056**	**1,000**	sctap2	**1,278**	**1,294**
bore3d	**1,435**	**2,261**	ganges	**1,106**	**1,149**
capri	0,867	**3,990**	scsd8	**2,137**	**1,791**
agg	0,864	0,909	ship08s	**1,296**	**1,305**
scorpion	**1,141**	**1,141**	ship12s	**1,142**	**1,164**
bandm	**2,155**	**2,068**	sctap3	**1,339**	**1,312**
sctap1	0,967	**1,033**	stocfor2	0,649	0,667
scfxm1	**1,114**	**1,029**	czprob	0,761	0,851
agg2	0,737	0,789	ship08l	**1,778**	**1,745**
agg3	0,805	0,732	bnl2	**1,403**	**1,490**
stair	**1,632**	**2,090**	ship12l	**1,657**	**1,653**
scsd1	**1,226**	**1,097**	d2q06c	**1,002**	0,839
grow15	**1,244**	**1,195**	woodw	0,924	0,830
scagr25	**4,551**	**3,461**	truss	**1,179**	0,984
fit1d	**1,048**	0,911	fit2d	**1,021**	0,945
finnis	**1,000**	**1,050**	maros-r7	**1,367**	**1,317**
Mean				1,20	1,35

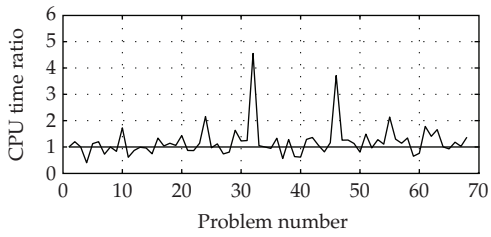
(ii) In terms of iterations number, SupportSav is competitive with SimplexFab in solving 79% of the test problems with an average iteration ratio of 1.57. In terms of CPU time, SupportSav is competitive with SimplexFab in solving 68% of the test problems with an average CPU time ratio of 1.35. The peaks of the graph ratios correspond to the problems where our approach is 2 up to 5 times faster than SimplexFab. Particularly, for the LP "capri" (Problem number 21), SupportSav is 5.29 times faster than SimplexFab in terms of iterations number and solves the problem 3.99 times faster than SimplexFab in terms of CPU time.

Table 8: Iteration ratio of the different solvers over SupportSav.

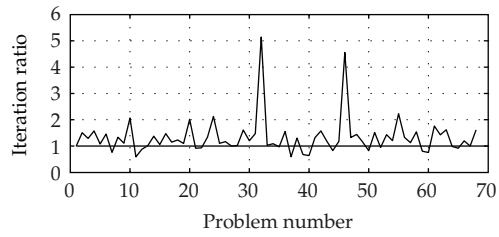
LP\Ratio	$nitr_{12}$	$nitr_{13}$	$nitr_{14}$	LP\Ratio	$nitr_{12}$	$nitr_{13}$	$nitr_{14}$
afiro	**1,000**	**1,000**	0,944	fffff800	0,972	**1,125**	**1,037**
kb2	**1,505**	**1,316**	**1,011**	grow22	**1,556**	**1,490**	**1,493**
sc50a	**1,289**	**1,342**	**1,316**	standata	0,594	**3,283**	0,274
sc50b	**1,576**	**1,576**	**1,545**	scsd6	**1,304**	**1,233**	**1,398**
adlittle	**1,076**	**1,042**	**1,458**	standmps	0,675	**3,038**	0,427
blend	**1,457**	**1,444**	**1,593**	standgub	0,634	**3,545**	0,283
share2b	0,759	0,771	**1,088**	scfxm2	**1,318**	**1,192**	0,946
sc105	**1,333**	**1,333**	**1,250**	scrs8	**1,573**	**1,379**	**1,655**
stocfor1	**1,108**	**1,108**	**1,676**	gfrd-pnc	**1,179**	**1,114**	0,963
scagr7	**2,065**	**2,304**	**1,333**	bnl1	0,829	0,794	0,976
israel	0,587	0,591	**1,029**	ship04s	**1,176**	**1,174**	**1,066**
share1b	0,876	0,851	0,842	fit1p	**4,557**	**4,459**	**4,269**
vtpbase	**1,012**	**4,191**	0,791	modszk1	**1,322**	**1,293**	0,987
sc205	**1,373**	**1,325**	**1,479**	shell	**1,442**	**2,870**	**1,436**
beaconfd	**1,056**	**1,056**	0,683	scfxm3	**1,153**	**1,132**	0,918
grow7	**1,472**	**1,384**	**1,432**	25fv47	0,823	0,690	0,982
lotfi	**1,145**	**1,031**	**1,298**	ship04l	**1,514**	**1,505**	**1,407**
brandy	**1,228**	**1,234**	**2,636**	wood1p	0,947	0,947	0,782
e226	**1,097**	**1,033**	**1,317**	sctap2	**1,431**	**1,472**	**1,443**
bore3d	**2,014**	**3,471**	**1,379**	ganges	**1,203**	**1,210**	**1,151**
capri	0,915	**5,286**	**1,127**	scsd8	**2,227**	**1,984**	**9,588**
agg	0,929	**1,047**	**4,591**	ship08s	**1,327**	**1,322**	0,828
scorpion	**1,339**	**1,320**	**1,222**	ship12s	**1,129**	**1,128**	0,954
bandm	**2,125**	**1,986**	**1,651**	sctap3	**1,538**	**1,519**	**1,507**
sctap1	**1,103**	**1,100**	**1,056**	stocfor2	0,803	0,793	**1,609**
scfxm1	**1,166**	**1,058**	0,955	czprob	0,754	0,897	**1,015**
agg2	**1,000**	0,988	**3,533**	ship08l	**1,760**	**1,754**	**1,124**
agg3	**1,017**	**1,006**	**3,562**	bnl2	**1,427**	**1,461**	**1,084**
stair	**1,609**	**2,104**	**1,335**	ship12l	**1,619**	**1,617**	**1,309**
scsd1	**1,202**	**1,255**	**1,324**	d2q06c	0,986	0,847	0,860
grow15	**1,471**	**1,424**	**1,723**	woodw	0,916	0,849	0,726
scagr25	**5,143**	**3,775**	**2,169**	truss	**1,198**	**1,065**	**1,520**
fit1d	**1,029**	0,995	**1,539**	fit2d	0,999	0,996	**1,048**
finnis	**1,090**	**1,197**	**1,005**	maros-r7	**1,613**	**1,574**	**1,286**
Mean					1,33	1,57	1,49

(iii) In terms of iterations number, SupportSav is competitive with LP_SOLVE_PSA in solving 72% of the test problems with an average iteration ratio of 1.49 (the majority of iteration ratios are above the line $y = 1$ in Figure 1(e)). Particularly, for the LP "scsd8" (Problem number 55), our method is 9.58 times faster than the primal simplex implementation of the open-source solver LP_SOLVE.

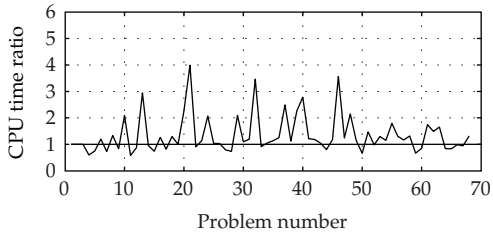
(iv) We remark from the last row of Table 6 that the total number of artificial variables added in order to find the initial support for SupportSav (13234) is considerably less than the total number of artificial variables added for SimplexFab (27389) and SupportFab (27389).



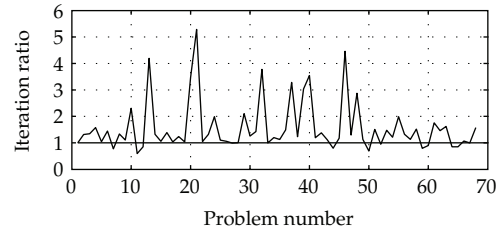
(a) Ratio of the CPU time of SupportFab over the CPU time of SupportSav (cputr_{12})



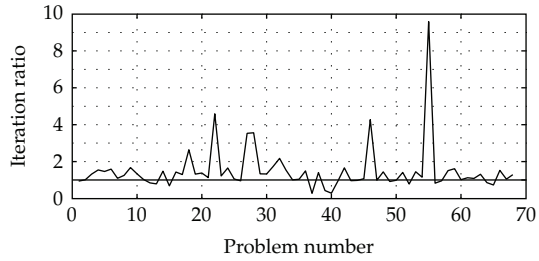
(b) Ratio of the iterations number of SupportFab over the iterations number of SupportSav (nitr_{12})



(c) Ratio of the CPU time of SimplexFab over the CPU time of SupportSav (cputr_{13})



(d) Ratio of the iterations number of SimplexFab over the iterations number of SupportSav (nitr_{13})



(e) Ratio of the iterations number of LP_SOLVE_PSA over the iterations number of SupportSav (nitr_{14})

Figure 1: Ratios of the different solvers over SupportSav.

(v) If we compute the total number of iterations necessary to solve the 68 LPs for each solver, we find (143737) for SupportSav, (159306) for SupportFab, (163428) for SimplexFab, and (158682) for LP_SOLVE_PSA. Therefore, the total number of iterations is the smallest for our solver.

5. Conclusion

In this work, we have proposed a single artificial variable technique to initialize the primal support method with bounded variables. An implementation under the MATLAB environnement has been developed. In the implementation, we have used the *LU* factorization of the basic matrix to solve the linear systems and the Sherman-Morrison-Woodbury formula to update the *LU* factors. After that, we have compared our approach (SupportSav) to the full artificial basis support method (SupportFab), the full artificial basis simplex method (SimplexFab), and the primal simplex implementation of the open-source solver LP_SOLVE (LP_SOLVE_PSA). The obtained numerical results are encouraging.

Indeed, the suggested method (SupportSav) is competitive with SupportFab, SimplexFab, and LP_SOLVE_PSA.

Note that during our experiments, we have remarked that the variation of the initial support and the initial point x^+ affects the performances (CPU time and iterations number) of the single artificial variant of the support method. Thus, how to choose judiciously the initial point and the initial support in order to improve the efficiency of the support method? In future works, we will try to apply some crash procedure like those proposed in [25, 27] in order to initialize the support method with a good initial support. Furthermore, we will try to implement some modern sparse algebra techniques to update the *LU* factors [47].

References

[1] L. V. Kantorovich, *Mathematical Methods in the Organization and Planning of Production*, Publication House of the Leningrad State University, 1939, Translated in Management Science, vol. 6, pp. 366–422, 1960.

[2] G. B. Dantzig, "Maximization of a linear function of variables subject to linear inequalities," in *Activity Analysis of Production and Allocation*, R. C. Koopmans, Ed., pp. 339–347, Wiley, New York, NY, USA, 1951.

[3] G. B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, USA, 1963.

[4] V. Klee and G. J. Minty, "How good is the simplex algorithm?" in *Inequalities III*, O. Shisha, Ed., pp. 159–175, Academic Press, New York, NY, USA, 1972.

[5] R. Gabasov and F. M. Kirillova, *Methods of Linear Programming*, vol. 1–3, The Minsk University, 1977, 1978 and 1980.

[6] S. Radjef and M. O. Bibi, "An effective generalization of the direct support method," *mathematical Problems in Engineering*, vol. 2011, Article ID 374390, 18 pages, 2011.

[7] R. Gabasov, F. M. Kirillova, and S. V. Prischepova, *Optimal Feedback Control*, Springer, London, UK, 1995.

[8] R. Gabasov, F. M. Kirillova, and O. I. Kostyukova, "Solution of linear quadratic extremal problems," *Soviet Mathematics Doklady*, vol. 31, pp. 99–103, 1985.

[9] R. Gabasov, F. M. Kirillova, and O. I. Kostyukova, "A method of solving general linear programming problems," *Doklady AN BSSR*, vol. 23, no. 3, pp. 197–200, 1979 (Russian).

[10] R. Gabasov, F. M. Kirillova, and A. I. Tyatyushkin, *Constructive Methods of Optimization. Part1. Linear Problems*, Universitetskoje, Minsk, Russia, 1984.

[11] E. A. Kostina and O. I. Kostyukova, "An algorithm for solving quadratic programming problems with linear equality and inequality constraints," *Computational Mathematics and Mathematical Physics*, vol. 41, no. 7, pp. 960–973, 2001.

[12] E. Kostina, "The long step rule in the bounded-variable dual simplex method: numerical experiments," *Mathematical Methods of Operations Research*, vol. 55, no. 3, pp. 413–429, 2002.

[13] M. O. Bibi, "Support method for solving a linear-quadratic problem with polyhedral constraints on control," *Optimization*, vol. 37, no. 2, pp. 139–147, 1996.

[14] B. Brahmi and M. O. Bibi, "Dual support method for solving convex quadratic programs," *Optimization*, vol. 59, no. 6, pp. 851–872, 2010.

[15] M. Bentobache and M. O. Bibi, "Two-phase adaptive method for solving linear programming problems with bounded variables," in *Proceedings of the YoungOR 17*, pp. 50–51, University of Nottingham, UK, 2011.

[16] M. Bentobache and M. O. Bibi, "Adaptive method with hybrid direction: theory and numerical experiments," in *Proceedings of the Optimization*, pp. 24–27, Universidade Nova de Lisboa, Portugal, 2011.

[17] M. O. Bibi and M. Bentobache, "The adaptive method with hybrid direction for solving linear programming problems with bounded variables," in *Proceedings of the colloque International sur l'Optimisation et les Systèmes d'Information (COSI' 11)*, pp. 80–91, University of Guelma, Algeria, 2011.

[18] M. O. Bibi and M. Bentobache, "An hybrid direction algorithm for solving linear programs," in *Proceedings of the International Conference on Discrete Mathematics & Computer Science (DIMACOS'11)*, pp. 28–30, University of Mohammedia, Morocco, 2011.

[19] L. G. Khachian, "A polynomial algorithm for linear programming," *Soviet Mathematics Doklady*, vol. 20, pp. 191–194, 1979.

[20] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorica*, vol. 4, no. 4, pp. 373–395, 1984.

[21] S. I. Gass, *Linear Programming: Methods and Applications*, McGraw-Hill, New York, NY, USA, 1964.

[22] C. M. Shetty, "A simplified procedure for quadratic programming," *Operations Research*, vol. 11, pp. 248–260, 1963.

[23] C. B. Millham, "Fast feasibility methods for linear programming," *Opsearch*, vol. 13, pp. 198–204, 1976.

[24] P. Wolfe, "The Composite Simplex Algorithm," *SIAM Review*, vol. 7, no. 1, pp. 42–54, 1965.

[25] R. E. Bixby, "Implementing the simplex method: the initial basis," *ORSA Journal on Computing*, vol. 4, no. 3, pp. 1–18, 1992.

[26] N. I. M. Gould and J. K. Reid, "New crash procedures for large systems of linear constraints," *Mathematical Programming*, vol. 45, no. 1–3, pp. 475–501, 1989.

[27] I. Maros and G. Mitra, "Strategies for creating advanced bases for large-scale linear programming problems," *INFORMS Journal on Computing*, vol. 10, no. 2, pp. 248–260, 1998.

[28] B. A. Murtagh and M. A. Saunders, "MINOS 5.5 User's Guide," Tech. Rep. SOL 83–20, Systems Optimization Lab., Stanford University, Stanford, Calif, USA, 1998.

[29] W. Orchard-Hays, *Advanced Linear-Programming Computing Techniques*, McGraw-Hill, New York, NY, USA, 1968.

[30] S. Lim and S. Park, "LPAKO: a simplex-based linear programming program," *Optimization Methods and Software*, vol. 17, no. 4, pp. 717–745, 2002.

[31] H. D. Sherali, A. L. Soyster, and S. G. Baines, "Nonadjacent extreme point methods for solving linear programs," *Naval Research Logistics Quarterly*, vol. 30, no. 1, pp. 145–161, 1983.

[32] M. Bentobache, *A new method for solving linear programming problems in canonical form and with bounded variables*, M.S. thesis, University of Bejaia, Algeria, 2005.

[33] M. Bentobache and M. O. Bibi, "Two-phase support method for solving linear programming problems with nonnegative variables: numerical experiments," in *Proceedings of the Colloque International sur l'Optimisation et les Systèmes d'Information (COSI'08)*, pp. 314–325, University of Tizi Ouzou, Algeria, 2008.

[34] M. Bentobache and M. O. Bibi, "Two-phase support method for solving linear programming problems with bounded variables: numerical experiments," in *Proceedings of the Colloque International sur l'Optimisation et les Systèmes d'Information (COSI'09)*, pp. 109–120, University of Annaba, Algeria, 2009.

[35] G. Mitra, M. Tamiz, and J. Yadegar, "Experimental investigation of an interior search method within a simplex framework," *Communications of the ACM*, vol. 31, no. 12, pp. 1474–1482, 1988.

[36] R. J. Vanderbei, *Linear Programming: Foundations and Extensions*, Kluwer Academic Publichers, Princeton University, 2001.

[37] ILOG CPLEX, "9.0 User's Manual," 2003, http://www.ilog.com.

[38] "LP_SOLVE," http://sourceforge.net/projects/lpsolve/files/lpsolve.

[39] A. Makhorin, "GNU linear programming Kit," Reference Manual Version 4.9, Draft Edition, 2006, http://www.gnu.org/software/glpk/glpk.html .

[40] P. G. García and Á. Santos-Palomo, "A deficient-basis dual counterpart of Paparrizos, Samaras and Stephanides's primal-dual simplex-type algorithm," in *Proceedings of the 2nd Conference on Optimization Methods & Software and 6th EUROPT Workshop on Advances in Continuous Optimization*, Prague, Czech Republic, 2007.

[41] S. R. Thorncraft, H. R. Outhred, and D. J. Clements, "Evaluation of open-source LP optimization codes in solving electricity spot market optimization problems," in *Proceedings of the 19th Mini-Euro Conference on Operations Research Models and Methods in the Energy Sector*, Coimbra, Portugal, 2006.

[42] M. C. Ferris, O. L. Mangasarian, and S. J. Wright, "Linear programming with MATLAB," *MPSSIAM Series on Optimization*, 2007.

[43] R. H. Bartels and G. H. Golub, "The simplex method of linear programming using LU decomposition," *Communications of the ACM*, vol. 12, no. 5, pp. 266–268, 1969.

[44] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright, "A practical anti-cycling procedure for linearly constrained optimization," *Mathematical Programming*, vol. 45, no. 1-3, pp. 437–474, 1989.

[45] M. Gay, "Electronic mail distribution of linear programming test problems," Mathematical Programming Society COAL, Bulletin no. 13, 1985, http://www.netlib.org/lp/data.

[46] K. Paparrizos, N. Samaras, and G. Stephanides, "An efficient simplex type algorithm for sparse and dense linear programs," *European Journal of Operational Research*, vol. 148, no. 2, pp. 323–334, 2003.

[47] J. J. H. Forrest and J. A. Tomlin, "Updated triangular factors of the basis to maintain sparsity in the product form simplex method," *Mathematical Programming*, vol. 2, no. 1, pp. 263–278, 1972.