Applied Mathematics, 2012, 3, 1423-1429
http://dx.doi.org/10.4236/am.2012.330200 Published Online October 2012 (http://www.SciRP.org/journal/am)

A Modified Centered Climbing Algorithm for Linear Programming

Ming-Fang Ding, Yanqun Liu, John Anthony Gear
School of Mathematical & Geospatial Sciences, RMIT University, Melbourne, Australia
Email: mingfang_ding@yahoo.cn

Received July 9, 2012; revised August 9, 2012; accepted August 16, 2012

ABSTRACT

In this paper, we propose a modified centered climbing algorithm (MCCA) for linear programs, which improves the centered climbing algorithm (CCA) developed for linear programs recently. MCCA implements a specific climbing scheme where a violated constraint is probed by means of the centered vector used by CCA. Computational comparison is made with CCA and the simplex method. Numerical tests show that, on average CPU time, MCCA runs faster than both CCA and the simplex method in terms of tested problems. In addition, a simple initialization technique is introduced.

Keywords: Linear Programming; Ladder Method; Climbing Rules; Simplex Method

1. Introduction

In the last decades a variety of algorithms have been proposed for solving linear programming (LP) problems (see, e.g., [1-5]). However, so far there hasn't been an available single best LP algorithm which is suitable for solving all types of LP problems. Considerable research is still under way to find a faster LP algorithm.

Recently, a ladder method was developed in [5] for solving general LP problems. In this method, the inclusive normal cone is updated at each iteration by climbing in the associated inclusive region (ladder) until the problem is solved. A climbing rule used to update the inclusive normal cone is of crucial importance, directly determining the performance of a ladder algorithm. The climbing rule involves picking up a violated constraint and dropping a constraint from the current inclusive cone. Ladder algorithms applying various climbing rules were proposed in [5,6]. In this paper, we present a new ladder algorithm called "the Modified Centered Climbing Algorithm (MCCA)", where a specific climbing rule is employed by means of the centered vector used by CCA [5]. At each iteration, a violated constraint is selected whose associated outer normal vector forms the minimum angle with the centered vector. Computational results show that, the proposed ladder algorithm has surprising superiority to CCA as well as the simplex method in terms of average CPU time for randomly generated test problems. In addition, the single artificial constraint technique is presented to initialize the ladder method for a certain class of LP problems.

The paper is organized as follows. In the remaining of this section, for the sake of readability, we introduce concepts used in the ladder method. A new ladder algorithm is proposed in Section 2, followed by an initialization technique of the ladder method in Section 3. In Section 4, a specific example is provided to illustrate effectiveness of the new algorithm. We report computational results in Section 5, followed by a brief conclusion in Section 6.

Consider the following linear programming problem:

$$(\mathrm{P}): \quad \min \ c^T x \qquad \text{s.t.} \ \ Ax \le b$$

where $x \in R^n$ is decision variables, $A \in R^{m\times n}$ with $m \ge n$, $c = [c_1; c_2; \cdots; c_n] \in R^n$ ($c \ne 0$), and $b = [b_1; b_2; \cdots; b_m] \in R^m$. Here, square brackets with entries separated by semi-colons indicate column vectors. Throughout the paper, we assume that rank(A) = n.

We denote the constraint index set $\{1, 2, \cdots, m\}$ by $\mathcal{J}$. Let J be an ordered subset of $\mathcal{J}$. Denote by $J(i \leftrightarrow j)$ the ordered subset with i-th entry of J replaced by $j \in \mathcal{J}/J$. The j-th row of A is denoted by a_j. We denote by $A(J)$ the $k \times n$ submatrix with its j-th row as the i_j-th row of A. Also, denote by $b(J)$ the k-vector with j-th element of $b(J)$ as the i_j-th element of b. In addition, $\|\cdot\|$ denotes the Euclidean norm.

Before proceeding, we present the following definitions developed in [5], which will be used in this paper.

Definition 1 [5] *Consider problem* (*P*). *Let*

$J=\{j_1,j_2,\cdots,j_n\}\subset\mathcal{J}$ be an ordered subset. A convex cone generated by n linearly independent vectors $a_{j_1}^T,a_{j_2}^T,\cdots,a_{j_n}^T$, where $a_{j_l}\ (1\le l\le n)$ are rows of A, is said to be an inclusive normal cone generated by J if it contains the vector –c, where c is the objective vector. The generated cone is denoted by N(J). If J generates an inclusive cone, the set defined by

$$L(J)=\{x\in R^n : a_j x\le b_j, for\ j\in J\}$$

is called the inclusive region or the ladder associated with J. The corresponding ordered index set J is called the generator of L(J), and the unique solution of A(J)x = b(J), denoted by x_J, is called the base point of the ladder L(J).

According to Theorem 2.2 in [5], problem (P) has an optimal solution (if an optimal solution exists) if and only if it has a feasible base point. A feasible base point is exactly an optimal solution.

Definition 2 [5] *A ladder L(J) of problem (P) is said to be degenerate if at least one of its n edges is normal to the objective vector c. Problem (P) is said to be non-degenerate if it does not have a degenerate ladder.*

Throughout the paper, we assume that problem (P) is non-degenerate since it is shown in [5] that the degenerate case can be readily treated by imposing an appropriate perturbation on the objective function of the original problem without affecting the optimal solution of the original problem. On the basis of the above assumption, we give the following ladder algorithm.

2. The Modified Centered Climbing Algorithm (MCCA)

Step 0 Initialization.

Start with a known ladder generator, which is denoted by $J_0=\{j_1^0,j_2^0,\cdots,j_n^0\}\subset\mathcal{J}$ (Refer to [5] or the subsequent section for how to find such a generator if it is not immediately available). Denote by $x^0=x_{J_0}$ the base point associated with J_0. Calculate the initial base point

$$x^0=\left[A(J_0)\right]^{-1}b(J_0).$$

Set $k=0$ and $D_{k-1}=\varnothing$.

Step 1 Check optimality.

Let $V^k=\{j\in\mathcal{J}\setminus(J_k\bigcup D_{k-1}) : a_j x^k>b_j\}$.

- If $V^k=\varnothing$, exit with "the problem attains optimality".
- Otherwise, go to Step 2.

Step 2 Updating the ladder.

2.1 Picking up a new index as a pick.

Let $$v_{J_k}=-\left[A(J_k)\right]^{-1}1_{n\times 1},$$

where $1_{n\times 1}=[1;1;\cdots;1]\in R^n$, and v_{J_k} is the center vector of the current ladder $L(J_k)$ [5]. Select $p^k\in V^k$ as a pick such that

$$t_{p^k}=\max_{j\in V^k}\{t_j\},$$

where

$$t_j=\frac{a_j v_{J_k}}{\|a_j\|}. \tag{1}$$

2.2 Try to find an index $j_d^k\in J_k$ as a drop such that $J_{k+1}=J_k(j_d^k\leftrightarrow p^k)$ is a ladder generator and the associated base point $x^{k+1}\in L(J_k)$ (See Procedure 2 in [5] for how to identify).

- If such an index does not exist, exit with "the problem is infeasible".
- Otherwise, go to next step.

2.3 Let $J_{k+1}=J_k(j_d^k\leftrightarrow p^k)$ and $D_k=\{j_d^k\}$. Calculate the updated base point

$$x^{k+1}=\left[A(J_{k+1})\right]^{-1}b(J_{k+1}).$$

Set $k:=k+1$. Go to Step 1.

Note that existence of an initial ladder (generator) implies that the case of unboundedness can not occur (Refer to Theorem 2.5 (d) in [5] for details). Step 2.1 constitutes the main difference with respect to the previous ladder algorithms. At each iteration, a violated constraint is selected as a pick whose associated outer normal vector forms the minimum angle with the centered vector. Before numerically examining its efficiency, we would like to introduce the single artificial constraint technique to construct an initial ladder for a certain class of LP problems.

3. Finding an Initial Ladder for LP Problems with Bounded Variables

An initial ladder is required to get the ladder algorithm started. To find a ladder $L(J)$ is to find the associated generator $J=\{j_1,j_2,\cdots,j_n\}\subset\mathcal{J}$, equivalently, to find n independent outward normals a_{j_k} (k = 1, 2, ⋯, n) such that there exist n constants $\lambda_{j_k}\ge 0$ (k = 1, 2, ⋯, n) satisfying

$$-c=\sum_{k=1}^n\lambda_{j_k}a_{j_k}^T.$$

Various approaches were developed in [5] to obtain an initial ladder. In the following, we present an initialization technique for LP problems involving bounded variables.

3.1. Variables with Upper Bounds

In this subsection, we consider the case where variables of problem (P) have upper bounds. Temporarily, we assume that at least one component of c is positive. For

convenience of discussion, write the problem as below:

$$\begin{aligned}
\text{(P1)}\qquad \min\ & c_1x_1 + c_2x_2 + \cdots + c_dx_d + \cdots + c_nx_n \\
\text{s.t.}\ & a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \le b_1 \\
& a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \le b_2 \\
& \cdots \\
& a_{(m-n)\times 1}x_1 + a_{(m-n)\times 2}x_2 + \cdots + a_{(m-n)\times n}x_n \le b_{m-n} \\
& x_1 \le b_{m-n+1} \\
& x_2 \le b_{m-n+2} \\
& \cdots \\
& x_d \le b_{m-n+d} \\
& \cdots \\
& x_n \le b_m
\end{aligned}$$

With this assumption that c contains at least one positive component, it is easily seen that the index set $\{m-n+1, m-n+2, \cdots, m-n+d, \cdots, m\}$ is not a ladder generator. In order to obtain a ladder for the above problem, we add an artificial constraint

$$-\sum_{i=1}^{n} x_i \le M\ ,$$

where M is a sufficiently large number. For clarity, display the problem with the additional constraint as below:

$$\begin{aligned}
\min\ & c_1x_1 + c_2x_2 + \cdots + c_dx_d + \cdots + c_nx_n \\
\text{s.t.}\ & a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \le b_1 \\
& a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \le b_2 \\
& \cdots \\
& a_{(m-n)\times 1}x_1 + a_{(m-n)\times 2}x_2 + \cdots + a_{(m-n)\times n}x_n \le b_{m-n} \\
& x_1 \le b_{m-n+1} \\
& x_2 \le b_{m-n+2} \\
& \cdots \\
& x_d \le b_{m-n+d} \\
& \cdots \\
& x_n \le b_m \\
& -x_1 - x_2 - \cdots - x_n \le M
\end{aligned}$$

Executing the following simple procedure, we can readily find an initial ladder for the above problem. At start, take $J = \{m-n+1, m-n+2, \cdots, m-n+d, \cdots, m\}$. Let $c_d = \max\{c_i\} > 0$ $(1 \le d \le n)$. Take $j = m-n+d$ as a drop (the associated constraint is $-x_d \le b_{m-n+d}$) and $p = m+1$ a pick (the associated constraint is $-x_1 - x_2 - \cdots - x_n \le M$). It is easy to verify that $J(j \leftrightarrow p)$ is a ladder generator of the above problem. Indeed, from

$$-\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{d-1} \\ c_d \\ c_{d+1} \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 & -1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & -1 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 1 & -1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & -1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & -1 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 0 & -1 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_{d-1} \\ \lambda_d \\ \lambda_{d+1} \\ \vdots \\ \lambda_n \end{bmatrix}$$

we have

$$\lambda_i = c_d - c_i \ge 0\,(i \ne d),\quad \lambda_d = c_d > 0$$

which implies $J(j \leftrightarrow p)$ is a ladder generator of the above problem.

3.2. Variables with Lower Bounds

In this subsection, we consider the case where variables of problem (P) have lower bounds. For convenience of discussion, we rewrite the problem in the following form:

$$\begin{aligned}
\text{(P2)}\qquad \min\ & c_1x_1 + c_2x_2 + \cdots + c_dx_d + \cdots + c_nx_n \\
\text{s.t.}\ & a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \le b_1 \\
& a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \le b_2 \\
& \cdots \\
& a_{(m-n)\times 1}x_1 + a_{(m-n)\times 2}x_2 + \cdots + a_{(m-n)\times n}x_n \le b_{m-n} \\
& -x_1 \le b_{m-n+1} \\
& -x_2 \le b_{m-n+2} \\
& \cdots \\
& -x_d \le b_{m-n+d} \\
& \cdots \\
& -x_n \le b_m
\end{aligned}$$

Note that here we use the same notations in problem (P2) as in (P1) for convenience. In this subsection, we temporarily assume that c contains at least one negative component. With this assumption, it is easy to be seen that the index set $\{m-n+1, m-n+2, \cdots, m-n+d, \cdots, m\}$ is not a ladder generator. Adding an artificial constraint $\sum_{i=1}^{n} x_i \le M$, we get the following system:

$$\begin{aligned}
\min\ & c_1x_1 + c_2x_2 + \cdots + c_dx_d + \cdots + c_nx_n \\
\text{s.t.}\ & a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \le b_1 \\
& a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \le b_2 \\
& \cdots \\
& a_{(m-n)\times 1}x_1 + a_{(m-n)\times 2}x_2 + \cdots + a_{(m-n)\times n}x_n \le b_{m-n} \\
& -x_1 \le b_{m-n+1}
\end{aligned}$$

$$-x_2 \le b_{m-n+2}$$

$$\cdots$$

$$-x_d \le b_{m-n+d}$$

$$\cdots$$

$$-x_n \le b_m$$

$$x_1 + x_2 + \cdots + x_n \le M$$

Performing the similar procedure as the above subsection, we can easily obtain an initial ladder for the above problem. Initially, take $J = \{m - n + 1, m - n + 2, \cdots, m - n + d, \cdots, m\}$. Let $c_d = \min\{c_i\} < 0$ $(1 \le d \le n)$. Take $j = m - n + d$ as a drop (the associated constraint is $-x_d \le b_{m-n+d}$) and $p = m + 1$ as a pick (the associated constraint is $x_1 + x_2 + \cdots + x_n \le M$). It is easy to verify that $J(j \leftrightarrow p)$ is a ladder generator of the above problem. In fact, from

$$-\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{d-1} \\ c_d \\ c_{d+1} \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} -1 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ 0 & -1 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & -1 & 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 1 & -1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & -1 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_{d-1} \\ \lambda_d \\ \lambda_{d+1} \\ \vdots \\ \lambda_n \end{bmatrix}$$

we have

$$\lambda_i = -c_d + c_i \ge 0 \,(i \ne d), \quad \lambda_d = -c_d > 0$$

which implies $J(j \leftrightarrow p)$ is a ladder generator of the above problem.

If variables in an LP problem are bounded from both below and above, that is, an LP problem contains n constraints taking the form of $l_i \le x_i \le u_i$ $(1 \le i \le n)$, where l_i and u_i denote the lower and upper bounds of x_i and $l_i < u_i$, then after rewriting the above constraints as two constraints $-x_i \le -l_i$ and $x_i \le u_i$ we can follow the procedure in either Subsection 3.1 or Subsection 3.2 to obtain an initial ladder for the problem.

4. A Specific Example

To illustrate the efficiency of the above ladder algorithm, we use both the simplex method and MCCA to solve a Klee-Minty problem [7,8].

Example 1 *Consider the following Klee-Minty problem with* $n = 3$

$$\min \ -100x_1 - 10x_2 - x_3$$

$$\text{s.t. } x_1 \le 1$$

$$20x_1 + x_2 \le 100$$

$$200x_1 + 20x_2 + x_3 \le 10,000$$

$$x_1, x_2, x_3 \ge 0.$$

On the one hand, we use the simplex method to solve the above problem. Introducing the slack variables s_1, s_2, $s_3 \ge 0$, write the above problem as the standard form

$$\min \ -100x_1 - 10x_2 - x_3$$

$$\text{s.t. } x_1 + s_1 = 1$$

$$20x_1 + x_2 + s_2 = 100$$

$$200x_1 + 20x_2 + x_3 + s_3 = 10,000$$

$$x_1, x_2, x_3, s_1, s_2, s_3 \ge 0$$

The tableau in **Table 1** shows that the simplex method with the most negative rule requires $2^n - 1 = 2^3 - 1 = 7$ iterations to attain the optimality.

On the other hand, we solve the same problem using MCCA. Firstly we rewrite all constraints as ≤–type:

$$\min \ -100x_1 - 10x_2 - x_3$$

$$\text{s.t. } x_1 \le 1$$

$$20x_1 + x_2 \le 100$$

$$200x_1 + 20x_2 + x_3 \le 10,000$$

$$-x_1 \le 0$$

$$-x_2 \le 0$$

$$-x_3 \le 0$$

To find an initial ladder, we add an artificial constraint $x_1 + x_2 + x_3 \le M$. For clarity, write the problem with the additional constraint as below.

$$\min \ -100x_1 - 10x_2 - x_3$$

$$\text{s.t. } x_1 \le 1$$

$$20x_1 + x_2 \le 100$$

$$200x_1 + 20x_2 + x_3 \le 10000$$

$$-x_1 \le 0$$

$$-x_2 \le 0$$

$$-x_3 \le 0$$

$$x_1 + x_2 + x_3 \le M$$

It is easy to verify that the index set {7, 5, 6} is an initial ladder generator (see Subsection 3.2). With the known ladder generator at hand, it takes only two iterations to reach an optimal solution for MCCA. For solution detail, see **Table 2**.

Clearly MCCA is much more efficient for solving the above Klee-Minty problem. Firstly, our algorithm requires no additional variables (slack, surplus and artificial variables). Secondly, the number of iterations is reduced greatly. In addition, we would like to stress that

Table 1. The tableau obtained from simplex for Example 1.

Iteration		x_1	x_2	x_3	s_1	s_2	s_3	rhs
0	z	–100	–10	–1	0	0	0	0
	s_1	1	0	0	1	0	0	1
	s_2	20	1	0	0	1	0	100
	s_3	200	20	1	0	0	1	10,000
1	z	0	–10	–1	100	0	0	100
	x_1	1	0	0	1	0	0	1
	s_2	0	1	0	–20	1	0	80
	s_3	0	20	1	–200	0	1	9800
2	z	0	0	–1	–100	10	0	900
	x_1	1	0	0	1	0	0	1
	x_2	0	1	0	–20	1	0	80
	s_3	0	0	1	200	–20	1	8200
3	z	100	0	–1	0	10	0	1000
	s_1	1	0	0	1	0	0	1
	x_2	20	1	0	0	1	0	100
	s_3	–200	0	1	0	–20	1	8000
4	z	–100	0	0	0	–10	1	9000
	s_1	1	0	0	1	0	0	1
	x_2	20	1	0	0	1	0	100
	x_3	–200	0	1	0	–20	1	8000
5	z	0	0	0	100	–10	1	9100
	x_1	1	0	0	1	0	0	1
	x_2	0	1	0	–20	1	0	80
	x_3	0	0	1	200	–20	1	8200
6	z	0	10	0	–100	0	1	9900
	x_1	1	0	0	1	0	0	1
	s_2	0	1	0	–20	1	0	80
	x_3	0	20	1	–200	0	1	9800
7	z	100	10	0	0	0	1	10,000
	s_1	1	0	0	1	0	0	1
	s_2	20	1	0	0	1	0	100
	x_3	200	20	1	0	0	1	10,000

Table 2. The table obtained from MCCA for Example 1.

Iteration	Ladder generator	Base point	Optimal value
0	{7,5,6}	$[M; 0; 0]$	
1	{7,5,3}	$\left[\frac{10,000-M}{199}; 0; \frac{-10,000+200M}{199}\right]$	
2	{4,5,3}	[0; 0; 10,000]	–10,000

although here we use an example with non-negativity variables to illustrate the efficiency of our algorithm, there is no non-negativity requirement for variables in our problem form. Thus, our algorithm is suitable for a wider range of LP problems.

5. Numerical Tests

In this section, we make computational tests to demonstrate the efficiency of MCCA. The ladder algorithms were coded in MATLAB 7.11.0. Test problems are randomly generated in the same way as in [5], which is presented as below.

Example 2 [5] (*Randomly generated feasible problem*) *Generate a linear programming problem by specifying* $A \in R^{m\times n}$, $c = [c_1; c_2; \cdots; c_n] \in R^n$, *and* $b = [b_1; b_2; \cdots; b_m] \in R^m$ *in the following method.*

1. Randomly generate $c \in R^n$ and a vector $\bar{x} \in R^n$ such that components of c take values between –25 and 25, and components of $\bar{x}$ between 0 and 20.

2. Generate A and b by two steps.

(a) For $1 \le j \le n$, the j-th row a_j of A is $a_j = -c^T + 2\text{sign}(c_j)e_j$, where e_j is the j-th row of the $n \times n$ identity matrix. Then, b_j is defined by $b_j = a_j\bar{x} + \gamma_j$, where γ_j is a random number in (0, 1).

(b) For $n + 1 \le j \le m$, randomly generate a row vector $\alpha_j \in R^n$ and a number $\beta_j \in R$ such that β_j and all the components of α_j are between –25 and 25. If $\alpha_j\bar{x} \le \beta_j$, then the j-th row a_j of A and the j-th element of b are defined by $a_j = \alpha_j$, $b_j = \beta_j$. Otherwise, they are defined by $a_j = -\alpha_j$, $b_j = -\beta_j$.

Tests were run on a desk-top computer (HP Intel(R) Core(TM), i7-2600 CPU@3.40GHz, 3.39GHz, 3.24GB of RAM) under Microsoft Windows XP operating system. For comparison, the centered climbing algorithm (CCA) [5] and the linprog solver in MATLAB optimization toolbox (Version 5.1, (R2010b)) were used for solving the same test problems. The medium-scale simplex algorithm (SP) was implemented. **Tables 3** and **4** present computational results for 20 test problems with various dimensions. The average CPU time is reported in seconds. Since our algorithm and the simplex method actually work on problems with different forms and dimensions, the number of iterations does not provide much helpful information. Therefore, here we do not take the comparison of iteration numbers into account.

Tables 3 and **4** reveal that, MCCA has the absolute advantage over CCA as well as the simplex method for tested problems. We would like to point out that in the present code we adopt the traditional technique of the inverse of matrix to calculate base points. If the advanced numerical technique was incorporated into the current code, the computational performance would promise further improvement.

Table 3. Average CPU time (in seconds) for test problems in Example 2 (*m* = 2*n*).

Size	Algorithms		
(*m*, *n*)	MCCA	CCA	SP
(40. 20)	0.0843	0.0953	0.6406
(80, 40)	0.3937	0.4375	1.2609
(120, 60)	0.4921	0.5093	2.1656
(160, 80)	0.8484	1.2281	3.9640
(200, 100)	1.4312	1.775	7.0625
(240, 120)	2.4078	2.9609	10.3703
(280, 140)	4.4765	4.7421	18.5488
(320, 160)	6.6347	7.6992	28.5644
(360, 180)	10.9609	12.3066	42.7851
(400, 200)	16.1437	16.9937	61.7343

Table 4. Average CPU time (in seconds) for test problems in Example 2 (*m* – *n* = 100).

Size	Algorithms		
(*m*, *n*)	MCCA	CCA	SP
(600, 500)	35.8554	40.8424	126.6692
(650, 550)	44.75	57.9263	148.9732
(700, 600)	52.25	72.5273	185.5312
(750, 650)	62.2291	73.6718	225.5625
(800, 700)	75.7187	118.7760	239.6666
(850, 750)	97.3125	99.4531	240.1953
(900, 800)	94.9375	122.8281	252.8281
(950, 850)	112.5312	155.4531	280.9843
(1000, 900)	117.5937	147.9531	345.1093

6. Conclusion

A new ladder algorithm, termed "the Modified Centered Climbing Algorithm", was proposed in this paper. Computational results demonstrated that the ladder algorithm outperforms CCA as well as the simplex algorithm in terms of average CPU time for randomly generated test problems. In addition, the single artificial constraint technique was presented to initialize the ladder method for LP problems with bounded variables. An illustration showed that this initialization technique is intuitive and simple.

REFERENCES

[1] G. B. Dantzig, "Linear Programming and Extensions," Princeton University Press, Princeton, 1963.

[2] N. Karmarkar, "A New Polynomial-Time Algorithm for

Linear Programming," *Combinatorica*, Vol. 4, No. 4, 1984, pp. 373-395. doi:10.1145/800057.808695

[3] K. G. Murty and Y. Fathi, "A Feasible Direction Method for Linear Programming," *Operations Research Letters*, Vol. 3, No. 3, 1984, pp. 121-127. doi:10.1016/0167-6377(84)90003-8

[4] L. G. Khachian, "A Polynomial Algorithm in Linear Programming," *Soviet Mathematics Doklady*, Vol. 20, 1979, pp. 191-194.

[5] Y. Liu, "An Exterior Point Linear Programming Method Based on Inclusive Normal Cones," *Journal of Industrial and Management Optimization*, Vol. 6, No. 4, 2010, pp. 825-846. doi:10.3934/jimo.2010.6.825

[6] M.-F. Ding, Y. Liu and J. A. Gear, "An Improved Targeted Climbing Algorithm for Linear Programs," *Numerical Algebra, Control and Optimization* (*NACO*), Vol. 1, No. 3, 2011, pp. 399-405. doi:10.3934/naco.2011.1.399

[7] R. J. Vanderbei, "Linear Programming: Foundations and Extensions," 3rd Edition, Springer, New York, 2008.

[8] V. Klee and G. J. Minty, "How Good Is the Simplex Algorithm?" In: O. Shisha, Ed., *Inequalities III*, Academic Press, New York, 1972, pp. 159-175.

