Journal of Computational Mathematics, Vol.14, No.3, 1996, 249–255.

A MODIFIED BISECTION SIMPLEX METHOD FOR LINEAR PROGRAMMING*

P.Q. Pan
(*Department of Mathematics and Mechanics, Southeast University, Nanjing, China*)

Abstract

In this paper, a modification of the bisection simplex method[7] is made for more general purpose use. Organized in an alternative simpler form, the modified version exploits information of the optimal value, as does the original bisection method, but no bracket on the optimal value is needed as part of input; in stead, it only requires provision of an estimate b_0 of the optimal value and an estimate of the error bound of b_0 (it is not sensitive to these values though) . Moreover, a new, ratio-test-free pivoting rule is proposed, significantly reducing computational cost at each iteration. Our numerical experiments show that the method is very promising, at least for solving linear programming problems of such sizes as those tested.

1. Introduction

The bisection method, proposed in an earlier paper by the author[7], exploits information of the optimal value to speed up the solution process. However, the method requires a bracket on the optimal value as part of its input, and its promisingly good performance depends on whether a suitable bracket is available; it may even fail to solve a problem if the initial interval provided does not contain the optimal value actually. In this paper, the method is modified for more general purpose use. Organized in an alternative simpler form, the new version no longer needs any bracket on the optimal value as part of input; in stead, it only requires an estimate b_0 of the optimal value and an estimate of the error bound of b_0, to which it is not sensitive though.

Nevertheless, it might be the new pivoting rule proposed that makes a more important improvement. The original rule (Rule 3.9 of [7]) of the bisection method may be regarded as a variant of Dantzig's classical rule, applied in an alternative administration; features of rules of this type are as follows:

(1) The incoming variable takes a feasible value.

(2) The outgoing variable takes the value of zero.

(3) All the feasible variables remain feasible after a basis change.

Although these classical conditions are widely accepted, and employed in different contexts, some authors such as Wolfe[13,14], Greenberg[4], Maros[5] and Belling-Seib[1]

* Received April 18, 1994.

1) The Project Supported by National Natural Science Foundation of China.

suggest relaxing condition (3); they reduce the amount of total infeasibility in stead. Rules of this type usually do require less iterations than classical methods. Unfortunately, they give a rise in computational cost per iteration. The new proposed rule, which is a ratio-test-free one, not only relaxes condition (3) but also gets rid of measuring infeasibility, consequently reducing computational effort at each iteration. Such type of rules have been very successful in other contexts[8,9,10,11]. Since numerical results of our tests show that the number of iterations required by the modified version is slightly less than that required by the original bisection method, total computational cost is reduced.

In Section 2, we propose the pivoting rule first, and then establish a procedure using the rule. In Section 3, we describe the modified algorithm in which the procedure is employed as its subalgorithm. Finally, in Section 4, we report our numerical results obtained, which are very encouraging although still preliminary.

2. The Ratio-Test-Free Rule

Consider linear programming problem in the standard form:

$$\max \; z = cx \tag{2.1a}$$

$$\text{s.t. } Ax = b \tag{2.1b}$$

$$x \geq 0, \tag{2.1c}$$

where $A \in R^{m \times n}$, $b \in R^m$, and c and x are row and column n-vectors, respectively.

In this section, an attempt is made for achieving feasibility under some *fixed* objective function value. For this purpose, the procedure, given in Section 3 of [7], is modified in an alternative simpler form in which a ratio-test-free pivoting rule is employed.

View $cx = z$ as a constraint, and take it as the 0-th constraint among others. Then, setting

$$A := \left(\frac{c}{A}\right), \qquad b := \left(\frac{z}{b}\right), \tag{2.2}$$

allows to denote the augmented constraint system by (2.1b) again. Thus now we have $A \in R^{(m+1) \times n}$ and $b \in R^{m+1}$ with $a_{0j} \equiv c_j$, $j = 1, \ldots, n$ and $b_0 \equiv z$, which may be referred to as *objective value parameter*. Assume that $Ax = b$ is consistent with $\operatorname{rank}(A) = k + 1 \leq m + 1 < n$.

Let $B \in R^{(m+1) \times (k+1)}$ be the basis of A with the basic index set

$$J_B = \{j_0, \ldots, j_k\}. \tag{2.3}$$

Introduce notation

$$\bar{J}_B = \{1, \ldots, n\} \backslash J_B \; . \tag{2.4}$$

The corresponding canonical form can then be represented by the tableau below:

$$B^+A \mid B^+b, \tag{2.5}$$

where B^+ is the Moore-Penrose inverse of B.

For a fixed b_0, if the row index set

$$I = \{i \mid (B^+b)_i < 0, i = 0, \dots, k\} \tag{2.6}$$

is empty, a feasible solution with objective value b_0 is already present. When this is not the case, we use the following pivoting rule to make the basis change:

Rule 2.1. *If set I is nonempty, select the pivot row index r such that*

$$r = \arg\min\{(B^+b)_i \mid i \in I\}. \tag{2.7}$$

If set

$$J = \{j \mid (B^+a_j)_r < 0, j \in \bar{J}_B\} \tag{2.8}$$

is nonempty, select the pivot column index s such that

$$s = \arg\min\{(B^+a_j)_r \mid j \in J\}. \tag{2.9}$$

If I and J are both nonempty, and pivot row and column indices, r and s, are determined under the preceding rule, the new tableau yielding from the basis change will be

$$\bar{B}^+A \mid \bar{B}^+b, \tag{2.10}$$

where $\bar{B}^+$ can be computed via the following formulae:

$$\bar{B}^+ = B^+ + \frac{(e_r - B^+a_s)e_r^TB^+}{e_r^TB^+a_s}. \tag{2.11}$$

The new right-hand side of the system can be written in terms of the old:

$$(\bar{B}^+b)_r = (B^+b)_r/(B^+a_s)_r > 0, \tag{2.12a}$$
$$(\bar{B}^+b)_i = (B^+b)_i - ((B^+b)_r/(B^+a_s)_r)(B^+a_s)_i, \qquad i \neq r \tag{2.12b}$$

where inequality follows from (2.7) with (2.6) and (2.9) with (2.8). It is clear that such a basis change turns the most negative component of the right-hand side into a positive one though may not maintain current feasibilities. One iteration step is then complete. Such step is repeated until I is empty, implying that the feasible solution with value b_0 is reached, or I is nonempty but J is empty, implying that there exists no feasible solution with value of b_0. After that, the current solution may be purified into a *basic* solution via the same steps as those of Algorithm 3.11 of [7], and the latter is then tested for optimality.

The following model algorithm summarizes the preceding procedure:

Subalgorithm 2.2. *Let b_0 be given and let B^+ be the Moore-Penrose inverse of a basis.*

1. *Compute B^+b.*
2. *If I defined by* (2.6) *is empty then:*
   (i) *stop if set $\underline{I} = \{i \mid (B^+e_0)_i < 0, i = 0, \dots, k\}$ is empty;*
   (ii) *Determine σ and $\underline{i}$ by $\sigma = -(B^+b)_{\underline{i}}/(B^+e_0)_{\underline{i}} = \min\{-(B^+b)_i/(B^+e_0)_i$*

$\mid i \in \underline{I}\} \geq 0$, *then set* $b_0 = b_0 + \sigma$ *and* $B^+b = B^+b + \sigma(B^+e_0)$ *if* $\sigma > 0$*;*

(iii) *stop if set* $\underline{J} = \{j \mid (B^+a_j)_{\underline{i}} < 0, j \in \bar{J}_B\}$ *is empty;*

(iv) *stop.*

3. *Determine row index* r *by* (2.7)*.*
4. *If set* J *defined by* (2.8) *is empty then:*

(i) *stop if* $(B^+e_0)_r = 0$*;*

(ii) *stop if* $(B^+e_0)_r > 0$*;*

(iii) *compute* $\sigma = -(B^+b)_r/(B^+e_0)_r < 0$*, then set* $b_0 = b_0 + \sigma$ *and* $B^+b = B^+b + \sigma(B^+e_0)$*;*

(iv) *stop if set* I *defined by* (2.6) *is empty;*

(v) *stop.*

5. *Determine column index* s *by* (2.9)*.*
6. *Update* B^+ *via* (2.11)*, and* B^+b *by* (2.12)*.*
7. *Go to Step* 2*.*

We state the following theorem to clarify meanings of different outlets and end products of the preceding algorithm, proof of which is similar to that of Theorem 3.13 in [7]:

Theorem 2.3. *Assume termination of Subalgorithm* 2.2*. The termination occurs at either*

(a) *Step* 2(iii) *or* 4(iv)*, with an optimal solution obtained; or*

(b) *Step* 2(iv)*, with a basic feasible(not optimal) solution reached; or*

(c) *Step* 4(ii) *or* 4(v)*, indicating absence of optimal value over interval* $(-\infty, b_0]$ *(at Step* 4(ii)*), or over* $[b_0, \infty)$ *(at Step* 4(v)*); or*

(d) *Step* 2(i)*, indicating up-unboundedness of the program; or*

(e) *Step* 4(i)*, indicating infeasibility of the program.*

The question remaining now is whether or not Subalgorithm 2.2 terminates. Since the number of all possible bases is finite, it does not terminate if and only if cycling occurs, i.e., some bases are repeated infinitely many times. It has not been possible to rule out the possibility of cycling. However, cycling might rarely happen to occur in practice, a standpoint of view that is supported by our computational experiments, reported in the final section of this paper (See also [8, 9, 10, 11]).

3. The Main Procedure

In this section, we describe the main procedure for solving the linear program (2.1), in which Subalgorithm 2.2 is employed.

Assume the existence of optimal solution. In order to speed up solution process, the main procedure again exploits information about the optimal value; more flexibly, however, it no longer needs a bracket (α, β) on the optimal value provided, rather it only requires an estimate of it, as initial b_0, and an estimate δ of the error bound of b_0. It produces a bracket (α, β) by itself in the following way. It first call Subalgorithm 2.2. If termination occurs with the indication that the initial b_0 is a lower (upper) bound on the optimal value, it sets $\alpha = b_0$, $b_0 = \alpha + \delta$ ($\beta = b_0$ and $b_0 = \beta - \delta$), and then call Subalgorithm 2.2 with the new b_0 again. In the process of repeating such steps, once current b_0 is found to be an upper (lower) bound, setting $\beta = b_0$ ($\alpha = b_0$) gives a bracket (α, β) on the optimal value, which allows the algorithm to go on its bisection steps thereafter. In the whole process, if some b_0 happens to fall into the "ideal interval" desirably, Subalgorithm 2.2 will terminate with an optimal solution achieved, and therefore no further steps are required.

The preceding can be put in the basic algorithm below:

Algorithm 3.1. *Let b_0 be an estimate of the optimal value and δ be an estimate of the error bound of b_0. Let B^+ be the Moore-Penrose inverse of an initial basis of A. This algorithm solves linear program* (2.1).

```
(1) Set $\alpha = -\infty$ and $\beta = +\infty$.
(2) Execute Subalgorithm 2.2.
(3) Stop if termination occurs at either Step 2(iii) or 4(iv).
(4) Stop if termination occurs at Step 2(i).
(5) Stop if occurs at Step 4(i).
(6) If occurs at Step 2(iv) or 4(ii), then:
    (i) set $\alpha = b_0$;
    (ii) if $\beta = +\infty$, set $b_0 = \alpha + \delta$, and go to Step (2).
(7) If occurs at Step 4(v), then
    (i) set $\beta = b_0$;
    (ii) if $\alpha = -\infty$, set $b_0 = \beta - \delta$, and go to Step (2).
(8) set $b_0 = (\alpha + \beta)/2$, and then go to Step (2).
```

We state the following companion theorem, which can be simply proved by making the use of Theorem 2.3:

Theorem 3.2. *Assume termination of Algorithm* 3.1*. Then it occurs at either*

(a) *Step* (3)*, with an optimal solution reached; or*

(b) *Step* (4)*, indicating upper unboundedness of the problem; or*

(c) *Step* (5) *indicating infeasibility of it.*

Notes: If Subalgorithm 2.2 is finite, Algorithm 3.1 must terminates at Step (3) if any linear programming problem encountered has an optimal solution, or at Step (4) if the feasible value set is nonempty, and upper unbounded; however, the termination of Algorithm 3.1 is not guaranteed whenever the problem has no feasible solution.

4. Computational Experience

In this final section, we report numerical results of our preliminary tests. Algorithm 3.1, coded in a FORTRAN program, is implemented, and compared with Algorithm 4.2 of [7] as well as the revised two-phase simplex algorithm using Dantzig's original rule. The model algorithm below, a revised version of Algorithm 6.5 of [7], was used for producing the Moore-Penrose inverse B^+ of an initial basis B as input for both Algorithm 3.1 and Algorithm 4.2 of [7]:

Subalgorithm 4.1.. *Given $A \in R^{(m+1)\times n}$, pivoting indices α_j, $j = 1, \dots, n$, and a small positive number ϵ.*

1. *Set* $k = 0$, $r = 1$ *and* $J = 1, \dots, n$.
2. *Determine* $s_1 = \mathrm{arg}max\alpha_j | j \in \bar{J}$.
3. *Set* $\bar{J} = \bar{J} \backslash \{s_1\}$.
4. *Set* $A_0 = a_{s_1}$, *and compute* $A_0^+ = (1/a_{s_1}^T a_{s_1}) a_{s_1}^T$.
5. *Set* $k = k + 1$.
6. *Set* $r = r + 1$.
7. *Determine* $s_r = \arg\max\{\alpha_j | j \in \bar{J}\}$.
8. *Set* $\bar{J} = \bar{J} \backslash \{s_r\}$.
9. *Compute* $d_k = A_{k-1}^+ a_{s_r}$ *and* $c_k = a_{s_r} - A_{k-1} d_k$.
10. *If* $c_k^T c_k < \epsilon$ *and* $r < n$, *go to step* 6.
11. If $c_k^T c_k \geq \epsilon$, set $A_k = (A_{k-1} \vdots a_{s_r})$ and compute $A_k^+ = \begin{pmatrix} A_{k-1}^+ - d_k c_k^+ \\ c_k^+ \end{pmatrix}$.
12. If $k < m$ and $r < n$, go to step 5.
13. Set $B = A_k$ and $B^+ = A_k^+$, and stop.

The machine precision used was about 16 decimal places. 10^{-6} was taken to be as the tolerance, and also taken as the value of the ϵ for Subalgorithm 4.1. Tested linear programming problems fall into three sets. The first set contains 62 arbitrarily collected problems with strict inequality constraints, involving up to 22 decision variables and constraints. The second one includes 4 larger sparse problems in the standard form, for each the price coefficients were simply taken to be pivoting indices, used as input of Subalgorithm 4.1. The third are three Klee-Minty problems (see, for example, [12]). For each of these problems, a relatively good estimate b_0 of the optimal value and an estimate δ of the error bound of b_0 were supplied to Algorithm 3.1; accordingly, $\alpha = b_0 - \delta$ and $\beta = b_0 + \delta$ were supplied to Algorithm 4.2 of [7].

In terms of number of pivots required, numerical results obtained are summarized in the table below, where problems of set 2 are designated by P1, P2, P3 and P4, and the Klee-Minty problems by KM1, KM2 and KM3, respectively.

Table 1. Numerical Results

Problem	Algorithm 3.1	Algorithm 4.2 of [7]	The Classical
Total for Group 1	195	204	779
P1: $m = 27$ $n = 51$	8	9	27
P2: $m = 28$ $n = 56$	10	12	38
P3: $m = 55$ $n = 137$	50	64	170
P4: $m = 56$ $n = 138$	85	101	172
Total for Group 2	153	186	407
KM1: $n = 8$	6	7	255
KM2: $n = 10$	9	10	1023
KM2: $n = 12$	11	11	4095

The results clearly show that both the original bisection algorithm and the modified version outperformed the classical method on these tested problems with quite large margins. In fact, this is even valid for each of these problems. As for the two bisection codes, pivots required by the modified one are slightly less than the original; noting that the pivoting rule, used in the modified algorithm, is a ratio-test-free one, and reduces computational effort per iteration significantly, we conclude that the performance of the modified algorithm is better than the original. It should be indicated that cost with Subalgorithm 4.1 alone is limited, which is no more than that spent by the classical method of its $m/2$ pivot steps.

References

[1] K. Belling-Seib, An improved general Phase-I method in linear programming, *European Journal of Operations Research*, 36(1988), 101–106.

[2] K.Z. Chen, G.X. Xiao, K.J. Wu and H.J. Cao, An improved simplex- like bisection method for LP, Proceedings of Second National Symposium on Mathematical Programming, Xidian University Press, 1994, Xian, China, 338–346.

[3] G.B. Dantzig, Linear Programming and Extensions, Princeton University Press, Princeton, New Jersey, 1963.

[4] H.J. Greenberg, Pivot selection tactics, in H.J. Greenberg (Ed.) Design and Implementation of Optimization Software, Sijthoff and Noordhoff, 1978, 109–143.

[5] Istvan Maros, A general Phase-I method in linear programming, *European Journal of Operations Research*, 34(1986) 64–77.

[6] P.Q. Pan, Practical finite pivoting rules for the simplex method, *OR Spektrum* 12(1990) 219–225.

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

[8] P.Q. Pan, Ratio-test-free pivoting rules for a dual Phase-1 method, in: S.T Xiao and F. Wu (Ed.), Proceedings of the Third Conference of Chinese SIAM, Tsinghua University Press, Beijing, 1994, 245–249.

[9] P.Q. Pan, Achieving primal feasibility under the dual pivoting rule, *Journal of Information and Optimization Sciences* , 15(1994b), 405–413.

[10] P.Q. Pan, Composite Phase-1 Methods without measuring infeasibility, in: M.Y. Yue (Ed.), Theory of Optimization and its Applications, Xidian University Press, Xian, 1994, 359–364.

[11] P.Q. Pan, Ratio-test-free pivoting rules for the bisection simplex method, Theory and Application of Decision Making Science, (Proceedings of Chinese Symposium on Decision Making Science), 1994, 24–29.

[12l A. Schrijver, The Theory of Linear and Integer Programming, John Wiley & Sons, Chichester, 1986.

[13] P. Wolfe, An extended composite algorithm for linear programming, the RAND Corporation, P-2373, 1961.

[14] P. Wolfe, The composite simplex algorithm, *SIAM Review* , 7(1965), 42-54.