

# An Algorithm to Solve a Linear Program

Rajan Alex

Dept. of CIS, College of Business, WTAMU, Canyon, TX 79016

*Abstract: This work is about an algorithm for solving a linear program which is simple to apply. There are three algorithms in this work. The first algorithm solves a two-variable linear program. This algorithm is built on simple concepts such as the slope and the intercept of a line. The core idea of the algorithm is the Deleting Principle based on the consistency between slopes and intercepts along the boundary of the feasible region. The second algorithm is the main algorithm. It solves a general linear program. The algorithm starts from the origin and moves to other points on the boundary of the feasible region. The algorithm depends on a two-dimensional intersection process: a systematic and repeated application of the first algorithm that will lead to a feasible solution on the boundary of the feasible region called a stable solution. The third algorithm or the judging algorithm is applied on every stable solution reached in the main algorithm. The judging algorithm either verifies that a stable solution is optimal or outputs a new feasible solution on the boundary. In the latter case, the judging algorithm shows how to get out of a trapped stable solution. The main algorithm may then be repeated by applying the first algorithm along the new direction suggested by the judging algorithm where the objective function is guaranteed to improve. Thus the main algorithm will eventually lead to a stable solution that is optimal, if the problem is feasible.*

**Keywords: feasible region, corner point, upper wall, and vertical wall.**

## 1.0 Introduction

This paper is about a simple algorithm to solve a general linear programming problem that searches for an optimal solution among the feasible solutions lying on the boundary of the feasible region. Simplex Methods [1] are practical Methods for solving linear programs and searches for optimal solutions among the corner points of the feasible region. Yet, the algorithm performs not very well with many application problems including the Klee-Minty and Chvatal (KMC) problems [2]. The Ellipsoid algorithm [4] was acclaimed on the front page of the newspapers throughout the world when it appeared in 1979. Although the algorithm is the first polynomial-time algorithm, it differs radically from the Simplex Methods. To understand the algorithm, one should have some knowledge of affine transformations and ellipsoids. The algorithm searches for optimal solution from outside of the feasible region: by approximating the simplex with an ellipsoid and eventually arrives at an optimal solution. Today, the Ellipsoid algorithm is considered to be impractical because of its complexity. A well-known polynomial-time algorithm is the one presented by Karmarkar [3]. The algorithm also differs radically from the Simplex Methods. Karmarkar's algorithm solves directly the linear program using the steepest descent method, starting from the interior of the feasible region. In this paper, I present a simple algorithm that searches for the optimal solution among the boundary points of the feasible region.

In section 2, I briefly describe a linear program and the definitions for classifying constraints. In section 3, I give the algorithm for a two-variable linear program. In section 4, I present two algorithms which use the two-variable algorithm to solve a general linear program. Section 5 is the conclusion section.

## 2.0 Preliminary

A general linear programming problem may be stated as

$$(\text{LP}_n) \begin{cases} \max z = C^T x \\ \text{st} \\ Ax \leq b \end{cases} \quad (2.1)$$

where  $A$  is a  $m \times (n+1)$  real matrix with  $m \geq 1$  and  $n \geq 1$ ,  $b$  is a vector in  $R^m$ , and  $x, c$  are vectors in  $R^{n+1}$ . Denote the  $(n+1)^{\text{th}}$  variable as  $y$ . We have the following definitions.

**Definition 2.1** The plane  $\eta_i: a_{i1}x_1 + \dots + a_{in}x_n + a_{i,n+1}y = b_i$  corresponding to the  $i^{\text{th}}$  constraint in (2.1) is called a  $\alpha$ -,  $\beta$ - and  $\gamma$ - planes according to  $a_{i,n+1} > 0$ ,  $a_{i,n+1} < 0$  and  $a_{i,n+1} = 0$ , respectively.

According to the definition, the constraints in (2.1) corresponding to the  $\alpha$ -,  $\beta$ - and  $\gamma$ - planes take the forms:

$$\alpha_i: y \leq [b_i - (a_{i1}x_1 + \dots + a_{in}x_n)] / a_{i,n+1} \quad (2.2)$$

$$\beta_k: y \geq [b_k - (a_{k1}x_1 + \dots + a_{kn}x_n)] / a_{k,n+1} \quad (2.3)$$

$$\gamma_l: a_{l1}x_1 + \dots + a_{ln}x_n \leq b_l \quad (2.4)$$

Assume that the problem  $(\text{LP}_n)$  has a feasible region and that the origin belongs to the feasible region. As a core idea of the algorithm, we view each constraint in (2.1) as a cut which cuts off a half-space in  $R^{n+1}$ . Thus,  $m$  cuts result in the feasible region for the  $(\text{LP}_n)$ . In particular, the  $\alpha$ -planes contribute to the upper boundary, the  $\beta$ -planes contribute to the lower boundary, and the  $\gamma$ -planes contribute to the vertical boundary of the feasible region. Assume that the index set of the  $\alpha$ -planes,  $\beta$ -planes and the  $\gamma$ -planes of (2.1) are

$$I = \{1, 2, \dots, I\}, \quad K = \{I+1, I+2, \dots, I+K\}, \quad \text{and} \quad L = \{I+K+1, \dots, I+K+L\}, \quad (2.5)$$

respectively. Thus we can say that when  $i \in I \cup K \cup L$ , the plane  $\eta_i$  is a  $\alpha$ -plane,  $\beta$ -plane,  $\gamma$ -plane according to  $0 \leq i \leq I$ ,  $I \leq i \leq I+K$ ,  $I+K \leq i \leq I+K+L$ , respectively. For any  $i \in I \cup K$ , denote

$$y = \lambda_i(P) = \eta_i(P) = [b_i - (a_{i1}x_1 + \dots + a_{in}x_n)] / a_{i,n+1}, \quad P = (x_1, x_2, \dots, x_n) \in R^n. \quad (2.6)$$

We call  $\lambda_i(P)$  or  $\eta_i(P)$  in (2.6) as the *intercept* of the plane  $\eta_i$  at  $P$ . We also call  $\lambda_i(P)$  the *height* ( $y$ -value) of  $\eta_i$  at  $P$ .

**Definition 2.2** The three mappings  $y = u(P)$ ,  $y = l(P)$ , and  $y = h(P)$  are called the upper wall, lower wall, and wall-height of problem  $(\text{LP}_n)$ , respectively. They are defined by

$$u(P) = \min \{ \lambda_i(P) \mid i \in I \}, \quad P \in R^n, \quad (2.7)$$

$$l(P) = \max \{ \lambda_k(P) \mid k \in K \}, \quad P \in R^n, \quad \text{and} \quad (2.8)$$

$$h(P) = u(P) - l(P), \quad P \in R^n. \quad (2.9)$$

Denote  $G = \{P \in R^n \mid P \text{ satisfies all } \gamma\text{-inequalities in (2.4)}\}$  (2.10)

as the  $\gamma$ -domain of  $(\text{LP}_n)$ . I use a bar notation to denote a feasible boundary point on the upper wall as  $\bar{P}(p_1, p_2, \dots, p_n, y) \in R^{n+1}$  where (without the bar notation)  $P(p_1, p_2, \dots, p_n) \in G$ , and  $y = u(P)$  as in (2.7).

## 3.0 Basic Algorithm BICUT for problem $(\text{LP}_1)$ with Two Variables

The two variable problem has  $n=1$ , and the variables are denoted as  $x$ , and  $y$ . The problem is

$$(\text{LP}_1) \begin{cases} \max z = c_1x + c_2y \\ \text{st} \\ a_{i1}x + a_{i2}y \leq b_i, \quad i \in E = I \cup K \cup L \end{cases} \quad (3.1)$$

Choose the variable  $y$  so that  $c_2 > 0$ . Let  $\mu_i, \lambda_i$  be the slope and the intercept of the line  $\eta_i$ ,  $i \in I \cup K$ .

The basic idea of the algorithm is to find the lines that lie on the boundary of the feasible region. The Deleting Principle is to delete the lines that do not lie on the boundary of the feasible region. Then we will have fewer lines to work with and find the boundary of the feasible region. To do this, we have the following definitions.

**Definition 3.1:** An  $\alpha$ -line,  $\alpha_i, i \in I$ , is *delete-able* on an interval  $(a, b)$  if

$$\alpha_i(x) > u(x), \text{ for all } x \in (a, b), \quad (3.7)$$

where  $\alpha_i(x) = \lambda_i(x)$  in (2.6) with  $P = x$ , and  $u(x)$  as in (2.7). We say an  $\alpha_i, i \in I$  holds the upper wall on  $(a, b)$ , if  $\alpha_i(x) = u(x), x \in (a, b)$ .

A  $\beta$ -line,  $\beta_k, k \in K$ , is *delete-able* on an interval  $(a, b)$  if

$$\beta_k(x) < l(x), \text{ for all } x \in (a, b), \quad (3.8)$$

where  $\beta_k(x) = \lambda_k(x)$  in (2.6) with  $P = x$ , and  $l(x)$  as in (2.8). We say a  $\beta_k, k \in K$ , holds the lower wall on  $(a, b)$ , if  $\beta_k(x) = u(x), \text{ for all } x \in (a, b)$ .

**Theorem 3.1** (a) For any fixed point  $e$  on the  $x$ -axis an index  $i' \in I$  is *delete-able* or the line  $\alpha_{i'}$  is *delete-able* on  $(e, +\infty)$ , if there is a  $i \in I - \{i'\}$  such that

$$\mu_i \leq \mu_{i'} \text{ and } \alpha_i(e) \leq \alpha_{i'}(e). \quad (3.9)$$

(b) For any fixed point  $e$  on the  $x$ -axis an index  $k' \in K$  is *delete-able* or the line  $\beta_{k'}$  is *delete-able* on  $(e, +\infty)$ , if there is  $k \in K - \{k'\}$  such that

$$\mu_k \geq \mu_{k'} \text{ and } \beta_k(e) \geq \beta_{k'}(e). \quad (3.10)$$

Proof of (a): Suppose that two lines  $\alpha_i, \alpha_{i'}$  meet at  $x = x^*$ , from (3.9) we have  $-\infty \leq x^* \leq e$  and

$$\lambda_{i'}(x) > \lambda_i(x), \text{ for all } x > e. \quad (3.11)$$

By the definition of upper wall in (2.7),  $\lambda_i(x) \geq u(x)$ , for  $x > e$ . So,

$$\lambda_{i'}(x) > u(x), \text{ for all } x > e. \quad (3.12)$$

Thus  $i'$  is *delete-able* on  $(e, +\infty)$ .

Similarly, we can prove the second part of the theorem.

**Theorem 3.2 (Deleting Principle):** For any given real number  $e$ , delete index numbers from  $I, K$  and get  $I', K'$  as follows.

$$I' = I - \{i' \in I / \exists i \in I - \{i'\}, \mu_i \leq \mu_{i'}, \alpha_i(e) \leq \alpha_{i'}(e)\} \quad (3.13)$$

and

$$K' = K - \{k' \in K / \exists k \in K - \{k'\}, \mu_k \geq \mu_{k'}, \beta_k(e) \geq \beta_{k'}(e)\}. \quad (3.14)$$

The upper and lower walls on  $(e, +\infty)$  will not change when  $I', K'$  are used instead of  $I, K$ .

The proof is obvious.

We will introduce the algorithm BICUT based on Theorems 3.1 and 3.2.

**Algorithm:** BICUT [  $A(m, 3), c$  ]

Input The information given by  $LP_1[ A(m, 3), c ]$ .

Output If  $(LP_1)$  is unbounded then output  $A_{unbounded}$ .@ If  $(LP_1)$  is non-feasible then output  $A_{non-feasible}$ .@ If  $(LP_1)$  is feasible and bounded then output the optimal solution of the problem  $(x^*, y^*, z^*)$ .

Begin Algorithm

Step 1 Compute the slopes and intercepts of the constraint equations. Classify the constraints according to definition 2.1. Arrange the  $\alpha$ -lines in the non-increasing order of their slope. Arrange the  $\beta$ -lines in the non-decreasing order of their slopes.

Step 2 For the  $\gamma$ -lines, set

$$A := \begin{cases} -\infty & \text{if there is no } a_{i1} < 0 \\ \max\{\frac{b_1}{a_{i1}} \mid i \in L, a_{i1} < 0\} & \text{otherwise} \end{cases} \quad (3.1)$$

$$B := \begin{cases} +\infty & \text{if there is no } a_{i1} > 0 \\ \min\{\frac{b_1}{a_{i1}} \mid i \in L, a_{i1} > 0\} & \text{otherwise} \end{cases} \quad (3.2)$$

If  $A > B$ , then stop and output Anon-feasible.@

Step 3 Begin tracing the upper and lower walls for the feasible region as follows:

3.0 Set  $t := 0$ ,  $e(0) := 0$ ,  $\Delta(0) := A$ ; Denote the sets  $UR := \phi$ ,  $LR := \phi$ ,  $HT := \phi$ .

3.1  $t := t + 1$ ,  $e(t) := e(t-1) + \Delta(t-1)$

3.2 If  $e(t) = -\infty$  go to 3.3; otherwise set

$$\lambda_i := \lambda_i + \mu_i \Delta(t-1), \text{ for all } i \in I, \text{ and} \quad (3.3)$$

$$\lambda_k := \lambda_k + \mu_k \Delta(t-1), \text{ for all } k \in K. \quad (3.4)$$

3.3 Delete indices according to the Deleting Principle (Theorem 3.1). Purge  $I$ , the index set of the  $\alpha$ -lines and get

$$I := I - \{i' \in I \mid \exists i \in I - \{i'\}, \mu_i \leq \mu_{i'}, \lambda_i \leq \lambda_{i'}\}. \quad (3.5)$$

Purge  $K$ , the index set of the  $\beta$ -lines and get

$$K := K - \{k' \in K \mid \exists k \in K - \{k'\}, \mu_k \geq \mu_{k'}, \lambda_k \geq \lambda_{k'}\}. \quad (3.6)$$

3.4 Trace the boundary points on the upper and lower walls.

Set  $i_0 := \min\{i \mid i \in I\}$ , and  $k_0 := \min\{k \mid k \in K\}$ .

Set  $u(e(t)) := \lambda_{i_0}$  and add the point  $(e(t), u(e(t)))$  to the set  $UR$ .

Set  $l(e(t)) := \lambda_{k_0}$  and add the point  $(e(t), l(e(t)))$  to the set  $LR$ .

Set  $h(e(t)) := u(e(t)) - l(e(t))$  and add the point  $(e(t), h(e(t)))$  to the set  $HT$ .

3.5 Begin to generate the next point by setting

$$\Delta 1 := \begin{cases} \min_i \{x_{i_0} \mid i = \frac{\lambda_{i_0} - \lambda_i}{\mu_i - \mu_{i_0}} / i \in I, i > i_0\}, & \text{if } |I| > 1 \\ B - e(t), & \text{otherwise,} \end{cases} \quad (3.7)$$

$$\Delta 2 := \begin{cases} \min_k \{x_{k_0} \mid k = \frac{\lambda_{k_0} - \lambda_k}{\mu_k - \mu_{k_0}} / k \in K, k > k_0\}, & \text{if } |K| > 1, \\ B - e(t), & \text{otherwise,} \end{cases} \quad (3.8)$$

and  $\Delta(t) := \min\{\Delta 1, \Delta 2\}$ .

3.6 If  $e(t) + \Delta(t) < B$  then go to 3.1; otherwise, set

$u(B) := \lambda_{i_0} + \mu_{i_0} (B - e(t))$  and add the point  $(B, u(B))$  to the set  $UR$ .

$l(B) := \lambda_{k_0} + \mu_{k_0} (B - e(t))$  and add the point  $(B, l(B))$  to the set  $LR$ .

$h(B) := u(B) - l(B)$  and add the point  $(B, h(B))$  to the set  $HT$ .

Step 4 Let  $h^* := \max\{h(a) \mid a \in \{A, e(2), \dots, e(t), B\}\}$ . (3.9)

Step 5 If  $h^* < 0$ , then stop and output: Anon-feasible.@ Otherwise, find the upper wall as follows:

5.1 If  $h(A) < 0$ , then

$$\text{set } t^* := \min \{ t' / h(e(t')) \geq 0 \}, \quad A := e(t^*) - \left[ \frac{e(t^*) - e(t^* - 1)}{h(e(t^*)) - h(e(t^* - 1))} \right] h(e(t^*))$$

$$\text{and } u(A) := l(e(t^*)) + \left[ \frac{e(t^*) - A}{e(t^*) - e(t^* - 1)} \right] (l(e(t^* - 1)) - l(e(t^*))).$$

If  $h(B) < 0$ , then set  $s^* := \max \{ t' \mid h(e(t')) \geq 0 \}$ ,

$$B := e(s^*) - \left[ \frac{e(s^* + 1) - e(s^*)}{h(e(s^* + 1)) - h(e(s^*))} \right] h(e(s^*)), \text{ and}$$

$$u(B) := u(e(s^*)) + \left[ \frac{u(e(s^* + 1)) - u(e(s^*))}{e(s^* + 1) - e(s^*)} \right] (B - e(s^*)).$$

5.2 The upper wall of the feasible region is the joining of points

$(A, u(A)), \dots, (e(t), u(e(t))), \dots, (B, u(B))$ . Denote its  $x$ -components by the set

$$S = \{ A, \dots, e(t), \dots, B \}.$$

Step 6  $z^* := \max \{ c_1 a + c_2 u(a) \mid a \in S \}$ , since  $c_2 > 0$ . If  $z^* = +\infty$ , then stop and output “unbounded.”

Step 7 Set  $x^* := \begin{cases} 0, & \text{if } 0 \in S \text{ and } h(0) = z^* \\ e(\min \{ t' \mid c_1 e(t') + c_2 u(e(t')) = z^*, t' \in S \}), & \text{otherwise,} \end{cases}$

and  $y^* := u(x^*)$ . Output the solution  $(x^*, y^*, z^*)$ .

#### 4.0 The New Algorithm LPCUT for a linear program

LPCUT is an algorithm that will find an optimal solution for the problem  $(LP_n)$ . The main idea of the algorithm is to apply BICUT repeatedly on the two-dimensional intersections of the problem  $(LP_n)$ . Suppose we set all the constraint variables  $x_{jj} = 0$  except a fixed  $x_j$  and the  $y$  variable, then the  $(LP_n)$  problem becomes a  $(LP_1)$  problem and the algorithm BICUT can be applied. This process is called Applying BICUT along the  $x_j$ -axis. Start with the feasible solution  $P = (0, \dots, 0) \in R^n$ . Each time we change the origin to a new point and get a renewed coordinate system, by the application of BICUT along the  $x_j$ -axis from the origin, for  $j = 1, \dots, n$ , if the objective function improves along the direction. The aim is to improve the objective function for each application of BICUT. In order to make this work easily understandable to the readers, I make an assumption as follows:

**Assumption 4.1** The point  $(0, \dots, 0) \in R^{n+1}$  is a feasible solution if the  $(LP_n)$  is feasible.

However, such an assumption is not necessary to solve a linear program using the algorithm. I am working on a modified algorithm to solve a linear program without the strong assumption and is the topic of another paper that is under preparation. Since the origin is feasible, we are guaranteed that each BICUT algorithm applied from the main algorithm will not result in the output “infeasible.”

**Definition 4.1** A point  $P = P(t) \in R^n$  is called a *stable point* of  $LP[A(m, n+2), c]$ , if  $P$  does not change when BICUT algorithm is applied along any of the  $x_j$ -axis from  $P$ , for  $j = 1, 2, \dots, n$ .

With the assumption and the definition, we have the following algorithm.

**Algorithm** LPCUT  $[A(m, n+2), c]$ ,  $n > 1$

Input:  $LP[A(m, n+2), c]$  and an accuracy  $\varepsilon > 0$ .

Output: If  $(LP_n)$  is unbounded then output “unbounded.” If  $(LP_n)$  is feasible and bounded then output the optimal solution  $(x_1^*, x_2^*, \dots, x_n^*, y^*)$  with  $z^*$ .

Begin Algorithm

Step 0 Set the following variables

$$0.0 \quad (i) := i, \text{ for } i = 1, 2, \dots, m \text{ and } [j] = j, \text{ for } j = 1, 2, \dots, n+2. \quad (4.1)$$

0.1 Select a column  $A_{j'}$ ,  $j' \in \{1, \dots, n+1\}$  so that the coefficient  $c_{j'} > 0$  and the  $A_{j'}$  has the largest number of non-zero entries. If there are more than one such indices, then select the one with the highest optimizing coefficient. Then exchange the columns  $A_j$ , coefficients  $c_j$  according to the choice of  $j'$  as follows.

$$Z := j'; [j] := j+1, j = 1, \dots, n \text{ and } [n+1] := Z. \quad (4.2)$$

Denote the variable  $x_{[n+1]}$  by  $y$ . Rewrite  $A(m, n+2) = [\underline{A}, b]$  and

$$c = (c_{[1]}, c_{[2]}, \dots, c_{[n+1]})^T \text{ according to (4.2).}$$

0.2 Label the constraints and identify the index set as in (2.5) and get  $I = \{1, 2, \dots, I\}$ ,  $K = \{I+1, \dots, I+K\}$ ,  $L = \{I+K+1, \dots, I+K+L\}$ , and the index set of all the constraints is  $E = I \cup K \cup L$ ,  $|E| = I+J+K = m$ .

$$0.3 \quad \text{Set } z := -\infty. \quad (4.3)$$

Step 1 Let  $t := 0$ ,  $P(t) := (p_1(t), \dots, p_n(t)) = (0, \dots, 0) \in R^n$ . According to the Assumption 4.1, if  $(P(0), 0) \in R^{n+1}$  does not satisfy all the constraints then stop and output "non-feasible."

Step 2 Prepare for the successive application of the BICUT Algorithm

2.0 Let  $Q := (q_1, \dots, q_n)$ , where  $q_k := p_k(t)$ , for  $k = 1, \dots, n$ .

2.1  $t := t+1$ ,  $j := 0$ .

2.2 Increment  $j$ .

2.3 Find the two-variable problem along the  $x_j$ -axis from  $Q$ . To do this set

$$A'(m, 3) = [A'_1, A'_2, b'] \text{, where}$$

$$A'_1 = (a_{(1)[j]}, \dots, a_{(m)[j]})^T, \quad A'_2 = (a_{(1)[n+1]}, \dots, a_{(m)[n+1]})^T, \text{ and } b' \in R^{n+1} \text{ with its } i^{\text{th}}$$

component  $b_{(i)} = b_{(i)} - (a_{(i)[1]}q_{[1]} + \dots + a_{(i)[n]}q_{[n]})$ ,  $i \in E$  and set an new optimizing vector

$$s = (s_1, s_2), \text{ where } s_1 = c_{[j]}, \quad s_2 = c_{[n+1]}.$$

2.4 Apply BICUT  $[A'(m, 3), s]$ . If the output is "unbounded," then stop and output "unbounded." Otherwise use the output  $(x^*, y^*, z^*)$  and set

$$z^* := z^* + (c_{[1]}q_{[1]} + \dots + c_{[n]}q_{[n]}), \quad \Delta z := z^* - z, \quad z := z^*.$$

Update the  $j^{\text{th}}$  component of  $Q$  by

$$q_{[j]} := \begin{cases} q_{[j]} + x^*, & \text{if } \Delta z > \varepsilon \\ q_{[j]}, & \text{if } \Delta z \leq \varepsilon \end{cases} \quad (4.4)$$

2.5 If  $j < n$ , go to Step 2.2

2.6 Set  $P(t) := Q$ , find  $|P(t) - P(t-1)| = \max_{1 \leq i \leq n} |p_i(t) - p_i(t-1)|$ .

2.7 If  $|P(t) - P(t-1)| \leq \varepsilon$ , then go to Step 3; else go to Step 2.

Step 3 Applying BICUT along the plane containing a line  $\overline{P^1 P^2}$  and parallel to the  $y$ -axis.

3.0 If  $t = 1$  then go to Step 4. Otherwise proceed to the next step to apply BICUT along  $\overline{P^1 P^2}$ .

3.1 Set  $P^1 = (x_1, \dots, x_n)$ ,  $P^2 = (x_1, \dots, x_n)$ , where  $x_k := p_k(t)$ ,  $x_k := p_k(t-2)$ , for  $1 \leq k \leq n$ .

3.2 Let  $A'(m, 3) = [A_1, A_2, b']$ , where  $A_1, A_2$  and  $b'$  are column vectors and their  $i^{\text{th}}$

components are

$$a_{(i)1} := \frac{a_{(i)[1]}(x_{[1]} - x_{[1]}) + \dots + a_{(i)[n]}(x_{[n]} - x_{[n]})}{\sqrt{(x_1 - x_1)^2 + \dots + (x_n - x_n)^2}}, a_{(i)2} := a_{(i)[n+1]}, i \in E \quad (4.5)$$

and

$$b_{(i)} := b_{(i)} - (a_{(i)[1]}x_{[1]} + \dots + a_{(i)[n]}x_{[n]}), i \in E, \text{ respectively.} \quad (4.6)$$

$$\text{Set } s_1 := \frac{c_{[1]}(x_{[1]} - x_{[1]}) + \dots + c_{[n]}(x_{[n]} - x_{[n]})}{\sqrt{(x_1 - x_1)^2 + \dots + (x_n - x_n)^2}}, s_2 := c_{[n+1]}, \text{ and let } s := (s_1, s_2) \quad (4.7)$$

3.3 Apply BICUT [A'(m,3), s], increment  $t$  and get the output  $(x^*, y^*, z^*)$ . Set

$z^* := z^* + (c_{[1]}p_{[1]}(t-1) + \dots + c_{[n]}p_{[n]}(t-1))$ ,  $\Delta z := z^* - z$ ,  $z := z^*$ , and the components of  $P(t)$  as

$$p_{[j]}(t) := \begin{cases} p_{[j]}(t-1) + \frac{x^* p_{[j]}(t-1)}{\sqrt{(x_1 - x_1)^2 + \dots + (x_n - x_n)^2}}, & \text{if } \Delta z > \varepsilon \\ p_{[j]}(t-1), & \text{if } \Delta z \leq \varepsilon \end{cases} \quad (4.8)$$

for  $j = 1, \dots, n$ .

3.4 Find  $|P(t) - P(t-1)| = \max_i |p_i(t) - p_i(t-1)|$ .

If  $|P(t) - P(t-1)| > \varepsilon$ , then go to step 2.

Step 4 To judge whether the stable point  $P(t)$  is optimal:

4.1 Find  $u(P(t)) = \min \{\lambda_i(P(t)) / i \in I\}$ , where  $\lambda_i(P(t))$  is given by (2.6).

4.2 Choose the set of indices among the  $\alpha$ -,  $\beta$ -, and  $\gamma$ -planes that contains  $(P(t), u(P(t)))$  and denote the set by  $D$ .

4.3 Apply Algorithm DETRAP [ $D, (P(t), u(P(t))), c$ ] and do one of the following:

1) If the output of the algorithm is 'top,' then go to step 5.

2) If the output is 'trap' with two points  $P^1, P^2$  then go to step 3.2.

Step 5 Stop and output the optimal solution  $(x_1^*, \dots, x_n^*, y^*)$  with the optimal value  $z^*$ .

**Algorithm DETRAP** [ $D, \bar{P}, c$ ] (The judging algorithm)

Input: 1) Augmented matrix  $A_D$  formed by the planes  $\{\eta_d\}, d \in D \subseteq E$ . Assume that  $|D| = p \leq n+1$ .

2) The point  $\bar{P} = (p_1, \dots, p_{n+1}) \in R^{n+1}$  where the planes meet and the optimizing vector  $c$

Output: If  $\bar{P}$  is an optimal point of the feasible cone formed by planes of  $A_D$ , then output the message ATop. Otherwise output the message ATrap and two points  $\overrightarrow{P^1 P^2}$ .

Algorithm:

Step 0 Set the following variables

0.1  $d := 1$

0.2 Let  $D = \{d_1, d_2, \dots, d_p\}$ . Take  $(i) = d_i, i = 1, 2, \dots, p$  and  $j = 1, 2, \dots, n+1$ .

0.3 Form the matrix  $A' = [a_{(i)[j]}; b_{(i)}]$ , with entries

$$b_{(i)} := 0, i = 1, 2, \dots, n+1$$

$$a_{(i)[j]} := a_{(i)[j]}, i = p+1, \dots, n+1; j = 1, 2, \dots, n+1$$

Step 1 Perform elementary row operations on  $A'$  and reduce it to an upper triangular matrix.

Step 2 Let  $r$  be the number of rows with all zeroes. If  $r > 0$  then go to Step 6. Otherwise go to step 3

Step 3 Case 1: In this case assume that the elementary row operations on  $A'$  gives

$$A' = \begin{bmatrix} & [1] & [2] & \cdots & [n] & [n+1] & b \\ (1) & 1 & a'_{(1)[2]} & \cdots & a'_{(1)[n]} & a'_{(1)[n+1]} & 0 \\ (2) & 0 & 1 & \cdots & a'_{(2)[n]} & a'_{(2)[n+1]} & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ (n) & 0 & 0 & \cdots & 1 & a'_{(n)[n+1]} & 0 \\ (n+1) & 0 & 0 & \cdots & 0 & 1 & 0 \end{bmatrix}$$

3.1 Find a point as follows :

3.1.1 Set  $x_{[n+1]} = 1$

3.1.2 For  $i = n, n-1, \dots, 1$

For  $j = i+1, \dots, n+1$

$$x'_{[i]} := -a'_{(i)[j]} x'_{[j]}$$

(End for loop  $j$ )

(End for loop  $i$ )

3.2 Verify if the following inequality holds. That is, whether the (transformed) point in the step 3.1 is point on the edge  $S'_{(n+1)}$ . Set  $-w = a_{(n+1)[1]}x'_{[1]} + \cdots + a_{(n+1)[n+1]}x'_{[n+1]} < \varepsilon$

3.3 If the inequality in step 3.2 is not true then repeat step 3.1 with  $x'_{[n+1]} := -1$  in step 3.1.1

3.4 To Check whether the edge  $S'_{(n+1)}$  is *stationary*. If  $c_{[1]}x'_{[1]} + \cdots + c_{[n]}x'_{[n]} + c_{[n+1]}x'_{[n+1]} > \varepsilon$  in Steps 3.2 and 3.3, then stop and output “Trap” with  $P^1 := (x_1, \dots, x_n)$  and  $P^2 := (x'_1, \dots, x'_n)$ . Otherwise go to Step 4.

Step 4 If  $d = n+1$ , then stop and send the output message: “Top.” Otherwise set  $d := d+1$ .

Step 5 Renew edge by exchanging the rows and columns of  $A'$  as follows:

5.1  $Z := [n+1]; [n+1] := [n], [n] := [n-1], \dots, [2] := [1], [1] := Z$

5.2  $Z := (n+1); (n+1) := (n), (n) := (n-1), \dots, (2) := (1), (1) := Z$

5.3 If  $a'_{(1)[1]} = 0$ , then repeat step 5.2 until  $a'_{(1)[1]} \neq 0$

5.4 Perform elementary row operation and reduce it to a upper triangle.

5.5 Go to Step 3.

Step 6 Case 2

6.1 Perform further elementary row operation on  $A'$  and get triangular block matrix. Assume that the row-reduce form of  $A'$  is

$$A' = \begin{bmatrix} & [1] & [2] & \cdots & [n-r+1] & [n-r+2] & \cdots & [n+1] & b \\ (1) & 1 & 0 & \cdots & 0 & a'_{(1)[n-r+2]} & \cdots & a'_{(1)[n+1]} & 0 \\ (2) & 0 & 1 & \cdots & 0 & a'_{(2)[n-r+2]} & \cdots & a'_{(2)[n+1]} & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ (n-r+2) & 0 & 0 & \cdots & 1 & a'_{(n-r+2)[n-r+2]} & \cdots & a'_{(n-r+2)[n+1]} & 0 \end{bmatrix},$$

6.2 Begin judging

6.2.1  $j := n-r+1$ , with  $r > 0$ .

6.2.2 If  $j < n+1$ , then  $j := j+1$ . Otherwise set  $n := n-r$  and go to Step 0

6.2.3 If  $|c_{[1]}a'_{(1)[j]} + \cdots + c_{[n-r+1]}a'_{(n-r+1)[j]} - c_{[j]}| \leq \varepsilon$  then go to 6.2.2. Otherwise proceed to the next step.

6.2.3 Set  $x_{[1]} := p_{[1]}, \dots, x_{[n]} := p_{[n]}$  and

$$x'_{[1]} := p'_{(1)} - a'_{(1)[j]}, \dots, x'_{[n-r+1]} := p'_{[n-r+1]} - a'_{(n-r+1)[j]},$$

$$x'_{[n-r+2]} := p'_{[n-r+2]}, \dots, x'_{[j-1]} := p'_{[j-1]},$$

$$x'_{[j]} := p'_{[j]} + 1, x'_{[j+1]} := p'_{[j+1]}, \dots, x'_{[n]} := p'_{[n]}.$$

Stop and output "Trap" with two points  $P^1 := (x_1, \dots, x_n)$ ,  $P^2 := (x'_1, \dots, x'_n)$ .

### 5.0 Conclusion

This work is about an algorithm to solve a linear programming problem that is simple to understand and also easy for hand calculation or computer implementation. In this work, the constraints in general contribute to the upper wall or the lower wall or the vertical wall of the feasible region. Using the upper and lower walls, a height function can be formed. The height function plays an important role in this work. Whenever the BICUT algorithm outputs a solution, it precisely falls on the boundary of the feasible region. Therefore, we can say that the LPCUT algorithm searches for optimal solutions on the boundary of the feasible region.

### 6.0 References

- [1] G.B. Danzig. "Linear Programming and Extensions." Princeton University Press, Princeton, NJ., 1963.
- [2] R. Fletcher. "Practical Methods of Optimization." John Wiley & Sons Ltd., 1987.
- [3] N. Karmarkar. "A New Polynomial-Time Algorithm for Linear Programming." *Combinatorica* 4, 191-194, 1984.
- [4] L.G. Khachiyan. "A Polynomial Algorithm for Linear Programming." (in Russian), *Doklady Akademiia Nauk USSR* 244, 1093-1096. A translation in *Soviet Mathematics Doklady* 20 (1979) 191-194, 1979.