Mathematical Programming 56 (1992) 361-364
North-Holland

Short communication

On affine scaling and semi-infinite programming

Michael C. Ferris
Computer Sciences Department, University of Wisconsin, Madison, WI 53706, USA

Andrew B. Philpott
University of Auckland, New Zealand

Received 8 May 1990
Revised manuscript received 25 February 1991

We consider an extension of the affine scaling algorithm for linear programming problems with free variables to problems having infinitely many constraints, and explore the relationship between this algorithm and the finite affine scaling method applied to a discretization of the problem.

Key words: Affine scaling, semi-infinite linear programming, free variables.

In this note we are concerned with the generalization given by Ferris and Philpott [3] of the affine scaling algorithm discovered by Dikin [2] to solve semi-infinite linear programming problems, in which the number of variables is finite, but the number of constraints is not. In [3] a discrepancy is pointed out between the classical algorithm and its generalization. The purpose of this note is to explain the discrepancy.

Ferris and Philpott [3] propose an affine scaling algorithm for linear programs of the following form:

$$\begin{aligned} \text{minimize} \quad & c^{\mathrm{T}}x \\ \text{subject to} \quad & Ax - z = b, \\ & z \geqslant 0. \end{aligned}$$

Since the variables x are unrestricted in sign, it seems natural that only the z variables should undergo the scaling transformation at each stage. Given a feasible point $(x^{(k)}, z^{(k)})$, with $z_j^{(k)} > 0$ for each component j, a single iteration of the algorithm in [3] performs the following steps.

This material is based on research supported by Air Force Office of Scientific Research Grant AFOSR 89-0410.

Algorithm 1.

Step 1. $Z = \mathrm{diag}(z^{(k)})$.

Step 2. $d = -[I + A^{\mathrm{T}}Z^{-2}A]^{-1}c, f = -Z^{-1}A[I + A^{\mathrm{T}}Z^{-2}A]^{-1}c$.

Step 3. $x^{(k+1)} = x^{(k)} + \alpha d$, $z^{(k+1)} = z^{(k)} + \alpha Zf$.

Here $\begin{bmatrix} d \\ f \end{bmatrix}$ is the projection of $\begin{bmatrix} -c \\ 0 \end{bmatrix}$ onto the nullspace of $[A \ -Z]$, and the step length α is chosen in Step 3 so that $z^{(k+1)}$ retains strictly positive components.

The algorithm described here differs both from that for free variables derived by Vanderbei [4], and the dual affine scaling algorithm described by Adler et al. [1]. In terms of the scaled variables, the constraints above become

$$[A \ -Z]\begin{bmatrix} x \\ Z^{-1}z \end{bmatrix} = b.$$

Observe that Algorithm 1 scales each free variable x_j by unity. If one interprets the aim of affine scaling to be to place the (scaled) iterate at distance one coordinatewise from each of the (scaled) lower bounds on the variables, then Algorithm 1 scales x_j *as if it had a lower bound of* $x_j^{(k)} - 1$. The algorithm thus may be viewed as adjusting a set of hypothetical lower bounds for the free variables on each iteration. Notwithstanding this it gives a sequence of iterates which can be shown to converge to the solution.

Of course, such an algorithm is not the most natural extension of affine scaling to free variables. The canonical extension is obtained by Vanderbei [4], who derives an algorithm which explicitly uses bounds on the free variables, and then lets these bounds tend to infinity. The algorithm which this procedure produces performs the following steps in each iteration.

Algorithm 2.

Step 1. $Z = \mathrm{diag}(z^{(k)})$.

Step 2. $d = -[A^{\mathrm{T}}Z^{-2}A]^{-1}c, f = -Z^{-1}A[A^{\mathrm{T}}Z^{-2}A]^{-1}c$.

Step 3. $x^{(k+1)} = x^{(k)} + \alpha d$, $z^{(k+1)} = z^{(k)} + \alpha Zf$.

When it is normalized to have unit length, the direction d for Algorithm 2 is the solution to the following problem:

$$\begin{aligned} \text{minimize} \quad & c^{\mathrm{T}}y \\ \text{subject to} \quad & Ay - w = 0, \\ & w^{\mathrm{T}}Z^{-2}w \leq 1. \end{aligned}$$

This might be contrasted with the direction given by Algorithm 1, which when normalized solves:

$$\begin{aligned} \text{minimize} \quad & c^{\mathrm{T}}y \\ \text{subject to} \quad & Ay - w = 0, \\ & w^{\mathrm{T}}Z^{-2}w + y^{\mathrm{T}}y \leq 1. \end{aligned}$$

Observe that here the inequality constraint explicitly restricts the size of y in exactly the same fashion that y would be restricted if each x_j had a lower bound of $x_j^{(k)} - 1$. This has a deleterious effect on the algorithm as one might expect.

In [3] Algorithm 1 is extended to solve semi-infinite programming problems of the following form:

$$\begin{aligned} \text{minimize} \quad & c^{\mathrm{T}}x \\ \text{subject to} \quad & a(s)^{\mathrm{T}}x - z(s) = b(s), \\ & z(s) \geqslant 0, \quad s \in S, \end{aligned}$$

where S is some infinite index set, which in all the examples discussed in [3] is the interval $[0, 1]$.

Since the nullspace of the constraint operator is finite dimensional, at each iterate $(x^{(k)}, z^{(k)})$ one can compute the direction d of Algorithm 1 by solving

$$[I + B]d = -c$$

where the matrix B has elements given by

$$b_{ij} = \int_S \frac{a_i(s)a_j(s)}{(z^{(k)}(s))^2}\,\mathrm{d}s.$$

The components b_{ij} can be computed by some suitable numerical quadrature technique such as Simpson's Rule.

The computational tests reported in [3] give conclusive evidence that this algorithm has an inferior performance to that of Algorithm 1 applied to a discretization of the semi-infinite problem. Furthermore, this discrepancy does not disapper with refinement of the discretization. Indeed, for the discrete problem, the direction d^N computed by Algorithm 1 is the solution to

$$[I + B^N]d^N = -c$$

where

$$b_{ij}^N = \sum_{r=1}^{N} \frac{a_i(s_r)a_j(s_r)}{(z(s_r))^2}.$$

Thus, since $b_{ij}^N \approx N b_{ij}$, d^N can differ significantly from d.

If we now consider Algorithm 2 then the direction d is given by $d = -B^{-1}c$ where B is computed as above. If Algorithm 2 is applied to a discretization then d^N is given by $d^N = -(B^N)^{-1}c$, which in the limit has the same direction as d. Computational tests on the problems in [3] confirm that Algorithm 2 gives the same number of iterations when applied to a discretization of the semi-infinite linear program as it does when Simpson's Rule is used to perform the quadrature. Furthermore, computational results show Algorithm 2 outperforms Algorithm 1.

In conclusion, it is clear that the natural method to extend to the semi-infinite case is Algorithm 2. One can similarly define generalizations of the finite dimensional

interior point methods which use logarithmic barrier functions. However, the main drawback to obtaining an efficient interior point method for semi-infinite programming is in the computation of a step length. To ascertain the maximum step at any iteration, it appears that the method has to find the global maximum of the function

$$\frac{-z^{(k)}(s)}{a(s)^{\mathrm{T}}d}$$

over $s \in S$. This would seem to be a difficult problem.

References

[1] I. Adler, M.G.C. Resende, G. Veiga and N.K. Karmarkar, "An implementation of Karmarkar's algorithm for linear programming," *Mathematical Programming* 44 (1989) 297–336.

[2] I.I. Dikin, "Iterative solution of problems of linear and quadratic programming," *Soviet Mathematics Doklady* 8 (1967) 674–675.

[3] M.C. Ferris and A.B. Philpott, "An interior point algorithm for semi-infinite linear programming," *Mathematical Programming* 43 (1989) 257–276.

[4] R.J. Vanderbei, "Affine-scaling for linear programs with free variables," *Mathematical Programming* 43 (1989) 31–44.