Annals of Operations Research 43(1993)229–248

The empirical performance of a polynomial algorithm for constrained nonlinear optimization

Dorit S. Hochbaum*
School of Business Administration and Department of IEOR, University of California, Berkeley, CA 94720, USA

and

Sridhar Seshadri
School of Business Administration, University of California, Berkeley, CA 94720, USA

Abstract

In [6], a polynomial algorithm based on successive piecewise linear approximation was described. The algorithm is polynomial for constrained nonlinear (convex or concave) optimization, when the constraint matrix has a polynomial size subdeterminant. We propose here a practical adaptation of that algorithm with the idea of successive piecewise linear approximation of the objective on refined grids, and the testing of the gap between lower and upper bounds. The implementation uses the primal affine interior point method at each approximation step. We develop special features to speed up each step and to evaluate the gap. Empirical study of problems of size up to 198 variables and 99 constraints indicates that the procedure is very efficient and all problems tested were terminated after 171 interior point iterations. The procedure used in the implementation is proved to converge when the objective is strongly convex.

0. Introduction

In this paper, we consider the nonlinear minimization (maximization) problem:

$$\min\left\{\sum_{i=1}^{n} f_i(x_i) \mid Ax \geq b\right\}.$$

The f_i's are convex (concave) functions (with no additional properties assumed). The polyhedron $\{x|Ax \geq b\}$ is bounded, or alternatively, if unbounded, we assume

*Supported in part by the Office of Naval Research under Grant No. N00014-88-K-0377 and Grant No. ONR N00014-91-J-1241.

that there is a known bound on the optimal solution which, when incorporated into the constraint set, will convert it into a (bounded) polytope. A is an $m \times n$ integer matrix and b an integer vector of dimension m.

In [6], an algorithm is described for solving the above nonlinear problem. Although this algorithm runs in polynomial time for constraint matrices with small subdeterminants, the algorithm may not be practically efficient. The idea of this algorithm is to scale the variables along a grid, and then to use the piecewise linear approximation of the objective function on that grid. Such a method is usually referred to in the literature as successive approximations. This algorithm, in contrast to others, specifies the refinement of the grid, which together with a proximity theorem on the distance between a scaled solution to the piecewise linear problem and the optimal solution, guarantees a polynomial running time. From the proximity theorem in [6], it follows that the solution to the linear programming problem can serve as a center to a new "box" for the variables, which in each dimension is half the previous box. It is proved that the optimal solution to the original problem lies in that smaller box. The number of variables in the linear programming resulting from the piecewise linear approximation depends on the refinement of the grid. Since, in general, the running time of any linear programming algorithm depends on the number of variables, this is the main impediment to the direct usage of the algorithm of [6].

Here, we propose a practical adaptation of this algorithm. The piecewise linear program need not be solved to optimality to guarantee the same result as in the polynomial algorithm. Moreover, instead of the usually large number of grid points required by the algorithm, we use only 8 grid points. After solving the linear programming problem in an iteration, the solution (if it passes a test) serves as a center to a new box used in the next iteration. This new box is contained in the previous one, and its sides are 1/4 of the previous ones. This is unless that solution lies at one endpoint of any of the intervals. In that case, we use Lagrangian relaxation to determine a new center for the box. Consequently, this procedure does not guarantee a reduction of volume at each step, but it works well in practice. We prove that this adjustment is similar to a step taken under some reduced gradient projection. If the solution is always interior to the box, then the algorithm runs in polynomial time. Otherwise, it is proved that when the objective is strongly convex [12], the algorithm terminates in a finite number of iterations. As it turns out, in our experiments the linear programming solution found was rarely on the boundary, and hence we achieved a reduction in volume by a factor of $1/4^n$ in most iterations.

Another aspect of our implementation is the use of the interior point method (the primal affine algorithm, see [10] or [1]). Here, our implementation takes into consideration the fact that the linear programming problem to be solved is of a special structure. First, all variables are in intervals, that is, they have upper and lower bounds. Therefore, a desirable feature of the procedure is the implicit handling of those bounds. Furthermore, the linear program contains eight duplicated columns for each of the n variables in the original problem. We develop an efficient

implementation of the interior point method for this case, so the number of operations required for the problem with $8n$ variables is only $104n$ more per iteration compared to a problem on n variables.

In [6], a tolerance of ε in the solution space was specified as a stopping criterion. Here, we decide on termination based on tolerance on the relative error. In order to evaluate the error, we use the same Lagrangian relaxation procedure as above to deliver a lower bound. Hence, this Lagrangian relaxation procedure serves a dual purpose. The first is a use as a lower bound, and the second is to identify a new center for the subsequent iteration, as described earlier. The Lagrangian relaxation bound together with linear programming solution determines an upper bound on the error. In our experiments, the tolerance level on the relative error was set to 0.0001. We have used randomly generated problems as well as real-world problems in the experiments. A solution within such relative error was found for all tested problems within less than 171 iterations of the interior point method.

The plan of the paper is as follows. In section 1, we review the features of the algorithm in [6] and introduce terminology and notation. Section 2 describes the implementation of the primal affine method when there are multiple copies of each column of the constraint matrix. In section 3, we give the Lagrangian relaxation lower bound and the derivation of one endpoint of the updated interval. In section 4, we give a comprehensive description of the algorithm, and in section 5 a proof of convergence. Section 6 describes the empirical study and the generation of the problems studied. Section 7 has concluding remarks.

1. Review of the successive piecewise approximation algorithm

In this section, we give an overview of ref. [6], which will be used as a framework for the implementation. The proofs to all claims in this section are in [6].

The algorithm maintains a box that contains an optimal solution to the problem. At each iteration, the volume of this box is reduced by a factor of 2^n. This is achieved by reducing each dimension of the box by a factor of 2. A direct implementation will therefore require a total of $\log_2 B$ iterations, where B is the length of a side of the initial cube for the integer problem. For the continuous problem solved to ε-accuracy, there are $\log_2(B/2\varepsilon)$ such iterations.

At a given iteration, the interval for each variable x_i is divided into a grid of $O(n\Delta)$ points, where Δ is the bound on the absolute value of a subdeterminant of the matrix A. The nonlinear function is approximated by a piecewise linear function on that particular grid. Due to the convexity, this piecewise linear approximation can be solved as an ordinary linear program (see, for example, [4, pp. 482–486]).

The main result that makes the algorithm polynomial (for a polynomial Δ) is a proximity result between the optimal solution to the nonlinear problem (integer or continuous) and the optimal solution derived for the piecewise linear approximation. This proximity is a function of n, Δ and the size of the grid (i.e. the scaling

constant). The choice of the grid size is such that an optimal solution to the piecewise linear approximation is at most (1/4)th the length of the box away from an optimal solution. This allows us to update the box in which the optimal solution is to be found, and reduce its size by a factor of 2^n.

The iterations continue until the optimal solution interval is reduced to a size at most 2ε. The complexity of the algorithm is summarized in the following theorem:

THEOREM 1.1

Let the complexity of Linear Programming $\text{Min}\{cx | Ax \geq b, 0 \leq x \leq 1\}$ be $T(n, m, \Delta)$; then the complexity of solving for an ε-accurate optimal solution to a nonlinear separable and convex (concave) minimization (maximization) problem on $\{x | Ax \geq b\}$ is $\log_2(B/2\varepsilon(T(8n^2\Delta, m, \Delta)$.

We now formally define the algorithm of successive piecewise linear approximations. Let $f_i : \mathbb{R} \to \mathbb{R}$, $i = 1, \ldots, n$ be n convex functions and define

$$F(x) := \sum_{i=1}^{n} f_i(x_i), \quad x := (x_1, \ldots, x_n) \in \mathbb{R}^n. \tag{1.2}$$

We are interested in the solutions to the nonlinear programming problem

(RP) minimize $F(x)$

subject to $Ax \geq b$.

Here, A is an integral $m \times n$ matrix and b is an m-vector. The solution to (RP) will be obtained by solving a sequence of scaled and linearized versions of (RP). For any scaling constant $s \in \mathbb{R}_+$, let the scaled problem (RP-s) be defined by

(RP-s) minimize $F(sy)$

subject to $Ay \geq b/s$.

By setting $x = sy$ in (RP-s), it can be observed that it is identical to (RP). For any $s > 0$, let $f_i^{L:s} : \mathbb{R} \to \mathbb{R}$ be the piecewise linearized version of f_i such that $f_i^{L:s}$ takes the same value as f_i at all integer multiples of s: that is, $f_i^{L:s}(sy) = f_i(sy)$ for y integer and

$$f_i^{L:s}(x_i) = \left(\left\lfloor \frac{x_i}{s} + 1 \right\rfloor - \frac{x_i}{s}\right) f_i\left(s\left\lfloor \frac{x_i}{s} \right\rfloor\right)$$

$$+\left(\frac{x_i}{s} - \left\lfloor \frac{x_i}{s} \right\rfloor\right) f_i\left(s\left\lfloor \frac{x_i}{s} + 1 \right\rfloor\right), \quad x_i \in \mathbb{R}, \tag{1.3}$$

where $\lfloor x_i/s \rfloor$ is the largest integer value smaller than or equal to x_i/s. Clearly, $f_i^{L:s}$ is a piecewise linear function which is convex if f_i is convex. Now define

(LP-s) minimize $F^{L:s}(x)$

subject to $Ax \geq b$,

where

$$F^{L:s}(x) := \sum_{i=1}^{n} f_i^{L:s}(x_i), \quad x \in \mathbb{R}^n. \tag{1.4}$$

Since the optimal solution is enclosed in a bounded box, we will incorporate those bounds into the feasible solution set. That is, the following constraints are added to the problem (RP):

$$L_i \leq x_i \leq U_i, \quad i = 1, \ldots, n.$$

These constraints will be scaled as well. In the procedure, at iteration k, we shall work with the upper and lower bounds $U_i^{(k)}$ and $L_i^{(k)}$ and a scaling constant s, such that the length $(U_i^{(k)} - L_i^{(k)})/s$ is an integer constant independent of i. Denote $N = (U_i^{(k)} - L_i^{(k)})/s$. Each variable x_i is substituted by a sum of N 0–1 variables:

$$x_i = s\left\{\left\lfloor \frac{L_i}{s} \right\rfloor + \sum_{j=1}^{N} z_{ij}\right\}, \quad 0 \leq z_{ij} \leq 1 \text{ for all } i \text{ and } j. \tag{1.5}$$

With this substitution, (LP-s) is a piecewise linear convex (concave) minimization (maximization) problem on the variables z_{ij}.

The modified objective function for the linear programming formulation (e.g. [4, pp. 482–486]) is

$$\min \sum_{i=1}^{n} f_i^{L:s}\left(s\left\lfloor \frac{L_i}{s} \right\rfloor\right) + \sum_{i=1}^{n} \sum_{j=1}^{N} \left[f_i^{L:s}\left(s\left(\left\lfloor \frac{L_i}{s} \right\rfloor + j\right)\right) - f_i^{L:s}\left(s\left(\left\lfloor \frac{L_i}{s} \right\rfloor + j - 1\right)\right)\right] z_{ij}.$$

Denoting the columns of A by $a_1, \ldots, a_n$, then the constraint set

$$\sum_{i=1}^{n} a_i x_i \geq b$$

is converted using the substitution (1.5) into the constraint set

$$\sum_{i=1}^{n} \sum_{j=1}^{N} a_i z_{ij} \geq b',$$

where $b' = b/s - \sum_{i=1}^{n} a_i \lfloor L_i/s \rfloor$. So the linear programming version of the (LP-s) problem (omitting the constant from the objective function) is

(LP′-s)

$$\text{minimize} \quad \sum_{i=1}^{n} \sum_{j=1}^{N} \left[f_i^{L:s}\left(s\left(\left\lfloor \frac{L_i}{s} \right\rfloor + j \right) \right) - f_i^{L:s}\left(s\left(\left\lfloor \frac{L_i}{s} \right\rfloor + j - 1 \right) \right) \right] z_{ij}$$

$$\text{subject to} \quad \sum_{i=1}^{n} \sum_{j=1}^{N} a_i z_{ij} \geq b', \quad 0 \leq z_{ij} \leq 1, \quad j = 1, \ldots, N, \quad i = 1, 2, \ldots, n.$$

Hereafter, we shall refer to $1/s$ of the coefficients of z_{ij} in (LP′-s) as c_{ij}. That is,

$$\frac{1}{s}\left[f_i^{L:s}\left(s\left(\left\lfloor \frac{L_i}{s} \right\rfloor + j \right) \right) - f_i^{L:s}\left(s\left(\left\lfloor \frac{L_i}{s} \right\rfloor + j - 1 \right) \right) \right] = c_{ij}. \tag{1.6}$$

From well-known results (e.g. [4]), one has:

LEMMA 1.6

Let $\hat{z}$ be an optimal solution to (LP′-s). If f_i is convex for each $i = 1, \ldots, n$, then $\hat{x}$ defined by $\hat{x}_i = s\lfloor L_i/s \rfloor + \sum_{j=1}^{N} \hat{z}_{ij}$, $i = 1, \ldots, n$, is an optimal solution to (LP-s).

Note that in (LP′-s) the constraint matrix is A^N with each column of A duplicated precisely N times, $A^N = [a_1, \ldots, a_1; a_2, \ldots, a_2; \ldots a_n, \ldots, a_n]$.

Let γ be such that $2^{\gamma+1} n\Delta \geq B$, where B is an upper bound on the length of the interval containing each variable. In particular, $\gamma = \lceil \log_2(B/n\Delta) \rceil - 1$ satisfies this inequality. Note that an optimal solution to

$$\begin{aligned} \text{(RP′)} \qquad & \text{minimize} \quad F(x) \\ & \text{subject to} \quad Ax \geq b, \\ & \qquad\qquad -2^{\gamma+1} n\Delta be \leq x \leq 2^{\gamma+1} n\Delta be \end{aligned}$$

is also an optimal solution to (RP). A feasible solution $\hat{x}$ for (RP) is said to be ε-accurate optimal if there exists an optimal solution x^* to (RP) such that $\|\hat{x} - x^*\|_\infty \leq \varepsilon$. The following algorithm gives an ε-accurate optimal solution for (RP). We call this algorithm *Successive Piecewise Linear Approximation*, or *SPLA*.

ALGORITHM SPLA

Step 1: Let $K = \lceil \gamma + \log_2 n + \log_2 \Delta - \log_2 \varepsilon \rceil + 1$, $s_k = 2^{\gamma-k}$, $k = 0, \ldots, K$, $\hat{x}^{(0)} = 0$.

Step 2: For $k = 1, \ldots, K$, using the substitution (1.5) and solving the resulting linear program, obtain an optimial solution $\hat{x}^{(k)}$ to

$$(\mathrm{LP}^*\text{-}s_k) \qquad \begin{aligned} &\text{minimize} && F^{L:s_k}(x) \\ &\text{subject to} && Ax \geq b, \\ & && \hat{x}^{(k-1)} - 2ns_{k-1}\Delta e \leq x \leq \hat{x}^{(k-1)} + 2ns_{k-1}\Delta e. \end{aligned}$$

The number of times step 2 is applied, i.e. a linear programming problem is solved, depends only on K.

THEOREM 1.7

Algorithm SPLA produces $\hat{x}^{(K)}$, which is an ε-accurate optimal solution for (RP). The running time of algorithm SPLA is $\lceil \log_2 B/\varepsilon \rceil T(8n^2\Delta, m, \Delta)$.

Consequently, the number of applications of the linear programming problem depends only on the upper bound. Each linear programming, however, consists of a number of columns that is at least one order of magnitude more than in the original nonlinear problem. Since only n of those columns are distinct, in the next section we shall show how to implement the algorithm while taking advantage of this column duplication. Our algorithm varies from SPLA in the updating of the intervals' lower and upper bounds and in the number of grid points set in each interval, which is 8 rather than $4n\Delta$.

2. The implementation of the Primal Affine algorithm with duplicated columns

We describe the modification of the Primal Affine algorithm for solving a linear program in bounded variables and repeated columns. The reader is referred to [10] for the theoretical details of the Primal Affine algorithm and an implementation for bounded variables. Here, we describe an efficient adaptation for repeated columns that is novel.

Consider first a linear program (LP), of size $m \times n$, and upper bounds on each variable x_i, $i = 1, 2, \ldots, n$.

$$(\mathrm{LP}) \qquad \begin{aligned} &\text{minimize} && \sum_{i=1}^{n} c_i x_i \\ &\text{subject to} && Ax \geq b, \\ & && 0 \leq x_i \leq u_i, \quad i = 1, 2, \ldots, n. \end{aligned}$$

The Primal Affine algorithm proceeds as follows:

PRIMAL AFFINE ROUTINE FOR SOLVING LP WITH UPPER BOUNDS

Step 0: $\alpha = 0.97$, $e = 1.0e-06$

Add a column to A, equal to b minus the sum of the columns of A. Assign an artificial variable with a high cost (typically 1.0e06) to this column. Set the initial solution equal to the $(n+1) \times 1$ sum vector e.

Step 1: $D_x = \text{diag}(\min(x_i, u_i - x_i))$

Step 2: $P_x = 1 - D_x A^T (A D_x^2 A^T)^{-1} A D_x$

Step 3: $C_p = P_x D_x c$; $w = (A D_x^2 A^T)^{-1} A D_x^2 c$

Step 4: $g = \max_i(\max((e_i D_x C_p)/x_i, -(e_i D_x C_p)/(u_i - x_i)))$

where e_i is the unit vector with a one in the ith position

Step 5: $x = x - (\alpha/g) D_x^2 (c - A^T w)$

Step 6: Check whether the duality gap and dual feasibility conditions are satisfied within ε. If yes, return current solution. Else, go to step 1.

Now consider (LP^r), the linear program with repeated columns, as a modification of (LP). We substitute $x_i = \sum_{j=1}^{a} x_{ij}$ and let $A_{\cdot ij_1} = A_{\cdot ij_2}$, for all $1 \le j_1, j_2 \le a$, to obtain

(LP^r)

$$\text{minimize} \quad \sum_{i=1}^{n} \sum_{j=1}^{a} c_{ij} x_{ij}$$

$$\text{subject to} \quad \sum_{i=1}^{n} \sum_{j=1}^{a} A_{\cdot ij} x_{ij} = b, \quad 0 \le x_{ij} \le u_i; \quad j = 1, 2, \ldots, a; \quad i = 1, 2, \ldots, n.$$

$A_{\cdot ij}$ is the ith column of A. It should be noted here that the major effort involved in solving (LP) is in step 2, where the matrix $AD_x^2A^T$ is inverted. The size of this matrix does not change in (LP^r). Hence, an efficient implementation of the Primal Affine algorithm can be obtained by reducing the computational work involved in the other steps. Denote the matrix with the repeated columns as $\tilde{A}$. Let $\tilde{x}_{ij}$ be $\min(x_{ij}, u_{ij} - x_{ij})$. Then,

$$\begin{aligned} \tilde{A} D_x^2 \tilde{A}^T &= \sum_{i=1}^{n} \sum_{J=1}^{a} A_{\cdot ij} A_{ij}^T \tilde{x}_{ij}^2 \\ &= \sum_{i=1}^{n} A_i A_i^T \left(\sum_{j=1}^{a} \tilde{x}_{ij}^2 \right) \\ &= A D_x'^2 A^T, \end{aligned}$$

where

$$D_x'^2 = \operatorname{diag}\left(\sum_{j=1}^{a} \tilde{x}_{ij}^2 \right). \tag{2.1}$$

Let $\tilde{c}$ be the cost vector of (LP^r). Then we can modify and write

$$\tilde{A} D_x^2 \tilde{c} = Ac',$$

where c' is an $m \times 1$ vector whose ith element is given by

$$(c')_i = \sum_{j=1}^{a} c_{ij} \tilde{x}_{ij}^2. \tag{2.2}$$

In practice, we carry out the calculations in steps 4 and 5 of the Primal Affine algorithm using the values of $\tilde{c}$ and $\tilde{A}^{\mathrm{T}} w$. Using (2.1) and (2.2), we can obtain w. For calculating $\tilde{A}^{\mathrm{T}} w$, the reductions in (2.1) and (2.2) allow us to write

$$(\tilde{A}^{\mathrm{T}} w)_{ij} = \left(A^{\mathrm{T}} \left(A D_x'^2 A^{\mathrm{T}} \right)^{-1} Ac' \right)_i. \tag{2.3}$$

The algorithm for (LP^r) is now almost identical to that for (LP), except for the modifications given in (2.1)–(2.3). We present only the analysis of the additional effort and storage required for solving (LP^r) compared to (LP). The extra work arises due to:

(i) determining whether a variable should be kept closer to its upper bound in step 1;

(ii) computing $D_x'^2$ as in (2.1) above in step 2;

(iii) calculating c' as in (2.2) for step 3;

(iv) determining g in step 4;

(v) updating the solution in step 5.

By carrying out (ii) and (iii) together, we save one operation per variable. By (2.3), just one value of $(A^{\mathrm{T}} w)_{ij}$ suffices for all $j = 1, 2, \ldots, a$. For (iv), we first compute $z = c_{ij} - (\tilde{A}^{\mathrm{T}} w)_{ij}$. Then, if this quantity is positive and the value of x_{ij} is closer to zero than its upper bound, compute z/x_{ij}. Else, if z is negative and x_{ij} is closer to its upper bound u_{ij}, we compute $-z/(u_{ij} - x_{ij})$. This sequence of calculations requires at most two comparisons, a multiplication and a division.

The additional computation per iteration can then be evaluated as follows:

	For calculating	Number of operations
(i)	D_x in step 1	$2an$
(ii)	$D_x'^2$ in step 2	an

	For calculating	Number of operations
(iii)	$\sum_{j=1}^{a} c_{ij}\tilde{x}_{ij}^2$ in step 3	$3an$
(iv)	Max(max(. . .)) in step 4	$5an$
(v)	New solution in step 5	$2an$
	Total	$13an$

The additional storage requirement of (LP^r) with respect to (LP) is due to storing the cost associated with each repeated column, the larger solution vector, and whether a variable has been set to its upper bound or not for each repeated column. In practice, in order to speed up computations, it is found necessary to provide extra storage of $[2n + 4an]$ double precision numbers. Thus, it is seen that (LP^r) can be solved using the Primal Affine algorithm with additional storage of $[2a + 4an]$ double precision numbers, and added computational effort of $13an$ operations per iteration compared to (LP).

3. The Lagrangian relaxation lower bound

The idea of using a Lagrangian relaxation to bound the solution appears in [7]. Here, we adopt this idea with a particular set of multipliers.

We call the vector w the *vector of dual variables* corresponding to a primal solution x. w is given by

$$w = \left(AD_x^2A^{\mathrm{T}}\right)^{-1}AD_x^2c,$$

where c is the linearized objective function vector (c_{ij}) in (LP'-s). Alternatively, we may replace c by the gradient of the objective function at x. With such a substitution, the procedure of finding $\bar{x}$, described below, is identical to finding the next iteration vector in a reduced gradient method. It is readily checked that for the submatrix B of A corresponding to the basic columns, $w = (B^{\mathrm{T}})^{-1} \cdot c_B$. w is also dual feasible if, for the primal vector x, it satisfies

$$f_j'(x_j) - A^{\mathrm{T}} \cdot w \geq 0.$$

If w is not dual feasible, we use it as a vector of multipliers in the Lagrangian relaxation of the original nonlinear problem:

$$L(w) = \min_{L^{(0)} \leq x \leq U^{(0)}} \{F(x) - w^{\mathrm{T}}(Ax - b)\}. \tag{3.1}$$

This problem is separable into n subproblems:

$$L(w) = \sum_{j=1}^{n} \min_{L_j^{(0)} \leq x_j \leq U_j^{(0)}} \{f(x_j) - w^{\mathrm{T}}(A_{\cdot j}x_j)\} + w^{\mathrm{T}}b.$$

We denote the solution to $L(w)$ by $\bar{x}$. To identify $\bar{x}_j$, we check the value of $f'(x_j) - w^T A_{.j}$ at $L_j^{(0)}$ and $U_j^{(0)}$. If these derivatives have opposite sign, we solve for

$$f'(x_j) - w^T A_{.j} = 0. \tag{3.2}$$

Otherwise, we compare the value of $f(x_j) - w^T(A_{.j}x_j)$ at the endpoint of the interval, and set $\bar{x}_j$ to be the endpoint of the lower value.

$L(w)$ is obviously a lower bound, and it is used to estimate an upper bound on the relative error. The solution $\bar{x}$ is also used for another purpose. We make sure that $\bar{x}$ is included in the new box. If $\bar{x}_j$ falls outside the current interval $[L_j, U_j]$, then the update of the box is done so that $\bar{x}_j$ is an endpoint of the updated interval for x_j. Since $\bar{x}$ is a point that would have been derived by a reduced gradient method, the number of iterations required by our algorithm in case the new solution falls outside the interval (and hence the box is no longer shrinking) can be only less than the number required by a reduced gradient method that uses w.

One nice property of $\bar{x}$ is that it usually falls very close to the solution x. More precisely, if x_j is interior to the current interval, i.e. lies at least one grid point away from either boundary of the current interval $[L_j^{(k)}, U_j^{(k)}]$, then $\bar{x}_j$ falls within one grid point away from x_j, as proved in the following lemma.

LEMMA 3.3

For A of full row rank, and for $i = 1, \ldots, n$, $\bar{x}_i$ lies within a single grid point away from x_i, when x_i is interior.

Proof

Let x be an optimal solution to the linear programming problem (LP'-s). That is, x is derived from the solution z according to (1.5).

The optimal solution z to the linear programming problem (LP'-s) has a particular structure. For each index i, the vector (z_{ij}) has at most one fractional value (basic variable) between 0 and 1, say, for $z_{i\alpha_i}$, while all z_{ij}, for $j < \alpha_i$, are equal to 1, and z_{ij}, for $j > \alpha_i$, are equal to 0. (Recall that a variable can be nonbasic at either the lower or upper bound.) If for some i, z_{ij} is basic, we shall refer to x_i as a basic variable. If a variable x_i is nonbasic, then there exists an index β_i such that for $j \le \beta_i$, $z_{ij} = 1$, and for $j > \beta_i$, $z_{ij} = 0$.

We now identify a new vector x' solving the equations (3.2). If x_i is basic, then, due to convexity and from the definition of $c_{i\alpha_i}$ in (1.6), there exists an x_i' and a $z'_{i\alpha_i}$ such that $x_i' = s\{\lfloor L_i/s \rfloor + \sum_{j=1}^{\alpha_i - 1} z_{ij} + z'_{i\alpha_i}\}$, and such that $f_i'(x_i') = c_{i\alpha_i}$. The variable x_i is basic, hence $c_{i\alpha_i} = c_B B^{-1} A_{.i} = w^T A_{.i}$.

For x_i nonbasic, the reduced costs satisfy

$$c_{i\beta_i} - c_B B^{-1} A_{.i} \le 0, \quad c_{i,\beta_i+1} - c_B B^{-1} A_{.i} \ge 0.$$

From the convexity, there exist $x_i^{(1)}$ and $x_i^{(2)}$ satisfying

$$f_i'(x_i^{(1)}) - c_B B^{-1} A_{\cdot i} \leq 0, \quad f_i'(x_i^{(2)}) - c_B B^{-1} A_{\cdot i} \geq 0.$$

Hence, there is an x_i' in the interval $[s\{\lfloor L_i/s \rfloor + \beta_i - 1\},\ s\{\lfloor L_i/s \rfloor + \beta_i + 1\}]$ such that $f_i'(x_i') - c_{i\beta_i} = 0$.

Since $c_{i\beta_i} = c_B B^{-1} A_{\cdot i} = w^{\mathrm{T}} A_{\cdot i}$, the desired result follows. □

4. A description of the algorithm

In section 2, we showed that the additional storage requirements and computational effort per iteration vary linearly with the number of repeated columns in the linear program. If the algorithm SPLA were to be used to solve the separable convex program, then the number of repeated columns would depend on the size of the maximum subdeterminant of the constraint matrix A and the number of variables. This would be large in practice. Therefore, instead of using the linearization scheme suggested in SPLA, we propose the use of eight segments for each variable. Although this approach does not guarantee the running time as in SPLA, it is implementaed with a monitoring scheme to verify that the optimal solution is not missed, and to provide the correction when necessary. As it turns out in our empirical study, the monitoring scheme never activates the correction mechanism. This indicates that the theoretical result can be satisfied in many practical cases with less generous computational requirements. The algorithm is initiated with given upper and lower bounds for each variable and stopping conditions given in terms of the duality gap. The algorithm proceeds as follows:

EIGHT SEGMENT ROUTINE

Input: δ, A, b, U, L, α, ε; **Output:** *Feasibility check,* and δ *optimal solution*

Step 0: $\alpha = 0.97$, $\varepsilon = 1.0\text{e} - 06$ are the parameters for the Primal Affine algorithm. U and L are the initial upper and lower bounds, U_j, L_j, $j = 1, 2, \ldots, n$. The grid size is intialized to $\delta_j = (U_j - L_j)/8$, $j = 1, 2, \ldots, n$.

Step 1: Using the current grid size, and upper and lower bounds, solve ($\mathbf{LP}^{\mathbf{r}}$) with eight repeated columns for each variable. If ($\mathbf{LP}^{\mathbf{r}}$) is infeasible, then report that the original problem is infeasible, and stop. Else report the optimal solution x^*, $w(x^*)$, and objective function value $F(x^*)$ of ($\mathbf{LP}^{\mathbf{r}}$).

Step 2: Check if x^* and $w(x^*)$ satisfy the dual feasibility condition (3.2). If yes, go to step 6. Else

Step 3: Compute a lower bound on the optimal solution to the original problem using $w(x^*)$ and by solving $L(w) = \min(f(x) - w^{\mathrm{T}}(Ax - b))$. Let the minimum be attained at $x = \bar{x}$.
Set $Gap = |(L(w) - F(x^*))/F(x^*)|$.

Step 4: *Gap* is the upper bound on the duality gap as a ratio of the current value of the objective function. If $Gap < \delta$, go to step 6.

Step 5: Compute new bounds:
If $\bar{x}_j < x_j^*$, then $L_j = \bar{x}_j$, $U_j = x_j^* + (x_j^* - \bar{x}_j)/7$
Else $L_j = x_j^* - (\bar{x}_j - x_j^*)/7$, $U_j = \bar{x}_j$
Update grid size and go to step 1.

Step 6: Output the current solution x^*, and the duality gap estimate.

In the above algorithm, at each stage the objective functions $f_j(x_j)$ are linearized using eight segments. The grid length is set equal to one-eighth of the difference between the current upper and lower bounds for each variable. The Primal Affine algorithm is used to solve the resulting linear program with repeated columns (LP^r) with $a = 8$. We check the dual feasibility of the vector x^* with respect to $w(x^*)$ using (3.2), where x^* is the optimal solution to (LP^r) at a given stage and $w(x^*)$ is the dual solution to (LP^r). If x^* is not equal to $\bar{x}$, then a lower bound on the objective function value $L(w)$ is computed as described in section 3. Note that $\bar{x}$ is the point at which the function in (3.1) is minimized. The duality gap is then checked in step 4 of the algorithm. If the gap relative to the current value of the objective function is greater than δ, then the grid is updated in step 5 for the next stage of the algorithm. A variation of this algorithm is presented in the next section that guarantees that the algorithm will terminate in a finite number of iterations, if the objective function is strongly convex. The change is in step 1, where a line search is first made whenever the solution to (LP^r) touches the side of the constraining box. In practice, the need for doing so is rare, as is explained in section 6.

5. A convergence proof

In this section, we prove that the algorithm described in section 4 will terminate in a finite number of steps if the objective function is strongly convex in the constraining box, and a small change is made in the algorithm as explained below. We begin with the definition of strongly convex functions.

A function $f(x)$ from $\mathbb{R}^n \to \mathbb{R}^1$ is defined to be strongly convex on $[L, U]$ if $f(x)$ is twice differentiable and there is a positive constant q that is a uniform lower bound on all the eigenvalues of the Hessian matrix [12].

Let the upper and lower bounds at stage k of the Eight Segment routine be $[L^k, U^k]$ if the solution to (LP^r) x touches the side of the constraining box, i.e. $x_j = L_j^k$ or $x_j = U_j^k$. In addition, this bound is not the original bound, that is, $x_j \neq U_j^0, L_j^0$. We term such a solution *artificially* constrained at this stage.

As defined before, let the initial bounds on the variables define a box $[L^0, U^0]$. Let M and m be the maximum and minimum absolute values of the objective function in this box. We will assume that m is not zero. From the assumption of strong convexity, we will then have the following relation:

$$f(y) \geq f(x) + \nabla f(x)^{\mathrm{T}}(y - x) + q\|y - x\|^2/2; \quad x, y \in [L^0, U^0], \tag{5.1}$$

where $\|\cdot\|$ is the Euclidean norm. Also, since we are optimizing within a given box, there is a constant Q such that

$$Q\|y - x\| \geq |f(y) - f(x)|; \quad x, y \in [L^0, U^0]. \tag{5.2}$$

LEMMA 5.3

The lower bound on the grid size $\delta_{\min}$, below which the value of *Gap* is less than the given δ, is $m\delta/(8Q\sqrt{n})$. This is reached, when the solution at no stage of the Eight Segment routine is artificially constrained, in $N \leq \lceil \log(\max_j(U_j - L_j)8Q\sqrt{n}/(m\delta))/\log 4 \rceil$ steps.

Proof

When the grid size is $\delta_{\min}$, the maximum distance within the box is $8\sqrt{n}\delta_{\min}$. The minimum absolute value of the objective function is greater than m. Because of (5.2), it then suffices to satisfy

$$\begin{aligned} &8Q\sqrt{n}\delta_{\min}/m \leq \delta, \\ \text{or} \qquad &\delta_{\min} \geq m\delta/(8Q\sqrt{n}). \end{aligned} \tag{5.4}$$

However, by lemma 3.3, as the grid size decreases by at least a factor of 1/4 at every stage when the solution is not artificially constrained, we also have the relation

$$\max_j(U_j - L_j)/4^N = \delta_{\min},$$

or

$$N \leq \left\lceil \log\left(\max_j(U_j - L_j)8Q\sqrt{n}/(m\delta) \right) / \log 4 \right\rceil. \tag{5.5}$$

We now modify the Eight Segment routine a little to guarantee convergence for strongly convex functions. Let the solution to (LP^{r}) at stage k be $\hat{x}^k$. Whenever $\hat{x}^k$ is artificially constrained by $[L^k, U^k]$, conduct a line search at the end of step 1 to minimize

$$\min_{\lambda} f(x^{k-1} + \lambda(\hat{x}^k - x^{k-1})), \quad \lambda \in (0, 1].$$

Let the optimal value of λ be λ^* and set $x^k = x^{k-1} + \lambda^*(\hat{x}^k - x^{k-1})$. Further, whenever the grid size of any variable drops below $\delta_{\min}$, the grid size is held at $\delta_{\min}$ for that variable, and ensure that the upper and lower bounds are fixed at least $3\delta_{\min}$ away from the current value of this variable in the next stage. Denote this algorithm as the Modified Eight Segment algorithm (MESA).

LEMMA 5.6

Every time the solution to (LP^r) is artificially constrained at any stage of MESA, we obtain a decrease in the objective function value of at least $q\delta_{\min}^2/8$.

Proof

Let k be an iteration index at which the solution is artificially constrained. First note that x^{k-1} is the solution at the *end* of stage $k-1$ of the Eight Segment routine, is one of the grid points in stage k of the routine, and is at least one grid distance away from the sides of $[L^k, U^k]$. Second that, due to convexity, the values of the piecewise-linear objective function within the box at stage k are greater than or equal to the actual value of the objective function at all points of the box. Therefore, we have

$$f(\hat{x}^k) \leq f(x^{k-1}). \tag{5.7}$$

Let λ^* be the optimal value of λ. Then there are two cases to consider.

Case (i) : $[\lambda^* = 1]$.

In this case, we have $\nabla f(\hat{x}^k)^T(x^{k-1} - \hat{x}^k) \geq 0$. So by (5.1), we obtain

$$\begin{aligned} f(x^{k-1}) &\geq f(\hat{x}) + q\|x^{k-1} - \hat{x}^k\|^2/2 \\ &\geq f(\hat{x}^k) + q\delta_{\min}^2/2. \end{aligned}$$

Case (ii) : $[\lambda^* \in (0, 1)]$.

In this case, either $\|\hat{x}^k - x^k\|$ or $\|x^{k-1} - x^k\|$ is greater than or equal to $\delta_{\min}/2$.

If $\|\hat{x}^k - x^k\| \geq \delta_{\min}/2$, then, as $\nabla f(x^k)^T(\hat{x}^k - x^k) \geq 0$, by combining (5.1) and (5.7), we obtain

$$\begin{aligned} f(x^{k-1}) &\geq f(x^k) + q\|\hat{x}^k - x^k\|^2/2 \\ &\geq f(x^k) + q\delta_{\min}^2/8. \end{aligned}$$

And if $\|x^{k-1} - x^k\| \geq \delta_{\min}/2$, then we directly obtain this inequality using (5.1). □

LEMMA 5.8

MESA, the Eight Segment routine with the line search incorporated as described, will terminate in at most $8(M-m)N/(q\delta_{\min}^2)$ steps, under the assumption of strong convexity.

Proof

From lemma (5.3) and lemma (5.6), we have

$$\text{Number of stages} \leq N(M-m)/(q\delta_{\min}^2/8)$$

$$= 8\left\lceil \log\left(\max_j (U_j - L_j)8Q\sqrt{n}/(m\delta)\right) / \log 4 \right\rceil (M-m)/(q\delta_{\min}^2). \quad \square$$

6. Empirical results

The algorithm MESA has been implemented using *f77* on a SUN workstation running SunOS 4.3, and was tested on several randomly generated problems with dense constraint matrices and two test problems provided by S. Nabar. We report our computational experience in the following sections.

6.1. EXPERIENCE WITH RANDOMLY GENERATED PROBLEMS

The randomly generated problems were created following the scheme described in [10]. The coefficients of the constraint matrix were randomly generated numbers in the range $[-0.5, 0.5]$. To ensure a feasible solution, first a value for each variable was randomly chosen in the range $[0, 5]$. The right-hand side vector was then set equal to the product of the constraint matrix with this solution. The cost functions tested were polynomials including terms up to the fourth power. The coefficients were randomly generated starting with the linear coefficient in the range of $[-3, 3]$, a positive square coefficient in the range $[1, 10]$ and suitable third and fourth power coefficients to guarantee convexity in the range $(0, 1000)$ for each variable. The initial upper bounds for all variables were fixed at 1000 and lower bounds at 0.

Values of $\alpha = 0.97$ and $\varepsilon = 1.0\text{e} - 06$ were used for the Primal Affine algorithm. The relative gap error δ was set to 0.1%. Table 1 summarizes the results for randomly generated problems of size (m, n) ranging from (5, 10) to (95, 195). The number of stages never exceeded eight in these experiments, and the total number of interior point iterations grew very slowly from 137 to 168 with increasing size of the problem instance. This leads to the belief that even for very large problems, the number of interior point iterations would not grow significantly. In fact, given the experience with interior point methods, the governing factor for the total number of iterations will be the number of stages. The number of stages will depend on the speed with which the dual and the primal solution converge to one another, and this speed will be high if very few variables are artificially constrained during the solution procedure. We checked the number of times any variable was artificially constrained in our experiments and term this *AVAC*.

Table 1

Number of iterations for generated problems.

m	n	Lower bound	Primal solution	Number of iterations
5	10	2020.8	2020.9	137
10	20	3444.0	3444.4	142
10	100	2782.4	2783.1	140
20	30	8029.0	8029.6	141
20	40	6597.8	6598.7	141
20	190	15166.1	15168.2	152
25	65	6714.4	6715.4	154
25	75	7498.2	7499.3	152
30	50	6863.1	6864.1	149
30	80	9979.9	9981.2	150
35	195	20344.6	20347.5	158
39	100	11926.0	11927.8	153
39	110	12417.1	12419.1	155
39	120	16078.7	16080.7	152
39	130	10506.9	10508.6	158
39	140	12357.3	12359.5	157
39	150	14524.9	14526.8	157
39	160	14454.0	14456.3	156
39	170	18949.6	18952.5	163
39	180	21773.3	21776.4	160
39	190	19647.4	19650.5	158
60	190	28552.5	28556.4	158
70	190	21397.3	21400.0	171
80	190	26453.4	26457.5	168
90	190	34596.7	34601.0	165
91	191	32566.4	32570.3	163
92	190	28376.9	28380.3	167
95	195	33518.3	33522.4	161
99	198	39001.3	39005.1	166

Table 2 shows the convergence to solution for two instances. The duality gap is seen to reduce by two orders of magnitude at each stage of MESA. In none of the solved problems did the solution ever become artificially constrained (*AVAC* was zero). This would then explain the rate of convergence seen in table 2.

Table 2

Convergence to solution.

Problem size	60 × 190			92 × 190		
	No. of iterations	Lower bound	Primal solution	No. of iterations	Lower bound	Primal solution
Step 1	33	−109e08	685652	34	−141e08	194649
Step 2	28	−436e05	361608	29	−546e05	141638
Step 3	24	−75948	159456	27	−100774	127896
Step 4	22	26019	32922	23	25973	32970
Step 5	20	28345	28952	22	28128	28781
Step 6	17	28538	28556	18	28349	28406
Step 7	14	28552	28556	14	28376	28380
	158			167		

6.2. EXPERIENCE WITH TEST PROBLEMS

SAMPLE 1 is a stratified sampling problem created by S. Nabar using data provided by G. Easton. The objective is to minimize the number of points sampled subject to the constraints on the sampling error. Each of the variables in this problem has a pre-specified upper and lower bound. The number of variables is 35 and the number of constraints is 13. We solved a relaxed version of the problem with continuous variables. The constraints of the original problem involve the reciprocal of each variable, and we change these into linear constraints by transforming the variables. The upper bound of each transformed variable is unity. Hence, the problem can be "solved" in one iteration using a large number of pieces. We solved the problem using 80 pieces and a single stage to compare with the performance of MESA, which used 8 pieces and several stages depending on the duality gap and the specified stopping criterion. The results are shown in table 3. MESA took four stages and a total of 150 interior point iterations to solve the problem to 0.017% of the best lower bound in 11.0 seconds of CPU time. In contrast to the randomly generated problems, *AVAC* was found to be four. The 80-piece approach needed 89 iterations of the interior point method, gave a duality gap estimate of 0.094%, and took more CPU time (18.2 sec).

NET3 is a Network problem created by S. Nabar and L. Schrage. There are 24 variables and 18 constraints. The objective is to minimize the cost of establishing links between three cities and the cost of traffic delays. The congestion costs associated with traffic are convex functions of the load on a link. We solved a relaxed version of the problem, and obtained an integer solution to the three decision variables corresponding to the links. Once again, the upper bounds on the decision

Table 3

Test problems.

Problem	Method	Number of iterations	Best lower bound	Primal solution	Gap [%]	CPU time [sec]
SAMPLE1	MESA	150	267.9805	268.0256	0.017	11.0
	80 piece	89	267.9805	268.2312	0.094	18.2
NET3	MESA	90	212034.5311	212034.5332	1.0e – 06	5.1
	80 piece	31	212034.5311	212035.6774	1.0e – 04	3.5

variables are small (less than 24), so we compared MESA with the 80-piece approach. From table 3, it is seen that both solution approaches gave very high accuracy (within 10^{-4}% of the best lower bound). The total number of interior point iterations used in MESA was 90 compared to the 31 iterations for the 80-piece method. We observed that the solution was artificially constrained 11 times in MESA, i.e. *AVAC* equalled 11.

7. Concluding remarks

We present an implementation of a polynomial algorithm that delivers a solution within a prescribed accuracy in polynomial time which depends on the magnitude of the largest subdeterminant of the linear constraint matrix. A direct implementation of this algorithm is not efficient, since the grid along which we piecewise approximate the objective function is too fine. To circumvent this problem, our approach combines several elements. One is a heuristic approach where we use eight (8) segments in the grid instead of the prescribed number. This may result in violation of the invariant property of the polynomial algorithm, namely that the optimal solution may no longer exist in the shrinking box in which we search for a solution. We incorporate a detection mechanism to find out whether this condition holds, and use an alternative technique to take a corrective action.

Although our approach does not yield a provably polynomial algorithm, in the empirical study this corrective action had rarely to be invoked. Hence, each implementation was de facto polynomial and fast. We show that the correction is such that the algorithm behaves no worse than some reduced gradient method, and it converges in a finite number of steps when the function is strongly convex.

Another feature of the algorithm is the use of an interior point method to solve the linear programs. We use a special implementation of the interior point that efficiently handles the feature of duplicated columns. Although the duplicated columns increase the size of the linear program by a factor of eight, the running time is only increased by an additive linear function.

This algorithm can also be used in the case that the objective function is quadratic. Preliminary testing on quadratic cost network flow problems of small size shows the same computational behavior as seen above. Of course, for this class of problems there are other efficient solution procedures available [9]. It will be interesting to compare the performance of our algorithm with existing procedures.

This implementation is yet another illustration that the average performance of an algorithm is frequently much better than indicated in complexity analysis of worst-case behavior.

Acknowledgements

We are grateful to S. Nabar for providing us with sample problems. We wish to express our thanks to the anonymous referee for his detailed comments, which helped to improve the presentation.

References

[1] E.R. Barnes, A variation on Karmarkar's algorithm for solving linear programming problems, Manuscript, IBM T.J. Watson Research Center, Yorktown Heights, NY (1985).

[2] M.S. Bazaraa and C.M. Shetty, *Nonlinear Programming: Theory and Algorithms* (Wiley, 1979).

[3] I.I. Dikin, Iterative solution of problems of linear and quadratic programming, Sov. Math. Dokl. 8(1967)674–675.

[4] G.B. Dantzig, *Linear Programming and Extensions* (Princeton University Press, 1963).

[5] R. Fourer, Simplex algorithm for piecewise-linear programming I: Derivation and proof, Math. Progr. 33(1985)204–233.

[6] D.S. Hochbaum and J.G. Shanthikumar, Convex separable optimization is not much harder than linear optimization, J. ACM 37(1990)843–862.

[7] R.R. Meyer, Computational aspects of two-segment separable programming, Math. Progr. 26(1983) 21–39.

[8] A. Premoli, Piecewise linear programming: The compact (CPLP) algorithm, Math. Progr. 36(1986) 210–227.

[9] R.T. Rockafellar, *Network Flows and Monotropic Optimization* (Wiley, 1984).

[10] R.J. Vanderbei, M.S. Meketon and B.A. Freedman, A modification of Karmarkar's linear programming algorithm, Algorithmica 1(1986)395–407.

[11] P. Wolfe, Methods of nonlinear programming, in: *Recent Advances in Mathematical Programming*, ed. R.L. Graves and P. Wolfe (1963).

[12] P. Wolfe, Convergence theory in nonlinear programming, in: *Integer and Nonlinear Programming*, ed. J. Abadie (North-Holland, 1970).