MULTIPOINT METHODS FOR SEPARABLE NONLINEAR NETWORKS

by

P. V. Kamesam and R. R. Meyer

Computer Sciences Technical Report #468

November 1982

MULTIPOINT METHODS FOR SEPARABLE NONLINEAR NETWORKS

by

P. V. Kamesam and R. R. Meyer[†]

Abstract

Iterative piecewise-linear approximation methods are considered for separable, convex nonlinear network problems. A comparison is made between "fixed grid" approximations of 2, 4, and 6 segments per variable and "implicit grid" strategies that generate segments as needed, but store at most a 2-segment approximation at any time. It is shown that the implicit grid methods are linearly convergent, and this predicted behavior is confirmed by highly accurate solutions within 7 iterations of problems with up to 2238 variables. Since the computing time per iteration is only slightly more for implicit grids than for fixed grids, the numerical results presented show overall computing times are less for implicit grids. A lower bounding technique based on the error of approximation is also developed. This technique is highly useful if Lagrangian relaxations are difficult to solve.

[†]This material is based upon work supported by the National Science Foundation under Grant No. MCS-8200632

1. Introduction

Throughout this paper we are concerned with algorithms for the following minimization problem:

$$\begin{aligned} \underset{x \in R^n}{\text{minimize}} \quad & f(x) = \sum_{i=1}^{n} f_i(x_i) \\ \text{s.t.} \quad & Ax = b \qquad\qquad (1.1) \\ & \ell \leq x \leq u , \end{aligned}$$

where $\ell \in R^n$, $u \in R^n$, $x = (x_1,\ldots,x_n) \in R^n$, $b \in R^m$, A is an $m \times n$ node-arc incidence matrix for a directed network, and each f_i is a continuous convex function on $[\ell_i, u_i]$.

There are many optimization problems that may be represented in the form (1.1). Examples are estimation of data elements in input-output tables [Bachem and Korte 77], determination of steady-state flows in electrical and water distribution networks [Collins, et al 78], and reservoir control problems [Bertsekas 76]. Details of these problems are given in Section 7.

Problem (1.1) has a simple structure and numerous nonlinear programming algorithms can be readily used or specialized for (1.1). Because of the network constraints, however, it is tempting to consider linearizing the objective function, particularly since in applications problems of type (1.1) tend to be fairly large and the speed of linear network flow codes can therefore be used to best advantage. The proposed algorithms do so by constructing piecewise-linear approximations and then solving a sequence of linear network subproblems.

2. Review of Previous Work

Several authors have proposed a variety of solution approaches. [Collins, et al 78] discusses the Frank-Wolfe method and piecewise linearization of the objective function. [Dembo and Klincewicz 78] and [Beck, et al 83] specialize the reduced gradient method to solve (1.1) and show how some second order information may be used to obtain an approximation to Newton's method. [Bachem and Korte 77] and [Cottle and Duvall 82] propose algorithms for a subclass wherein each f_i is a quadratic function and the constraints define a transportation polytope.

Non-iterative separable programming techniques can also be used. These standard approaches, though readily applicable, have the disadvantage that there is a trade-off between the accuracy of the solution and the size of the linearized subproblems to be solved. [Thakur 78] gives error bounds on the optimal objective value and the optimal solution. [Bazaraa and Shetty 79] discusses an iterative separable programming procedure based on generalized programming.

These separable programming methods use what amounts to a global approximation of the objective function. [Meyer 80] and [Kao and Meyer 81], on the other hand, develop local approximation methods and show these methods to be computationally effective.

The primary advantages of the local approximation methods are (1) the size of the linearized subproblem is kept small, (2) they avoid unnecessary function evaluations which would be required for a global piecewise-linear approximation of the objective function, and (3) in the small neighborhood of the current iterate in which it is used, the approximation is quite accurate.

3. Solution Methodology

In what follows, we discuss two algorithms for (1.1). The first of these is an extension of the algorithms proposed by [Kao and Meyer 81] and ~~uses a local piecewise-linear approximation with a fixed number of segments~~, which in the computational experiments reported in Section 7 was 2, 4, or 6.

The second algorithm uses an implicit global approximation (to the objective function) that is modified each iteration. The approximation is implicitly global in the sense that we initially use only two segments, and generate additional approximating segments only as needed. Computationally, the approximation utilizes at most two piecewise-linear segments per variable at any time. In essence, we try to retain the computational advantages of a local approximation method, while using (in theory) a global approximation to the objective function. A similar implicit grid approach is discussed in [Jensen and Barnes 1980], but there a uniform grid size for all variables is considered and no results are given on lower bounds or convergence.

4. Local Approximation Methods in Separable Programming

These local approximation methods use piecewise-linear approximations over appropriately chosen neighborhoods. In order to describe these methods clearly, we introduce the following notation, and define a local approximation around $\hat{m} \in R^n$.

An ordered triple $(\hat{\ell},\hat{m},\hat{u})$ is said to be admissible if it satisfies the following conditions

$$\ell \leq \hat{\ell} \leq \hat{m} \leq \hat{u} \leq u \,, \tag{4.1}$$

$$\hat{\ell}_i < \hat{m}_i \quad \text{if} \quad \ell_i < \hat{m}_i \,, \tag{4.2}$$

$$\text{and} \quad \hat{m}_i < \hat{u}_i \quad \text{if} \quad \hat{m}_i < u_i \,, \quad i=1,2,..,n. \tag{4.3}$$

An approximating problem $P(\hat{\ell},\hat{m},\hat{u})$ is defined as

$$P(\hat{\ell},\hat{m},\hat{u}): \quad \begin{array}{ll} \min\limits_{x} & \sum \hat{f}_i(x_i) \\ \text{s.t.} & x \in \{x | Ax=b\} \cap [\hat{\ell},\hat{u}] \end{array}$$

where $\hat{f}_i(x_i)$ is a piecewise-linear approximation (with 2s segments) of $f_i(x_i)$ over $[\hat{\ell}_i,\hat{u}_i]$ that agrees with f_i at $\hat{m}_i \pm k\beta$, $k=0,1,...s$, where $\hat{m}_i + s\beta = \hat{u}_i$ and $\hat{m}_i - s\beta = \hat{\ell}_i$. ($\beta > 0$ is the length of the subdivision interval except (perhaps) for the end segments and may depend on i; for simplicity of notation, however, we avoid reference to these dependencies). It is well-known ([Kao and Meyer 81]) that the problem $P(\hat{\ell},\hat{m},\hat{u})$ may be converted into an equivalent linear network flow problem under the convexity assumption on f.

Algebraically, we have

$$\hat{f}_i(x_i) = \begin{cases} f_i(\hat{m}_i) \quad \text{if} \quad x_i = \hat{m}_i \\ f_i(\hat{m}_i + (k-1)\beta) + c_i^{+k}\{x_i - (\hat{m}_i + (k-1)\beta)\} \text{ for } (k-1)\beta \le x_i - \hat{m}_i \le k\beta \\ f_i(\hat{m}_i - (k-1)\beta) + c_i^{-k}\{x_i - (\hat{m}_i - (k-1)\beta)\} \text{ for } -k\beta \le x_i - \hat{m}_i \le -(k-1)\beta \\ \qquad (k=1,2,\ldots,s) \end{cases} \tag{4.1.1}$$

where

$$\begin{aligned} c_i^{+k} &= \{f_i(\hat{m}_i+k\beta) - f_i(\hat{m}_i+(k-1)\beta)\}/\beta \\ -c_i^{-k} &= \{f_i(\hat{m}_i-k\beta) - f_i(\hat{m}_i-(k-1)\beta)\}/\beta \end{aligned} \tag{4.1.2}$$

with the understanding that if $\hat{m}_i = \hat{\ell}_i = \ell_i$, then the approximation is defined only on the interval $[\hat{m}_i, \hat{u}_i]$. Similar comments hold if $\hat{m}_i = \hat{u}_i = u_i$. Because of the convexity of f_i, we have $\hat{f}_i(x_i) \ge f_i(x_i)$ for $x_i \in [\hat{\ell}_i, \hat{u}_i]$. Letting $\hat{S}$ be the feasible region of $P(\hat{\ell},\hat{m},\hat{u})$, this implies that if $\tilde{x} \in \hat{S}$, and $\hat{f}(\tilde{x}) < \hat{f}(\hat{m})$, then $f(\tilde{x}) < f(\hat{m})$. We thus have strict monotonicity of the iterates, since in the algorithms proposed, $\hat{m}$ is the most recently generated feasible solution and a new iterate must yield an improved value of $\hat{f}$. From this viewpoint, $\hat{f}$ is a local piecewise-linear approximation in the neighborhood $[\hat{\ell},\hat{u}]$ of the most recent iterate and $\hat{S}$ is not empty, since it contains $\hat{m}$.

4.2 <u>Error Bounds and Optimality Conditions</u>

Let $\hat{E}_i \equiv \max_{\hat{\ell}_i \le x_i \le \hat{u}_i} [\hat{f}_i(x_i) - f_i(x_i)]$. Then $\hat{E}_i$ satisfies $\hat{E}_i \le \max_{[\hat{\ell}_i,\hat{u}_i]} \hat{f}_i(x_i) - \min_{[\hat{\ell}_i,\hat{u}_i]} f_i(x_i) \le \max_{[\hat{\ell}_i,\hat{u}_i]} f_i(x_i) - \min_{[\hat{\ell}_i,\hat{u}_i]} f_i(x_i)$. Thus, as the length of the interval $[\hat{\ell}_i,\hat{u}_i]$ goes to zero, $\hat{E}_i$ must tend to zero. Letting $E(\hat{\ell},\hat{m},\hat{u}) \equiv \max_{[\hat{\ell},\hat{u}]} [\hat{f}(x) - f(x)] = \sum_{i=1}^{n} \hat{E}_i$, we have

<u>4.2.1 Lemma</u>: If (ℓ^k, m^k, u^k) is a sequence of admissible triples such that $\ell^k \to \bar{m}$, $m^k \to \bar{m}$, $u^k \to \bar{m}$ then $E(\ell^k, m^k, u^k) \to 0$.

The following lemma relating the optimal values of $P(\hat{\ell}, \hat{m}, \hat{u})$ and (1.1) where x^* is optimal for $P(\hat{\ell}, \hat{m}, \hat{u})$ and x^{**} is optimal for (1.1), is taken from [Kao and Meyer 81].

<u>4.2.2 Lemma</u>: If x^* is an optimal solution of an approximating problem $P(\hat{\ell}, \hat{m}, \hat{u})$, where $(\hat{\ell}, x^*, \hat{u})$ is admissible, then the following lower and upper bounds hold for the optimal value z^{**} of (1.1).

$$f(x^*) - \sum_{i=1}^{n} \hat{e}_i \leq z^{**} \leq f(x^*)$$

where

$$\hat{e}_i \geq \max_{\hat{\ell}_i \leq x_i \leq \hat{u}_i} (\hat{f}_i(x_i) - f_i(x_i)) . \tag{4.2.1}$$

The following theorem from [Kao and Meyer 81] gives conditions under which $\bar{x} \in S := \{x \mid Ax=b, \ \ell \leq x \leq u\}$ is an optimal solution of (1.1).

<u>4.2.3 Theorem</u>: Let $\{(\bar{\ell}^k, \bar{x}, \bar{u}^k)\}$ be a sequence of admissible triples such that $\bar{\ell}^k \to \bar{x}$, $\bar{u}^k \to \bar{x}$, where $\bar{x} \in S$. Then $\bar{x}$ is optimal for (1.1) iff $\bar{x}$ is optimal for each of the problems $P(\bar{\ell}^k, \bar{x}, \bar{u}^k)$, $k=1,2,\ldots$.

The first algorithm given below, starts with a $\bar{x} \in S$ and conducts a search procedure at $\bar{x}$ by considering a sequence of approximating problems of the type described in Theorem 4.2.3. If $\bar{x}$ is optimal for (1.1), then the search procedure establishes the optimality of $\bar{x}$ by the above optimality conditions. Otherwise, the search procedure is terminated as soon as an improved feasible solution is obtained. This improved feasible solution is used in the next iteration, where again a search procedure is carried out. This algorithm is very similar to algorithm 2 of [Kao and Meyer 81] except that in their algorithm these authors use an approximating function consisting of at most two piecewise-linear segments. In algorithm (5.1) below, the approximating function may have up to 2s piecewise-linear segments. Thus, algorithm (5.1) is a straightforward extension of algorithm 2 of [Kao and Meyer 81].

5. Algorithm Based on Local Piecewise-Linear Approximation

5.1 Algorithm:

(a) Let $\alpha \in (0,1)$, let $\bar{x}$ be a feasible solution available at the start of the current iteration, and let $\hat{\lambda}$ be a grid size vector (determined by a procedure described below) for the current iteration.

(b) Denoting by $\bar{x}(\delta)$ an optimal solution of $P(\bar{\ell}(\delta), \bar{x}, \bar{u}(\delta))$, where $\bar{\ell}_i(\delta) \equiv \max\{\ell_i, \bar{x}_i - s\delta\lambda_i\}$ and $\bar{u}_i(\delta) \equiv \min\{u_i, \bar{x}_i + s\delta\lambda_i\}$, (with the understanding that $\bar{x}(\delta)$ is taken to be $\bar{x}$ if $\bar{x}$ is optimal for $P(\bar{\ell}(\delta), \bar{x}, \bar{u}(\delta))$), then $\bar{x}$ is optimal for (1.1) if $\bar{x}(\delta) = \bar{x}$ for $\delta = \alpha^0, \alpha^1, \alpha^2, \ldots$.

(c) Otherwise, let $\hat{\alpha}$ be the first power of α such that $f(\bar{x}(\hat{\alpha})) < f(\bar{x})$ and use $\bar{x}(\hat{\alpha})$ as a starting feasible solution for the next iteration, for which a new grid size vector is also generated.

In step (b) of the algorithm, we consider an infinite sequence of approximating problems to test the optimality of $\bar{x}$ for (1.1). In practice, this is avoided by computing a lower bound on the optimal value of (1.1) and terminating the computation if the current solution agrees with the lower bound to a desired accuracy. Theoretically, in order to establish a convergence theorem, step (c) of algorithm (5.1) must be modified and grid size vector generation must obey certain rules (see [Kao and Meyer 81]. Computationally, step (c) is quite adequate, and a good approach to the determination of $\hat{\lambda}$ is described in the next section.

5.2 Determination of the Bounds

From a theoretical point of view, it is necessary to ensure that the search procedure in step (c) be started with bounds $\hat{\ell}$ and $\hat{u}$ sufficiently far away from $\bar{x}$. On the other hand, if $\hat{\ell}$ and $\hat{u}$ are far from $\bar{x}$, then step (c) is time-consuming. The choices of $\hat{\ell}$ and $\hat{u}$ thus have a significant impact on the computational efficiency of the algorithm. We now describe a strategy for determining $\hat{\ell}$ and $\hat{u}$ which proved to be quite satisfactory.

At the beginning of the jth iteration, we have available a feasible solution x^{*j-1}, which is the optimal solution of the (j-1)st iteration along with a set of optimal multipliers μ^* associated with the constraints $\{Ax=b\}$. Consider the following Lagrangian relaxation problem

$$\min_{\ell \leq x \leq u} \{f(x) - \mu^{*T}(Ax-b)\} \tag{5.2.1}$$

Because f is separable, (5.2.1) reduces to

$$\sum_{i=1}^{n} \min_{\ell_i \leq x_i \leq u_i} \{f_i(x_i) - \mu^{*T}A^i x_i\} + \mu^{*T}b, \tag{5.2.2}$$

where A^i is the ith column of A.

It is well known that the minimum value $\underline{z}$ of (5.2.1) gives us a lower bound on the optimal value of (1.1). Let $\hat{x}$ solve (5.2.1). Unless $\hat{x}$ is optimal for (1.1), $\hat{x}$ is not feasible for (1.1). Nevertheless, $\hat{x}$ can be used to determine the initial bounds, $\hat{\ell}$, $\hat{u}$ for the jth iteration. We have used the following rule and found it to be quite satisfactory:

Let λ_i^{j-1} be the grid size for $\hat{f}_i$ in the (j-1)st iteration. Set $\lambda_i^j = \min\,[\max\{\frac{1}{2}\lambda_i^{j-1}, |x_i^{*j-1} - \hat{x}_i|\}, (f(x^{*j-1}) - \underline{z})^{\frac{1}{2}}]$. (The motivation for this rule is that we wish to combine the observations that the errors in the values of the variables are bounded by a multiple of the square root of the function value error (see Appendix for details) and that a large change in the value of a variable in a particular iteration is often followed by a relatively large change in the variable in the succeeding iteration.) In Section 7, we present computational results with Algorithm 5.1 using this heuristic rule.

In the next section, we describe a different algorithm based on implicit global approximation.

6. Implicit Global Approximations in Separable Programming

Separable programming traditionally consists of choosing a fixed grid of points in an interval $[\ell_i, u_i]$ and then approximating the function f_i on $[\ell_i, u_i]$ by the piecewise-linear segments determined by the function evaluations at the grid points. Unless a very fine grid of points in $[\ell_i, u_i]$ is chosen, the resulting error of approximation is likely to be high. [Collins, et al 78] considers the following problem: for a specified number of grid points, determine a grid so that the error of approximation is minimized. After determining the grid, an LP is solved to get an approximate solution to (1.1). They, however, report that determining the grid took more computational effort, than solving the LP (with 8 segments per variable). Even with such strategies, the error of approximation is still likely to be high, unless a very fine grid is used.

We propose an alternative to these traditional approaches. Initially start with a crude approximation defined by a coarse grid. After solving the resulting approximation, set up another approximating problem around the current iterate, this time with a finer grid. Proceeding iteratively, we stop when a solution to a desired accuracy is obtained.

The approximating function used is similar to (4.1.1) except that the approximation is (in theory) not restricted to a neighborhood of the current iterate. Thus $\hat{f}_i$ below is a piecewise-linear approximation of f_i over the interval $[\ell_i, u_i]$.

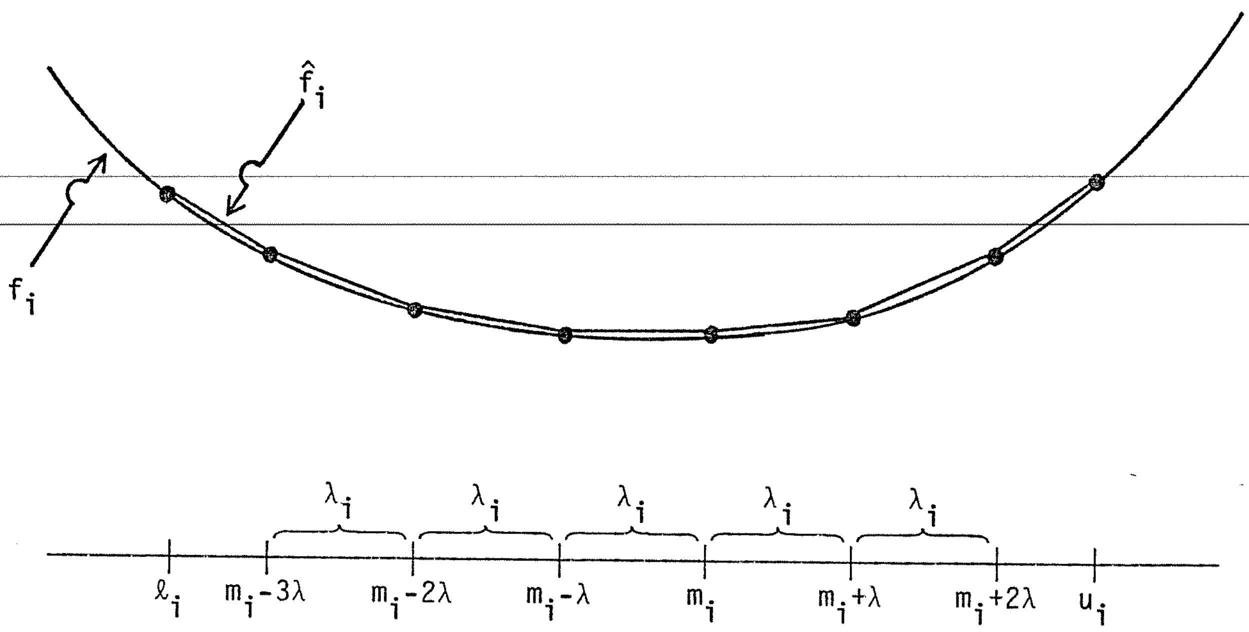

Fig. 6.1: A global piecewise linear approximation $\hat{f}_i$ of f_i.

The approximating function $\hat{f}_i$ agrees with f_i at $m_i \pm k\lambda_i$, k=0,1,2,... and also at the end points ℓ_i and u_i (see Figure 6.1). λ_i is the length of the subdivision interval except (perhaps) the end segments. For simplicity of notation, however, we assume that the length of each segment is equal to λ_i. The difference between this approximating function and (4.1.1) is that the latter is a local approximation restricted to the chosen neighborhood.

Let λ^j be the vector of subdivision interval lengths in the jth iteration. The corresponding approximating problem is

$$AP^j(m^j,\lambda^j): \quad \begin{array}{ll} \min\limits_x & \hat{f}^j(x) = \sum\limits_{i=1}^{n} \hat{f}_i^j(x_i) \\ \text{s.t.} & Ax = b,\ \ell \le x \le u \end{array}$$

<u>6.2 Proposition</u>: Let $\hat{f}_i^j$ be an approximation of f_i over $[\ell_i,u_i]$. Let

$$\varepsilon_i^j \equiv \max_{x_i \in [\ell_i, u_i]} \{\hat{f}_i^j(x_i) - f(x_i)\} .$$

Then

$$\varepsilon_i^j \to 0 \quad \text{as} \quad \lambda^j \to 0 .$$

Further, if

$$\varepsilon^k = \sum_{i=1}^{n} \varepsilon_i^j = \max_{x \in [\ell, u]} \{\hat{f}^j(x) - f(x)\}$$

then

$$\varepsilon^j \to 0 \quad \text{as} \quad \lambda^j \to 0 .$$

<u>Proof</u>: Follows from the continuity of f_i.

<u>6.3 Proposition</u>: Let x^* be an optimal solution of $AP^j(m^j, \lambda^j)$ and x^{**} an optimal solution of (1.1). Then the following lower and upper bounds hold on the optimal value of (1.1).

$$\hat{f}^j(x^*) - \varepsilon^j \leqq f(x^{**}) \leqq f(x^*) .$$

<u>Proof</u>:

$$\begin{aligned} \hat{f}^j(x^*) &\leqq \hat{f}^j(x^{**}) \\ &= f(x^{**}) + [\hat{f}^j(x^{**}) - f(x^{**})] \\ &\leqq f(x^{**}) + \varepsilon^j \end{aligned}$$

Since x^* is feasible for (1.1), the upper bound holds.

<u>6.4 Lemma</u>: Let $\{\lambda^j\}$ be such that $\lambda^j > 0$, $j=1,2,\ldots$ and $\{\lambda^j\} \to 0$. Then a point x^* is an optimal solution of (1.1) iff x^* is an optimal solution of each of the approximating problems $AP^j(x^*, \lambda^j)$, $j=1,2,\ldots$.

Proof:

($\Leftarrow$) By proposition (6.3),

$$f(x^*) - \varepsilon^j \leqq f(x^{**}) \leqq f(x^*), \ \forall j=1,2,\dots .$$

But by proposition (6.2), $\varepsilon^j \to 0$ as $\lambda^j \to 0$, and therefore, $f(x^*) = f(x^{**})$.

($\Rightarrow$) Since the objective function $\hat{f}^j$ of $AP^j(x^*,\lambda^j)$ satisfies $\hat{f}^j(x) \geq f(x), \forall x \in [\ell,u]$ and agrees with f at x^*, x^* is an optimal solution of each of the problems $AP^j(x^*,\lambda^j)$, $j=1,2,\dots$.

We are now ready to state our second algorithm to solve (1.1). At the jth iteration, we solve the approximating problem $AP^j(x^{*j-1},\lambda^j)$, where x^{*j-1} is the optimal solution of the previous iteration. (Both our algorithms (5.1) and (6.5) assume that a starting feasible solution is available. Various methods can be used to find an $x \in S$. In all our experiments, however, we solved an approximating problem of type (AP), (with a small number of segments per function) to obtain the initial feasible solution. If the original problem is infeasible, this is detected at the initial state.)

6.5 Algorithm:

Step 0: Let $\alpha \in (0,1)$ and let x^{*0} be a starting feasible solution. Let $0 < \lambda_i^0 \leq u_i - \ell_i, \forall i=1,2,\dots,n$. Set $j \leftarrow 1$.

Step 1: Let $\lambda_i^j = \alpha\lambda_i^{j-1} (i=1,2,\dots,n)$.

Step 2: Determine $x^{*j}(\lambda^j)$, an optimal solution of $AP^j(x^{*j-1}, \lambda^j)$.

Set $j \leftarrow j + 1$ and go to Step 1.

6.6 Theorem: Let Algorithm (6.5) generate a sequence of points $\{x^{*j}\}$ and let the corresponding sequence of dual multipliers associated with the constraints $Ax = b$ be $\{\mu^{*j}\}$. Then $f(x^{*j}) \rightarrow z^{**}$, the optimal objective value of (1.1). Each accumulation point of $\{x^{*j}\}$ is an optimal solution of (1.1). If, in addition, the right and left derivatives $f'_{i+}(\ell_i)$ and $f'_{i-}(u_i)$ $i=1,2,\ldots,n$ are finite, then the sequence $\{\mu^{*j}\}$ is bounded and each accumulation point of $\{\mu^{*j}\}$ is an optimal set of dual multipliers for (1.1).

Proof: See Appendix.

6.7 Theorem: Let Algorithm (6.5) generate a sequence of points $\{x^{*j}\}$ and let x^{**} be an optimal solution of (1.1). If

i) $f(x)$ is strongly convex on $[\ell,u]$

and ii) $f(x)$ is boundedly convex on $[\ell,u]$ or $\nabla f(x)$ is Lipschitz continuous on $[\ell,u]$,

then $\{x^{*j}\}$ converges to x^{**} linearly.

Proof: See Appendix.

6.8 Computational Aspects

While stating Algorithm 6.5, we have not mentioned how the subproblems $(AP)^j$ are to be solved. It is well-known that $(AP)^j$ can be converted to an equivalent linear program. Specifically, $AP^j(x^{*j-1}, \lambda^j)$ is equivalent to the following problem.

$$(\mathrm{LPAP})^j \qquad \min_{x_i^{+k},\,x_i^{-\ell}} \sum_{i=1}^{n} \sum_k c_i^{+k} x_i^{+k} - \sum_{i=1}^{n} \sum_\ell c_i^{-\ell} x_i^{-\ell}$$

$$\text{s.t.} \quad \sum_k Ax^{+k} - \sum_\ell Ax^{-\ell} = 0$$

$$0 \leq x^{+k} \leq \lambda^j$$

$$0 \leq x^{-\ell} \leq \lambda^j$$

where c_i^{+k} and $c_i^{-\ell}$ are as defined in (4.1.2).

The variables x_i^{+k} and $x_i^{-\ell}$ are the deviations of x_i from x_i^{*j-1} in incremental steps of λ_i^j, where it is understood that if x_i^{*j-1} is at its lower bound, then the variables $x_i^{-\ell}$ (corresponding to decrements from x_i^{*j-1}) would not be present in $(\mathrm{LPAP})^j$. Similar comments hold for x_i^{+k} if x_i^{*j-1} is at its upper bound. The number of variables x_i^{+k}, $x_i^{-\ell}$ vary depending on λ_i^j and also the value of x_i^{*j-1} in the interval $[\ell_i, u_i]$. (see Figure 6.2.)

In (1.1), the variable x_i corresponds to flow on an arc in the network, while in $(\mathrm{LPAP})^j$ the variables x_i^{+k}, $x_i^{-\ell}$ correspond to flows on parallel arcs between the same pair of nodes. Clearly, the computational efficiency of (6.5) depends on how efficiently the subproblems $(\mathrm{LPAP})^j$ are solved. As j (iteration number) increases, λ^j gets smaller and smaller and the number of variables x_i^{+k} and $x_i^{-\ell}$ increases. It would seem that an enormous number of function evaluations would be required to set up the approximation and because of their size, the subproblems $(\mathrm{LPAP})^j$ would be unmanageable. It is, however, not necessary to generate a global approximation of f_i. It suffices to work locally around x_i^{*j-1} and generate additional arcs only as needed, while simultaneously discarding arcs. It is therefore sufficient to work, at anytime, with at most two arcs x_i^+ and x_i^- for each of the variables x_i.

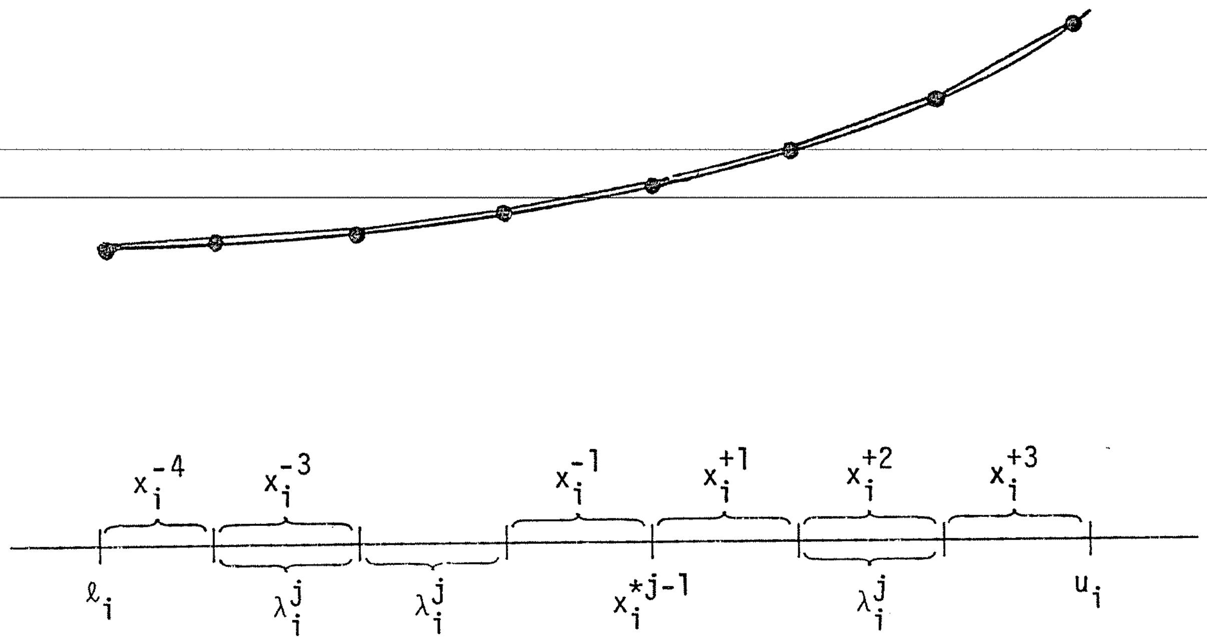

Fig. 6.2: Correspondence between segments and variables

<u>Strategy for modifying the current approximation</u>

Because of the convexity of f_i, we have

$$\ldots. \; c_i^{-2} \leq c_i^{-1} \leq c_i^{+1} \leq c_i^{+2} \; \ldots. \tag{6.8.1}$$

(Note, however, that in LPAP^j, the cost coefficient of $x_i^{-\ell}$ is $-c_i^{-\ell}$ and not $c_i^{-\ell}$.) It is clear from (6.8.1) that if x_i^{+1} prices out unfavorably, then so do all x_i^{+k}. Similar comments hold for x_i^{-1} and $x_i^{-\ell}$. We can, therefore, initially start with two variables x_i^+ and x_i^- with cost coefficients c_i^{+1} and $-c_i^{-1}$. It is again easy to see that if x_i^+ prices out favorably, then x_i^- cannot and vice versa. In the process of solving LPAP^j, suppose variable $x_i^+(x_i^-)$ reaches its temporary bound λ_i^j and becomes nonbasic. We can then modify the current value of x_i and then use $x_i^+(x_i^-)$ itself, as if it were a new variable corresponding to the immediately succeeding (preceding) segment in the approximation of $\hat{f}_i$. An additional functional evaluation of f_i is thus made at this stage and the cost coefficients are modified. If, however, $x_i^+(x_i^-)$ reaches its temporary bound λ_i^j, but remains basic, then the succeeding and preceding arcs do not price favorably and nothing need be done.

Pivoting in this fashion, we would either reach a stage where $x_i^+(x_i^-)$ no longer reaches its temporary bound λ_i^j, or the current value of x_i reaches its original upper bound $u_i(\ell_i)$. We can declare the current value as an optimal solution of LPAP^j if none of the variables x_i^+ and $x_i^-(i=1,2,\ldots n)$ price out favorably. The strategy thus is <u>column generation</u> combined with <u>column dropping</u>. Since the column generated is the same as the column dropped, from a computational viewpoint <u>columns are neither added nor dropped</u>.

In Algorithm (6.5), the grid size is reduced from iteration to iteration by a factor of α. It has been our experience that if α is set to 0.25, we need not move more than two or three segments from the current iterate. (Experimentation with $\alpha = 0.5$ has shown that the overall computing time does not change significantly, though the number of iterations to ε-optimality increases.) Thus, though the problems (LPAP)^j look formidable, they can be solved efficiently by the above computational strategy.

In (LPAP)^j for Algorithm 6.5, there are parallel arcs between every pair of nodes. We have experimented with pricing on multiple arcs successively. To be specific, suppose arc x_i^- reaches its lower bound, thus increasing the current value of x_i. Then, we can price out x_i^+ and select it as the entering variable if it prices favorably. Similarly, when an additional arc is added, it is possible to select this arc as the entering variable if it prices favorably. It has been our experience that such strategies do not help. In fact, experimentation has shown that such strategies increase the total number of pivots by 15% to 20%. In solving $(LPAP)^j$, we use a very simple unsophisticated pricing strategy as used in the primal network code RNET. It is possible that a suitable pricing strategy would improve the computational efficiency significantly.

6.9 Computing Lower Bounds on the Optimal Value

If the Lagrangian relaxation (5.2.1) can be solved in closed form, then it gives us an easy means for computing a lower bound on the optimal value. This, however, may not be possible in some problems.

Lemma (4.2.2) gives lower and upper bounds on the optimal value when a locally approximating problem $P(\hat{\ell},\hat{m},\hat{u})$ is solved. In the implicit grid method, after solving $LPAP^j$, we have available a local 2-segment approximation for which also x^{*j} (the optimal solution of $LPAP^j$) is optimal. Lemma (4.2.2) requires that x^{*j} be not artificially bounded. Note that in the implicit grid method nonbasic variables are never artifically bounded. If in x^{*j}, a basic variable x_i is at its temporary bound λ_i^j, then adding an additional (succeeding or preceding) segment does not affect the optimality of x^{*j}. Thus, a 2-segment approximation is on hand, such that x^{*j} is optimal for this approximation and further x^{*j} is not artifically bounded.

In order to compute the lower bound, we need to estimate $\hat{e}_i$, as given by (4.2.1). If the functions f_i are differentiable, then we can compute the linear support to f_i at x^{*j} and estimate $\hat{e}_i$ as the maximum deviation between this linear support and the 2-segment approximation.

Computationally, this requires only $\nabla f(x^{*j})$ and no further function evaluations are needed.

For all the test problems given in Section 7, we have computed the lower bounds by both Lagrangian relaxation and this method. Close to an optimal solution, the two lower bounds agreed up to 6 or 7 figures. We thus have a lower bounding technique which would be highly useful in situations where Lagrangian relaxations are difficult to solve. [Meyer 80] describes yet another approach that uses only function values, but is less accurate than the methods used here.

7. Computational Results

This section describes the test problems solved in this study, and ~~presents computational experience with both the Algorithms (5.1) and (6.5).~~ Algorithm (6.5) was implemented as per the strategies described in Section (6.8). Algorithm (5.1) was tested with the local approximations consisting of 2, 4 and 6 segments. The slopes of all the segments were computed and stored explicitly. It is, however, possible to implement Algorithm (5.1) also by generating additional segments only as needed. In all cases, the starting feasible solution was obtained in the first iteration, wherein a linearized subproblem ($LPAP^1$) was solved with a full artificial basis.

The linearized subproblems were solved by a suitably modified version of RNET [Grigoriadis and Hsu 79]. RNET was modified so as to maintain the cost coefficients and dual variables in double precision. All the test problems were run with the same partial pricing frequency. (The RNET pricing frequency was set to 4.0.) The test problems were run using the FORTRAN 77 compiler (with optimization) on the VAX 11/780 running under the UNIX operating system. The execution times reported are in CPU seconds exclusive of input and and include output time. (The output was only two lines per iteration).

7.1 Quadratic Data Fitting Problems

Except for the first problem (with 100 arcs), these problems were generated from actual data on input-output tables of the West German economy. The data was obtained from [Bachem and Korte 77]. The problem is to adjust the input-output coefficients of the previous year given the row and column sums for the current year, with an additional constraint that if an input-output coefficient in the previous year is zero, it must be zero for the current year as well. The problem can be formulated as

$$\min_{x_{ij}} \sum_i \sum_j (x_{ij}-t_{ij})^2$$

$$\text{s.t.} \quad \sum_{i=1}^{k} x_{ij} = d_j, \quad i=1,\ldots,k$$

$$\sum_{j=1}^{k} x_{ij} = s_i, \quad i=1,\ldots,k$$

$$0 \le x_{ij} \le u_{ij}, \quad i,j=1,\ldots,k$$

with the understanding that some arcs in this transportation problem may not be present. The upper bound u_{ij} for the ijth arc is determined as $u_{ij} = \min \{t_{ij}+1000, 100\times t_{ij}\}$, where t_{ij} is the target flow on the ijth arc. The numerical values of t_{ij} range from 1 to 6500. For these problems, computation was terminated if the lower and upper bounds on the optimal value agree up to 7 figures. (See Table 7.1.)

7.2 Reservoir Control Problems

These are the problems described in [Bertsekas 76]. The problem is to schedule water release from a reservoir subject to upper and lower bounds on the total volume of water in the reservoir. These problems can be formulated as separable nonlinear network flow problems. We have tested two sets (four problems in each set) of these problems; the first with quadratic objective function terms and the second with exponential objective function terms. In all cases, computation was terminated if the lower and upper bounds on the optimal value agree up to 6 figures. Computational results with quadratic objectives are presented in Table 7.2 and the computational results with exponential objective functions are presented in Table 7.3.

7.3 Water Distribution Problems

The third class of problems are pipe flow problems as described by [Collins, et al 78]. The objective function consists of a few linear terms and many nonlinear terms (a few integrals and the rest of the form $c_i|x_i|^{2.85}$). We have tested three problems of this type. These problems were obtained from [Dembo and Klincewicz 78]. For all the 3 problems, computation was terminated if the lower and upper bounds agree up to 7 figures and the computational results are as shown in Table 7.4.

TABLE 7.1: QUADRATIC DATA FITTING PROBLEMS

PROBLEM		ALGORITHM 6.5 (IMPLICIT GRID)	ALGORITHM 5.1			BEST KNOWN LOWER BOUND
			2-SEGMENTS	4-SEGMENTS	6-SEGMENTS	
100 ARCS 20 NODES	OBJ. VALUE	6600.0001	6600.0001	6600.0002	6600.0001	6599.9999
	CPU TIME(SEC)	3.233	5.967	5.200	7.149	
	# ITERATIONS	7	13	9	9	
	# PIVOTS	1026	1850	1640	2053	
144 ARCS 24 NODES	OBJ. VALUE	7302243.43	7302243.70	7302243.77	7302243.37	7302243.20
	CPU TIME(SEC)	7.117	9.683	12.734	14.407	
	# ITERATIONS	7	14	13	12	
	# PIVOTS	2193	2707	3760	4116	
144 ARCS 24 NODES	OBJ. VALUE	2714896.11	2714898.11	2714896.52	2714896.17	2714895.87
	CPU TIME(SEC)	7.867	10.183	12.982	17.100	
	# ITERATIONS	7	13	13	13	
	# PIVOTS	2445	2900	3636	4536	
2202 ARCS 109 NODES	OBJ. VALUE	5634201.59	5634201.78	not run	not run	5634201.37
	CPU TIME(SEC)	258.700	312.367			
	# ITERATIONS	7	15			
	# PIVOTS	53498	53477			
2238 ARCS 109 NODES	OBJ. VALUE	12022985.31	12022985.79	not run	not run	12022984.64
	CPU TIME(SEC)	239.834	320.350			
	# ITERATIONS	7	15			
	# PIVOTS	46672	53148			

TABLE 7.2: RESERVOIR CONTROL PROBLEMS (QUADRATIC OBJECTIVE TERMS)

PROBLEM		ALGORITHM 6.5 (IMPLICIT GRID)	ALGORITHM 5.1			BEST KNOWN LOWER BOUND
			2-SEGMENTS	4-SEGMENTS	6-SEGMENTS	
23 ARCS 13 NODES	OBJ. VALUE	-1975.6491	-1975.6491	-1975.6491	-1975.6491	-1975.6492
	CPU TIME(SEC)	.300	.417	.433	.500	
	# ITERATIONS	6	12	11	10	
	# PIVOTS	47	50	52	57	
103 ARCS 53 NODES	OBJ. VALUE	-8731.0245	-8731.0257	-8731.0257	-8731.0257	-8731.0261
	CPU TIME(SEC)	1.116	1.650	1.866	2.217	
	# ITERATIONS	6	13	12	11	
	# PIVOTS	219	279	248	330	
207 ARCS 105 NODES	OBJ. VALUE	-17393.533	-17393.553	-17393.553	-17393.553	-17393.555
	CPU TIME(SEC)	2.467	3.384	3.817	4.300	
	# ITERATIONS	6	12	11	10	
	# PIVOTS	492	530	569	554	
729 ARCS 366 NODES	OBJ. VALUE	-60750.487	-60750.488	-60750.488	-60750.488	-60750.491
	CPU TIME(SEC)	12.316	17.450	24.500	30.750	
	# ITERATIONS	6	13	12	11	
	# PIVOTS	1993	2007	1912	2319	

TABLE 7.3: RESERVOIR CONTROL PROBLEMS (EXPONENTIAL OBJECTIVE TERMS)

PROBLEM		ALGORITHM 6.5 (IMPLICIT GRID)	ALGORITHM 5.1			BEST KNOWN LOWER BOUND
			2-SEGMENTS	4-SEGMENTS	6-SEGMENTS	
	OBJ. VALUE	12.6411	12.6412	12.6412	12.6412	12.6411
23 ARCS	CPU TIME(SEC)	.400	.667	.683	.700	
13 NODES	# ITERATIONS	6	13	10	10	
	# PIVOTS	40	50	44	49	
	OBJ. VALUE	56.5602	56.5603	56.5602	56.5602	56.5601
103 ARCS	CPU TIME(SEC)	1.450	2.400	2.700	2.917	
53 NODES	# ITERATIONS	6	13	11	10	
	# PIVOTS	215	242	245	264	
	OBJ. VALUE	124.758	124.758	124.758	124.758	124.7578
207 ARCS	CPU TIME(SEC)	3.050	5.433	5.800	5.933	
105 NODES	# ITERATIONS	6	16	12	10	
	# PIVOTS	456	541	532	507	
	OBJ. VALUE	476.266	476.266	476.266	476.266	476.266
729 ARCS	CPU TIME(SEC)	16.650	25.316	37.650	40.717	
366 NODES	# ITERATIONS	7	15	12	12	
	# PIVOTS	1841	1910	1872	2125	

TABLE 7.4: WATER DISTRIBUTION PROBLEMS

PROBLEM		ALGORITHM 6.5 (IMPLICIT GRID)	ALGORITHM 5.1			BEST KNOWN LOWER BOUND
			2-SEGMENTS	4-SEGMENTS	6-SEGMENTS	
	OBJ. VALUE	-32392.731	-32392.730	-32392.730	-32392.730	-32392.731
46 ARCS	CPU TIME(SEC)	2.300	3.883	6.383	5.967	
30 NODES	# ITERATIONS	7	15	13	11	
	# PIVOTS	414	347	580	589	
	OBJ. VALUE	-48199.858	-48199.858	-48199.859	-48199.859	-48199.859
196 ARCS	CPU TIME(SEC)	11.633	20.567	31.184	32.167	
150 NODES	# ITERATIONS	7	17	17	14	
	# PIVOTS	1823	1719	2888	2753	
	OBJ. VALUE	-206175.21	-206175.21	-206175.22	-206175.22	-206175.24
906 ARCS	CPU TIME(SEC)	73.834	172.584	163.617	195.400	
666 NODES	# ITERATIONS	8	23*	15	14	
	# PIVOTS	12166	15755	15036	18185	

*COMPUTATION TERMINATED AFTER 23 ITERATIONS.

8. Conclusions

The numerical results of the preceding section indicate that the implicit grid strategy produces in 6-8 iterations highly accurate solutions for the broad set of test problems ranging from 23 to more than 2000 variables. With a fixed grid strategy, 7-23 iterations are required to achieve comparable accuracy. Since the time per iteration for the implicit grid method is generally comparable to or only slightly higher than the simplest of the fixed grid approaches (which is generally the best of the fixed grid methods), overall computing times are consistently better for the implicit grid approach. Further significant improvements in computing times appear attainable through the reduction of the time spent in pivoting by exploitation of pricing and basis preservation strategies.

Appendix

In this appendix, we present the proofs of Theorems 6.6 and 6.7. First, we give the definitions of strongly convex and boundedly convex functions, which were introducted by [Wolfe 70].

Def: Let $f(x)$ be a twice differentiable function from $R^n \to R^1$. $f(x)$ is strongly convex on $[\ell,u]$ with constant $q > 0$, iff q is a uniform lower bound on all the eigen values of the Hessian matrix $H(x) = \left[\frac{\partial^2 f(x)}{\partial x_i \partial x_j}\right]$ for $x \in [\ell,u]$. $f(x)$ is boundedly convex on $[\ell,u]$ with constant Q, if it is convex on $[\ell,u]$ and Q is a uniform upper bound on all the eigen values of the Hessian matrix for $x \in [\ell,u]$.

Before proving Theorem 6.6, we would like to establish the following lemma.

Lemma: Let Algorithm 6.5 generate a sequence of points $\{x^{*j}\}$ and let $\{\mu^{*j}\}$ be the corresponding sequence of dual multipliers associated with the constraints $Ax = b$. If the right and left derivatives $f'_{i+}(\ell_i)$ and $f'_{i-}(u_i)$ are finite for all $i=1,2,\dots n$, then the sequence $\{\mu^{*j}\}$ is bounded.

Proof: As shown in Section 6.8, each of the problems $AP^j(x^{*j-1}, \lambda^j)$ can be converted to an LP(LPAP^j). Given an optimal basis matrix of (LPAP)^j, we define a basis matrix of (1.1) as follows.

If any of the variables x_i^{+k} or $x_i^{-\ell}$ is basic in LPAP^j, then define x_i to be basic for (1.1). Let B^j be the corresponding basis submatrix of A. Then, the optimal multipliers can be expressed as

$\mu^{*j} = c_B^{j^T} B^{j^{-1}}$, where $c_B^j = (c_1^j, c_2^j, \ldots c_n^j)^T$ and each of c_i^j is the slope of a segment of $\hat{f}_i^j$.

~~For $\ell_i \leq y < z \leq u_i$, by the convexity of f_i, we have~~

$$f'_{i+}(\ell_i) \leq f'_{i+}(y) \leq \frac{f_i(z)-f_i(y)}{z-y} = \frac{f_i(y)-f_i(z)}{y-z} \leq f'_{i-}(z) \leq f'_{i-}(u)$$

Thus c_B^j remains bounded for all j as $j \to \infty$.

Since

$$\|\mu^{*j}\| \leq \|c_B^j\| \; \|B^{j^{-1}}\|,$$

and there are only finitely many basis submatrices of A, the sequence $\{\mu^{*j}\}$ remains bounded. ▲

Proof of Theorem 6.6: For convenience of notation, we rewrite the constraint set $S := \{x \mid Ax=b,\ \ell \leq x \leq u\}$ as $S' := \{x \mid h(x)=0,\ x \in X\}$ where $h: R^n \to R^m$ is affine and $X = \{x \mid \ell \leq x \leq u\}$.

We have that x^{*j} and μ^{*j} are optimal primal and dual solutions of $AP^j(x^{*j-1}, \lambda^j)$, and hence $\{x^{*j}, \mu^{*j}\}$ satisfy the saddle point necessary optimality conditions:

$$\hat{f}^j(x^{*j}) + \langle \mu, h(x^{*j}) \rangle \leqq \hat{f}^j(x^{*j}) + \langle \mu^{*j}, h(x^{*j}) \rangle \leqq \hat{f}^j(x) + \langle \mu^{*j}, h(x) \rangle \qquad (A.1)$$

where the first inequality holds for all $\mu \in R^m$ and the second inequality holds for all $x \in X$. Since $f(x) \leqq \hat{f}^j(x)$ and $\hat{f}^j(x) \leqq f(x) + \varepsilon^j$, $\forall x \in X$, from the second inequality of (A.1), we have

$$f(x^{*j}) + \langle \pi^{*j}, h(x^{*j}) \rangle \leqq f(x) + \varepsilon^j + \langle \pi^{*j}, h(x) \rangle, \quad \text{for all } x \in X \qquad (A.2)$$

The sequence $\{x^{*j}\}$ is always bounded and by the previous lemma, $\{\mu^{*j}\}$ is also bounded. Let $\{x^{**}, \mu^{**}\}$ be an accumulation point of $\{x^{*j}, \mu^{*j}\}$.

By proposition (6.2), $\varepsilon^j \to 0$ as $j \to \infty$. Thus, for any fixed $x \in X$, as $j \to \infty$, by the continuity of f and h,

$$f(x^{**}) + \langle \mu^{**}, h(x^{**}) \rangle \leqq f(x) + \langle \mu^{**}, h(x) \rangle \tag{A.3}$$

Since the LHS of (A.2) is independent of x, (A.3) holds for all $x \in X$.

By the continuity of h, $h(x^{**}) = 0$ and we have for all $\mu \in R^m$,

$$f(x^{**}) + \langle \mu, h(x^{**}) \rangle \leqq f(x^{**}) + \langle \mu^{**}, h(x^{**}) \rangle \tag{A.4}$$

(A.3) and (A.4) together constitute the saddle point sufficient optimality conditions for (1.1). ▲

In order to prove the convergence rate result as stated in Theorem 6.7, we need the following lemma, [see, e.g., Thakur 78], which gives the maximum error of approximation in approximating a boundedly convex function (or a function with a Lipschitz continuous derivative) by a piecewise-linear function.

Lemma: Let $g: [a,b] \to R$ be twice differentiable on $[a,b]$. Let $\hat{g}(x)$ be a piecewise-linear approximation of $g(x)$ in $[a,b]$, with a subdivision interval length λ. If $g(x)$ satisfies the condition

i) $|g''(x)| \leqq Q \quad \forall x \in [a,b]$,

or ii) $|g'(x) - g'(y)| \leqq Q|x-y| \quad \forall x, y \in [a,b]$.

Then,

$$\max_{x \in [a,b]} |g(x) - \hat{g}(x)| \leqq Q\lambda^2/8 .$$

Proof of Theorem 6.7: By the strong convexity of f, we have

$$f(y) - f(x) - \nabla f(x)^T(y-x) \geqq \frac{q}{2}\|y-x\|^2, \ \forall x,\ y \in [\ell,u] .$$

Let $\{x^{*j}\}$ be the sequence of points generated by Algorithm 6.5. Let x^{**} be an optimal solution of (1.1). From the above inequality, we get

$$f(x^{*j}) - f(x^{**}) - \nabla f(x^{**})^T(x^{*j} - x^{**}) \geqq \frac{q}{2}\|x^{*j} - x^{**}\|^2 .$$

Since x^{**} is optimal for (1.1) and x^{*j} is feasible for (1.1), by the minimum principle, we have that

$$\nabla f(x^{**})^T(x^{*j} - x^{**}) \geqq 0 .$$

Therefore, $$f(x^{*j}) - f(x^{**}) \geqq \frac{q}{2}\|x^{*j} - x^{**}\|^2$$

or $$\|x^{*j} - x^{**}\|^2 \leqq \frac{2}{q}\{f(x^{*j}) - f(x^{**})\} .$$

Combining the above lemma with proposition (6.2), we have

$$f(x^{*j}) - f(x^{**}) \leqq \sum_{i=1}^{n} \varepsilon_i^j \leqq \frac{Q\|\lambda^j\|^2}{8} ,$$

where λ^j is the subdivision interval length in the jth iteration.

Thus, $$\|x^{*j} - x^{**}\|^2 \leqq \frac{2}{q} \; \frac{Q\|\lambda^j\|^2}{8}$$

Since $\lambda^j = \alpha\lambda^{j-1}$, $j=1,2,\ldots,$ we have

$$\|x^{*j} - x^{**}\| \leqq \left[\left(\frac{Q}{4q}\right)^{\frac{1}{2}} \|\lambda_0\| \frac{1}{\alpha}\right] \alpha^j, \; j=1,2,\ldots$$

where $0 < \alpha < 1$. Thus, the sequence $\{x^{*j}\}$ converges to x^{**} linearly at rate α from the very first iteration.

References

1. Bachem, Achim and Korte, Bernhard, "Mathematical Programming and the Estimation of Input-Output Matrices," in Mathematical Programming and its Economic Application" (G. Castellani and P. Mazzolani, eds.), Angeli, Milano, 1981.

2. Bazaraa, M. S. and Shetty, C. M., "Nonlinear Programming, theory and algorithms", John Wiley & Sons, 1979.

3. Beck, P., Lasdon, L., and Engquist, M., "A reduced gradient algorithm for nonlinear network flow problems", ACM Transactions on Mathematical Software, 9, 57-70, 1983.

4. Bertsekas, D. P., "On the Goldstein-Levitin-Poljak gradient projection method", IEEE Transactions on Automatic Control, vol. AC-21, 2, 1976.

5. Collins, M., Cooper, L., Helgason, R., Kennington, J., and Leblanc, L., "Solving the pipe network analysis problems using optimization techniques", Man. Sci., 24, pp. 747-760, 1978.

6. Cottle, Richard W. and Duvall, Steven G., "A Lagrangian Relaxation Algorithm for the Constrained Matrix Problem", Tech. Rpt. SOL 82-10, Stanford University, Stanford, 1982.

7. Dembo, R. and Klincewicz, J., "An approximate second order algorithm for network flow problems with convex separable costs", Series #21, School of Business, Yale University, New Haven, Conn., 1978.

8. Grigoriadis, M. D. and Hsu, Tau, "RNET the Rutgers minimum cost network flow subroutine", SIGMAP Bulletin, pp. 17-18, 1979.

9. Jensen, Paul A. and Barnes, Wesley J., "Network Flow Programming", John Wiley and Sons, 1980.

10. Kao, C. Y. and Meyer, R. R., "Secant approximation methods for convex optimization", Mathematical Programming Study 14, pp. 143-162, 1981.

11. Meyer, R. R., "Computational aspects of two-segment separable programming", Technical Report #382, Computer Sciences Department, University of Wisconsin, Madison, 1980.

12. Rockafellar, R. T., "Optimization in Networks", Lecture Notes, University of Washington, Seattle, 1976.

13. Rockafellar, R. T., "Monotropic Programming", Nonlinear Programming 4, Academic Press, 1981.

14. Thakur, L. S., "Error Analysis for convex separable programs", SIAM Journal of Applied Mathematics, pp. 704-714, 1978.

15. Wolfe, P., "Convergence theory in nonlinear programming", Integer and Nonlinear Programming, ed. J. Abadie, North Holland, pp. 1-36, 1970.